DB2® ユニバーサル・データベース

コール・レベル・インターフェースの手引き および解説書

バージョン 7

SC88-8517-00
(英文原典：SC09-2950-00)

DB2® ユニバーサル・データベース

コール・レベル・インターフェースの手引き および解説書

バージョン 7

SC88-8517-00
(英文原典：SC09-2950-00)

ご注意!

本書、および本書がサポートする製品をご使用になる前に、 947ページの『付録M. 特記事項』にある一般的な情報を必ずお読みください。

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミング、またはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミング、またはサービスを、日本で発表する意図があることを必ずしも示すものではありません。

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)

原 典:	SC09-2950-00 IBM® DB2® Universal Database Call Level Interface Guide and Reference Version 7
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 2000.6

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

本書について

本書には、DB2 コール・レベル・インターフェースを使用してアプリケーションを作成し、IBM DB2 ファミリーのサーバーにアクセスするのに必要な情報が記載されています。また、ODBC を使用して IBM DB2 ファミリーのサーバーにアクセスするアプリケーションを (ODBC ソフトウェア開発キットを使って) 作成する場合にも、補足情報として利用する必要があります。

本書で「DB2」に言及している場合、製品バージョン番号が付いていてもいなくても「DB2 ユニバーサル・データベース」製品を示しています。その他のプラットフォーム上の DB2 に言及する場合は、特定のプロダクト名 (たとえば、DB2 (OS/390 版) または DB2 (AS/400 版など) を用いています。

本書の対象読者

SQL および「C」または「C++」プログラミング言語の知識がある DB2 アプリケーション・プログラマー。

SQL および「C」または「C++」プログラミング言語の知識がある ODBC アプリケーション・プログラマー。

本書の構成

本書は、以下の章に分かれています。

- 1ページの『第1章 CLI の紹介』では、DB2 CLI について紹介し、インターフェースの背景および組み込み SQL との関係について述べます。
- 11ページの『第2章 DB2 CLI アプリケーションの作成』では、典型的な DB2 CLI アプリケーションについて概説します。この章では、簡単な DB2 CLI アプリケーション内で実行される基本的なタスクまたはステップについて述べます。一般的な概念に加え、基本関数およびそれら相互間の対話についても述べます。
- 45ページの『第3章 拡張機能の使用』では、さらに拡張されたタスクおよびそれらを実行するのに使用する関数について概説します。
- 155ページの『第4章 CLI の構成およびサンプル・アプリケーションの実行』には、 DB2 CLI アプリケーションをコンパイルおよび実行するのに必要な環境をセットアップするための情報が含まれています。ユーザーの環境を検査するため、また独自のアプリケーションを開発するのに役立つテンプ

レートを提供するために、サンプル・アプリケーションが用意されています。 CLI/ODBC 構成キーワードとその意味の一覧表もこの章に含まれています。

- 233ページの『第5章 DB2 CLI 関数』では、DB2 CLI を構成している関数を解説しています。
- 付録は次のとおりです。
  - 811ページの『付録A. プログラミングのヒントと提案』では、DB2 CLI アプリケーションのパフォーマンスや可搬性を向上させるためのいくつかの共通のヒントおよび提案を示します。
  - 821ページの『付録B. アプリケーションの移行』では、直前のリリースから何が変更されたか、および既存のアプリケーションを移行するのに必要な互換性および必須ステップについて要約します。
  - 835ページの『付録C. DB2 CLI と ODBC』では、ODBC と DB2 CLI の違いについて述べます。
  - 841ページの『付録D. 拡張スカラー関数』では、 DB2 関数としてアクセスしたり ODBC ベンダー・エスケープ文節を使用できるスカラー関数について述べます。
  - 855ページの『付録E. SQLSTATE 相互参照』には SQLSTATE 表があり、この表には個々の SQLSTATE を生成する関数がリストされています。
    (233ページの『第5章 DB2 CLI 関数』の関数の説明のところに、各関数で返される可能性のある SQLSTATE をリストしてあります。)
  - 875ページの『付録F. データ変換』には、SQL および C のデータ・タイプ、両者間での変換についての説明があります。
  - 895ページの『付録G. ストアード・プロシージャーのカタログ表示』には、ストアード・プロシージャーおよびその属性に関する情報を保管するために使用するカタログ表についての説明があります。
  - 897ページの『付録H. ストアード・プロシージャー登録の疑似カタログ表』では、 DB2CLI.PROCEDURES 疑似カタログ表の作成と保守の方法について説明します。
  - 901ページの『付録I. サポートされている SQL ステートメント』には、DB2 ユニバーサル・データベースでサポートされている SQL ステートメントのリストおよび DB2 CLI でサポートされている SQL (構造化照会言語) のサブセットが示されています。
  - 907ページの『付録J. CLI サンプル・コード』には、具体的な例に関する完全なソースがリストされています。

– 915ページの『付録K. DB2 CLI/ODBC/JDBC トレース機能の使用』には、トレース・ファイル形式、非同期アプリケーションのトレース、および関連トピックに関する情報があります。

第1章 CLI の紹介

DB2 コール・レベル・インターフェース (DB2 CLI) は、データベース・サーバーの DB2 ファミリーに対する IBM の呼び出し可能な SQL インターフェースです。これは、関係データベース・アクセス用の 'C' および 'C++' アプリケーション・プログラミング・インターフェースで、関数呼び出しを使用して、動的 SQL ステートメントを関数引き数として渡します。これは組み込み動的 SQL の代替方法ですが、組み込み SQL とは違って、ホスト変数またはプリコンパイラーを必要としません。

DB2 CLI は、 Microsoft** オープン・データベース・コネクティビティー (Open Database Connectivity** (ODBC)) 仕様、および SQL/CLI 用国際規格 (International Standard for SQL/CLI) に基づいています。業界の標準に従う努力の一環として、これらの仕様が DB2 コール・レベル・インターフェース の基盤として採用されました。これは、上記のデータベース・インターフェースのいずれかについてすでに精通しているアプリケーション・プログラマーが短期間で学習できるようにするためです。さらに、複数の DB2 特定の拡張が追加されており、アプリケーション・プログラマーが DB2 機能を特に活用するのに役立ちます。

DB2 CLI ドライバーは、ODBC ドライバー・マネージャーによってロードされる際、 ODBC ドライバーとしても働きます。この ODBC ドライバーは、ODBC 2.0 のレベル 2、および ODBC 3.0 のレベル 1 に適合しています。さらに、種々の ODBC 3.0 のレベル 2 インターフェース適合項目 (202、 203、 205、 207、 209、および 211) にも合致しています。 ODBC サポートおよびレベル 2 インターフェースの適合項目に関する情報は、 835ページの『付録C. DB2 CLI と ODBC』に示されています。

DB2 CLI の背景情報

DB2 CLI または呼び出し可能 SQL インターフェースを理解するには、それが何に基づいているのかを理解し、それを既存のインターフェースと比較するとわかりやすくなります。

X/Open Company と SQL アクセス・グループは共同で、 *X/Open コール・レベル・インターフェース* と呼ばれる呼び出し可能 SQL インターフェースの仕様を開発しました。このインターフェースの目標は、アプリケーションがいずれか 1 つのデータベース・ベンダーのプログラミング・インターフェースから

独立できるようにすることによって、アプリケーションの可搬性を高めることです。 X/Open コール・レベル・インターフェース仕様のほとんどは、 ISO コール・レベル・インターフェース国際規格 (ISO/IEC 9075-3:1995 SQL/CLI) の一部として受け入れられています。

Microsoft 社は、X/Open CLI の準備草案に基づいて、 Microsoft オペレーティング・システム用のオープン・データベース・コネクティビティー (ODBC) と呼ばれる呼び出し可能 SQL インターフェースを開発しました。

バージョン 3 の ODBC は、ISO SQL/CLI のほとんどに適合しています。 ODBC 3.0 には、国際規格となっていない機能もかなり含まれていますが、この多くは次回の規格草案に加えられることになっています。

また、ODBC 仕様には、接続要求時に与えられるデータ・ソース (データベース名) に基づいて、ドライバー・マネージャーによってデータベース特定の ODBC ドライバーが実行時に動的にロードされる操作環境が含まれています。 アプリケーションは、各 DBMS のライブラリーではなく、単一のドライバー・マネージャーのライブラリーに直接リンクされます。ドライバー・マネージャーは、アプリケーションの関数呼び出しを実行時に仲介して、それが該当する DBMS 特定の ODBC ドライバーに確実に宛てられるようにします。 ODBC ドライバー・マネージャーは、ODBC 特定の関数だけを認識しているので、 DBMS 特定の関数は ODBC 環境ではアクセスできません。 DBMS 特定の動的 SQL ステートメントは、 151ページの『ベンダー・エスケープ文節の使用』で説明されているエスケープ文節と呼ばれるメカニズムによってサポートされます。

ODBC は、Microsoft オペレーティング・システムに限られるものではなく、他のインプリメンテーションをさまざまなプラットフォームで利用できます。

DB2 CLI ロード・ライブラリーは、 ODBC ドライバーとして ODBC ドライバー・マネージャーによってロードできます。 ODBC アプリケーションの開発の際には、 ODBC ソフトウェア開発キットを (Microsoft プラットフォームの場合は Microsoft から、 Microsoft 以外のプラットフォームの場合はその他のベンダーから) 入手してください。 DB2 サーバーに接続可能な ODBC アプリケーションを開発する場合、このマニュアル (DB2 特定の拡張に関する情報および診断情報) と、 *ODBC 3.0 Programmer's Reference and SDK Guide* を併せてご使用ください。

DB2 CLI に対して直接記述されたアプリケーションは、DB2 CLI ロード・ライブラリーに直接リンクします。 DB2 CLI では、DB2 特定の関数はもとよ

り、複数の ODBC および ISO SQL/CLI 関数のサポートが含まれています。サポートされる関数のリストについては、235ページの『DB2 CLI 関数の要約』を参照してください。

DB2 CLI と ODBC との関連の詳細については、835ページの『付録C. DB2 CLI と ODBC』を参照してください。

次の DB2 機能は、ODBC と DB2 CLI の両方のアプリケーションで利用可能です。

- 2 バイトの (図形) データ・タイプ
- ストアード・プロシージャー
- 分散作業単位 (DUOW)、2 フェーズ・コミット
- 複合 SQL
- ユーザー定義タイプ (UDT)
- ユーザー定義関数 (UDF)

また DB2 CLI には、 ODBC アプリケーションによってアクセスできない DB2 機能にアクセスするための次の拡張が含まれています。

- ラージ・オブジェクト (LOB) および LOB ロケーターのサポート
- 詳細な DB2 特定診断情報に関する SQLCA アクセス

DB2 CLI と組み込み SQL との違い

組み込み SQL インターフェースを使用するアプリケーションには、 SQL ステートメントをコードに変換するプリコンパイラーが必要で、変換後そのコードはコンパイルされ、データベースにバインドされ、実行されます。対照的に、DB2 CLI アプリケーションは、プリコンパイルまたはバインドする必要がなく、代わりに関数の標準セットを使用して実行時に SQL ステートメントおよび関連サービスを実行します。

この違いは重要です。というのはこれまでプリコンパイラーは、個々のデータベース製品に特定のものであったからです。これは効果的にユーザーのアプリケーションをその製品に結び付けるものでした。 DB2 CLI を使用すると、どの特定のデータベース製品からも独立した可搬性のあるアプリケーションを作成することが可能になります。この独立性によって、 DB2 CLI アプリケーションはさまざまな DB2 データベース (DRDA データベースを含む) にアクセスするために再コンパイルまたは再バインドする必要がなく、ただ該当するデータベースに実行時に接続するだけで済むようになります。

組み込み SQL と DB2 CLI との比較

DB2 CLI と組み込み SQL は、次の点でも異なります。

- DB2 CLI では、カーソルの明示宣言は必要ありません。必要に応じて DB2 CLI で生成されます。そして、アプリケーションはその生成されたカーソルを通常のカーソル取り出しモデルとして、複数行の SELECT ステートメント、および定位置 UPDATE と DELETE ステートメント用に使用します。
- OPEN ステートメントは DB2 CLI では使用しません。その代わりに、SELECT の実行によって自動的にカーソルがオープンされます。
- 組み込み SQL とは違って、DB2 CLI では、 EXECUTE IMMEDIATE ステートメントに相当するステートメント (SQLExecDirect() 関数) でパラメーター・マーカーの使用が可能です。
- DB2 CLI の COMMIT または ROLLBACK は、 SQL ステートメントとして渡されるのではなく、 SQLEndTran() 関数呼び出しによって発行されます。
- DB2 CLI はステートメント関連情報をアプリケーションのために管理し、*ステートメント・ハンドル* を提供してそれを要約オブジェクトとして参照できるようにします。このハンドルによって、アプリケーションが製品特有のデータ構造を使用する必要がなくなります。
- ステートメント・ハンドルと同様に、*環境ハンドル* および*接続ハンドル* が、すべてのグローバル変数および接続特定情報を参照する方法として備えられています。*記述子ハンドル* は、 SQL ステートメントのパラメーターか、結果セットの列のどちらかの状況を記述します。
- DB2 CLI は、X/Open SQL CAE 仕様で定義された SQLSTATE 値を使用します。この形式および値のほとんどは、 IBM 関係データベース製品で使用する値と一貫性がありますが、違いもあります。 (ODBC SQLSTATES と X/Open 定義の SQLSTATES の間にも違いがあります。) すべての DB2 CLI SQLSTATE の相互参照は、855ページの『SQLState 相互参照表』を参照してください。
- DB2 CLI はスクロール可能カーソルをサポートします。スクロール可能カーソルを使用すると、静的な読み取り専用のカーソルによって次のようなスクロールが可能になります。
  - 1 行または複数行ごとに下方へ
  - 1 行または複数行ごとに上方へ
  - 最初の行から 1 行または複数行ごとに
  - 最後の行から 1 行または複数行ごとに

上記の違いは別にして、組み込み SQL と DB2 CLI には次の重要な共通の概念があります。 *DB2 CLI は、組み込み SQL で動的に作成できる SQL ステートメントを実行することができます。*

注: さらに、DB2 CLI は複合 SQL ステートメントのような、動的に準備できない一部の SQL ステートメントも受け入れることができます。

901ページの表215 に各 SQL ステートメントをリストし、 DB2 CLI を使用して実行できるかどうかを示してあります。また、表には、コマンド行プロセッサーを使用してステートメントを対話式で実行できるかどうかも示してあります (これは SQL ステートメントをプロトタイピングするのに便利です) 。

各 DBMS には動的に作成できるステートメントがさらにある場合もありますが、この場合には DB2 CLI がステートメントを DBMS に渡します。 1 つの例外があります。一部の DBMS では COMMIT および ROLLBACK ステートメントを動的に準備できますが渡されることはありません。この場合は SQLEndTran() 関数を使用して、 COMMIT または ROLLBACK ステートメントのいずれかを指定する必要があります。

DB2 CLI を使用する場合の利点

DB2 CLI インターフェースには、組み込み SQL よりも優れている点がいくつかあります。

- これはクライアント・サーバー環境に理想的です。この環境では、アプリケーションの作成時には宛先データベースは不明です。アプリケーションにどのデータベース・サーバーが接続するかに関係なく、 SQL ステートメント実行用の一貫性のあるインターフェースが得られます。
- プリコンパイラーへの従属性が排除されているので、アプリケーションの移植性が高まります。アプリケーションは、データベース製品ごとに事前処理する必要がある組み込み SQL ソース・コードとしてではなく、コンパイル済みアプリケーションまたは実行時ライブラリーとして配布されます。
- 個々の DB2 CLI アプリケーションを各データベースにバインドする必要はなく、すべての DB2 CLI アプリケーションについて、 DB2 CLI に付いているバインド・ファイルを一度バインドする必要があるだけです。いったんこれを汎用にすると、アプリケーションに必要な管理の量を著しく減らすことができます。
- DB2 CLI アプリケーションを複数のデータベースに接続することができます。同一アプリケーションから同一データベースに複数接続することもできます。各接続には、それぞれのコミット範囲があります。アプリケーションでマルチスレッド化の使用により同一結果になる組み込み SQL を使用するよりは CLI を使用する方がずっと簡単です。
- DB2 CLI では、アプリケーション制御の、複雑になることの多いデータ域の必要がなくなります (たとえば SQLDA や SQLCA など、一般に組み込み

SQL アプリケーションに関連しているものです)。その代わりに、DB2 CLI が必要なデータ構造を割り振って制御し、アプリケーションが参照できるようハンドル を与えます。

- DB2 CLI では、複数スレッドのスレッド保護アプリケーションの開発が可能になります。この場合、各スレッドは、独自の接続および他のスレッドとは別のコミット効力範囲を持つことができます。 DB2 CLI は、上記のデータ域を除去し、特定のハンドルでアプリケーションにアクセス可能なデータ構造のすべてを関連付けることにより、このことを成し遂げます。組み込み SQL とは違って、複数スレッドの CLI アプリケーションではコンテキスト管理の DB2 API のいずれかを呼び出す必要はありません。これは、DB2 CLI ドライバーで自動的に操作されます。
- DB2 CLI では、拡張パラメーターの入力と取り出しの機能が備えられており、データの配列が入力時に指定され、結果セットの複数行を直接配列に取り出し、複数の結果セットを生成するステートメントを実行します。
- DB2 CLI では、スキーマ (表、列、外部キー、 1 次キーなど) 情報を照会するための一貫性のあるインターフェースが与えられます。この情報はさまざまな DBMS カタログ表に入っています。返される結果セットは DBMS 間で一貫性があります。したがって、アプリケーションは、データベース・サーバーのリリース間のカタログ変更や、さまざまなデータベース・サーバー間のカタログの違いの影響を受けません。その結果、アプリケーションでは、バージョン固有およびサーバー固有のカタログ照会は書き込まれません。
- 拡張データ変換も DB2 CLI に備えられており、さまざまな SQL と C データ・タイプ間での情報の変換時にアプリケーション・コードが少なくて済みます。
- DB2 CLI には、ODBC と X/Open CLI の両方の関数が組み込まれており、両方とも業界仕様として受け入れられています。 DB2 CLI は、最新の ISO CLI 標準にも合わせています。アプリケーション開発者がこれらの仕様で得た知識は、 DB2 CLI の開発に直接応用することができ、その逆も可能です。このインターフェースは、関数ライブラリーについての知識はあるが、ホスト言語に SQL ステートメントを組み込む製品特定の方法についてはあまり知らないプログラマーにとって、直観的に理解できるものです。
- DB2 CLI には、 DB2 ユニバーサル・データベース (つまり DB2 (MVS/ESA 版) バージョン 5 またはそれ以降) のサーバーにあるストアード・プロシージャーから生成される複数行と結果セットを取り出す機能が備えられています。しかし、この機能は DataJoiner のバージョン 2 サーバーによりアクセス可能なサーバーにストアード・プロシージャーがある場合

に、組み込み SQL を使用しているバージョン 5 の DB2 ユニバーサル・データベースのクライアントのために用意されているものです。

- DB2 CLI は、サーバー側でスクロール可能カーソルをサポートし、配列出力と共に使用することが可能です。これは、「Page Up」、「Page Down」、「Home」、および「End」キーを使用するスクロール・ボックスで、データベース情報を表示する GUI アプリケーションに役立ちます。読み取り専用カーソルをスクロール可能と宣言した後、結果セットを通して 1 行または複数行単位で下方または上方に移動することができます。下記において、オフセットを指定することにより、複数の行を取り出すことも可能です。
  - 現在行
  - 結果セットの開始または終了位置
  - 以前にブックマークで設定した特定の行
- DB2 CLI アプリケーションは、 CLI および組み込み SQL アプリケーションが結果セットを記述するのと同じように、 SQL ステートメントにパラメーターを動的に記述できます。これによって、あらかじめパラメーター・マーカーのデータ・タイプを知らなくても、パラメーター・マーカーを含む SQL ステートメントを動的に処理することが可能になります。 SQL ステートメントが準備されると、記述子情報がパラメーターのデータ・タイプの詳細と共に返されます。

組み込み SQL か DB2 CLI かの決定

どのインターフェースを選択するかは、使用するアプリケーションによって決まります。

DB2 CLI は、移植性が要求される、照会ベースのグラフィカル・ユーザー・インターフェース (GUI) アプリケーションに理想的です。前述の利点のため、DB2 CLI の方が明らかにどんなアプリケーションにもふさわしいように見えるかもしれません。しかし、考慮しなければならない要素が 1 つあります。静的 SQL と動的 SQL の比較です。組み込みアプリケーションでは、静的 SQL を使った方がずっと簡単です。

CLI アプリケーションでの静的 SQL の使用に関する詳細については、次の Web ページを参照してください。

```
http://www.ibm.com/software/data/db2/udb/staticcli
```

静的 SQL には、次のようないくつかの利点があります。

- パフォーマンス

  動的 SQL は実行時に準備され、静的 SQL はプリコンパイル時に準備されます。より多くの処理が必要になると同時に、準備ステップのために実行時

に追加のネットワーク通信量が生じることがあります。しかし、この準備ステップ (およびネットワーク通信量) は、 DB2 CLI のアプリケーションが据え置き準備の場合には、必須ではありません。

静的 SQL が動的 SQL より常に良いパフォーマンスが得られるわけではない、というのも重要なことです。動的 SQL では、新規の索引などのデータベースに対する変更事項を利用でき、現行のデータベース統計を使って最適のアクセス・プランを選択することができます。さらに、ステートメントの事前コンパイルは、キャッシュされる場合に避けることができます。

- カプセル化およびセキュリティー

  静的 SQL では、オブジェクト (表や視点など) に対する権限はパッケージに関連付けられ、パッケージのバインド時に妥当性検査されます。このため、データベース管理者は、特定のパッケージに関する実行権を 1 つのユーザーの集まりに付与する (つまり、特権をパッケージにカプセル化する) だけで済み、各データベース・オブジェクトへの明示アクセス権を付与する必要はありません。動的 SQL では、権限は実行時にステートメント単位で妥当性検査されます。したがって、ユーザーは各データベース・オブジェクトへの明示アクセス権を付与してもらわなければなりません。これによってこれらのユーザーは、アクセスする必要のないオブジェクトの部分にアクセスすることが許可されます。

- 組み込み SQL は、C または C++ 以外の言語でサポートされます。
- 固定照会の選択の場合、組み込み SQL はより簡単です。

アプリケーションで両方のインターフェースの利点が必要な場合は、静的 SQL を含むストアード・プロシージャーを作成して、 DB2 CLI アプリケーションで静的 SQL を利用できます。ストアード・プロシージャーは、DB2 CLI アプリケーション内から呼び出され、サーバーで実行されます。ストアード・プロシージャーを作成すると、どの DB2 CLI または ODBC アプリケーションでもこれを呼び出すことができます。詳細については、 131ページの『ストアード・プロシージャーの使用』を参照してください。

CLI アプリケーションでの静的 SQL の使用に関する詳細については、次の Web ページを参照してください。

```
http://www.ibm.com/software/data/db2/udb/staticcli
```

DB2 CLI と組み込み SQL の両方を使う混合アプリケーションを作成して、それぞれの利点を活用することもできます。この場合、DB2 CLI を使用して基本のアプリケーションを作成し、パフォーマンスまたは機密保護上の理由のために静的 SQL を使用してキー・モジュールを作成します。このためアプリケーション設計が複雑になるので、ストアード・プロシージャーがアプリケーショ

ン要件に合わない場合に限りこの方法を使用してください。 143ページの『組み込み SQL と DB2 CLI の混合』を参照してください。

結局、それぞれのインターフェースをいつ使用するかの判断は、 1 つの要因によるというのではなく、個々の必要性と以前の経験によって決まります。

サポートされている環境

DB2 CLI 実行時サポートは、DB2 サーバーとクライアントの両方で提供されます。以前のバージョンでのサポートについては、821ページの『変更の要約』を参照してください。

DB2 CLI 開発サポートは DB2 アプリケーション開発クライアントに組み込まれています。このサポートは、必要なヘッダー・ファイル、リンク・ライブラリー、および特定の操作環境用の組み込みアプリケーションと DB2 CLI アプリケーションの開発に必要な資料から構成されます。たとえば、DB2 アプリケーション開発クライアントにより OS/2 の下で実行するアプリケーションを作成できますが、 DB2 クライアントを使用すると、これらのアプリケーションは DB2 (OS/2 版)、 DB2 (AIX 版)、またはその他の DB2 ユニバーサル・データベース・サーバー上のデータにアクセスできます。また、DB2 コネクトを用いれば、アプリケーションは DB2 ユニバーサル・データベース (AS/400 版) 、DB2 (OS/390 版)、 DB2 (VSE および VM 版) サーバーまたはその他の IBM の DRDA サーバーや IBM 以外の DRDA サーバーにアクセスできます。

DB2 ソフトウェア開発者キットの詳細、および DB2 CLI アプリケーション作成用にサポートされているコンパイラーの使用については、 *アプリケーション構築の手引き* を参照してください。

その他の情報源

DB2 CLI アプリケーションを作成する場合、接続性の問題、環境問題、SQL 言語サポートの問題、およびその他のサーバー特定の情報を理解するために、アクセスされているデータベース・サーバーについての情報を参照する必要が生じることがあります。 DB2 ユニバーサル・データベース・バージョンの場合、*SQL 解説書*、*アプリケーション開発の手引き*、および*管理 API 解説書* を参照してください。その他の DB2 サーバーにアクセスするアプリケーションを作成している場合は、すべての製品に共通している情報や、何らかの違いも含む情報を載せている *SQL 解説書* を参照してください。

第2章 DB2 CLI アプリケーションの作成

このセクションでは、一般的な DB2 CLI アプリケーションを概念的に示して説明します。

DB2 CLI アプリケーションは、タスクの集まりに分割することができます。これらのタスクのあるものは、独立したステップに編成されますが、他のものはアプリケーション全体を通して適用されます。個々のタスクは、1 つまたは複数の DB2 CLI 関数によって実行されます。

このセクションで述べるタスクは、すべてのアプリケーションに適用される基本タスクです。配列挿入を使用したり、ラージ・オブジェクト・サポートを使用するようなさらに拡張されたタスクについては、 45ページの『第3章 拡張機能の使用』で述べます。

例の中で使われている関数は、DB2 CLI アプリケーションでの使い方を示してあります。関数ごとのすべての説明および使用法については、233ページの『第5章 DB2 CLI 関数』を参照してください。

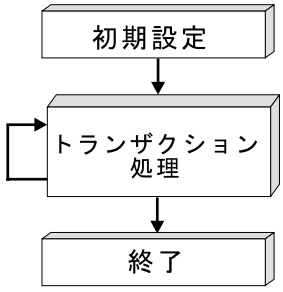


図 1. DB2 CLI アプリケーションの概念説明

どの DB2 CLI アプリケーションにも、 図1 に示されている 3 つの主タスクがあります。

初期設定

このタスクは、主要な*トランザクション処理* タスクの準備をするためにいくつかの資源を割り振り、初期設定します。詳細については、13ページの『初期設定および終了』を参照してください。

トランザクション処理

これはアプリケーションの主タスクです。 SQL ステートメントが DB2 CLI に渡されてデータを照会したり修正します。詳細については、17ページの『トランザクション処理』を参照してください。

終了 このタスクは割り振られた資源を解放します。資源は通常、固有なハンドルによって識別されるデータ域で構成されます。詳細については、13ページの『初期設定および終了』を参照してください。

前述の 3 つのタスクに加えて、診断メッセージの処理などの一般タスクがあります。これはアプリケーションによって行われます。

初期設定および終了

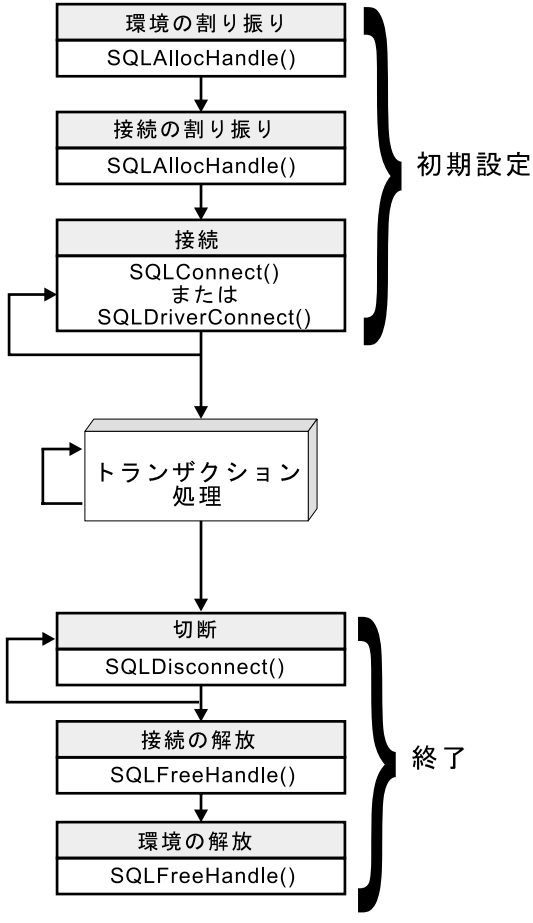


図 2. 初期設定タスクおよび終了タスクの概念説明

図2 は、初期設定と終了の両方のタスクの関数呼び出しの順序を示しています。図の中央にあるトランザクション処理タスクは、18ページの図3 に示してあります。

ハンドル

初期設定タスクは、環境ハンドルおよび接続ハンドルの割り振りと初期設定から構成されます (これらは後で終了タスクで解放される)。アプリケーションはその後、他の DB2 CLI 関数を呼び出すときに該当するハンドルを渡します。ハンドルとは、DB2 CLI によって制御されるデータ・オブジェクトを参照する変数のことです。ハンドルを使用すると、アプリケーションがグローバル変数またはデータ構造 (IBM の組み込み SQL インターフェースで使用される SQLDA や SQLCA など) を割り振り、管理する必要がなくなります。

SQLAllocHandle() 関数は、ハンドル・タイプおよび親ハンドル引き数とともに呼び出され、環境、接続、ステートメント、または記述子ハンドルを作成します。関数 SQLFreeHandle() は、 1 つのハンドルに割り当てられた資源を解放します。

ハンドルには次の 4 つのタイプがあります。

環境ハンドル

環境ハンドルは、アプリケーションのグローバル状態に関する情報 (属性や接続など) が入っているデータ・オブジェクトを参照します。接続ハンドルを割り振るためには、その前に環境ハンドルを割り振っておく必要があります。

接続ハンドル

接続ハンドルは、特定のデータ・ソース (データベース) への接続に関連する情報が入っているデータ・オブジェクトを参照します。この情報には、接続属性、一般状況情報、トランザクション状況、および診断情報が含まれます。

アプリケーションは、同時に複数のサーバーに接続でき、同じサーバーに複数の別個の接続を確立することもできます。アプリケーションには、データベース・サーバーへの並行接続ごとに 1 つの接続ハンドルが必要です。複数の接続に関する情報は、15ページの『1 つまたは複数のデータ・ソースへの接続』を参照してください。

ユーザーがコネクター・ハンドルの数の限度を設定しているかどうかを判別するには、 SQLGetInfo() を呼び出してください。

ステートメント・ハンドル

ステートメント・ハンドルは、次のセクション、17ページの『トランザクション処理』で解説します。

記述子ハンドル

記述子ハンドルは、以下の情報を含むデータ・オブジェクトを参照します。

- 結果セットにある列
- SQL ステートメントにある動的パラメーター

記述子および記述子ハンドルは、105ページの『記述子の使用』のセクションで述べます。

1 つまたは複数のデータ・ソースへの接続

1 つまたは複数のデータ・ソースへの並行接続 (または同一データ・ソースへの複数並行接続) をするには、アプリケーションが接続ごとに一度 SQL_HANDLE_DBC の *HandleType* を指定して、 SQLAllocHandle() を呼び出します。それ以降の接続ハンドルは、SQLConnect() を指定してデータベース接続を要求し、 SQLAllocHandle() と SQL_HANDLE_STMT の *HandleType* を指定してその接続内で使用されるステートメント・ハンドルを割り振ります。さらに拡張接続関数の SQLDriverConnect() があり、これによって追加の接続オプション、および図形ユーザー・インターフェースをサポートしている環境で接続ダイアログ・ボックスを直接オープンする機能が利用できるようになります。関数 SQLBrowseConnect() を使用すると、データ・ソースに接続するのに必要なすべての属性と属性値を知ることができます。

接続ハンドルを使用すると、スレッドごとに 1 つの接続を利用する複数スレッドのアプリケーションにおいて確実にスレッド保護できます。接続ごとに別々のデータ構造が DB2 CLI によって割り振られ、管理されるからです。

55ページの『複数サイト更新 (2 フェーズ・コミット)』に記述されている分散作業単位接続とは違って、さまざまな接続で実行されるステートメント間で調整は行われません。

初期設定および接続の例

```
/* ... */
#include <stdio.h>
#include <stdlib.h>
#include <sqlcli1.h>
/* ... */
SQLRETURN
prompted_connect( SQLHANDLE henv,
                  SQLHANDLE * hdbc);
#define MAX_UID_LENGTH   18
#define MAX_PWD_LENGTH   30
#define MAX_CONNECTIONS  2
#define MAX_CONNECTIONS 2
/* extern SQLCHAR server[SQL_MAX_DSN_LENGTH + 1] ;
extern SQLCHAR uid[MAX_UID_LENGTH + 1] ;
extern SQLCHAR pwd[MAX_PWD_LENGTH + 1] ;
*/
int main( ) {
```

```
    SQLHANDLE henv;
    SQLHANDLE hdbc[MAX_CONNECTIONS] ;
/* ... */
    /* allocate an environment handle */
    SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv ) ;
    /* Connect to first data source */
    prompted_connect( henv, &hdbc[0] ) ;
    /* Connect to second data source */
    prompted_connect( henv, &hdbc[1] ) ;
    /*********   Start Processing Step  *************************/
    /* allocate statement handle, execute statement, etc.       */
    /*********   End Processing Step  **************************/
    printf( "¥nDisconnecting .....¥n" ) ;
    SQLDisconnect( hdbc[0] ) ;  /* disconnect first connection */
    SQLDisconnect( hdbc[1] ) ;  /* disconnect second connection */
    /* free first connection handle */
    SQLFreeHandle( SQL_HANDLE_DBC, hdbc[0] ) ;
    /* free second connection handle */
    SQLFreeHandle( SQL_HANDLE_DBC, hdbc[1] ) ;
    /* free environment handle */
    SQLFreeHandle( SQL_HANDLE_ENV, henv ) ;
    return ( SQL_SUCCESS ) ;
}
/* prompted_connect - prompt for connect options and connect */
SQLRETURN prompted_connect( SQLHANDLE henv,
                            SQLHANDLE * hdbc
                          ) {
    SQLCHAR server[SQL_MAX_DSN_LENGTH + 1] ;
    SQLCHAR uid[MAX_UID_LENGTH + 1] ;
    SQLCHAR pwd[MAX_PWD_LENGTH + 1] ;
    /* allocate a connection handle     */
    if ( SQLAllocHandle( SQL_HANDLE_DBC,
                         henv,
                         hdbc
                       ) != SQL_SUCCESS ) {
        printf( ">---ERROR while allocating a connection handle-----¥n" ) ;
        return( SQL_ERROR ) ;
    }
    /* Set AUTOCOMMIT OFF */
    if ( SQLSetConnectAttr( * hdbc,
                            SQL_ATTR_AUTOCOMMIT,
                            ( void * ) SQL_AUTOCOMMIT_OFF, SQL_NTS
                          ) != SQL_SUCCESS ) {
       printf( ">---ERROR while setting AUTOCOMMIT OFF ------------¥n" ) ;
       return( SQL_ERROR ) ;
    }
    printf( ">Enter Server Name:¥n" ) ;
    gets( ( char * ) server ) ;
    printf( ">Enter User Name:¥n" ) ;
    gets( ( char * ) uid ) ;
    printf( ">Enter Password:¥n" ) ;
    gets( ( char * ) pwd ) ;
    if ( SQLConnect( * hdbc,
                     server, SQL_NTS,
                     uid,    SQL_NTS,
```

```
                    pwd,    SQL_NTS
                  ) != SQL_SUCCESS ) {
        printf( ">--- ERROR while connecting to %s -------------¥n",
                server
              ) ;
        SQLDisconnect( * hdbc ) ;
        SQLFreeHandle( SQL_HANDLE_DBC, * hdbc ) ;
        return( SQL_ERROR ) ;
    }
    else              /* Print Connection Information */
        printf( "Successful Connect to %s¥n", server ) ;
    return( SQL_SUCCESS ) ;
}
```

トランザクション処理

次の図は、DB2 CLI アプリケーションでの関数呼び出しの一般的な順序を示しています。関数またはあり得るパスのすべてが示されているわけではありません。

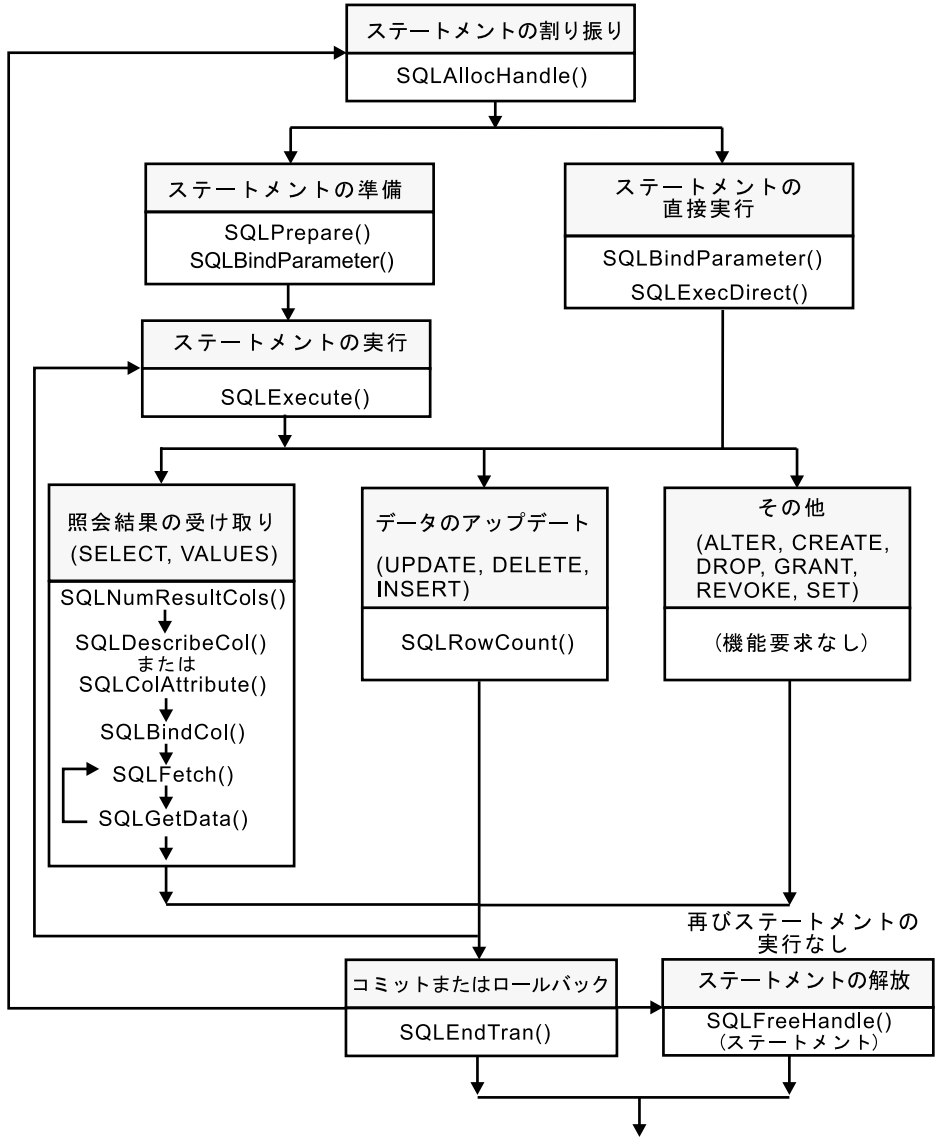


図 3. トランザクション処理

図3 は、トランザクション処理タスクのステップおよび DB2 CLI 関数を示しています。このタスクには次の 5 つのステップがあります。

- ステートメント・ハンドルの割り振り
- SQL ステートメントの準備と実行
- 結果の処理
- コミットまたはロールバック

- 任意選択で、万一ステートメントが再実行される場合のステートメント・ハンドルの解放

ステートメント・ハンドルの割り振り

SQLAllocHandle() は、SQL_HANDLE_STMT の *HandleType* を指定して呼び出され、ステートメント・ハンドルを割り振ります。ステートメント・ハンドルは、 1 つの SQL ステートメントの実行をトラックするのに使用されるデータ・オブジェクトを参照します。これには、ステートメント属性、SQL ステートメント・テキスト、動的パラメーター、カーソル情報、動的引き数と列のバインド、結果の値、および状況情報 (これらは後で説明します) などの情報が含まれています。各ステートメント・ハンドルは、1 つの接続ハンドルと関連付けられます。

ステートメント・ハンドルは、ステートメントを実行する前に割り振らなければなりません。

一度に割り振れるステートメント・ハンドルの最大数は、システム資源全体によって制限されます (通常は、スタック・サイズ)。しかし、実際に使用されるステートメント・ハンドルの最大数は、DB2 CLI で定義されます。アプリケーションが上記の制限を超えた場合、 SQLPrepare() または SQLExecDirect() の呼び出し時に HY014 SQLSTATE が返されます。

準備と実行

ステートメント・ハンドルがいったん割り振られた後、 SQL ステートメントの指定と実行には 2 つの方法があります。

1. 準備してから実行
   a. 引き数として SQL ステートメントを指定して SQLPrepare() を呼び出します。
   b. SQL ステートメントにパラメーター・マーカー がある場合は、SQLBindParameter() を呼び出します。
   c. SQLExecute() を呼び出します。
2. 直接実行
   a. SQL ステートメントにパラメーター・マーカー がある場合は、SQLBindParameter() を呼び出します。
   b. 引き数として SQL ステートメントを指定して、 SQLExecDirect() を呼び出します。

最初の方法では、ステートメントの準備を実行と分割します。この方法は以下の場合に使用します。

- そのステートメントが繰り返し実行される場合 (通常、異なるパラメーター値を指定して)。これにより、同一ステートメントを複数回準備する必要がなくなります。複数回、同じステートメントを準備する必要を省きます。以後の実行時には、準備の際にすでに生成されたアクセス・プランを利用します。
- ステートメントの実行よりも前に、結果セットの列についての情報をアプリケーションが必要とする場合。

2 番目の方法では、準備ステップと実行ステップを結合します。この方法は以下の場合に使用します。
- ステートメントが 1 度だけ実行される場合。ステートメントを実行するのに 2 つの関数を呼び出す必要を省きます。
- ステートメントを実行する前に、結果セットの列に関する情報をアプリケーションが必要としない場合。

注: SQLGetTypeInfo() および 45ページの『第3章 拡張機能の使用』に記述されているスキーマ (カタログ) 関数は、独自の照会ステートメントを実行して結果セットを生成します。スキーマ機能を呼び出すということは照会ステートメントを実行するのと同等で、照会ステートメントが実行されたかのように結果セットが処理されます。

DB2 ユニバーサル・データベースのバージョン 5 以降では、サーバーに*グローバルな動的ステートメント・キャッシュ* が保管されています。このキャッシュは、準備状態の SQL ステートメントに対する最も一般的なアクセス・プランを保管するのに使用します。各ステートメントが準備される前に、サーバーはこのキャッシュを検索して、 (このアプリケーションか別のアプリケーション、またはクライアントによって) この SQL ステートメント用のアクセス・プランが作成済みであるかどうかを調べます。作成済みであれば、サーバーが新たにアクセス・プランを生成する必要はなく、代わりに、キャッシュの中にあるものを使用します。現在では、グローバル動的ステートメント・キャッシュのないサーバー (DB2 共通サーバー v2 など) への接続でなければ、アプリケーションがクライアントで接続をキャッシュする必要はありません。クライアントでのキャッシュ接続については、移行のセクションの 825ページの『クライアントでのステートメント・ハンドルのキャッシュ』を参照してください。

SQL ステートメントのパラメーターのバインド: 上記のどちらの実行方法でも、式 (または組み込み SQL のホスト変数) の代わりに、パラメーター・マーカーを使用できます。

パラメーター・マーカーは、'?' 文字で表されるもので、SQL ステートメント内で、ステートメントの実行時にアプリケーション変数の内容が置換される位

置を示します。パラメーター・マーカーは、1 を先頭にして左方から右方へ順番に参照されます。ステートメント内のパラメーターの数を判別するのに、SQLNumParams() を使用することができます。

アプリケーション変数がパラメーター・マーカーに関連付けられるときには、そのパラメーター・マーカーにバインド されます。アプリケーションは SQL ステートメントを実行する前に、アプリケーション変数をそのステートメント内の各パラメーター・マーカーにバインドしなければなりません。バインドは SQLBindParameter() 関数の呼び出しによって実行しますが、その際、指示する引き数の数、パラメーターの数値位置、パラメーターの SQL タイプ、変数のデータ・タイプ、アプリケーション変数を指すポインター、および変数の長さを指定します。

バインドされたアプリケーション変数および関連する長さは、*据え置き* 入力引き数と呼ばれます。パラメーターがバインドされるときにはポインターだけが渡されるからです。そのステートメントが実行されるまで変数からデータが読み取られることはありません。アプリケーションは、据え置き引き数を使用すると、バインドされたパラメーター変数の内容を修正したり、新しい値でステートメントを繰り返し実行できるようになります。

各パラメーターの情報は、指定変更されるまでか、またはアプリケーションがパラメーターをアンバインドするかステートメント・ハンドルを除去するまで、そのまま有効です。アプリケーションがパラメーターのバインドを変更せずに SQL ステートメントを繰り返し実行すると、 DB2 CLI は同じポインターを使用して実行時ごとにデータを探し出します。アプリケーションは、パラメーターのバインドを、別の据え置き変数の集まりに変更することもできます。アプリケーションは、据え置き入力フィールドに使用される変数の割り振り解除や廃棄を、フィールドをパラメーター・マーカーにバインドする時と DB2 CLI が実行時にそれらにアクセスする時の間に行うことはできません。

SQL ステートメントが要求するものとは異なるタイプの変数にパラメーターをバインドすることが可能です。アプリケーションはソースの C データ・タイプおよびパラメーター・マーカーの SQL タイプを指示する必要があり、 DB2 CLI は指定された SQL データ・タイプと一致するよう変数の内容を変換します。たとえば、SQL ステートメントには整数値が必要なのに、アプリケーションには整数のストリング表示があるとします。このストリングをパラメーターにバインドすることができ、 DB2 CLI はステートメントの実行時にそのストリングを対応する整数値に変換します。

省略時設定では、DB2 CLI はパラメーター・マーカーのタイプの検査を行いません。アプリケーションが正しくないパラメーター・マーカーのタイプを示す

と、 DBMS によって余計な変換が行われるか、エラーになる可能性があります。データ変換の詳細については、32ページの『データ・タイプとデータ変換』を参照してください。

パラメーター・マーカーについての情報は、記述子を使用して見ることができます。実装パラメーター記述子 (implementation parameter descriptor (IPD)) の自動移植を使用可能にした場合、パラメーター・マーカーについての情報が収集されます。ステートメント属性 SQL_ATTR_ENABLE_AUTO_IPD は、この作業では SQL_TRUE に設定する必要があります。詳細については、105ページの『記述子の使用』を参照してください。

パラメーター・マーカーが照会に関する述部の一部であり、ユーザー定義タイプと関連付けられていると、そのパラメーター・マーカーをステートメントの述部部分で組み込みタイプにキャストしなければなりません。そうしないと、エラーが起こります。例については、831ページの『述部内でのユーザー定義タイプ』を参照してください。

グローバルな動的ステートメント・キャッシュは、もっと前のセクションで説明されています。アクセス・プランは、ステートメントが厳密に同一である場合、その間でのみ共用されます。パラメーター・マーカー付きの SQL ステートメントの場合、 SQL ステートメントそれ自身が同一であれば、そのパラメーターに結び付いている特定値は、同一である必要はありません。

アプリケーション記憶域をパラメーター・マーカーへバインドする場合のさらに拡張された方法については、下記を参照してください。
- 91ページの『配列の使用によるパラメーター値の入力』
- 87ページの『長形式データの分割送信 / 取り出し』
- 97ページの『パラメーター・バインドの相対位置』

結果の処理

ステートメントが実行された後の次のステップは、SQL ステートメントのタイプによって異なります。

照会 (SELECT、VALUES) ステートメントの処理: ステートメントが照会ステートメントの場合、結果セットの各行を取り出すために通常、次のステップが必要です。
1. 結果セットの構造、列の数、列のタイプおよび長さを確立 (記述) する。
2. データを受け取るためにアプリケーション変数を列に (任意選択で) バインドする。
3. データの次の行を繰り返し取り出し、それをバインドされたアプリケーション変数へ受け取る。

4. 正常なそれぞれの取り出しの後で SQLGetData() を呼び出して、それまでにバインドされなかった列を (任意選択で) 取り出す。

- 上記のステップのそれぞれに、診断チェックが必要です。
- 45ページの『第3章 拡張機能の使用』には、複数行を一度に取り出す拡張手法の SQLFetchScroll() の使用法について説明されています。
- また、DB2 CLI はスクロール可能な読み取り専用のカーソルもサポートしています。詳細は、75ページの『スクロール可能カーソル』を参照してください。

ステップ 1

最初のステップでは、実行済みまたは準備済みのステートメントの分析が必要です。アプリケーションは、列の数、各列のタイプ、およびおそらく結果セットにある各列の名前を照会する必要があります。この情報は、ステートメントの準備の後または実行の後で SQLNumResultCols() および SQLDescribeCol() (または SQLColAttributes()) を呼び出すことによって得ることができます。

ステップ 2

2 番目のステップでは、次の SQLFetch() の呼び出しのときに、アプリケーションが列データを直接アプリケーション変数へ取り出せるようにします。取り出される列ごとに、アプリケーションは SQLBindCol() を呼び出してアプリケーション変数を結果セットの列にバインドします。アプリケーションはステップ 1 で得られた情報を使用して、アプリケーション変数の C データ・タイプを判別したり、列値が占めることができる最大記憶域を割り振ったりします。 SQLBindParameter() を使ってパラメーター・マーカーに変数がバインドされる場合と同様に、列は据え置き引き数にバインドされます。この場合は、データが SQLFetch() の呼び出し時に記憶場所に書き込まれるので、変数は据え置き出力引き数です。

アプリケーションが長いデータの列を小さく分けて取り出す必要がある場合のように、アプリケーションがどの列もバインドしない場合には、SQLGetData() を使用することができます。バインドされる列とアンバインドされる列がある場合、 SQLBindCol() と SQLGetData() の両方の手法を結合することができます。アプリケーションは、据え置き出力フィールドに使用される変数の割り振り解除や廃棄を、結果セットの列にバインドする時と DB2 CLI がこれらのフィールドにデータを書き込む時の間に行ってはなりません。

ステップ 3

3 番目のステップでは、 SQLFetch() を呼び出して結果セットの最初または次の行を取り出します。バインドされた列があれば、アプリケーシ

ョン変数は更新されます。また、SQLFetchScroll() を使用すれば、結果セットの移動の時について、柔軟性が増します。詳細については 75ページの『スクロール可能カーソル』を参照してください。
SQLFetchScroll() を使用することにより、アプリケーションが結果セットの複数行を配列の中へ取り出せるようにすることもできます。詳細については、99ページの『配列への結果セットの取り出し』を参照してください。

SQLBindCol() の呼び出しのときに指定されたデータ・タイプによってデータ変換が指示された場合には、 SQLFetch() の呼び出し時に変換が行われます。説明については、32ページの『データ・タイプとデータ変換』を参照してください。

ステップ 4 (任意選択)

最後の (任意選択) ステップでは、 SQLGetData() を呼び出してアンバインドされた列を取り出します。このようにして、バインドされていなくてもすべての列を取り出せます。 SQLGetData() を繰り返し呼び出して、大きな列を小さく分けて取り出すこともできます。これは、バインドされた列には行うことはできません。

SQLBindCol() の場合と同様に、ご希望のアプリケーション変数のターゲット C データ・タイプを指定して、データ変換をこの時点でも指示することができます。詳細については、32ページの『データ・タイプとデータ変換』を参照してください。

結果セットの特定の列をアンバインドするには、アプリケーション変数引き数 (*TargetValuePtr*) にヌル・ポインターを指定して SQLBindCol() を使用してください。 1 回の関数呼び出しですべての列をアンバインドするには、 SQL_UNBIND のオプション を指定して SQLFreeStmt() を使用してください。

列がバインドされている場合は、 SQLGetData() を使用してバインドされていない列として検索する場合にくらべて、一般にアプリケーションのパフォーマンスがよくなります。ただし、アプリケーションは一度に取り出しと処理が可能な長データの量に関して制約される可能性があります。これが考えられる場合は、SQLGetData() の方が良い選択である場合があります。長データを処理する他の手法については、123ページの『ラージ・オブジェクトの使用』を参照してください。

アプリケーション記憶域を結果セット列へバインドする場合のさらに高度な方法については、下記を参照してください。

- 99ページの『配列への結果セットの取り出し』
- 87ページの『長形式データの分割送信 / 取り出し』

- 103ページの『列バインドの相対位置』

UPDATE、DELETE、および INSERT ステートメントの処理: ステートメントがデータを修正するもの (UPDATE、DELETE または INSERT) である場合、アクションは何も必要ありません。診断メッセージの通常チェックだけが必要です。この場合、SQLRowCount() を使用して、 SQL ステートメントによって影響を受けた行の数を得ることができます。

SQL ステートメントが、定位置 UPDATE または DELETE である場合、カーソル を使用する必要があります。カーソルとは、活動状態の照会ステートメントの結果表の行を指す、移動可能なポインターです。 (この照会ステートメントには、確実に照会が読み取り専用としてオープンされていないようにするための FOR UPDATE OF 文節が入っていなければなりません。) 組み込み SQL では、カーソル名は行の取り出し、更新、または削除に使用されます。 DB2 CLI では、定位置 UPDATE または DELETE の SQL ステートメントがカーソルを名前で参照する場合に限り、カーソル名が必要です。また、SQLAllocHandle() が、 SQL_HANDLE_STMT の *HandleType* を指定して呼び出されるときに、カーソル名は自動的に生成されます。

取り出された行を更新するには、アプリケーションが 2 つのステートメント・ハンドルを、 1 つは取り出しに 1 つは更新に使用します。アプリケーションは SQLGetCursorName() を呼び出してカーソル名を獲得します。アプリケーションは定位置 UPDATE または DELETE のテキストをこのカーソル名も含めて生成し、 2 番目のステートメント・ハンドルを使用してその SQL ステートメントを実行します。アプリケーションは、取り出しステートメント・ハンドルを再利用して、定位置 UPDATE または DELETE を実行することはできません。それはまだ使用中だからです。また、SQLSetCursorName() を使用して独自のカーソル名を定義することもできます。ただし、すべてのエラー・メッセージは SQLSetCursorName() で定義された名前ではなく生成された名前を参照するので、生成された名前を使用することをお勧めします。

その他のステートメントの処理: ステートメントがデータの照会も修正も行わない場合、診断メッセージの通常チェック以外にはアクションは必要ありません。

コミットまたはロールバック

トランザクション とは、回復可能な 1 つの作業単位、または 1 つのアトミック操作として扱うことができる SQL ステートメントのグループです。このことは、グループ内の全操作は、それらがあたかも単一操作のように完了する (コミットする) またはやり直しする (ロールバックする) ことが保証されてい

るということです。トランザクションを、作業単位または論理作業単位と呼ぶこともあります。トランザクションが複数の接続にわたる場合、それは分散作業単位と呼びます。

DB2 CLI は下記の 2 つのコミット・モードをサポートします。

自動コミット

自動コミット・モードでは、どの SQL ステートメントも完了トランザクションであり、自動的にコミットされます。照会以外のステートメントの場合、終了ステートメントの実行時にコミットが出されます。照会ステートメントの場合、カーソルのクローズ後にコミットが出されます。アプリケーションは最初の照会のカーソルをクローズする前に 2 番目の照会を開始することはできません。

手動コミット

手動コミット・モードでは、SQLPrepare()、SQLExecDirect()、SQLGetTypeInfo() または 72ページの『システム・カタログ情報の照会』に記述されているような、結果セットを返す関数を使用してデータベースに最初にアクセスすることで、トランザクションは暗示的に開始されます。この時点で、呼び出しが失敗してもトランザクションは開始されています。トランザクションは、 SQLEndTran() を使用してそのトランザクションをロールバックまたはコミットする時点で終了します。つまり、上記の 2 つの時点の間に (同じ接続で) 実行されたステートメントは、 1 つのトランザクションとして扱われることを意味します。

省略時のコミット・モードは自動コミットです (調整済みトランザクションが関係している場合を除く。 55ページの『複数サイト更新 (2 フェーズ・コミット)』を参照してください)。アプリケーションは SQLSetConnectAttr() を呼び出して、手動コミットと自動コミットのモードを切り替えることができます。一般に、照会専用のアプリケーションは自動コミット・モードにあることが望まれます。データベースに更新を行う必要があるアプリケーションでは、データベース接続が確立されたらすぐに、自動コミットをオフにする必要があります。

同じまたは別のデータベースに複数の接続が存在する場合、個々の接続に独自のトランザクションがあります。必ず意図した接続および関連したトランザクションだけが影響を受けるようにするために、 SQLEndTran() を呼び出す際には正しい接続ハンドルを指定して特に注意して行う必要があります。また、 SQLEndTran() 呼び出しで有効な環境ハンドルおよび NULL 接続ハンドルを指定して、すべての接続をロールバックまたはコミットすることも可能です。分

散作業単位接続 (55ページの『複数サイト更新 (2 フェーズ・コミット)』に記述) とは違って、個々の接続に関するトランザクション間で調整は行われません。

SQLEndTran() を呼び出す場合: アプリケーションが自動コミット・モードである場合、 SQLEndTran() を呼び出す必要は決してありません。各ステートメントの実行の終わりにコミットが暗黙に出されます。

手動コミット・モードでは、SQLDisconnect() を呼び出す前に、 SQLEndTran() を呼び出す必要があります。分散作業単位が関連しているときは、追加規則が適用される場合があります。詳細は、 55ページの『複数サイト更新 (2 フェーズ・コミット)』を参照してください。

更新を実行する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する前に切断が行われるまで待つことは、お勧めできません。もう一つの極端な手段は、自動コミット・モードでの操作です。これも、余分な処理が追加されるのでお勧めしません。手動コミットと自動コミット間の切り替えの詳細については、 46ページの『環境、接続、およびステートメントの属性』および 663ページの『SQLSetConnectAttr - 接続属性を設定する』を参照してください。

アプリケーションがトランザクションをいつ終了するかを決める際には、下記の点を考慮してください。
- 個々の接続に 1 つだけ未解決のトランザクションがある場合、従属ステートメントを同じトランザクション内に保持してください。
- 未解決のトランザクションがある間は、さまざまな資源を保持できます。トランザクションを終了すると、他のユーザーが使用するために資源が解放されます。
- トランザクションが正常にコミットまたはロールバックされると、このトランザクションは、システム・ログから完全に回復可能になります。オープン・トランザクションは回復可能ではありません。

SQLEndTran() 呼び出しの影響:

トランザクションが終了すると、以下のことが行われます。
- 保留カーソルに関連したロックを除いて、 DBMS オブジェクトに関するすべてのロックが解除されます。
- 準備済みステートメントはトランザクション間で保存されます。特定のステートメント・ハンドルに関するステートメントを準備すると、コミットやロールバックの後で再び準備する必要はありません。そのステートメントは引き続き同じステートメント・ハンドルに関連しています。

- カーソル名、バインドされたパラメーター、および列のバインドは、トランザクション間で保守されます。
- 省略時設定では、コミットした後 (ただしロールバックしない) カーソルは保存されます。つまり、省略時設定ではすべてのカーソルは WITH HOLD 文節で定義されます (SQL/DS に接続している場合を除きます。この場合は WITH HOLD 文節はサポートされません。さらに、分散作業単位環境で CLI アプリケーションが実行しているときには、 55ページの『複数サイト更新 (2 フェーズ・コミット)』を参照してください)。省略時の動作の変更の詳細については、756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』を参照してください。

詳細および例については、388ページの『SQLEndTran - 接続のトランザクションの終了』を参照してください。

ステートメント・ハンドルの解放

SQLFreeStmt() を呼び出して、特定のステートメント・ハンドルの処理を終了します。この関数は、次のうち 1 つまたは複数を行うのに使用できます。

- ブックマーク列 (使用されている場合) の例外を除いて、結果セットのすべての列のアンバインド。ブックマークの使用法の詳細については、 75ページの『スクロール可能カーソル』を参照してください。

  アプリケーションの行記述子 (ARD) の SQL_DESC_COUNT フィールドも、この場合ゼロにセットされます。記述子の使用法の詳細については、105ページの『記述子の使用』を参照してください。
- すべてのパラメーター・マーカーのアンバインド。

  アプリケーションのパラメーター記述子 (APD) の SQL_DESC_COUNT フィールドも、この場合ゼロにセットされます。記述子の使用法の詳細については、105ページの『記述子の使用』を参照してください。
- カーソルのクローズおよび保留中の結果の廃棄 (これも、 SQLCloseCursor() を使用して実行可能です)。

SQL_HANDLE_STMT の *HandleType* を指定して SQLFreeHandle() を呼び出し、次のことを行います。

- ステートメント・ハンドルのドロップ、およびすべての関連資源の解放

ハンドルを使用して、別の数またはタイプのパラメーターまたは別の結果セットでステートメントを処理するには、その前に必ず列およびパラメーターをアンバインドしてください。そうしないとアプリケーション・プログラミング・エラーが発生する場合があります。

診断

診断とは、アプリケーション内で生成された警告またはエラー条件に対処することです。 DB2 CLI 機能の呼び出し時に戻される診断には、以下の 2 つのレベルがあります。

- 戻りコード
- 詳細な診断 (SQLSTATE、メッセージ、SQLCA)

個々の CLI 関数は基本診断として関数戻りコードを戻します。 SQLGetDiagRec() 関数と SQLGetDiagField() 関数の両方で、さらに詳しい診断情報を通知します。データ・ソースで診断が報告される場合、 SQLGetSQLCA() 関数は SQLCA へアクセスできるようにします。この調整によって、アプリケーションは戻りコードに基づき基本的な制御の流れを扱えるようになり、さらに SQLSTATE により特定の障害原因および特定のエラー処理の判別を可能にします。

SQLGetDiagRec() も SQLGetDiagField() も両方とも、次の 3 つの部分からなる情報を返します。

- SQLSTATE
- ネイティブのエラー。データ・ソースで診断が検出されるときは、これは SQLCODE で、そうでなければこれは -99999 に設定されます。
- メッセージ・テキスト。これは SQLSTATE に関連したメッセージ・テキストです。

関数に関する詳細および使用例については、 522ページの『SQLGetDiagRec - 診断レコードの複数のフィールド設定を取得する』と 512ページの『SQLGetDiagField - 診断データのフィールドの入手』を参照してください。

SQLGetSQLCA() は特定のフィールドにアクセスするために SQLCA を返しますが、 SQLGetDiagRec() または SQLGetDiagField() の代替として使用されることは決してありません。

関数戻りコード

次の表に、DB2 CLI 関数の戻りコードをすべてリストします。 233ページの『第5章 DB2 CLI 関数』の関数説明のところに、各関数で返される可能性のあるコードをリストしてあります。

表 1. DB2 CLI 関数戻りコード

戻りコード	説明
SQL_SUCCESS	関数が正常に完了しました。他に利用可能な SQLSTATE 情報はありません。

表 1. DB2 CLI 関数戻りコード (続き)

戻りコード	説明
SQL_SUCCESS_WITH_INFO	関数は正常に完了しましたが、警告または他の情報があります。 SQLGetDiagRec() を呼び出して、SQLSTATE およびその他の通知メッセージまたは警告を受け取ってください。 SQLSTATE のクラスは '01' です。 855ページの『SQLState 相互参照表』を参照してください。
SQL_STILL_EXECUTING	関数は非同期に実行中で、まだ完了していません。DB2 CLI ドライバーは、関数を呼び出した後アプリケーションに制御を返しましたが、その関数は実行をまだ完了していません。詳細な説明については、145ページの『CLI の非同期実行』を参照してください。
SQL_NO_DATA_FOUND	関数が正常に戻りましたが、関連データが見つかりませんでした。 SQL ステートメント実行後に戻されるときは、追加情報が使用可能であり、SQLGetDiagRec() を呼び出してこれを得ることができます。
SQL_NEED_DATA	アプリケーションは SQL ステートメントを実行しようとしましたが、アプリケーションが実行時に渡されるよう指示したパラメーター・データが DB2 CLI にありませんでした。詳細については、87ページの『長形式データの分割送信 / 取り出し』を参照してください。
SQL_ERROR	関数は失敗しました。 SQLGetDiagRec() を呼び出して、 SQLSTATE およびその他のエラー情報を受け取ってください。
SQL_INVALID_HANDLE	関数は無効な入力ハンドル (環境、接続またはステートメント・ハンドル) のために失敗しました。これはプログラミング・エラーです。さらに情報はありません。

SQLSTATE

SQLSTATE は、5 文字 (バイト) の英数字ストリングで、形式は ccsss です (cc はクラスで、sss はサブクラス)。クラスが以下の SQLSTATE は、次のとおりになります。

- '01' の場合、警告です。
- 'HY' の場合、DB2 CLI または ODBC ドライバーによって生成されます。

- 'IM' の場合、ODBC ドライバー・マネージャーによってのみ生成されます。

注: 前のバージョンの DB2 CLI では、 'HY' ではなく 'S1' のクラスの SQLSTATE を返していました。 CLI ドライバーを指定するには 'S1' の SQLSTATE を返します。そして、アプリケーションは環境属性 SQL_ATTR_ODBC_VERSION を値 SQL_OV_ODBC2 に設定する必要があります。詳細については、734ページの『SQLSetEnvAttr - 環境属性を設定する』および 821ページの『付録B. アプリケーションの移行』を参照してください。

DB2 CLI の SQLSTATE には、データベース・サーバーによって返される追加の IBM 定義 SQLSTATE と、 ODBC v3 および ISO SQL/CLI 仕様では定義されていない条件に関する DB2 CLI 定義 SQLSTATE の両方が含まれています。これによって、最大量の診断情報が戻されます。また、ODBC 環境でアプリケーションを実行している場合、 ODBC 定義 SQLSTATE を受け取ることも可能です。

アプリケーション内で SQLSTATE を使用する場合、次のガイドラインに従ってください。

- `SQLGetDiagRec()` を呼び出す前に関数戻りコードを必ずチェックして、診断情報が使用可能かどうかを判別してください。
- ネイティブのエラー・コードよりも SQLSTATE を使用してください。
- アプリケーションの移植性を高めるためには、 ODBC v3 および ISO SQL/CLI 仕様で定義されている DB2 CLI SQLSTATE のサブセットだけに依存性を持たせ、追加情報は通知専用として戻すようにします。 (依存性があると、特定の SQLSTATE に基づくアプリケーション作成の論理フローの決定を参照します。)

  注: SQLSTATE のクラス (先頭 2 文字) に関する依存性を作成すると効果的な場合があります。
- 診断情報を最大限活用するために、 SQLSTATE とともにテキスト・メッセージを返してください (該当すれば、テキスト・メッセージには IBM 定義 SQLSTATE も含まれます)。アプリケーションがエラーを返した関数の名前を印刷することも効果的です。

DB2 CLI によって明示的に返される SQLSTATE のリストと説明は、 855ページの『SQLState 相互参照表』を参照してください。

アプリケーションが DB2 を呼び出す方法 (生じる何らかのエラーも含めて) をより良く理解するには、 CLI/ODBC のトレース機能を使用することができます。 915ページの『付録K. DB2 CLI/ODBC/JDBC トレース機能の使用』および、 CLI/ODBC 構成キーワード 223ページの『TRACE』を参照してください。

SQLCA

組み込みアプリケーションは、SQLCA からすべての診断情報を入手します。DB2 CLI アプリケーションは SQLGetDiagRec() を使用して同じ情報の大部分を取り出せますが、それでもまだ、ステートメントの処理に関する SQLCA にアクセスする必要が生じる場合があります (たとえば、ステートメントの準備の後、 SQLCA にはそのステートメントの実行に関する相対コストが含まれています)。その前の要求でデータ・ソースとの対話 (たとえば、接続、準備、実行、取り出し、切断) があった場合に限り、SQLCA には意味のある情報が入っています。

SQLGetSQLCA() 関数は、この構造を取り出す場合に使用します。詳細については、587ページの『SQLGetSQLCA - SQLCA データ構造を入手する』を参照してください。

SQLGetSQLCA() は、 SQLGetDiagRec() または SQLGetDiagField() の代替として決して使用すべきではありません。

データ・タイプとデータ変換

DB2 CLI アプリケーションを作成するときには、 SQL データ・タイプと C データ・タイプの両方で処理する必要があります。 DBMS は SQL データ・タイプを使用し、アプリケーションは C データ・タイプを使用しなければならないので、これは避けられないことです。そのため、アプリケーションは DBMS とアプリケーションとの間でデータを転送するとき (DB2 CLI 関数を呼び出すとき) に、 C データ・タイプを SQL データ・タイプと突き合わせなければなりません。

これをアドレス指定するのに役立つように、 DB2 CLI はさまざまなデータ・タイプに記号名を付け、 DBMS とアプリケーションとの間のデータ転送を管理します。また、必要に応じてデータ変換 (たとえば、 C 文字ストリングから SQL INTEGER タイプに) も行います。これを実行するには、 DB2 CLI はソースとターゲットの両方のデータ・タイプを認識している必要があります。アプリケーションは記号名を使用して両方のデータ・タイプを識別します。

C および SQL データ・タイプ

表2 では、各 SQL データ・タイプ、それに対応する記号名、および省略時の C 記号名をリストしています。

SQL データ・タイプ

この列には SQL CREATE DDL ステートメントに表示される SQL データ・タイプを示します。 SQL データ・タイプは DBMS に従属します。

記号 SQL データ・タイプ

この列には、整数値として (sqlcli.h で) 定義される SQL 記号名を示します。この値は、最初の列にリストした SQL データ・タイプを識別するために、さまざまな関数によって使用されます。この値を使用する例については、374ページの『例』を参照してください。

省略時の C 記号データ・タイプ

この列には C 記号名を示してあります。これも整数値として定義されています。この値は、 35ページの表3 で示す C データ・タイプを識別するために、さまざまな関数実引き数で使用されます。記号名は、さまざまな関数 (SQLBindParameter()、SQLGetData()、 SQLBindCol() など) によってアプリケーション変数の C データ・タイプを識別する際に使用されます。これらの関数の呼び出し時に C データ・タイプを明示的に識別する代わりに、 SQL_C_DEFAULT を指定することもでき、その場合 DB2 CLI は、この表で示したパラメーターまたは列の SQL データ・タイプに基づく省略時 C データ・タイプを想定します。たとえば、SQL_DECIMAL の省略時 C データ・タイプは SQL_C_CHAR です。

表 2. SQL 記号データ・タイプおよび省略時データ・タイプ

SQL データ・タイプ	記号 SQL データ・タイプ	省略時記号 C データ・タイプ
BIGINT	SQL_BIGINT	SQL_C_BIGINT
BLOB	SQL_BLOB	SQL_C_BINARY
BLOB LOCATOR [a]	SQL_BLOB_LOCATOR	SQL_C_BLOB_LOCATOR
CHAR	SQL_CHAR	SQL_C_CHAR
CHAR FOR BIT DATA [b]	SQL_BINARY	SQL_C_BINARY
CLOB	SQL_CLOB	SQL_C_CHAR
CLOB LOCATOR [a]	SQL_CLOB_LOCATOR	SQL_C_CLOB_LOCATOR
DATE	SQL_TYPE_DATE [d]	SQL_C__TYPE_DATE [d]
DBCLOB	SQL_DBCLOB	SQL_C_DBCHAR

表 2. SQL 記号データ・タイプおよび省略時データ・タイプ (続き)

SQL データ・タイプ	記号 SQL データ・タイプ	省略時記号 C データ・タイプ
DBCLOB LOCATOR [a]	SQL_DBCLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR
DECIMAL	SQL_DECIMAL	SQL_C_CHAR
DOUBLE	SQL_DOUBLE	SQL_C_DOUBLE
FLOAT	SQL_FLOAT	SQL_C_DOUBLE
GRAPHIC	SQL_GRAPHIC	SQL_C_DBCHAR
INTEGER	SQL_INTEGER	SQL_C_LONG
LONG VARCHAR [b]	SQL_LONGVARCHAR	SQL_C_CHAR
LONG VARCHAR FOR BIT DATA [b]	SQL_LONGVARBINARY	SQL_C_BINARY
LONG VARGRAPHIC [b]	SQL_LONGVARGRAPHIC	SQL_C_DBCHAR
NUMERIC [c]	SQL_NUMERIC [c]	SQL_C_CHAR
REAL	SQL_REAL	SQL_C_FLOAT
SMALLINT	SQL_SMALLINT	SQL_C_SHORT
TIME	SQL_TYPE_TIME [d]	SQL_C_TYPE_TIME [d]
TIMESTAMP	SQL_TYPE_TIMESTAMP [d]	SQL_C_TYPE_TIMESTAMP [d]
VARCHAR	SQL_VARCHAR	SQL_C_CHAR
VARCHAR FOR BIT DATA [b]	SQL_VARBINARY	SQL_C_BINARY
VARGRAPHIC	SQL_VARGRAPHIC	SQL_C_DBCHAR

a LOB ロケーター・タイプは持続性のある SQL データ・タイプではありません (列はロケーター・タイプでは定義できません。パラメーター・マーカーの記述か LOB 値の表記にのみ使用されます)。 123ページの『ラージ・オブジェクトの使用』を参照してください。

b LONG データ・タイプおよび FOR BIT DATA データ・タイプは、可能であれば必ず該当する LOB タイプに置き換えてください。

c DB2 (MVS/ESA 版)、DB2 (VSE および VM 版) および DB2 ユニバーサル・データベースでは、 NUMERIC は DECIMAL の同義語です。

d 以前のリリースで使用されたデータ・タイプに関する詳細については、 821ページの『付録B. アプリケーションの移行』を参照してください。

注: データ・タイプ DATE、DECIMAL、NUMERIC、TIME、および TIMESTAMP は、変換せずに省略時 C バッファー・タイプに転送することはできません。

35ページの表3 には、それぞれの記号 C タイプごとに総称型定義を示しています。

C 記号データ・タイプ

この列には C 記号名を示してあります。これは整数値として定義されています。この値は、最後の列に示された C データ・タイプを識別するために、さまざまな関数実引き数で使用されます。この値を使用する例については、265ページの『例』を参照してください。

C タイプ

この列には C 定義済みタイプを示してあります。これは C `typedef` ステートメントを使用して `sqlcli.h` で定義されるものです。この列にある値を使用して、アプリケーションの可搬性を高めるために、すべての DB2 CLI 関連の変数および引き数を宣言します。関数の引き数に使用される追加の記号データ・タイプのリストについては、 37ページの表5 を参照してください。

基本 C タイプ

この列は参考のためにのみ示してあります。すべての変数および引き数は、前の列にある記号タイプを使用して定義します。一部の値は、36ページの表4 で記述されている C 構造体です。

表3. C データ・タイプ

C 記号データ・タイプ	C タイプ	基本 C タイプ
SQL_C_CHAR	SQLCHAR	unsigned char
SQL_C_BIT	SQLCHAR	unsigned char または char (値 1 または 0)
SQL_C_TINYINT	SQLSCHAR	signed char (範囲 -128 〜 127)
SQL_C_SHORT	SQLSMALLINT	short int
SQL_C_LONG	SQLINTEGER	long int
SQL_C_DOUBLE	SQLDOUBLE	double
SQL_C_FLOAT	SQLREAL	float
SQL_C_SBIGINT	SQLBIGINT	_int64
SQL_C_UBIGINT	SQLBIGINT	unsigned _int64
SQL_C_NUMERIC [c]	SQL_NUMERIC_STRUCT	詳細は、36ページの表4 を参照
SQL_C_TYPE_DATE [b]	DATE_STRUCT	詳細は、36ページの表4 を参照
SQL_C_TYPE_TIME [b]	TIME_STRUCT	詳細は、36ページの表4 を参照
SQL_C_TYPE_TIMESTAMP [b]	TIMESTAMP_STRUCT	詳細は、36ページの表4 を参照
SQL_C_CLOB_LOCATOR [a]	SQLINTEGER	long int
SQL_C_BINARY	SQLCHAR	unsigned char
SQL_C_BLOB_LOCATOR [a]	SQLINTEGER	long int

表3. C データ・タイプ (続き)

C 記号データ・タイプ	C タイプ	基本 C タイプ
SQL_C_DBCHAR	SQLDBCHAR	wchar_t
SQL_C_DBCLOB_LOCATOR	SQLINTEGER	long int

a LOB ロケーター・タイプ。

b 以前のリリースで使用されたデータ・タイプに関する詳細は、 821ページの『付録B. アプリケーションの移行』を参照してください。

c 32 ビット Windows のみ。

注: DB2 CLI では、fcSQL ファイル参照データ・タイプ (組み込み SQL で使用される) は必要ありません。詳細は、123ページの『ラージ・オブジェクトの使用』を参照してください。

表4. C 構造体

C タイプ	総称構造	Windows 構造
DATE_STRUCT	typedef struct DATE_STRUCT { SQLSMALLINT year; SQLUSMALLINT month; SQLUSMALLINT day; } DATE_STRUCT;	typedef struct tagDATE_STRUCT { SWORD year; UWORD month; UWORD day; } DATE_STRUCT;
TIME_STRUCT	typedef struct TIME_STRUCT { SQLUSMALLINT hour; SQLUSMALLINT minute; SQLUSMALLINT second; } TIME_STRUCT;	typedef struct tagTIME_STRUCT { UWORD hour; UWORD minutes; UWORD second; } TIME_STRUCT;
TIMESTAMP_STRUCT	typedef struct TIMESTAMP_STRUCT { SQLUSMALLINT year; SQLUSMALLINT month; SQLUSMALLINT day; SQLUSMALLINT hour; SQLUSMALLINT minute; SQLUSMALLINT second; SQLINTEGER fraction; } TIMESTAMP_STRUCT;	typedef struct tagTIMESTAMP_STRUCT { SWORD year; UWORD month; UWORD day; UWORD hour; UWORD minute; UWORD second; UDWORD fraction; } TIMESTAMP_STRUCT;

表 4. C 構造体 (続き)

C タイプ	総称構造	Windows 構造
SQL_NUMERIC_STRUCT	(なし。Windows 32 ビットのみ)	typedef struct tagSQL_NUMERIC_STRUCT { SQLCHAR precision; SQLCHAR scale; SQLCHAR sign; [a] SQLCHAR val[SQL_MAX_NUMERIC_LEN];[b] [c] } SQL_NUMERIC_STRUCT;

SQLUSMALLINT C データ・タイプに関する詳細は、表5 を参照してください。

a 符号付きフィールド: 1 = 正数、2 = 負数

b 数値は、位取り整数として、 SQL_NUMERIC_STRUCT 構造体の値フィールドにリトル・エンディアン・モード (重要性の最も低いバイトが左端のバイトになる) で保管されます。たとえば、数値 10.001 基数 10 に 4 桁の位取りを指定すると、100010 という整数に位取りされます。この数値は 16 進形式では 186AA なので、 SQL_NUMERIC_STRUCT の値は 『AA 86 01 00 00 ... 00』 となり、そのバイト数は SQL_MAX_NUMERIC_LEN #define によって定義されます。

c SQL_C_NUMERIC データ・タイプの精度および位取りフィールドは、アプリケーションからの入力には使用されず、アプリケーションのドライバーからの入力のみに使用されます。ドライバーは、SQL_NUMERIC_STRUCT に数値を書き込むときに独自の省略時値を精度フィールドの値として使用し、アプリケーション記述子 (省略時設定は 0) の SQL_DESC_SCALE フィールドの値を位取りフィールドに使用します。アプリケーションは、アプリケーション記述子の SQL_DESC_PRECISION および SQL_DESC_SCALE フィールドを設定して、精度および位取りに指定する独自の値を提供することができます。

その他の C データ・タイプ

SQL データ・タイプにマップするデータ・タイプに加えて、ポインターやハンドルなどの他の関数実引き数に使用される C 記号タイプもあります。総称データ・タイプおよび ODBC データ・タイプの両方を次に示します。

表 5. C データ・タイプおよび基本 C データ・タイプ

定義済み C タイプ	基本 C タイプ	一般的な使用法
SQLPOINTER	void *	データおよびパラメーターの記憶域を指すポインター。
SQLHANDLE	long int	ハンドル情報の全 4 タイプを参照するハンドル。
SQLHENV	long int	環境情報を参照するハンドル。
SQLHDBC	long int	データベース接続情報を参照するハンドル。
SQLHSTMT	long int	ステートメント情報を参照するハンドル。
SQLUSMALLINT	unsigned short int	無符号の短整数値用の関数入力引き数。
SQLUINTEGER	unsigned long int	無符号の長整数値用の関数入力引き数。

表5. C データ・タイプおよび基本 C データ・タイプ (続き)

定義済み C タイプ	基本 C タイプ	一般的な使用法
SQLRETURN	short int	DB2 CLI 関数からの戻りコード。

バージョン 2.1: 以前のバージョンの DB2 CLI では、次のようになっていました。
- 長 (32 ビット) 整数として SQLRETURN が定義されていました。
- SQLUSMALLINT および SQLUINTEGER の代わりに SQLSMALLINT および SQLINTEGER が (無符号の代わりに符号付きが) 使用されていました。詳細については、821ページの『付録B. アプリケーションの移行』を参照してください。

データ変換

前述のように、DB2 CLI はアプリケーションと DBMS との間のデータの転送と、必要な変換を管理します。データ転送が実際に行われる前に、ソース、ターゲット、または両方のデータ・タイプが、 `SQLBindParameter()`、`SQLBindCol()`、または `SQLGetData()` の呼び出し時に指示されます。これらの関数は、33ページの表2 に示されている記号タイプの名前を使用して、必要なデータ・タイプを識別します。

たとえば、SQL データ・タイプ DECIMAL(5,3) に対応するパラメーター・マーカーを、アプリケーションの C バッファー・タイプ DOUBLE にバインドする場合、該当する `SQLBindParameter()` 呼び出しは次のようになります。

```
SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_DOUBLE,
                  SQL_DECIMAL, 5, 3, double_ptr, 0, NULL);
```

33ページの表2 は、省略時のデータ変換のみを示しています。前の段落で述べた関数を使用して、データを他のタイプに変換することができます。ただし、すべてのデータ変換がサポートされていたり、意味をなすわけではありません。 39ページの表6 は、DB2 CLI がサポートする変換をすべて示しています。

39ページの表6 の最初の列には SQL データ・タイプが入ります。その後の列は C データ・タイプを示しています。 C データ・タイプの列には、次のものが入ります。

D
: 変換はサポートされており、SQL データ・タイプの省略時変換です。

X
: すべての IBM DBMS が変換をサポートします。

ブランク
: いずれの IBM DBMS も変換をサポートしません。

例として、CHAR (または39ページの表6 に示されている C 文字ストリング) を SQL_C_LONG (符号付き LONG) へ変換できることが表に示されています。対照的に、LONGVARCHAR は SQL_C_LONG へ変換できません。

必須の形式およびデータ・タイプ間の変換結果については、 875ページの『付録F. データ変換』を参照してください。

精度と位取りに関する制限に関して *SQL 解説書* に指定されている規則、およびタイプ変換に関して切り捨ておよび丸めの規則があります。これらの規則はDB2 CLI に適用されますが、以下の例外があります。すなわち、数値の小数点の右側の値が切り捨てられると切り捨て警告が返され、小数点の左側が切り捨てられるとエラーが返されるというものです。エラーの場合には、アプリケーションが `SQLGetDiagRec()` を呼び出して SQLSTATE および障害についての追加情報を得る必要があります。浮動小数点データ値をアプリケーションと DB2 CLI 間で移動したり変換する場合、その対応が正確である保証はありません。値が精度および位取りの点で変わる可能性があるからです。

表 6. サポートされるデータ変換

SQL データ・タイプ	SQL_C_CHAR	SQL_C_LONG	SQL_C_SHORT	SQL_C_TINYINT	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_TYPE_DATE	SQL_C_TYPE_TIME	SQL_C_TYPE_TIMESTAMP	SQL_C_BINARY	SQL_C_BIT	SQL_C_DBCHAR	SQL_C_CLOB_LOCATOR	SQL_C_BLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR	SQL_C_BIGINT	SQL_C_NUMERIC
BLOB	X									D				X			
CHAR	D	X	X	X	X	X	X	X	X	X	X					X	X
CLOB	D									X			X				
DATE	X						D		X								
DBCLOB										X		D			X		
DECIMAL	D	X	X	X	X	X				X	X					X	X

表 6. サポートされるデータ変換 (続き)

SQL データ・タイプ	SQL_C_CHAR	SQL_C_LONG	SQL_C_SHORT	SQL_C_TINYINT	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_TYPE_DATE	SQL_C_TYPE_TIME	SQL_C_TYPE_TIMESTAMP	SQL_C_BINARY	SQL_C_BIT	SQL_C_DBCHAR	SQL_C_CLOB_LOCATOR	SQL_C_BLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR	SQL_C_BIGINT	SQL_C_NUMERIC
DOUBLE	X	X	X	X	X	D					X					X	X
FLOAT	X	X	X	X	X	D					X					X	X
GRAPHIC	X											D					
INTEGER	X	D	X	X	X	X					X					X	X
LONG VARCHAR	D									X							
LONG VARGRAPHIC	X									X		D					
NUMERIC	D	X	X	X	X	X					X						X
REAL	X	X	X	X	D	X					X						X
SMALLINT	X	X	D	X	X	X					X					X	X
BIGINT	X	X	X	X	X	X				X	X					D	X
TIME	X							D	X								
TIMESTAMP	X						X	X	D								
VARCHAR	D	X	X	X	X	X	X	X	X	X	X					X	X
VARGRAPHIC	X											D					

表 6. サポートされるデータ変換 (続き)

SQL データ・タイプ	SQL_C_CHAR	SQL_C_LONG	SQL_C_SHORT	SQL_C_TINYINT	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_TYPE_DATE	SQL_C_TYPE_TIME	SQL_C_TYPE_TIMESTAMP	SQL_C_BINARY	SQL_C_BIT	SQL_C_DBCHAR	SQL_C_CLOB_LOCATOR	SQL_C_BLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR	SQL_C_BIGINT	SQL_C_NUMERIC

注:

- データは LOB ロケーター・タイプには変換されません。ロケーターはデータ値を表します。詳細は、123ページの『ラージ・オブジェクトの使用』を参照してください。
- REAL は DB2 ユニバーサル・データベースではサポートされません。
- DB2 (MVS/ESA 版)、DB2 (VSE および VM 版)、および DB2 ユニバーサル・データベースでは、NUMERIC は DECIMAL の同義語です。
- SQL_C_NUMERIC は 32 ビット Windows オペレーティング・システムでのみ使用できます。

ストリング引き数の処理

以下に示す規則によって、DB2 CLI 関数のストリング引き数の処理のさまざまな面を取り扱います。

ストリング引き数の長さ

入力ストリング引き数には、1 つの関連した長さ引き数があります。この引き数が指示するのは、引き数の正確な長さ (ヌル終止符を除く)、ヌル終了ストリングを示す特殊値 SQL_NTS、または NULL 値を渡す SQL_NULL_DATA のうちのいずれかです。長さを SQL_NTS に設定すると、DB2 CLI はヌル終止符を見つけてストリングの長さを判別します。

出力ストリング引き数には、2 つの関連した長さ引き数があります。 1 つは割り振られる出力バッファーの長さを指定する入力長さ引き数で、もう 1 つは DB2 CLI が返したストリングの実際の長さを返す出力長さ引き数です。戻される長さの値は、戻りに使用できるストリングの全長です。それがバッファーに適合するかどうかとは関係ありません。

SQL 列データの場合、出力がヌルであれば、SQL_NULL_DATA が長さ引き数に戻され、出力バッファーは考慮されません。列の値が NULL 値の場合、記述子フィールド SQL_DESC_INDICATOR_PTR は SQL_NULL_DATA にセットされます。その他のフィールド設定を含む詳細については、SQLSetDescField() の 717ページの SQL_DESC_INDICATOR_PTR を参照してください。

出力長さ引き数にヌル・ポインターを指定して関数が呼び出される場合、 DB2 CLI は長さを戻さず、データ・バッファーがデータを保持できる大きさであると想定します。出力データが NULL 値であっても、DB2 CLI はその値が NULL 値であることを示すことはできません。結果セットの列に NULL 値が入る可能性があるときは、出力長さ引き数を指す有効なポインターを必ず指定しなければなりません。有効な出力長さ引き数を必ず使用することを強くお勧めします。

ストリングのヌル終了

省略時設定では、 DB2 CLI が戻すすべての文字ストリングがヌル終了記号 (16 進数 00) で終わります。ただし、図形および DBCLOB データ・タイプから SQL_C_CHAR アプリケーション変数へ戻されるストリングは除きます。SQL_C_DBCHAR アプリケーション変数に取り出される図形および DBCLOB データ・タイプは、 2 バイトのヌル終了記号でヌル終了します。このためすべてのバッファーが、予期される最大バイト数にヌル終了記号を加えた値が入る大きさのスペースを割り振る必要があります。

また、SQLSetEnvAttr() を使用し、環境属性を設定して、可変長出力 (文字ストリング) データのヌル終了を使用不能にすることもできます。この場合には、アプリケーションが予期される最長のストリングと同じ長さにバッファーを正確に割り振ります。アプリケーションは、出力長さ引き数の記憶域を指す有効なポインターを与えなければならず、これにより DB2 CLI は戻されるデータの実際の長さを示すことができます。こうしないと、アプリケーションにはこの長さを判別する方法が何もないことになります。 DB2 CLI の省略時は、常にヌル終了記号を書き込むことです。

PATCH1 CLI/ODBC 構成キーワードを使用すると、 DB2 CLI にヌル終了の図形および DBCLOB ストリングを挿入することが可能です。このキーワード

は、クライアント構成アシスタント (CCA) によりアクセス可能な CLI/ODBC 設定ノートブックから設定が可能です。 158ページの『CLI/ODBC アクセスのためのプラットフォーム固有の詳細情報』を参照してください。ご使用のプラットフォームの *CLI/ODBC ドライバーの構成* セクションには、上記キーワードを設定するのに必要なステップが載せられています。 174ページの『構成キーワード』にある PATCH1 の説明には、図形および DBCLOB ストリングのヌル終了記号を挿入するのに必要な設定を見いだす方法も示されています。

ストリングの切り捨て

出力ストリングがバッファーに入りきらない場合、 DB2 CLI はバッファーのサイズにストリングを切り捨て、ヌル終了記号を書き込みます。切り捨てが行われると、関数は SQL_SUCCESS_WITH_INFO と、切り捨てを示す SQLSTATE **01**004 を戻します。それからアプリケーションはバッファー長と出力長を比較して、どのストリングが切り捨てられたかを判別することができます。

たとえば、`SQLFetch()` が、 SQL_SUCCESS_WITH_INFO と SQLSTATE **01**004 を戻す場合、列にバインドされたバッファーのうちの少なくとも 1 つが小さ過ぎてデータを保持できないということになります。列にバインドされたバッファーごとに、アプリケーションはバッファー長と出力長を比較してどの列が切り捨てられたかを判別できます。

ストリングの解釈

通常、 DB2 CLI はストリング引き数を大文字と小文字の区別をして解釈し、値からスペースをトリムすることはありません。 1 つの例外は、`SQLSetCursorName()` 関数のカーソル名の入力引き数です。この場合、カーソル名が区切られ (二重引用符で囲まれ) ないと、前書きおよび後書きブランクが除去され、大文字小文字は無視されます。

環境およびデータ・ソース情報の照会

アプリケーションが、現行の DB2 CLI ドライバーまたは接続先のデータ・ソースの特性と機能に関する情報を必要とする場合が多くあります。 DB2 CLI には、この情報を戻す関数が多数備わっています。

アプリケーションがこの情報を必要とする状況としては、一般に次の 2 つがあります。

- アプリケーションがユーザーに関する情報を表示する場合。データ・ソースの名前とバージョン、または DB2 CLI ドライバーのバージョンなどの情報は、接続時に表示されたり、エラー報告処理の一部として表示されることがあります。

- 一部の (すべてではない) データベース・サーバーから利用できる機能に適合し、その機能の利点を活用するように作成された総称アプリケーション。

次の DB2 CLI 関数は、データ・ソースの特定情報を提供します。
- 365ページの『SQLDataSources - データ・ソースのリストを入手する』
- 529ページの『SQLGetFunctions - 関数の入手』
- 533ページの『SQLGetInfo - 一般情報の入手』
- 603ページの『SQLGetTypeInfo - データ・タイプ情報の入手』

環境情報の照会の例

576ページの『例』に示されている getinfo.c の例では、 DB2 に接続されている場合に次の出力が発生します。

```
       Server Name: SAMPLE
     Database Name: SAMPLE
     Instance Name: db2inst1
         DBMS Name: DB2/6000
      DBMS Version: 05.00.0000
    CLI Driver Name: libdb2.a
 CLI Driver Version: 05.00.0000
 ODBC SQL Conformance Level: Extended Grammar
```

第3章 拡張機能の使用

このセクションでは、高度なタスクを多数説明します。

このセクションでは、ユーザー定義関数やトリガーなどの、動的 SQL から通常得られる機能については扱いません。サポートされる SQL 言語の詳細な説明については、*SQL 解説書* を参照してください。

環境、接続、およびステートメントの属性

環境、接続、およびステートメントには、それぞれ定義済みの属性 (またはオプション) の集まりがあります。アプリケーションですべての属性を照会することができますが、省略時値を変更できるのは一部の属性だけです。アプリケーションで属性値を変更して、DB2 CLI の動作を変更することができます。

環境ハンドルには、その環境での DB2 CLI 関数の動作を制御する属性があります。アプリケーションでは、`SQLSetEnvAttr()` を呼び出して属性の値を指定したり、 `SQLGetEnvAttr()` を呼び出して現行属性値を得ることができます。`SQLSetEnvAttr()` は、接続ハンドルを割り振る前にのみ呼び出すことができます。

接続ハンドルには、その接続での DB2 CLI 関数の動作を制御する属性があります。変更できる属性は、次の種類に分かれます。
- 一度接続ハンドルが割り振られたら、いつでも設定できるもの
- 実際の接続の確立が完了する前に限り設定できるもの
- 接続が確立された後に限り設定できるもの
- 接続が確立された後で、未解決のトランザクションまたはオープン・カーソルがない場合に限り設定できるもの

アプリケーションでは、 SQLSetConnectAttr() を呼び出して接続属性の値を変更したり、 SQLGetConnectAttr() を呼び出して属性の現行値を得ることができます。ハンドルが割り振られた後であればいつでも設定できる接続属性の例としては、 25ページの『コミットまたはロールバック』で紹介した自動コミット・オプションがあります。各属性が設定可能なタイミングに関する詳細な説明については、 663ページの『SQLSetConnectAttr - 接続属性を設定する』を参照してください。

ステートメント・ハンドルには、そのステートメント・ハンドルを使って実行する CLI 関数の動作を制御する属性があります。変更できるステートメント属性は、次の種類に分かれます。
- 複数の属性を設定可能であるが、現行では 1 つの特定の値にのみ設定できるもの
- ステートメント・ハンドルが割り振られた後でいつでも設定できるもの
- オープン・カーソルがそのステートメント・ハンドルにない場合に限り設定できるもの

アプリケーションでは、 SQLSetStmtAttr() を呼び出して設定可能なステートメント属性の値を指定したり、 SQLGetStmtAttr() を呼び出して属性の現行値を得ることができます。各属性が設定可能なタイミングに関する詳細な説明については、 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』を参照してください。

SQLSetConnectAttr() 関数を、ステートメント属性を設定するために使用することはできません。この関数は、バージョン 5 より前の DB2 CLI でサポートされていました。詳しくは、824ページの『SQLSetConnectAttr() を使用してステートメント属性のサブセットを設定する』を参照してください。

アプリケーションの多くは、省略時の属性設定値だけを使用します。しかし、これらの省略時値の一部が、アプリケーションの特定のユーザーには適さない状況もありえます。 DB2 CLI には、エンド・ユーザーが実行時にこれらの省略時値の一部を変更する方式が 2 つ備えられています。 1 番目の方式は、 SQLDriverConnect() および SQLBrowseConnect() 関数に接続ストリングを入力

するときに新しい省略時属性値を指定するものです。 2 番目の方式は、DB2 CLI 初期設定ファイル内の新しい省略時属性値の仕様にかかわるものです。

DB2 CLI 初期設定ファイルを使用して、そのワークステーション上のすべての DB2 CLI アプリケーションについて省略時値を変更することができます。 SQLDriverConnect() 接続ストリング中に省略時属性値を指定する方法がアプリケーションで使用できない場合は、エンド・ユーザーにとってこの方法が省略時値を変更する唯一の方法になります。 SQLDriverConnect() に指定される省略時属性値によって、特定の接続に関する DB2 CLI 初期設定ファイル内の値が指定変更されます。エンド・ユーザーが DB2 CLI 初期設定ファイルを使用する方法に関する情報と、変更できる省略時値のリストについては、174ページの『DB2 CLI/ODBC 構成キーワードのリスト』を参照してください。

省略時値変更のメカニズムは、エンド・ユーザーの調整用です。アプリケーション開発者は、適切な属性設定関数を使用する必要があります。アプリケーションが属性設定やオプション関数を、初期設定ファイルや接続ストリングの指定とは違う値を指定して呼び出すと、初期省略時値は指定変更され、新しい値が有効になります。

変更できる属性は、*設定* 属性またはオプション関数の詳しい説明のところにリストされています。 233ページの『第5章 DB2 CLI 関数』を参照してください。読み取り専用オプションは、*獲得* 属性またはオプション関数の詳しい説明のところにリストされています。

一般に使用される属性の一部については、811ページの『付録A. プログラミングのヒントと提案』を参照してください。

次の図は、基本的な接続シナリオに属性関数を追加する様子を示しています。

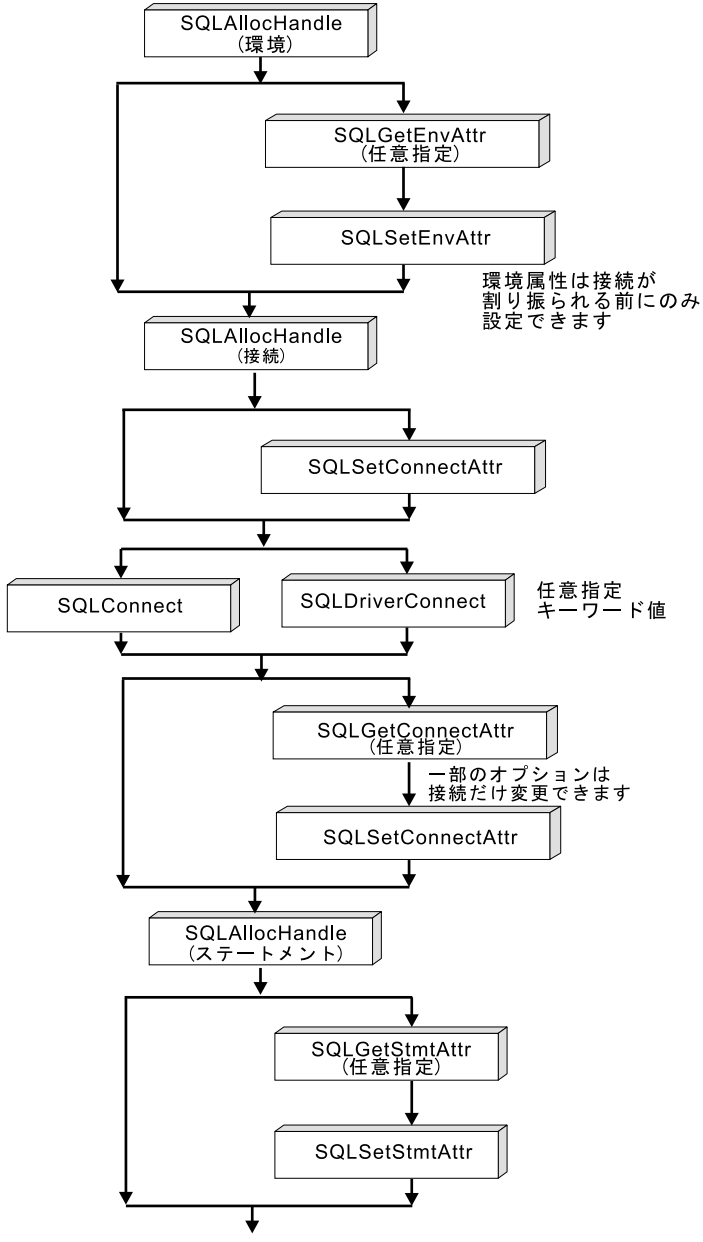


図4. 属性 (オプション) の設定と取り出し

マルチスレッドのアプリケーション作成

DB2 CLI は次のプラットフォーム上でスレッドの並行実行をサポートしています。

- AIX
- HP UX-11
- OS/2
- Silicon Graphics IRIX
- Sun Solaris
- 32 ビット Windows

スレッドをサポートするその他のプラットフォームでは、 DB2 CLI は DB2 CLI に対するすべての呼び出しをシリアル化することでスレッドの安全を保証しています。言い換えれば、DB2 CLI は常に再入可能ということです。

注: アプリケーションを作成していて、 DB2 CLI 呼び出しおよび組み込み SQL または DB2 API 呼び出しを使用する場合には、 53ページの『マルチスレッド混合アプリケーション』を参照してください。

並行実行とは、 2 つのスレッドが (同時に実行可能なマルチプロセッサー上で) それぞれ独立して実行できることを表しています。たとえば、アプリケーションはデータベース間のコピーを次の方法で実現することができます。

- 1 つのスレッドがデータベース A に接続し、 `SQLExecute()` および `SQLFetch()` 呼び出しを使って、 1 つの接続から共用アプリケーション・バッファーの中へデータを読み取ります。
- もう 1 つのスレッドがデータベース B に接続し、並行して上記共用バッファーからデータを読み取り、データベース B に挿入します。

対照的に、DB2 CLI がすべての関数呼び出しをシリアル化する場合は、一度に 1 つのスレッドだけが DB2 CLI 関数を実行することができます。その他のスレッドすべては現行スレッドの処理が終わるまで待ってからでなければ、実行の機会を獲得することはできません。

マルチ・スレッドの用途

DB2 CLI アプリケーション内にもう一つのスレッドを作成する一般的な理由の多くは、実行しているスレッド以外のスレッドを使用すると、(たとえば、長時間の照会の実行を避けて) `SQLCancel()` を呼び出せるようにすることができます。

注: 複数スレッドをサポートしているプラットフォームでは、非同期の SQL モデル (これは Windows 3.1 のようなスレッドのないオペレーティング・

システム (OS) 用に設計されている) を使うよりも、上記の方式を使用すべきです。ご使用になるアプリケーションでマルチ・スレッドを使用できない場合、 145ページの『CLI の非同期実行』を参照してください。

たいていの GUI ベースのアプリケーションではスレッドを使用して、ユーザーとの対話が優先度の高いスレッドで扱われるようにしています。それに比べると、他のアプリケーション・タスクは優先度が低くなっています。アプリケーションでは、 1 つのスレッドだけですべての DB2 CLI 関数 (SQLCancel() は例外です) を実行できるようにしています。この場合、スレッド関連のアプリケーション設計上の問題はありません。それは、 DB2 CLI との対話に使用するデータ・バッファーを 1 つのスレッドだけがアクセスできるようにしているからです。

複数の接続を使用し、幾らかの時間がかかるステートメントを実行しているアプリケーションでは、スループットを改善するために、マルチ・スレッドで DB2 CLI 関数を実行することを考慮してください。そのようなアプリケーションは、マルチスレッドのアプリケーション、特にデータ・バッファーの共用が関係するマルチスレッド・アプリケーションを作成する際の標準的な慣習に従ってください。以下のセクションでは、複雑なマルチスレッド・アプリケーションを作成するために、 DB2 CLI が保証することと、アプリケーションが保証する必要のあることについて、詳細を説明します。

プログラミングのヒント

DB2 CLI で割り振られる資源は、スレッドの安全が保証されています。これは、共用グローバルまたは接続特有のセマフォーのいずれかを使用して成し遂げられます。同時に 1 つのスレッドだけが、環境ハンドルを入力として受け入れる DB2 CLI 関数を実行することが可能です。接続ハンドル (つまりその接続ハンドル上で割り振られるステートメントまたは記述子) を受け入れるその他の関数すべては、接続ハンドル上でシリアル化されます。

このことは、スレッドが接続ハンドル (または接続ハンドルの子) を指定して関数の実行を一度開始すると、他のスレッドはブロックされ、実行中のスレッドが返されるまで待機することを意味しています。これに対する 1 つの例外は SQLCancel() で、別のスレッドで現在実行しているステートメントを取り消すことができます。この理由のために、最も無理のない設計とは、接続ごとに 1 つのスレッドを対応付け、 SQLCancel() 要求を処理するためにさらに 1 つのスレッドを加えることです。こうすれば、各スレッドは他のスレッドから独立して実行可能です。

一例として、スレッドが、あるスレッド内の 1 つのハンドルを使用していて、それから別のスレッドが関数呼び出しの間にそのハンドルを解放した場合、そのハンドルを使用する次の試みには結果として SQL_INVALID_HANDLE の戻りコードが生じることになります。

注: これは DB2 CLI アプリケーションにのみ適用されます。この場合のハンドルはポインターであり、別のスレッドがそのハンドルを解放していれば、そのポインターがもはや有効ではないので、ODBC アプリケーションはトラップします。この理由のために、下記のモデルに従うのが最善です。

注: マルチスレッド・アプリケーションには、プラットフォームやコンパイラーに固有のリンク・オプションが必要になることがあります。詳細な説明については、*アプリケーション構築の手引き* を参照してください。

サンプル・アプリケーションのモデル

次に 1 つの例を示します。

- 次のものを割り当てるマスター・スレッドを指定します。
  - *m* 個の ″子″ スレッド
  - *n* 個の接続ハンドル
- 接続が必要なそれぞれのタスクは、子スレッドの 1 つにより実行されます。そして、*n* 個の接続の 1 つがマスター・スレッドにより与えられます。
- 子スレッドが接続プールに接続を返すまで、各接続はマスター・スレッドにより使用中としてマークされます。
- `SQLCancel()` 要求がマスター・スレッドにより処理されます。

このモデルを使用すると、非 SQL 関連のタスクを実行するのに複数のスレッドが使用される場合には、マスター・スレッドは接続よりも多くのスレッドを持つことができ、アプリケーションが種々のデータベースに対するアクティブ接続のプールを維持し、しかもアクティブ・タスクの数を制限したい場合には、マスター・スレッドはスレッドよりも多くの接続を持つことができます。

さらに重要なことに、これにより 2 つのスレッドが同時に同一の接続またはステートメント・ハンドルを使用しようとすることがなくなります。 DB2 CLI はその資源へのアクセスを制御しますが、結合列やパラメーター・バッファーのようなアプリケーション資源は DB2 CLI により制御されません。したがってアプリケーションは、バッファーへのポインターが同時に 2 つのスレッドで使用されないように保証する必要があります。すべての据え置き引き数は、列またはパラメーターがアンバインドされるまで有効に保つ必要があります。

2 つのスレッドがデータ・バッファーを共用することが必要な場合、アプリケーションは何らかの形の同期メカニズムを実装する必要があります。たとえ

ば、上記に言及されているデータベース間のコピー・シナリオにおいて、共用バッファーの使用はアプリケーションによって同期をとる必要があります。

アプリケーションのデッドロック

アプリケーションは、データベースおよびアプリケーションにある共用資源でデッドロック状態が発生する可能性を考慮に入れておく必要があります。

DB2 はサーバーでデッドロックを検出すると、 1 つまたは複数のトランザクションをロールバックしてデッドロックを解消することができます。それでも、次のような場合、アプリケーションにはデッドロックの可能性があります。

- 2 つのスレッドが同一データベースに接続されている。さらに、
- 1 つのスレッドがアプリケーション資源を保留して、データベース資源を待っている。そして、
- アプリケーション資源を待っている間に、他のスレッドがデータベース資源にロックしている場合。

上記の場合には、DB2 サーバーはデッドロックではなく、ロックだけを探そうとします。それで、データベース LOCKTIMEOUT 構成の設定が変更されない限り、アプリケーションはいつまでも待ち続けることになります。

上記に提案されているモデルは、この問題を避け、スレッドが接続における実行を一度開始すると、スレッド間でアプリケーション資源を共有することはありません。

既存マルチスレッド・アプリケーションの問題

既存のマルチスレッドの DB2 CLI アプリケーションが、シリアル化されたバージョン (バージョン 5 より前) の DB2 CLI を使用して正常に実行したにもかかわらず、バージョン 5 またはそれ以降の DB2 CLI を使用して実行するときに同期の問題を被ることがあります。

この場合には、 DISABLEMULTITHREAD の CLI/ODBC 構成キーワードを、1 にセットしてください。それによって、DB2 CLI はすべての関数呼び出しをシリアル化することができます。これが必要な場合には、アプリケーションを分析の上、訂正してください。

マルチスレッド混合アプリケーション

マルチスレッド・アプリケーションで、 CLI 呼び出しを DB2 API 呼び出しや組み込み SQL と混合することができます。アプリケーションの編成を最善のものにするには、どのタイプの呼び出しを最初に行うかを考慮する必要があります。

最初に DB2 CLI 呼び出しを実行する場合

DB2 CLI ドライバーは自動的に DB2 のコンテキスト API を呼び出し、アプリケーション用のコンテキストを割り当てて管理します。このことは、他の DB2 API または組み込み SQL を呼び出す前に SQLAllocEnv() を呼び出すすべてのアプリケーションが、 SQL_CTX_MULTI_MANUAL に設定されるコンテキスト・タイプで初期設定されることを示します。

この場合には、アプリケーションで DB2 CLI を使用して、すべてのコンテキストを割り当てて管理する必要があります。 DB2 CLI を使用して、すべての接続ハンドルを割り振り、すべての接続を実行します。組み込み SQL を呼び出す前に、各スレッドで SQLSetConnect() 関数を呼び出してください。 DB2 CLI 関数が同一スレッドに呼び出された後に、DB2 API は呼び出し可能となります。

最初に DB2 API 呼び出しか組み込み SQL を実行する場合

アプリケーションが CLI 関数の前に DB2 API または組み込み SQL を呼び出す場合は、 DB2 CLI ドライバーは DB2 のコンテキスト API を自動的に呼び出しません。

DB2 API または組み込み SQL を呼び出すすべてのスレッドはコンテキストに結び付いている必要があります。そうでないと、その呼び出しは SQL1445N の SQLCODE により失敗します。スレッドをコンテキストに明示的に結び付ける DB2 API sqleAttachToCtx()、または DB2 CLI 関数 (たとえば、SQLSetConnection()) を呼び出すことでこのことを行えます。

この場合には、アプリケーションがすべてのコンテキストを明示的に管理しなければなりません。

コンテキスト API を用いて、DB2 CLI 関数を呼び出す前にコンテキストを割り当て接続します。 (SQLAllocEnv() は、既存のコンテキストを省略時のコンテキストとして使用します)。 SQL_ATTR_CONN_CONTEXT の接続属性を使用して、それぞれの DB2 CLI 接続が用いるコンテキストを明示的に設定します。

既存の混合アプリケーションの実行に関する詳細は、821ページの『付録B. アプリケーションの移行』を参照してください。

複数サイト更新 (2 フェーズ・コミット)

15ページの『1 つまたは複数のデータ・ソースへの接続』に記載されているトランザクションのシナリオは、 1 つのトランザクションでただ 1 つのデータベース・サーバーと対話するアプリケーションを例として取り上げています。並行トランザクションには並行接続を用いることができますが、異なるトランザクションどうしが調整されることはありません。

複数サイト更新のことを、分散作業単位 (DUOW)、2 フェーズ・コミット (2PC)、整合分散トランザクションともいいます。これを用いると、アプリケーションが複数のリモート・データベース・サーバー中のデータを更新しても、整合性が保証されます。

複数サイト更新の好例として、一般的な銀行用トランザクションがあります。ある口座から、データベース・サーバーの異なる別の口座にお金を移動する場合を考えてみましょう。このトランザクションの場合、一方の口座に対する借方記入操作という更新がコミットされるのは、他方の口座に対する貸方記入処理という更新もコミットされている場合に限られていることが重要です。複数サイト更新に関する考慮事項は、両方の口座を表すデータが 2 つの別々のデータベース・サーバーによって管理されている場合に適用されます。

複数サイト更新によっては、トランザクション・マネージャーを使用して、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ことが含まれます。複数サイト更新の詳細説明については、*管理の手引き* を参照してください。このセクションでは、さまざまなトランザクション・マネージャーを使用するために DB2 CLI アプリケーションを作成する方法について説明します。

- 『DB2 をトランザクション・モニターとして使用する場合』
- 62ページの『Microsoft Transaction Server (MTS) をトランザクション・モニターとして使用する場合』
- 70ページの『プロセス・ベースの XA 準拠トランザクション・プログラム・モニター (XA TP)』
- 71ページの『ホストおよび AS/400 データベース・サーバー』

DB2 をトランザクション・モニターとして使用する場合

DB2 CLI/ODBC アプリケーションで DB2 自体をトランザクション・マネージャー (DB2 TM) として使用し、すべての IBM データベース・サーバーに対して分散トランザクションの調整を行えます。 DB2 をトランザクション・マネージャーとして使用する場合の要件と機能に関する詳細は、 *管理の手引き* を参照してください。

構成 - DB2 をトランザクション・モニターとして使用する場合

DB2 トランザクション・マネージャーをセットアップするには、 *管理の手引き* の情報に従わなければなりません。

CLI/ODBC アプリケーション中で DB2 をトランザクション・マネージャーとして使用するには、 CLI/ODBC 構成キーワードを次のように設定しなければなりません。

```
[COMMON]
DISABLEMULTITHREAD = 1
CONNECTTYPE=2
SYNCPOINT=2
```

CLI/ODBC 構成キーワードの設定について詳しくは、158ページの『CLI/ODBC アクセスのためのプラットフォーム固有の詳細情報』を参照してください。

上記の構成キーワードのうち 2 つは、次の環境属性を使用して設定することもできます。

- SQL_ATTR_CONNECT_TYPE を SQL_COORDINATED_TRANS に変更する。
- SQL_ATTR_SYNCPOINT を SQL_TWOPHASE に変更する。

詳細については、`SQLSetEnvAttr()` を参照してください。
DISABLEMULTITHREAD キーワードには該当する環境属性はないので、今までどおり db2cli.ini ファイル内の [COMMON] セクション中で 1 に設定しなければなりません。

DISABLEMULTITHREAD キーワードが db2cli.ini ファイルの [COMMON] セクションになければならないということは、当該クライアント・インスタンスからすべてのデータ・ソースへのすべての接続に影響が及ぶことを意味します。したがって、 DB2 クライアント・インスタンスはプロセス・ベースの CLI アプリケーションかスレッド・ベースの CLI アプリケーションのうちどちらか一方だけをサポートできますが、両方ともサポートすることはできません。

プログラミングに関する考慮事項

DB2 CLI/ODBC 構成キーワード DISABLEMULTITHREAD は 1 に設定しなければなりません。したがって DB2 CLI/ODBC ドライバーでは、アプリケーション・プロセスで行われたすべての接続について 1 つの DB2 コンテキストが使用されることになります。データベース要求はすべてプロセス・レベルで直列化されます。

環境属性 SQL_ATTR_CONNECTTYPE は、アプリケーションを整合分散環境で実行するか、または非整合分散環境で実行するかを制御します。この属性に指定可能な 2 つの値は、以下のとおりです。

- SQL_CONCURRENT_TRANS - 第 2 章で説明されているトランザクションの持つ意味 1 つにつき 1 つのデータベースをサポートします。同一データベースおよび異なるデータベースへの複数接続が許可されています。これが省略時値です。
- SQL_COORDINATED_TRANS - トランザクションの持つ意味 1 つにつき複数のデータベースをサポートします (これについては、以下に説明します)。

アプリケーション内のすべての接続の SQL_ATTR_CONNECTTYPE 設定は、同じでなければなりません。この環境属性はアプリケーションが設定することをお勧めします。必要であれば、環境ハンドルが `SQLAllocHandle()` への呼び出しにより (SQL_HANDLE_ENV の *HandleType* を指定して) 作成された直後に設定してください。 ODBC アプリケーションは `SQLSetEnvAttr()` にアクセスできないため、接続を確立する前に `SQLSetConnectAttr()` を使用してこの設定を行うことになります。

複数サイト更新セマンティクスを管理する属性: 整合トランザクションは、複数のデータベース接続の間でコミットやロールバックが調整 (整合) されることを意味します。 SQL_ATTR_CONNECTTYPE 属性の SQL_COORDINATED_TRANS 設定は、 IBM の組み込み SQL にあるタイプ 2 CONNECT に対応するもので、 SQL_ATTR_SYNC_POINT 属性と関連づけて考える必要があります。この属性には、次の 2 つの設定が考えられます。

- SQL_ONEPHASE: 複数のデータベース・トランザクションの各データベースが行う作業をコミットするときは、 1 フェーズ・コミットが使用されます。データ保全性を確かにするには、 1 つのトランザクションで 2 つ以上のデータベースを更新することがないようにします。 1 つのトランザクションで実行された更新のうち最初のデータベースだけが、そのトランザクションでのただ 1 つの更新側になり、それ以外のすべてのデータベースはアクセスされても読み取り専用になります。このトランザクション内でこれらの読み取り専用のデータベースを更新しようとしても拒否されます。
- 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。

SQL_ATTR_CONNECTTYPE の場合と同じように、この環境属性はアプリケーションが設定することをお勧めします。必要であれば、環境ハンドルが SQLAllocHandle() への呼び出しにより (SQL_HANDLE_ENV の *HandleType* を指定して) 作成された直後に設定してください。 ODBC アプリケーションでは、SQLSetConnectAttr() を使用して、接続を確立する前にこの環境のもとで接続ハンドルごとに設定しなければなりません。

アプリケーション内のすべての接続の SQL_ATTR_CONNECTTYPE および SQL_ATTR_SYNC_POINT 設定は同じでなければなりません。最初の接続が確立された後は、それ以降のすべての接続タイプは最初のものと同じでなければなりません。整合接続は、手動コミット・モードに省略時設定されています。(自動コミット・モードについては、 25ページの『コミットまたはロールバック』を参照してください。)

DB2 をトランザクション・マネージャーとして実行している際には、複数サイト更新環境で関数 SQLEndTran() を使用しなければなりません。

59ページの図5 は、 2 つの SQL_CONCURRENT_TRANS 接続 ('A' と 'B') でステートメントを実行時のアプリケーションの論理フローを表すとともに、トランザクションの有効範囲を示しています。

60ページの図6 は、同一ステートメントが 2 つの SQL_COORDINATED_TRANS 接続 ('A' および 'B') で実行されているのを表しており、整合分散トランザクションの有効範囲を示しています。

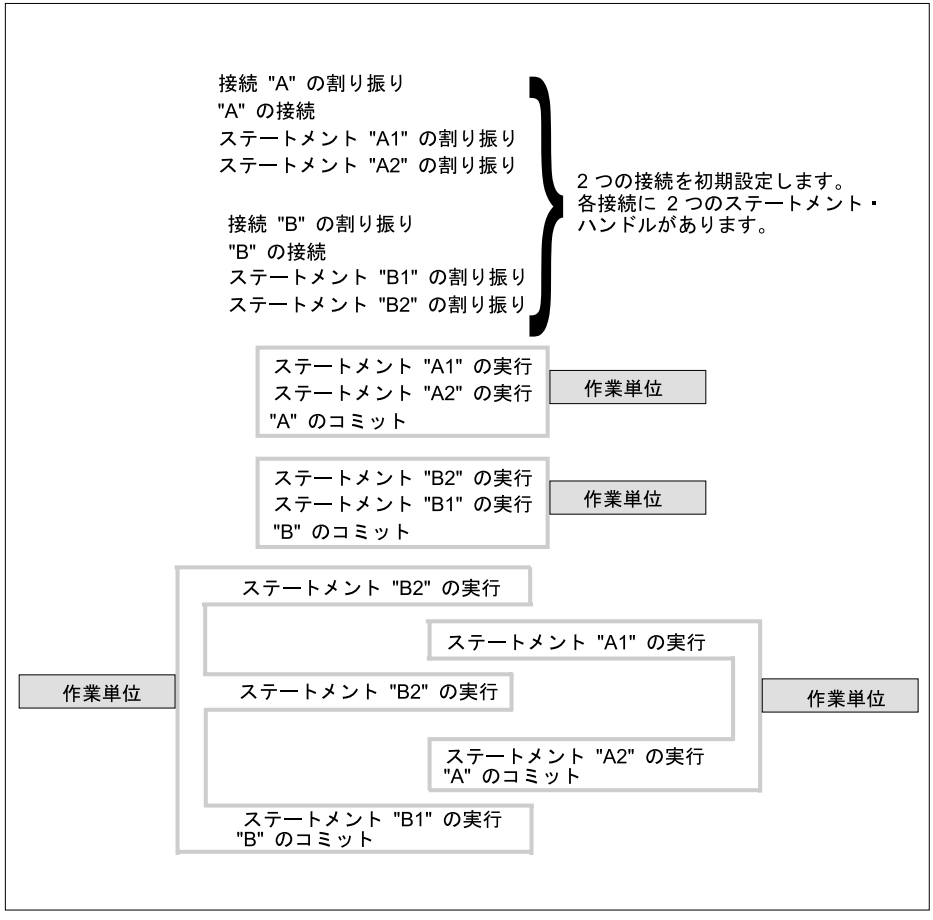


図5. 並行トランザクションを用いての複数接続

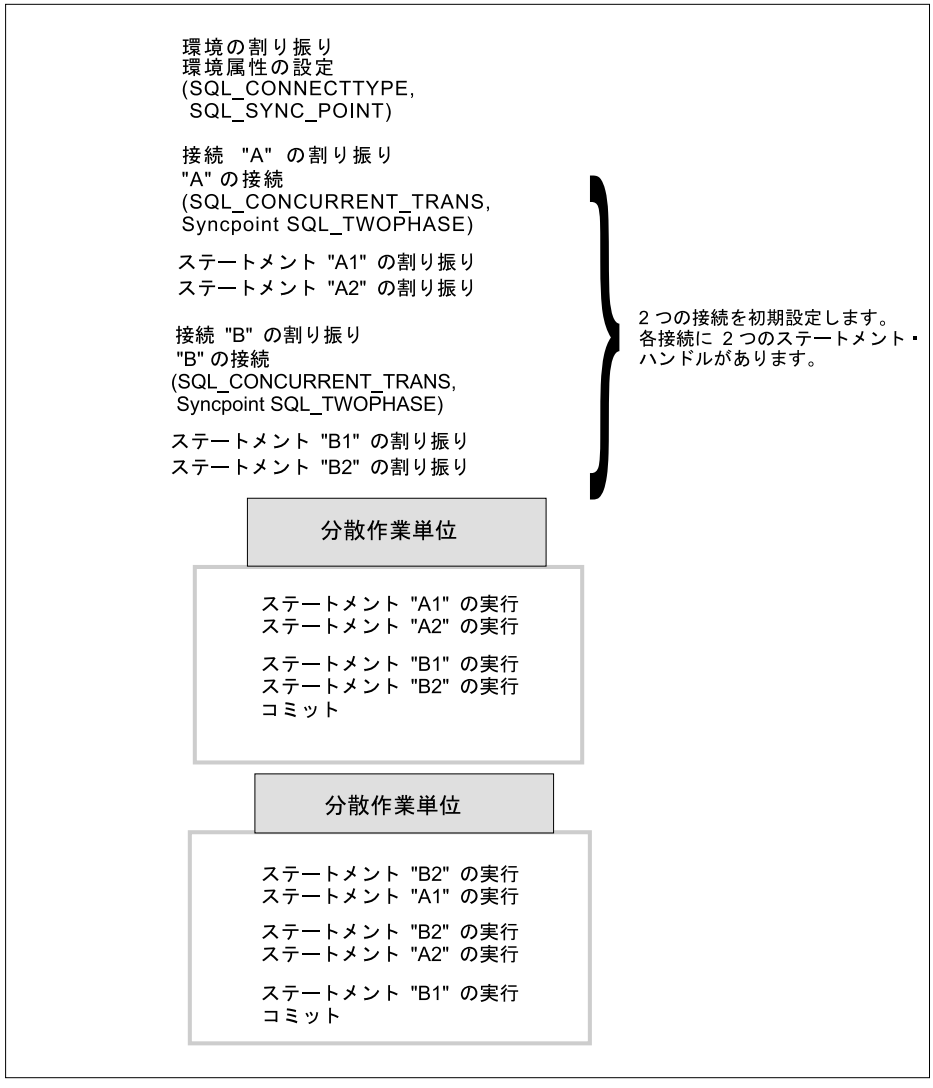


図 6. 整合トランザクションを用いての複数接続

整合トランザクション接続の確立: アプリケーションが整合トランザクション接続を確立するには、 SQLSetEnvAttr() 関数を呼び出すか、または CONNECTTYPE キーワードおよび SYNCPOINT キーワードを DB2 CLI 初期設定ファイルまたは SQLDriverConnect() の接続ストリングに設定します。初期設定ファイルは、SQLSetConnectAttr() 関数を使用しない既存のアプリケーションのためのものです。キーワードの詳細については、174ページの『DB2 CLI/ODBC 構成キーワードのリスト』を参照してください。

アプリケーションでは、並行接続と整合接続を混合して使うことはできません。最初の接続のタイプが決まると、それ以降のすべての接続のタイプはそれに従います。 SQLSetEnvAttr() は、接続活動中に接続タイプを変更しようとすると、エラーが返されます。

制限

複数サイト更新環境で組み込み SQL と CLI/ODBC 呼び出しを混合することはサポートされていますが、混合アプリケーションを作成する際の制約事項がすべて適用されます。詳細については、143ページの『組み込み SQL と DB2 CLI の混合』を参照してください。

例

次の例では、SQL_COORDINATED_TRANS に設定される SQL_ATTR_CONNECTTYPE、および SQL_ONEPHASE に設定される SQL_ATTR_SYNC_POINT を使用して 2 つのデータ・ソースへの接続を示しています。

```
/* ... */
/* main */
int main( int argc, char * argv[] ) {
    SQLHANDLE henv, hdbc[MAX_CONNECTIONS] ;
    SQLRETURN rc ;
/* ... */
    /* allocate an environment handle */
    SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv ) ;
    /*
     Before allocating any connection handles, set Environment wide
     Connect Options
     Set to Connect Type 2, Syncpoint 1
    */
    if ( SQLSetEnvAttr( henv,
                        SQL_CONNECTTYPE,
                        ( SQLPOINTER ) SQL_COORDINATED_TRANS,
                        0
                      ) != SQL_SUCCESS ) {
       printf( ">---ERROR while setting Connect Type 2 ------------¥n" ) ;
       return( SQL_ERROR ) ;
    }
/* ... */
    if ( SQLSetEnvAttr( henv,
                        SQL_SYNC_POINT,
                        ( SQLPOINTER ) SQL_ONEPHASE,
                        0
                      ) != SQL_SUCCESS ) {
       printf( ">---ERROR while setting Syncpoint One Phase ------------¥n" ) ;
       return( SQL_ERROR ) ;
    }
/* ... */
```

```
/* Connect to first data source */
prompted_connect( henv, &hdbc[0] ) ;
/* Connect to second data source */
DBconnect( henv, &hdbc[1] ) ;
/*********   Start Processing Step  ************************/
/* allocate statement handle, execute statement, etc.       */
/*********   End Processing Step  *************************/
/* Disconnect, free handles and exit */
```

Microsoft Transaction Server (MTS) をトランザクション・モニターとして使用する場合

Windows NT、Windows 95、および Windows 98 オペレーティング・システムの Microsoft Transaction Server (MTS) のもとで稼働しているアプリケーションでは、 MTS を使用して複数の DB2 UDB、ホスト、および AS/400 データベース・サーバーや MTS に準拠した他のリソース・マネージャーとの 2 フェーズ・コミットを調整できます。

Microsoft Transaction Server (MTS) をサポートするための新しい接続属性が、SQLSetConnectAttr() 中に作成されています。 SQL_ATTR_ENLIST_IN_DTC に関する情報は、663ページの『SQLSetConnectAttr - 接続属性を設定する』を参照してください。

MTS のソフトウェア前提条件

MTS をサポートするにはバージョン 5.2 以上の DB2 クライアントが必要です。また、MTS は Hotfix 0772 のバージョン 2.0 以降でなければなりません。

Microsoft Transaction Server の構成

DB2 UDB V5.2 以降を Microsoft Transaction Server (MTS) バージョン 2.0 と完全に統合することができます。 Windows 32 ビット オペレーティング・システムの MTS のもとで稼働しているアプリケーションでは、 MTS を使用して複数の DB2 UDB、ホスト、および AS/400 データベース・サーバーや MTS に準拠した他のリソース・マネージャーとの 2 フェーズ・コミットを調整できます。

DB2 で MTS サポートを使用可能にする: Microsoft Transaction Server サポートは自動的に使用可能になります。 *tp_mon_name* データベース・マネージャー構成パラメーターを MTS に設定することもできますが、これは必要ないので無視されます。

注: DB2 MTS サポートのインストールと構成に役立つ追加の技術情報が、IBM Web サイトにあります。 URL を

http://www.ibm.com/software/data/db2/library/ に設定し、 DB2 ユニバーサル・データベースの ″Technote″ をキーワード ″MTS″ で検索してください。

MTS のソフトウェア前提条件: MTS をサポートするには DB2 クライアント・アプリケーション・イネーブラー (CAE) バージョン 5.2 以降が必要です。また、MTS のバージョンは Hotfix 0772 のバージョン 2.0 以降でなければなりません。

DB2 ODBC ドライバーを Windows 32 ビット オペレーティング・システムにインストールすると、新しいキーワードがレジストリーに自動的に追加されます。

```
HKEY_LOCAL_MACHINE¥software¥ODBC¥odbcinit.ini¥IBM DB2 ODBC Driver:
Keyword Value Name: CPTimeout
Data Type: REG_SZ
Value: 60
```

インストールおよび構成: MTS のインストールと構成に関する考慮事項を次に要約します。 DB2 の MTS サポートを使用するには、以下の手順に従わなければなりません。

1. MTS アプリケーションを実行するマシンに MTS と DB2 クライアントを両方ともインストールします。
2. ホストまたは AS/400 データベース・サーバーが複数サイト更新に関係している場合は、次のようにします。
   a. ローカル・マシンかリモート・マシンに、 DB2 コネクト エンタープライズ・エディション (EE) をインストールします。 DB2 コネクト EE を使用すると、ホストまたは AS/400 データベース・サーバーを複数サイト更新トランザクションに加えることができます。
   b. 複数サイト更新に DB2 コネクト EE サーバーを使用できることを確認します。複数サイト更新に DB2 コネクトを使用できるようにすることに関する情報は、ご使用のプラットフォームの『DB2 コネクト エンタープライズ・エディション 概説およびインストール』を参照してください。

DB2 CLI/ODBC アプリケーションを実行するときは、次の構成キーワード (db2cli.ini ファイルに設定されている) を省略時値から変更しないでください。

- CONNECTYPE キーワード (省略時値 1)
- MULTICONNECT キーワード (省略時値 1)
- DISABLEMULTITHREAD キーワード (省略時値 1)

- CONNECTIONPOOLING キーワード (省略時値 0)
- KEEPCONNECTION キーワード (省略時値 0)

MTS サポートを使用できるようにするために作成された DB2 CLI アプリケーションが、上記のキーワードに対応している属性値を変更することのないようにしてください。また、アプリケーションが次の属性の省略時値を変更しないようにしてください。

- SQL_ATTR_CONNECT_TYPE 属性 (省略時値 SQL_CONCURRENT_TRANS)
- SQL_ATTR_CONNECTON_POOLING 属性 (省略時値 SQL_CP_OFF)

注: DB2 MTS サポートのインストールと構成に役立つ追加の技術情報が、IBM Web サイトにあります。 URL を http://www.ibm.com/software/data/db2/library/ に設定し、 DB2 ユニバーサル・データベースの ″Technote″ をキーワード ″MTS″ で検索してください。

インストールの検査:

1. DB2 クライアントと DB2 コネクト EE を、 DB2 UDB、ホスト、または AS/400 サーバーにアクセスするように構成します。
2. DB2 CAE マシンから DB2 UDB データベース・サーバーへの接続を検査します。
3. DB2 コネクト・マシンからホストまたは AS/400 データベース・サーバーへの DB2 CLP を使用した接続を検査し、 2 つか 3 つほど照会を発行します。
4. DB2 CAE マシンからホストまたは AS/400 データベース・サーバーへの DB2 コネクト・ゲートウェイを介した接続を検査し、 2 つか 3 つほど照会を発行します。

サポートされている DB2 データベース・サーバー: MTS 調整トランザクションを使用した複数サイト更新用に、次のサーバーがサポートされています。

- DB2 ユニバーサル・データベース エンタープライズ・エディション、バージョン 5.2
- DB2 エンタープライズ拡張エディション、バージョン 5.2
- DB2 (OS/390 版)
- DB2 (MVS 版)
- DB2 (AS/400 版)
- DB2 (VM および VSE 版)
- DB2 Common Server for SCO、バージョン 2

- DB2 ユニバーサル・データベース (AIX 版)、PTF U453782
- DB2 ユニバーサル・データベース (HP-UX 版)、PTF U453784
- DB2 ユニバーサル・データベース エンタープライズ・エディション (OS/2 版)、PTF WR09033
- DB2 ユニバーサル・データベース (Solaris 実行環境版)、PTF U453783
- DB2 ユニバーサル・データベース エンタープライズ・エディション (Windows NT 版)、PTF WR09034
- DB2 ユニバーサル・データベース エンタープライズ拡張エディション (UNIX 版) または (Windows NT 版)

MTS トランザクション・タイムアウトと DB2 接続の動作: MTS Explorer というツールでトランザクション・タイムアウト値を設定できます。詳細については、オンラインの *MTS Administrator Guide* を参照してください。

トランザクションがトランザクション・タイムアウト値 (省略時値は 60 秒) より長くかかる場合は、 MTS は関係しているすべてのリソース・マネージャーの打ち切りを非同期的に発行し、トランザクション全体が打ち切られます。

DB2 サーバーに対する接続の場合、打ち切りは DB2 ロールバック要求に変換されます。接続に関するロールバック要求は、他のデータベース要求と同様に直列化されるので、データベース・サーバー上のデータの保全性が保証されます。

次のような結果になります。

- 接続がアイドル中の場合は、ロールバックが即時に実行されます。
- 実行時間の長い SQL ステートメントを処理している場合、ロールバック要求は SQL ステートメントが完了するまで待ちます。

接続のプール処理: 接続のプール処理を行うと、アプリケーションが接続のプールから接続を使用できるようになり、接続を使用するたびに再確立する必要がなくなります。接続を作成してプールに入れると、アプリケーションは接続プロセスを全部実行する代わりに、その接続を再利用できます。接続がプールされるのは、アプリケーションが ODBC データ・ソースから切断され、同じ属性の接続が新しく行われた場合です。

接続のプール処理は ODBC ドライバー・マネージャー 2.x の機能です。 MTS には最新版の ODBC ドライバー・マネージャー (バージョン 3.5) が付属していますが、このバージョンでは接続のプール処理の構成に多少の変更が加えられており、またトランザクションの MTS COM オブジェクト間の ODBC 接

続に関する新機能があります (67ページの『同一トランザクションに加わっている COM オブジェクト間の ODBC 接続の再利用』を参照)。

ODBC ドライバー・マネージャー 3.5 で接続のプール処理を起動するには、ODBC ドライバーが新しいキーワードをレジストリーに登録する必要があります。該当するキーワードは次のとおりです。

```
Key Name: SOFTWARE¥ODBC¥ODBCINST.INI¥IBM DB2 ODBC DRIVER
Name: CPTimeout
Type: REG_SZ
Data: 60
```

32 ビット Windows オペレーティング・システム用の DB2 ODBC ドライバーのバージョン 6 以降は、接続のプール処理を完全にサポートしているので、このキーワードは登録済みです。バージョン 5.2 のクライアントの場合は修正パッケージ 3 (WR09024) 以降をインストールしなければなりません。

省略時値の 60 は、接続が 60 秒間プールされた後に切断されることを示しています。

業務の多い環境では、CPTimeout 値の数値を大きくして (Microsoft 社では特定の環境の場合に 10 分に設定するよう勧めることがある) 物理的な接続と切断の回数を抑えた方が賢明です。その理由は、この種の処理はシステム・メモリーや通信スタック資源を含む多大なシステム資源を使用するからです。

加えて、複数プロセッサー・マシンにおける同じトランザクション内のオブジェクト間で、同じ接続が確実に使用されるようにするために、「1 プロセッサーに対して複数のプール」のサポートをオフにする必要があります。そのためには、以下のレジストリー設定値を `odbcpool.reg` というファイルにコピーし、平文テキスト・ファイルとして保管し、 **odbcpool.reg** コマンドを発行します。 Windows オペレーティング・システムは以下のレジストリー設定値をインポートします。

```
REGEDIT4

[HKEY_LOCAL_MACHINE¥SOFTWARE¥ODBC¥ODBCINST.INI¥ODBC Connection Pooling]
"NumberOfPools"="1"
```

このキーワードを 1 に設定しないと、MTS は異なるプールに接続をプールすることがあり、したがって同じ接続を再利用しないことになります。

ADO 2.1 以降を使用した MTS 接続プール: MTS COM オブジェクトが ADO を使ってデータベースにアクセスする場合、 OLEDB リソース・プールをオフにする必要があります。そうすれば、 ODBC 用の Microsoft OLEDB プロバイダー (MSDASQL) が ODBC 接続プールを妨害することはありません。

この機能は ADO 2.0 では OFF に初期設定されますが、 ADO 2.1 では ON に初期設定されます。 OLEDB リソース・ポーリングをオフにするには、以下の行を oledb.reg というファイルにコピーし、平文テキスト・ファイルとして保管し、 **oledb.reg** コマンドを発行します。 Windows オペレーティング・システムは以下のレジストリー設定値をインポートします。

```
REGEDIT4

[HKEY_CLASSES_ROOT¥CLSID¥{c8b522cb-5cf3-11ce-ade5-00aa0044773d}]

@="MSDASQL"
"OLEDB_SERVICES"=dword:fffffffc
```

同一トランザクションに加わっている COM オブジェクト間の ODBC 接続の再利用: MTS COM オブジェクト間の ODBC 接続では、接続のプール処理が (COM オブジェクトがトランザクションかどうかに関係なく) 自動的にオンになります。

複数の MTS COM オブジェクトが同一トランザクションに加わっている場合、次の方法で接続を複数の COM オブジェクト間で再利用できます。

COM1 と COM2 という 2 つの COM オブジェクトが、同じ ODBC データ・ソースに接続しており、同じトランザクションに加わっているとします。

COM1 が接続して稼働した後、切断すると接続がプール処理されます。しかし、この接続は同一トランザクションの別の COM オブジェクトで使用するために予約されます。現行のトランザクションが終了した後に限り、他のトランザクションで使用できます。

同一トランザクション中で COM2 が起動されると、プール処理された接続が使用されます。 MTS は、同一トランザクションに加わっている COM オブジェクトだけに確実に接続が使用されるようにします。

一方、COM1 が明示的に切断されない場合は、トランザクションが終了するまで接続は占有されます。同一トランザクション中で COM2 が起動されると、別の接続が使用されます。その後このトランザクションでは 1 つの接続ではなく 2 つの接続が占有されます。

同一トランザクションに加わっている COM オブジェクト間で接続を再利用する機能には、次のような利点があります。

- クライアントとサーバーの両方で使用される資源が少なくて済む。必要な接続は 1 つだけになります。

- 同一トランザクション (同一データベース・サーバーおよび同一データにアクセスしている) に加わっている 2 つの接続が互いにロックされないようになる。ロックされない理由は、 DB2 サーバーでは MTS COM オブジェクトからの複数の接続が別のトランザクションとして扱われるからです。

TCP/IP 通信の調整: 同時に多数の物理接続や切断が行われるような作業負荷が大きい環境で、小さい CPTimeout 値が使用されていると、 TCP/IP スタックで資源に関する制約が生じることがあります。

この問題を軽減するには、TCP/IP レジストリー項目を使用してください。この点については、 *Windows NT Resource Guide* Volume 1 で説明されています。レジストリー・キー値は、 HKEY_LOCAL_MACHINE—> SYSTEM—> CurrentControlSet—> Services—> TCPIP—> Parameters にあります。

省略時値と推奨されている設定値は次のとおりです。

名前	省略時値	推奨値
KeepAlive 時間	7200000 (2 時間)	省略時値
KeepAlive 間隔	1000 (1 秒)	10000 (10 秒)
TcpKeepCnt	120 (2 分)	240 (4 分)
TcpKeepTries	20 (20 回再試行する)	省略時値
TcpMaxConnectAttempts	3	6
TcpMaxConnectRetransmission	3	6
TcpMaxDataRetransmission	5	8
TcpMaxRetransmissionAttempts	7	10
レジストリー値が定義されていない場合は、作成してください。		

MTS ″BANK″ サンプル・アプリケーションで DB2 のテストを行う: MTS に付属の ″BANK″ サンプル・プログラムを使用して、クライアントの製品と MTS のセットアップをテストできます。

次のステップに従ってください。

1. ファイル
   ¥Program Files¥Common Files¥ODBC¥Data Sources¥ MTSSamples.dsn に切り換えます。次のように表示されます。

```
[ODBC]
DRIVER=IBM DB2 ODBC DRIVER
UID=your_user_id
PWD=your_password
DSN=your_database_alias
Description=MTS Samples
```

ここで、

- *your_user_id* および *your_password* は、ホストに接続する際に使用するユーザー ID とパスワード
- *your_database_alias* はデータベース・サーバーに接続するのに使用するデータベース別名

2. 「コントロールパネル (Control Panel)」の「ODBC 管理 (ODBC Administration)」に進み、**「システム DSN (System DSN)」**タブを選択して、データ・ソースを追加します。
   a. 「IBM ODBC ドライバー (IBM ODBC Driver)」を選択してから**「完了 (Finish)」**を選択します。
   b. データベース別名のリストが表示されたら、以前に指定した別名を選択します。
   c. **「OK」**を選択します。
3. DB2 CLP を使用して、上記の ID *your_user_id* で DB2 データベースに接続します。
   a. `db2cli.lst` ファイルをバインドします。

      ```
      db2 bind @C:¥sqllib¥bnd¥db2cli.lst blocking all grant public
      ```

   b. ユーティリティーをバインドします。

      DRDA ホスト・サーバーをサーバーとして使用している場合は、接続先のホスト (OS/390、AS/400、または VSE&VM) に応じて、`ddcsmvs.lst`、 `ddcs400.lst`、または `ddcsvm.lst` をバインドします。たとえば、次のとおりです。

      ```
      db2 bind @C:¥sqllib¥bnd¥@ddcsmvs.lst blocking all grant public
      ```

      あるいは、`db2ubind.lst` ファイルをバインドします。

      ```
      db2 bind @C:¥sqllib¥bnd¥@db2ubind.lst blocking all grant public
      ```

   c. 次のように MTS サンプル・アプリケーションのサンプル表とデータを作成します。

      ```
      db2 create table account (accountno int, balance int)
      db2 insert into account values(1, 1)
      ```

4. DB2 クライアントで、データベース・マネージャー構成パラメーター *tp_mon_name* が `MTS` に設定されていることを確認します。

5. ″BANK″ アプリケーションを実行します。「**口座 (Account)**」および「**Visual C++**」オプションを選択してから、要求を実行依頼します。他のオプションは SQL サーバーに固有の SQL を使用する場合があるので、作動しないことがあります。

プログラミングに関する考慮事項

DB2 CLI/ODBC アプリケーションを実行するには、次の構成キーワード (db2cli.ini ファイル中に設定されている) を省略時値から変更しないでください。

- CONNECTYPE キーワード (省略時値 1)
- MULTICONNECT キーワード (省略時値 1)
- DISABLEMULTITHREAD キーワード (省略時値 1)
- CONNECTIONPOOLING キーワード (省略時値 0)
- KEEPCONNECTION キーワード (省略時値 0)

MTS サポートを使用できるようにするために作成された DB2 CLI アプリケーションが、上記のキーワードに対応している属性値を変更することのないようにしてください。また、アプリケーションが次の属性の設定値を変更しないようにしてください。

- SQL_ATTR_CONNECT_TYPE 属性 (省略時値 SQL_CONCURRENT_TRANS)
- SQL_ATTR_CONNECTON_POOLING 属性 (省略時値 SQL_CP_OFF)

接続を確立するには、 MTS を使用するために作成した DB2 CLI/ODBC アプリケーションで ODBC 関数が使用されていなければなりません。

制限

複数サイト更新環境で組み込み SQL と CLI/ODBC 呼び出しを混合することはサポートされていますが、混合アプリケーションを作成する際の制約事項がすべて適用されます。詳細については、143ページの『組み込み SQL と DB2 CLI の混合』を参照してください。

プロセス・ベースの XA 準拠トランザクション・プログラム・モニター (XA TP)

プロセス・ベースの XA TP (CICS や Encina など) は、プロセス当たり 1 つのアプリケーション・サーバーを始動します。個々のアプリケーション・サーバー・プロセスで、接続はすでに XA API (xa_open) を使用して確立されています。ここでは、環境構成について説明し、この環境で稼働するよう DB2 CLI/ODBC アプリケーションを作成する方法について説明します。

構成

XA トランザクション・マネージャーをセットアップするには、 *管理の手引き* の情報に従わなければなりません。

プロセス・ベースの XA TM の CLI/ODBC 構成キーワードの設定値は、 DB2 をトランザクション・マネージャーとして使用する場合と全く同じです。詳細については、56ページの『構成 - DB2 をトランザクション・モニターとして使用する場合』を参照してください。

プログラミングに関する考慮事項

この環境用の DB2 CLI/ODBC アプリケーションを作成する場合、そのアプリケーションが次のステップをすべて実行するようにしなければなりません。

- アプリケーションが最初に `SQLConnect()` または `SQLDriverConnect()` を呼び出して、 TM でオープンされる接続を CLI/ODBC 接続ハンドルに関連付けるようにしなければならない。データ・ソース名 (DSN キーワード) を指定しなければなりません。ユーザー ID とパスワードは任意指定です。
- アプリケーションが XA TM を呼び出してコミットまたはロールバックを行うようにしなければならない。したがって、CLI/ODBC ドライバーではトランザクションの終了が認識されないので、アプリケーションが終了前に次の処理を行うようにする必要があります。
  - CLI/ODBC ステートメント・ハンドルをすべて除去する。
  - `SQLDisconnect()` および `SQLFreeConnect()` (または `SQLFreeHandle()`) を呼び出して、接続ハンドルを解放する。実際のデータベース接続は、XA TM で xa_close が実行されるまで切断されません。

制限

複数サイト更新環境で組み込み SQL と CLI/ODBC 呼び出しを混合することはサポートされていますが、混合アプリケーションを作成する際の制約事項がすべて適用されます。詳細については、143ページの『組み込み SQL と DB2 CLI の混合』を参照してください。

ホストおよび AS/400 データベース・サーバー

DB2 UDB クライアントが DB2 コネクトを使用してホストまたは AS/400 DB2 データベース・サーバーに接続している場合にも、複数サイト更新はサポートされます。

構成

ホストまたは AS/400 データベース・サーバーに接続する際には、特定の DB2 CLI/ODBC 構成は必要ありません。

DB2 コネクトを実行しているマシンが、ホストに対して複数サイト更新モードで稼働できるようにするには、特定の構成設定値が必要になることがあります。詳しくは、ご使用のプラットフォームの *DB2 コネクト 概説およびインストール* を参照してください。

システム・カタログ情報の照会

アプリケーションが頻繁に行う最初のタスクの 1 つに表のリストの表示があり、このリストから処理を行う表を 1 つまたは複数選択します。アプリケーションからデータベース・システム・カタログに対して独自の照会を発行して、このタイプのカタログ情報を入手することもできますが、最善の方法はその代わりにアプリケーションから DB2 CLI カタログ関数を呼び出すことです。このようなカタログ関数を使用すると、総称インターフェースが得られ、 DB2 ファミリーのサーバー全体に照会を発行し、一貫性のある結果セットを返すことができます。そうすれば、アプリケーションはサーバーに固有のものではなくなり、カタログ照会もリリース固有のものではなくなります。

カタログ関数を使用すると、ステートメント・ハンドルによってアプリケーションに結果セットが返されます。この関数を呼び出すことは、`SQLExecDirect()` を使用してシステム・カタログ表に対して 1 つの選択を実行するのと概念的に同じです。この関数呼び出しの後で、アプリケーションは結果セットから個々の行を取り出すことができ、通常どおり `SQLFetch()` によって列データを処理します。 DB2 CLI カタログ関数は、次のとおりです。

- 343ページの『SQLColumnPrivileges - 表の列に関連した特権を入手する』
- 348ページの『SQLColumns - 表の列の情報を入手する』
- 459ページの『SQLForeignKeys - 外部キー列のリストを入手する』
- 636ページの『SQLPrimaryKeys - 表の基本キー列を入手する』
- 640ページの『SQLProcedureColumns - プロシージャーの入力 / 出力情報を入手する』
- 649ページの『SQLProcedures - プロシージャー名のリストを入手する』
- 782ページの『SQLSpecialColumns - 特殊な (行識別子) 列の入手』
- 788ページの『SQLStatistics - 基本表の索引および統計情報の入手』
- 795ページの『SQLTablePrivileges - 表に関連した特権の入手』
- 800ページの『SQLTables - 表情報の入手』
- 603ページの『SQLGetTypeInfo - データ・タイプ情報の入手』

この関数によって返される結果セットは、各カタログ関数の説明の部分で定義されています。列は、指定された順序で定義されます。今後のリリースでは、

それぞれの結果セットの定義の末尾に他の列が追加される可能性があります。したがって、そのような変更の影響を受けないような方法で、アプリケーションを作成する必要があります。

カタログ関数の中には、非常に複雑な照会を実行する結果となるものがあるので、そのような関数は必要なときにのみ呼び出すようにしてください。返された情報をアプリケーションが保管するようにし、同じ情報を入手するために繰り返し呼び出しを行うことがないようにすることをお勧めします。

カタログ関数での入力引き数

すべてのカタログ関数には、入力引き数リストに *CatalogName* および *SchemaName* (およびそれらに関連した長さ) があります。その他の入力引き数には、*TableName*、*ProcedureName*、または *ColumnName* (およびそれらに関連した長さ) があります。これらの入力引き数を使用して、返される情報の量を識別または制約します。ただし、 *CatalogName* は常にヌル・ポインター (長さは 0 に設定) でなければなりません。 DB2 CLI では現時点では 3 部分名がサポートされていないからです。

233ページの『第5章 DB2 CLI 関数』のカタログ関数の「関数の引き数」の項で、上記の入力引き数をそれぞれパターン値または普通の引き数のいずれかとして説明します。たとえば、`SQLColumnPrivileges()` は、 *SchemaName* および *TableName* を普通の引き数として扱い、 *ColumnName* をパターン値として扱います。

普通の引き数として扱われる入力はリテラルで、大文字小文字の区別が有効です。引き数は、照会を修飾はしませんが、ご希望の情報を識別します。この引き数にアプリケーションがヌル・ポインターを渡すとエラーになります。

パターン値として扱われる入力は、一致している行のみを含めることによって結果セットのサイズを制約するのに使用します。これは、基本照会を WHERE 文節で修飾した場合と同じです。パターン値入力についてアプリケーションがヌル・ポインターを渡すと、引き数は結果セットの制限に使用されません (つまり、WHERE 文節がない)。カタログ関数に複数のパターン値入力引き数があると、基本照会内の複数の WHERE 文節が AND で結合された場合と同じように扱われます。この結果セットでは、WHERE 文節のすべての条件を満たした場合に限り行が現れます。

各パターン値引き数には、次の文字が含まれています。

- 単一文字を表す下線 (_) 文字。

- ゼロ個以上の文字の順序列を表すパーセント (%) 文字。パターン値に % が 1 つ入っていることは、その引き数についてヌル・ポインターを渡すことと同じことであることに注意してください。
- 引き数自体を表す文字。大文字小文字の区別は有効です。

これらの引き数値は、WHERE 文節内の概念 LIKE 述部で使用されます。メタデータ文字 (_、%) をそのまま扱うには、エスケープ文字を _ または % の直前に入れなければなりません。エスケープ文字自体をパターンの一部として指定するためには、それを連続して 2 回入れます。アプリケーションは、SQL_SEARCH_PATTERN_ESCAPE を指定した SQLGetInfo() を呼び出すと、エスケープ文字を判別することができます。

カタログ関数の例

browser.c サンプル・アプリケーションでは、

- 指定されたスキーマ (修飾子) 名または探索パターンに関するすべての表のリストが表示されます。
- 選択された表に関する列、特殊列、外部キー、および統計情報が返されます。

browser.c サンプルの出力を下に示します。このサンプルに関連するセグメントは、カタログ関数ごとにリストされます。

```
Enter Search Pattern for Table Schema Name:
STUDENT
Enter Search Pattern for Table Name:
%
### TABLE SCHEMA              TABLE_NAME                TABLE_TYPE
    ------------------------- ------------------------- ----------
1   STUDENT                   CUSTOMER                  TABLE
2   STUDENT                   DEPARTMENT                TABLE
3   STUDENT                   EMP_ACT                   TABLE
4   STUDENT                   EMP_PHOTO                 TABLE
5   STUDENT                   EMP_RESUME                TABLE
6   STUDENT                   EMPLOYEE                  TABLE
7   STUDENT                   NAMEID                    TABLE
8   STUDENT                   ORD_CUST                  TABLE
9   STUDENT                   ORD_LINE                  TABLE
10  STUDENT                   ORG                       TABLE
11  STUDENT                   PROD_PARTS                TABLE
12  STUDENT                   PRODUCT                   TABLE
13  STUDENT                   PROJECT                   TABLE
14  STUDENT                   STAFF                     TABLE
Enter a table Number and an action:(n [Q | C | P | I | F | T |O | L])
|Q=Quit     C=cols     P=Primary Key I=Index   F=Foreign Key |
|T=Tab Priv O=Col Priv S=Stats       L=List Tables           |
1c
Schema: STUDENT  Table Name: CUSTOMER
   CUST_NUM, NOT NULLABLE, INTeger (10)
```

```
   FIRST_NAME, NOT NULLABLE, CHARacter (30)
   LAST_NAME, NOT NULLABLE, CHARacter (30)
   STREET, NULLABLE, CHARacter (128)
   CITY, NULLABLE, CHARacter (30)
   PROV_STATE, NULLABLE, CHARacter (30)
   PZ_CODE, NULLABLE, CHARacter (9)
   COUNTRY, NULLABLE, CHARacter (30)
   PHONE_NUM, NULLABLE, CHARacter (20)
>> Hit Enter to Continue<<
1p
Primary Keys for STUDENT.CUSTOMER
 1  Column: CUST_NUM            Primary Key Name: = NULL
>> Hit Enter to Continue<<
1f
Primary Key and Foreign Keys for STUDENT.CUSTOMER
  CUST_NUM  STUDENT.ORD_CUST.CUST_NUM
      Update Rule SET NULL , Delete Rule: NO ACTION
>> Hit Enter to Continue<<
```

スクロール可能カーソル

DB2 CLI はスクロール可能カーソルをサポートしています。カーソル間で次のようなスクロール能力があります。

- 1 行または複数行ごとに下方へ
- 1 行または複数行ごとに上方へ
- 最初の行から 1 行または複数行ごとに
- 最後の行から 1 行または複数行ごとに
- カーソル内のあらかじめ保管されている場所から

次の 2 つのタイプのスクロール可能カーソルが DB2 CLI にサポートされています。

- 『静的な読み取り専用カーソル』
- 76ページの『キーセット主導カーソル』

個々のカーソルのタイプに関する説明の後に、77ページの『使用するカーソル・タイプの決定』があります。

静的な読み取り専用カーソル

このタイプのスクロール可能カーソルは静的です。一度作成されたなら、行は加えられたりまたは削除されることはありません。さらに、どの行の値も変更されません。カーソルは同一のデータにアクセスしている他のアプリケーションに影響されません。

このカーソルは読み取り専用でもあります。アプリケーションはどの値も変更することができません。カーソル行がどのようにロックされるかは、カーソル

を作成するのに使用するステートメントの分離レベルで判別されます。分離レベルとその影響の完全な説明については、*SQL 解説書* を参照してください。

キーセット主導カーソル

このタイプのスクロール可能カーソルには、静的カーソルにはない機能が 2 つ追加されています。その機能とは、基礎となるデータに加えられた変更を検出する機能と、カーソルを使用して基礎となるデータに変更を加える機能です。

キーセット主導カーソルを初めてオープンする際には、結果セット全体が存続している間だけキーがキーセットに保管されます。このキーは、カーソルに含まれている行の順序とセットを判別するのに使用されます。結果セット全体をカーソル・スクロールする際に、このキーセット中のキーを使用して個々の行の現行データ値が検索されます。カーソルで行の再取り出しが行われるたびに、カーソルを初めてオープンした際にあった値ではなく、最新の値がデータベース中で検索されます。したがって、アプリケーションによるスクロールの際に行を通過するまで変更内容はその行に反映されません。

基礎となるデータに対する変更には、キーセット主導カーソルに反映されるものもされないものも含めて、さまざまなタイプがあります。

既存の行の値に対する変更

カーソルはこのタイプの変更内容を反映します。カーソルは必要になるたびにデータベースから行を再取り出しするので、キーセット主導カーソルはそのカーソル自体や他のカーソルによって加えられた変更を検出します。

行の削除

カーソルはこのタイプの変更内容も反映します。キーセットの生成後に行セット中の行が削除されると、カーソルには『穴』として示されます。カーソルがデータベースから行を再取り出ししようとする際に、その行がないと認識します。

行の追加

カーソルはこれらのタイプの変更を反映**しません**。行セットは、カーソルが初めてオープンする際に一度だけ判別されます。 SELECT ステートメントを再発行して、新しい行が追加されているはずなのでその真偽を調べるということはありません。

キーセット主導カーソルは、`SQLBulkOperations()` または `SQLSetPos()` を呼び出して結果セット中の行を修正する際にも使用されます。

SQLBulkOperations() をキーセット主導カーソルと共に使用する

SQLBulkOperations() を使用して、行セットの追加、更新、削除、または取り出しを行えます。行を更新する方法を指定するには、*Operation* 引き数を使用します。詳しくは、306ページの『SQLBulkOperations - 行のセットの追加、更新、削除、または取り出し』の次の箇所を参照してください。

- 307ページの『バルク挿入の実行』
- 308ページの『ブックマークを使用してバルク更新を実行する』
- 309ページの『ブックマークを使用してバルク取り出しを実行する』
- 310ページの『ブックマークを使用してバルク削除を実行する』

SQLSetPos() をキーセット主導カーソルと共に使用する

SQLSetPos() を使用して、行セットを更新したり削除したりできます。行を更新する方法を指定するには、*Operation* 引き数を使用します。詳しくは、744ページの『SQLSetPos - 行セット (Rowset) でカーソル位置を設定する』の次の箇所を参照してください。

- 747ページの『SQL_UPDATE オプション』
- 748ページの『SQL_DELETE オプション』

使用するカーソル・タイプの決定

まず最初に、静的カーソルとスクロール可能カーソルのどちらを使うかを決める必要があります。アプリケーションにスクロール可能カーソルの追加機構を付加する必要がなければ、静的カーソルを使用する必要があります。

スクロール可能カーソルが必要な場合は、静的カーソルかキーセット主導カーソルのどちらかに決める必要があります。静的カーソルを使用すると、オーバーヘッドを最小に抑えられます。アプリケーションにキーセット主導カーソルの追加機構を付加する必要がなければ、静的カーソルを使用する必要があります。

アプリケーションが、基礎となるデータに対する変更を検出したり、カーソルからデータの追加、更新、または削除を行うようにする必要がある場合は、キーセット主導カーソルを使用する必要があります。

ドライバーと DBMS でサポートされているカーソルのタイプを判別するには、アプリケーションが次の *InfoType* の SQLGetInfo() を呼び出すようにする必要があります。

- SQL_DYNAMIC_CURSOR_ATTRIBUTES1
- SQL_DYNAMIC_CURSOR_ATTRIBUTES2
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2
- SQL_KEYSET_CURSOR_ATTRIBUTES1

- SQL_KEYSET_CURSOR_ATTRIBUTES2
- SQL_STATIC_CURSOR_ATTRIBUTES1
- SQL_STATIC_CURSOR_ATTRIBUTES2

結果セットから返される行セットの指定

次の用語を理解するのは重要です。

結果セット SQL SELECT ステートメントにより生成される行の完全セット。一度作成されたなら、その結果セットは変更されません。

行セット 取り出し後に返される結果セットに入っている行のサブセット。アプリケーションは、初めてデータの取り出しが行われる前に行セットのサイズを指示し、 2 回目以降の取り出しが行われる前にそのサイズを修正できます。 SQLFetchScroll() への各呼び出しで、結果セットから該当する行を指定して行セットに移植します。

ブックマーク 結果セットにある特定の行へのポインター (ブックマーク) を保管することができます。一度保管したなら、アプリケーションは結果セットのどこにでも移動し続けることができ、そして行セットを生成するためブック・マークされた行に戻ります。詳細な説明については、85ページの『スクロール可能カーソルでのブックマークの使用』を参照してください。

結果セット内の行セットの位置は、 SQLFetchScroll() への呼び出しで指定されます。たとえば、次の呼び出しでは、結果セット内の 11 番目の行を開始する行セットを生成することになります (80ページの図7 のステップ 5)。

```
SQLFetchScroll(hstmt,   /* Statement handle */
    SQL_FETCH_ABSOLUTE, /* FetchOrientation value */
    11);                /* Offset value */
```

画面ベースのアプリケーションのスクロール・バー操作は、行セットの位置に直接対応付けることが可能です。画面に表示される行幅に対する行セット・サイズを設定することで、スクロール・バーの移動を SQLFetchScroll() への呼び出しに対応づけることができます。

取り出す行セット	FetchOrientation 値	スクロール・バー
最初の行セット	SQL_FETCH_FIRST	Home: スクロール・バーを先頭に
最後の行セット	SQL_FETCH_LAST	End: スクロール・バーを末尾に

取り出す行セット	FetchOrientation 値	スクロール・バー
次の行セット	SQL_FETCH_NEXT (SQLFetch() の呼び出しと同じ)	Page Down
直前の行セット	SQL_FETCH_PRIOR	Page Up
次の行で開始する行セット	SQL_FETCH_RELATIVE (*FetchOffset* を 1 にセット)	Line Down
直前の行で開始する行セット	SQL_FETCH_RELATIVE (*FetchOffset* を -1 にセット)	Line Up
特定行で開始する行セット	SQL_FETCH_ABSOLUTE (*FetchOffset* を、結果セットの開始 (正の値) または終了 (負の値) からの相対位置にセット)	アプリケーションにより生成
直前にブック・マークされた行で開始する行セット	SQL_FETCH_BOOKMARK (*FetchOffset* を、ブックマーク行 (詳細は、 85ページの『スクロール可能カーソルでのブックマークの使用』を参照) の正または負の相対位置にセット)	アプリケーションにより生成

以下の図では、いろいろな *FetchOrientation* 値を使用して SQLFetchScroll() へのいくつかの呼び出しを例示しています。結果セットにはすべての行 (1 ～ n) が含まれ、行セットのサイズは 3 です。呼び出しの順序は図の左側に、*FetchOrientation* 値は右側に表示しています。

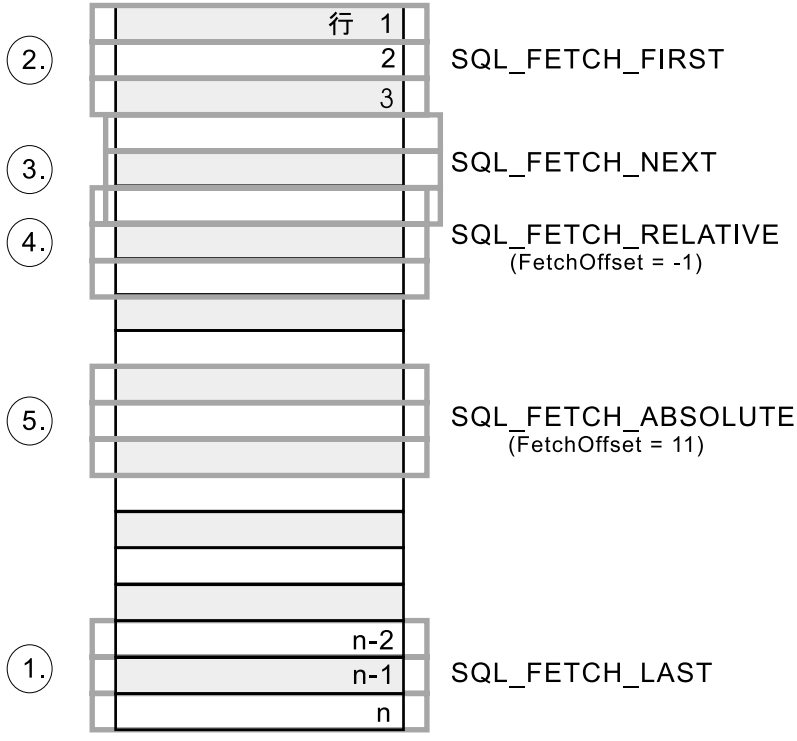


図 7. 行セットの取り出しの例

詳細については、 SQLFetchScroll() の 448ページの『カーソル位置の規則』を参照してください。

戻される行セットのサイズ

ステートメント属性 SQL_ATTR_ROW_ARRAY_SIZE を使用して、行セット内の行数を指定することができます。たとえば、35 行の行セットを宣言するには、以下の呼び出しを使用します。

```
/*...*/
#define ROWSET_SIZE 35
/*...*/
    rc = SQLSetStmtAttr(
                hstmt,
                SQL_ATTR_ROW_ARRAY_SIZE,
                (SQLPOINTER) ROWSET_SIZE,
                0);
```

アプリケーションは、全部の行セットにデータがあると決め付けないでください。それぞれの行セットが作成された後、行セットのサイズを検査する必要があります。それは行セットが行の完全セットを含んでいないという実例がある

からです。たとえば、行セット・サイズが 10 にセットされているケースを考えてください。それから、SQL_FETCH_ABSOLUTE および -3 にセットされた *FetchOffset* を使用して SQLFetchScroll() が呼び出されます。これによって、結果セットの終了行から 3 行を起点として 10 行を返そうとします。しかし、行セットの最初の 3 行だけが、意味のあるデータです。アプリケーションは残りの行を無視する必要があります。

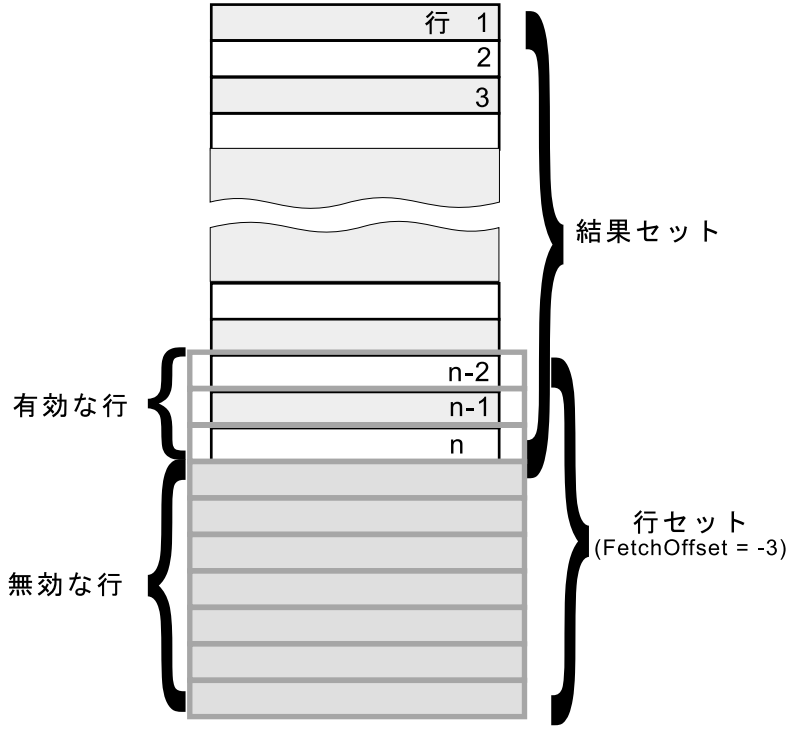


図 8. 一部の行セットの例

ステートメント属性 SQL_ATTR_ROW_ARRAY_SIZE の使用の詳細については、 83ページの『行セット・サイズの設定』を参照してください。

行状況の配列

行状況の配列は、行セットにある各行についての追加情報を提供します。SQLFetchScroll() への各呼び出し後に、配列は更新されます。アプリケーショ

ンは行セットのサイズとして同じ行数を指定して、配列 (SQLUSMALLINT タイプ) を宣言する必要があります (ステートメント属性 SQL_ATTR_ROW_ARRAY_SIZE)。それから、この配列のアドレスはステートメント属性 SQL_ATTR_ROW_STATUS_PTR により指定されます。

```
/* ... */
SQLUSMALLINT    row_statusオROWSET_SIZEコ;
/* ... */
/* Set a pointer to the array to use for the row status */
rc = SQLSetStmtAttr(
            hstmt,
            SQL_ATTR_ROW_STATUS_PTR,
            (SQLPOINTER) row_status,
            0);
/* ... */
```

SQLFetchScroll() への呼び出しで、SQL_SUCCESS または SQL_SUCCESS_WITH_INFO を返さない場合は、行状況のバッファー内容が未定義です。定義されている場合には、次の値が返されます。

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_NOROW	行セットが結果セットの終了行と重なり合いました。行状況配列のこの要素に対応した行が返されていません。
SQL_ROW_ADDED	SQLBulkOperations() によって行が挿入されました。その行がもう一度フェッチされるか、または SQLSetPos() により最新表示されると、状況は SQL_ROW_SUCCESS となります。 SQLFetch() や SQLFetchScroll() にはこの値はセットされません。
SQL_ROW_UPDATED	行は正常にフェッチされ、この結果セットからの最後のフェッチ以降に変更が加えられています。この結果セットからその行がもう一度フェッチされるか、または SQLSetPos() により最新表示されると、状況はその行の新しい状況に変更されます。

SQL_ROW_DELETED この結果セットからの最後のフェッチ以降にその行は削除されています。

81ページの図8 において、行状況配列の最初の 3 行には、SQL_ROW_SUCCESS の値が含まれることになります。そして、残りの 7 行には SQL_ROW_NOROW が含まれます。

典型的なスクロール可能カーソルのアプリケーション

スクロール可能カーソルを使用する各アプリケーションは、次のステップを指示通りの順序に行う必要があります。

1. 環境のセットアップ

以下の追加のステートメント属性は、 DB2 CLI アプリケーションのスクロール可能カーソルを使用する場合に必要とされるものです。詳細な説明については、756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』を参照してください。

行セット・サイズの設定

SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性を、SQLFetchScroll() への各呼び出しから返したい行数に設定します。

省略時値は 1 です。

スクロール可能カーソルのタイプ

DB2 CLI は静的な読み取り専用カーソルか、またはキーセット主導カーソルをサポートしています。 SQLSetStmtAttr() を使用して、SQL_ATTR_CURSOR_TYPE ステートメント属性を SQL_CURSOR_STATIC または SQL_CURSOR_KEYSET_DRIVEN に設定します。 ODBC はその他のスクロール可能カーソルを定義しますが、DB2 CLI では使用できません。

この値を自分で設定するか、または SQL_CURSOR_FORWARD_ONLY の省略時値が使用されます。

返される行数を保管する場所

アプリケーションには、SQLFetchScroll() への各呼び出しから行セットに返された行数を判別する手段が必要です。行セットに返される行数は、SQL_ATTR_ROW_ARRAY_SIZE を使用してセットされた行セットの最大サイズより小さいこともあります。

SQL_ATTR_ROWS_FETCHED_PTR ステートメント属性をポインターとして、 SQLUINTEGER 変数に設定します。そして、この変数には SQLFetchScroll() への各呼び出し後に行セットに返される行数が入ります。

行状況で使用する配列

SQL_ATTR_ROW_STATUS_PTR ステートメント属性をポインターとして、行状況を保管するのに使用する SQLUSMALLINT 配列に設定します。そして、SQLFetchScroll() への各呼び出し後に、この配列は更新されます。

詳細については、81ページの『行状況の配列』を参照してください。

ブックマークを使用しますか

スクロール可能カーソルにブックマークを使用する予定がある場合は、SQL_ATTR_USE_BOOKMARKS ステートメント属性を SQL_UB_VARIABLE に設定してください。

以下の例は、SQLSetStmtAttr() に対する必要な呼び出しを例示しています。

```
/* ... */
    /* Set the number of rows in the rowset */
    rc = SQLSetStmtAttr(
                hstmt,
                SQL_ATTR_ROW_ARRAY_SIZE,
                (SQLPOINTER) ROWSET_SIZE,
                0);
    CHECK_STMT(hstmt, rc);
    /* Set the SQL_ATTR_ROWS_FETCHED_PTR statement attribute to */
    /* point to the variable numrowsfetched: */
    rc = SQLSetStmtAttr(
                hstmt,
                SQL_ATTR_ROWS_FETCHED_PTR,
                &numrowsfetched,
                0);
    CHECK_STMT(hstmt, rc);
    /* Set a pointer to the array to use for the row status */
    rc = SQLSetStmtAttr(
                hstmt,
                SQL_ATTR_ROW_STATUS_PTR,
                (SQLPOINTER) row_status,
                0);
    CHECK_STMT(hstmt, rc);
    /* Set the cursor type */
    rc = SQLSetStmtAttr(
                hstmt,
                SQL_ATTR_CURSOR_TYPE,
                (SQLPOINTER) SQL_CURSOR_STATIC,
                0);
    CHECK_STMT(hstmt, rc);
    /* Indicate that we will use bookmarks by setting the */
    /* SQL_ATTR_USE_BOOKMARKS statement attribute to SQL_UB_VARIABLE: */
    rc = SQLSetStmtAttr(
                hstmt,
                SQL_ATTR_USE_BOOKMARKS,
```

```
                (SQLPOINTER) SQL_UB_VARIABLE,
                0);
    CHECK_STMT(hstmt, rc);
/* ... */
```

2. SQL SELECT ステートメントの実行および結果のバインド

SQL ステートメントの実行および結果セットのバインド用の通常の DB2 CLI プロセスに従ってください。アプリケーションは SQLRowCount() を呼び出し、全体の結果セットの行数を判別することができます。スクロール可能カーソルは、列方向および行方向のバインドの両方をサポートします。 CLI のサンプル・プログラム sfetch.c は、両方の方式を示しています。

3. 結果セットから一度に複数行の行セットを取り出す

この時点で、アプリケーションは以下のステップを使用して、結果セットに入っている情報を読み取ることができます。

1. SQLFetchScroll() を使用して、結果セットからデータの行セットを取り出す。 *FetchOrientation* 引き数を使用すると、結果セットにある行セットの場所を示すことができます。詳細については、78ページの『結果セットから返される行セットの指定』を参照してください。

   データの最初の行セットを検索するための SQLFetchScroll() への典型的な呼び出しは、次のようなものです。

   ```
   SQLFetchScroll(hstmt, SQL_FETCH_FIRST, 0);
   ```

2. 結果セット内に返される行数を計算する。この値は、SQLFetchScroll() への各呼び出し後に自動的にセットされます。上記の例において、ステートメント属性 SQL_ATTR_ROWS_FETCHED_PTR を変数 numrowsfetched にセットしています。これにより、この変数にはそれぞれの SQLFetchScroll() 呼び出しの後に取り出される行数が含まれます。

   SQL_ATTR_ROW_STATUS_PTR ステートメント属性をすでにセットしている場合には、行状況の配列も行セットのそれぞれ可能な行に応じて更新されます。詳細については、81ページの『行状況の配列』を参照してください。

3. 返される行のデータを表示または操作する。

4. 結果セットをクローズするステートメントの解放

一度アプリケーションが情報の検索を終了したなら、ステートメント・ハンドル解放用の通常の DB2 CLI プロセスに従ってください。

スクロール可能カーソルでのブックマークの使用

結果セットにある任意の行へのポインター (ブックマーク) を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。ブックマークの付いた行を起点として (つまり、正または負の相対位置を指定して) 行セットを検索できます。

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。

ブックマークはそれが作成された結果セット内でのみ有効です。 2 つの異なるカーソルで、同じ結果セットから同一行を選択した場合、そのブックマーク値は異なるものとなります。

唯一有効な比較は、同一の結果セットから得られる 2 つのブックマーク値の間のバイト対バイトの比較です。その比較が同じ場合には、その両方は同一行を指します。その他の数値計算またはブックマーク間の比較では、役立つ情報を提供できません。これには、結果セット内のブックマーク値の比較および結果セット間の比較が含まれます。

一般的なブックマークの使用法
ブックマークを使用するには、83ページの『典型的なスクロール可能カーソルのアプリケーション』で説明されているステップに加えて、下記のステップに従う必要があります。

環境のセットアップ: ブックマークを使用するには、SQL_ATTR_USE_BOOKMARKS ステートメント属性を SQL_UB_VARIABLE に設定する必要があります。これは、スクロール可能カーソルに必要なその他のステートメント属性に加えるものです。

ODBC は可変長および固定長のブックマークの両方を定義します。 DB2 CLI は、より新しい、可変長のブックマークだけをサポートします。

行セット内の希望する行からのブックマーク値の入手: アプリケーションは SQL SELECT ステートメントを実行し、 SQLFetchScroll() を使用して、希望する行で行セットを検索する必要があります。そして、SQLSetPos() を使用して、行セット内のカーソルの位置を指定します。最後に、SQLGetData() を用いて列 0 からブックマーク値が得られ、変数に保管されます。

ブックマーク値のステートメント属性のセット: ステートメント属性 SQL_ATTR_FETCH_BOOKMARK_PTR を使用して、ブックマークを用いる SQLFetchScroll() への次の呼び出しの場所を保管します。

SQLGetData() (変数 *abookmark* は下記を参照) を使用して、一度ブックマーク値を得たなら、次のようにして SQLSetStmtAttr() を呼び出します。

```
rc = SQLSetStmtAttr(
           hstmt,
           SQL_ATTR_FETCH_BOOKMARK_PTR,
           (SQLPOINTER) abookmark,
           0);
```

ブックマークに基づく行セットの検索: ブックマーク値が一度保管されたなら、アプリケーションは SQLFetchScroll() を使用して、結果セットからデータの検索を続けることができます。

そして、アプリケーションは結果セット全体を移動できますが、カーソルをクローズする前であればいつでも、ブックマークの付いた行の位置に基づいて行セットを検索できます。

SQLFetchScroll() に対する以下の呼び出しは、ブックマークの付いた行を起点として行セットを検索します。

```
rc = SQLFetchScroll(hstmt, SQL_FETCH_BOOKMARK, 0);
```

0 の値は相対位置を指定するものです。 -3 を指定すると、ブックマークの付いた行の 3 行前の行セットから始まり、 4 を指定すると 4 行後で始まります。

ブックマーク値を保管するのに使用する変数が、 SQLFetchScroll() 呼び出しでは指定されない点に注意してください。その変数は、ステートメント属性 SQL_ATTR_FETCH_BOOKMARK_PTR を使用して、 前のステップでセットされています。

長形式データの分割送信 / 取り出し

長形式データを扱う場合、ステートメントを実行する時、またはデータをデータベースから取り出す時に、アプリケーションがデータ全体を記憶域にロードするのは合理的ではないことがあります。そこでアプリケーションがデータを小さく分けて扱えるような方法が備えられています。長データを分けて送信する手法は、*実行時パラメーター値の指定* と呼ばれます。これは、整数などの固定サイズの非文字データ・タイプの値を指定する場合にも使用できるからです。

また、アプリケーションは SQLGetSubString() 関数を使用して、ラージ・オブジェクト値の一部分を検索することができます。詳細については、123ページの『ラージ・オブジェクトの使用』の 127ページの図16 を参照してください。

実行時パラメーター値の指定

SQLExecute() または SQLExecDirect() を呼び出す前に値がメモリーに保管されるのではなく、実行時に値がプロンプト指示されるバインド済みパラメーターのことを実行時データ・パラメーターと呼びます。 SQLBindParameter() 呼び出しでそのようなパラメーターを指定するには、アプリケーションは次のようにします。

- 実行時に値 SQL_DATA_AT_EXEC が入れられる変数を指す入力データ長ポインターを設定します。
- 複数の実行時データ・パラメーターがある場合は、個々の入力データ・ポインター引き数を、対象フィールドを固有に識別しているとアプリケーションが認識する値に設定します。

アプリケーションが SQLExecDirect() または SQLExecute() を呼び出したときに実行時パラメーターがあれば、呼び出しは SQL_NEED_DATA とともに返され、これらのパラメーターにアプリケーションが値を入れるよう入力を要求します。アプリケーションは次のように応答します。

1. SQLParamData() を呼び出して、最初の実行時データ・パラメーターへ概念的に進みます。 SQLParamData() は SQL_NEED_DATA を返し、関連した SQLBindParameter() 呼び出しで指定されている入力データ・ポインター引き数の内容を示して、必要な情報を識別するのを助けます。
2. SQLPutData() を呼び出して、パラメーターの実際のデータを渡します。 SQLPutData() を繰り返し呼び出すと、長いデータを小さく分けて送信することができます。
3. この実行時データ・パラメーターに関するデータ全体を渡した後で、再度 SQLParamData() を呼び出します。他に実行時データ・パラメーターがある場合は、 SQLParamData() は再度 SQL_NEED_DATA を返し、アプリケーションは上記のステップ 2 および 3 を繰り返します。

すべての実行時データ・パラメーターに値が割り当てられると、 SQLParamData() は SQL ステートメントの実行を完了し、元々 SQLExecDirect() または SQLExecute() が作成するはずであった戻り値および診断を作成します。 90ページの図9 の右側にこの流れを図示してあります。

実行時データ・フローが進んでいる間は、アプリケーションは次の DB2 CLI 関数だけ呼び出せます。

- 上記の SQLParamData() および SQLPutData()
- SQLCancel() 関数。これはこの流れを取り消すために使用するもので、SQL ステートメントを実行せずに、 90ページの図9 の右側にあるループを強制終了します。

- SQLGetDiagRec() 関数。アプリケーションはトランザクションの終了と接続属性の設定のどちらも行えません。

実行時パラメーターの手法を使用してラージ・オブジェクト・データを入力する際に、クライアントで一時ファイルを作成して使用することが必要となる場合があります。長形式データを入力する他の方式については、123ページの『ラージ・オブジェクトの使用』を参照してください。

データの分割取り出し

一般にアプリケーションは、結果セットの列に関する知識 (SQLDescribeCol() によって得た知識か、または以前の知識) に基づいて、列の値が使う可能性のある最大メモリーを割り振るよう選択し、その列を SQLBindCol() によってバインドします。しかし、文字および 2 進データの場合、列の長さが不定であることがあります。列値の長さが、アプリケーションが割り振る (または割り振れる) バッファーの長さを超えている場合に、 SQLGetData() の機能を使用すると、アプリケーションが繰り返して呼び出しを行い、 1 つの列の値を管理しやすい大きさに分けて連続して得ることができます。

基本的には、90ページの図9 の左側に示したように、 SQLGetData() を呼び出すと SQL_SUCCESS_WITH_INFO (および SQLSTATE 01004) が返され、この列に関するデータがさらに存在することを示します。 SQLGetData() が繰り返して呼び出され、 SQL_SUCCESS が返されるまでデータの残りの部分を獲得します。 SQL_SUCCESS は、この列に関するデータ全体が取り出されたことを知らせるものです。

また、関数 SQLGetSubString() を使用して、ラージ・オブジェクト値の特定部分を検索することができます。詳細については、598ページの『SQLGetSubString - ストリング値の部分を取り出す』を参照してください。
長形式データを取り出す他の方式については、123ページの『ラージ・オブジェクトの使用』を参照してください。

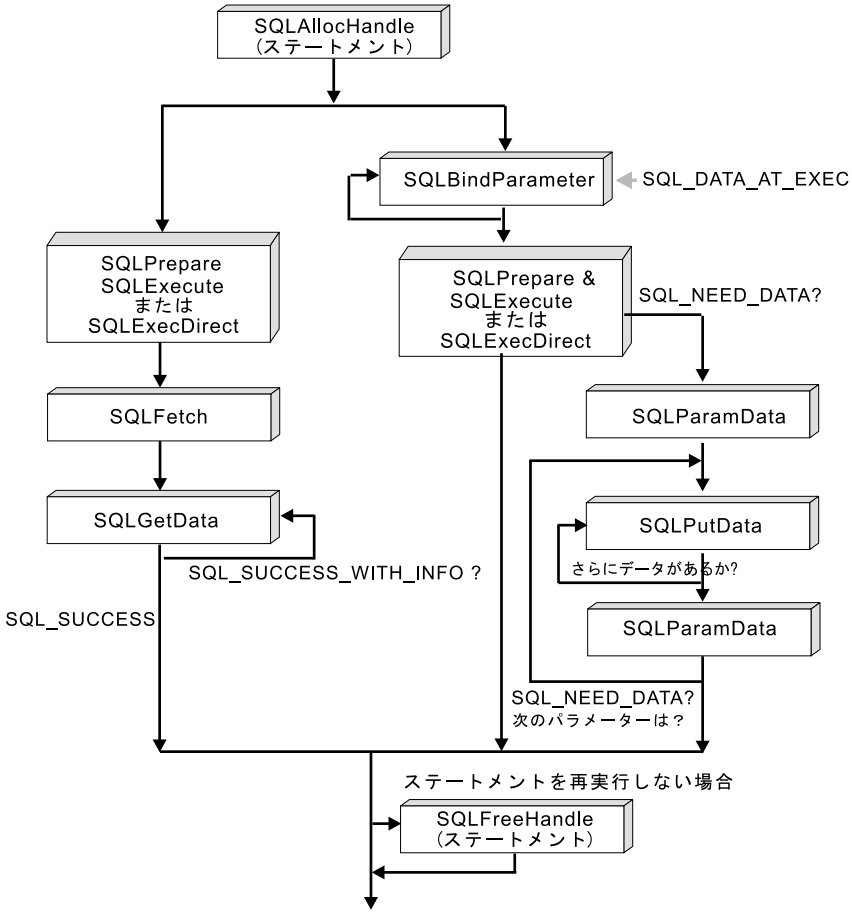


図 9. 分割入力および取り出し

分割入力および取り出しの例

イメージ blob の分割入力の例については、 659ページの『例』の `picin2.c` を参照してください。

イメージ blob の分割取り出しの例については、 496ページの『例』の `showpic2.c` を参照してください。

配列の使用によるパラメーター値の入力

データ入力項目および更新アプリケーション (特に図形アプリケーション) によっては、ユーザーがデータ入力書式のセルを多数挿入、削除、または変更してから、データベースへデータを送信するよう依頼するような状況が頻繁に生じます。このような大量の挿入、削除、または更新の状況に備えて、 DB2 CLI では配列入力方式が設けられており、アプリケーションが同一 INSERT、DELETE、または UPDATE ステートメントで繰り返し SQLExecute() を呼び出さなくても済むようになっています。加えて、ネットワーク・フローで有効な節約ができます。

アプリケーションが SQL ステートメントのパラメーター・マーカーを配列にバインドできる以下の 2 つの方法があります。

- *列方向配列の挿入* (列方向のバインドを使用する): 別の配列が各パラメーターに対してバインドされます。
- *行方向配列の挿入* (行方向のバインドを使用する): 1 つの構造体を作成して、ステートメントのパラメーターの完全セットを保管します。これらの構造体の配列を作成して、パラメーターにバインドします。パラメーターのバインド相対位置 (次の項で説明) は、行方向のバインドでのみ使用可能です。

SQLBindParameter() も、パラメーターに対するバッファーをバインドするために使用します。唯一の違いは、渡されるアドレスが配列アドレスであり単一変数のアドレスではないという点です。また、アプリケーションは、列方向または行方向のどちらのバインドを使用するかを指定するため、SQL_ATTR_PARAM_BIND_TYPE ステートメント属性をセットする必要があります。

列方向配列の挿入

この方式では、SQLBindParameter() 呼び出しによってパラメーター・マーカーを記憶場所の配列にバインドします。文字および 2 進入力データの場合は、アプリケーションが SQLBindParameter() 呼び出しで最大入力バッファー・サイズの引き数 (*BufferLength*) を使用して、 DB2 CLI に入力配列内の値の場所を示します。その他の入力データ・タイプの場合は、配列内の各要素の長さは C データ・タイプのサイズであるとみなされます。 SQL ステートメントの実行前に、ステートメント属性 SQL_ATTR_PARAMSET_SIZE に (SQLSetStmtAttr() への呼び出しを指定して) 配列のサイズをセットする必要があります。

93ページの図10 で、出退勤時間データ入力用紙の OVERTIME_WORKED 列と OVERTIME_PAID 列の値をユーザーが変更できるようなアプリケーションがあるとします。また、基礎 EMPLOYEE 表の基本キーは EMPLOY_ID であるとします。この場合、アプリケーションは次のような SQL ステートメントを準備するよう要求することができます。

```
UPDATE EMPLOYEE SET OVERTIME_WORKED= ? and OVERTIME_PAID= ?
WHERE EMPLOY_ID=?
```

ユーザーがすべての変更内容を入力すると、アプリケーションは変更される n 行をカウントし、変更されるデータと基本キーを保管するための m=3 配列を割り振ります。次に SQLBindParameter() を呼び出して、 3 つのパラメーター・マーカーをメモリー内の 3 つの配列の場所にバインドします。次に、変更する行数 (配列のサイズ) を指定するには、 (SQLSetStmtAttr() への呼び出しを指定して) ステートメント属性 SQL_ATTR_PARAMSET_SIZE をセットします。次に SQLExecute() を 1 回呼び出すと、すべての更新内容がデータベースへ送信されます。これが 93ページの図10 の右側に示した流れです。

基本方式については、93ページの図10 の左側に示してあります。この場合、SQLBindParameter() が呼び出されて、 3 つのパラメーター・マーカーをメモリー内の 3 つの変数の場所にバインドします。 SQLExecute() が呼び出されて、最初の変更の集まりをデータベースに送信します。変更箇所の次の行の値を反映するよう変数が更新され、再度 SQLExecute() が呼び出されます。この方式では、n-1 回余分に SQLExecute() 呼び出しがあることに注意してください。

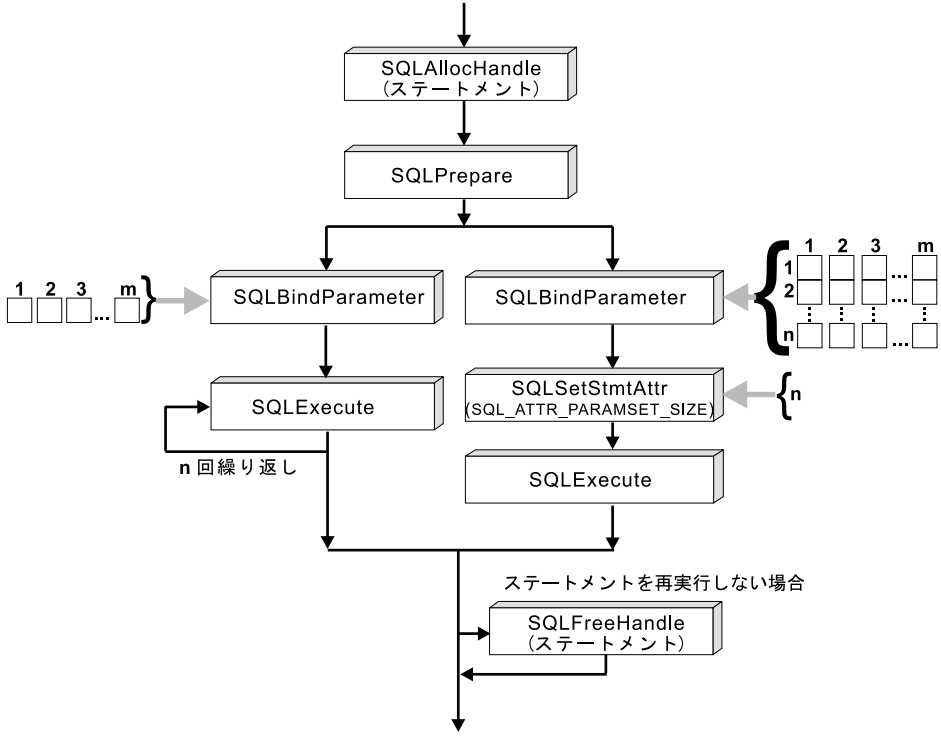


図 10. 列方向配列の挿入

アプリケーションでアクセスできるエラーに関する詳細については、 95ページの『診断情報の取り出し』を参照してください。

行方向配列の挿入

行方向配列の挿入を使用する場合の最初のステップは、パラメーターごとに 2 つの要素を含む構造体を作成することです。パラメーターごとの最初の要素では、長さ / 標識バッファーを保持し、2 番目の要素はその値を保持します。この構造体が一度定義されたなら、アプリケーションは構造体の配列を割り当てる必要があります。配列の行数は、パラメーターごとに使用される値の数に対応しています。

```
struct { SQLINTEGER La; SQLINTEGER A;  /* Information for parameter A */
         SQLINTEGER Lb; SQLCHAR B[4];  /* Information for parameter B */
         SQLINTEGER Lc; SQLCHAR C[11]; /* Information for parameter C */
       } R[n];
```

95ページの図11 には、3 つのパラメーターでなる構造体 R を示しています。それは *n* 行の配列にあります。その後、その配列には適切なデータを移植できます。

配列が一度作成され移植されたなら、アプリケーションは行方向のバインドを使用することを指示する必要があります。これは、ステートメント属性 SQL_ATTR_PARAM_BIND_TYPE を、作成された構造体の長さに設定することで行われます。また、ステートメント属性 SQL_ATTR_PARAMSET_SIZE を、配列の行数に設定する必要があります。

この段階で、各パラメーターは SQLBindParameter() を使用して、 (配列の最初の行における) 構造体の適切な 2 つの要素にバインドすることが可能になります。

```
/* Parameter A */
rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER,
         5, 0, &R[0].A, 0, &R.La);
/* Parameter B */
rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
         10, 0, R[0].B, 10, &R.Lb);
/* Parameter C */
rc = SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
         3, 0, R[0].C, 3, &R.Lc);
```

ここで、アプリケーションが SQLExecute() を一度呼び出すと、すべての更新内容がデータベースへ送信されます。

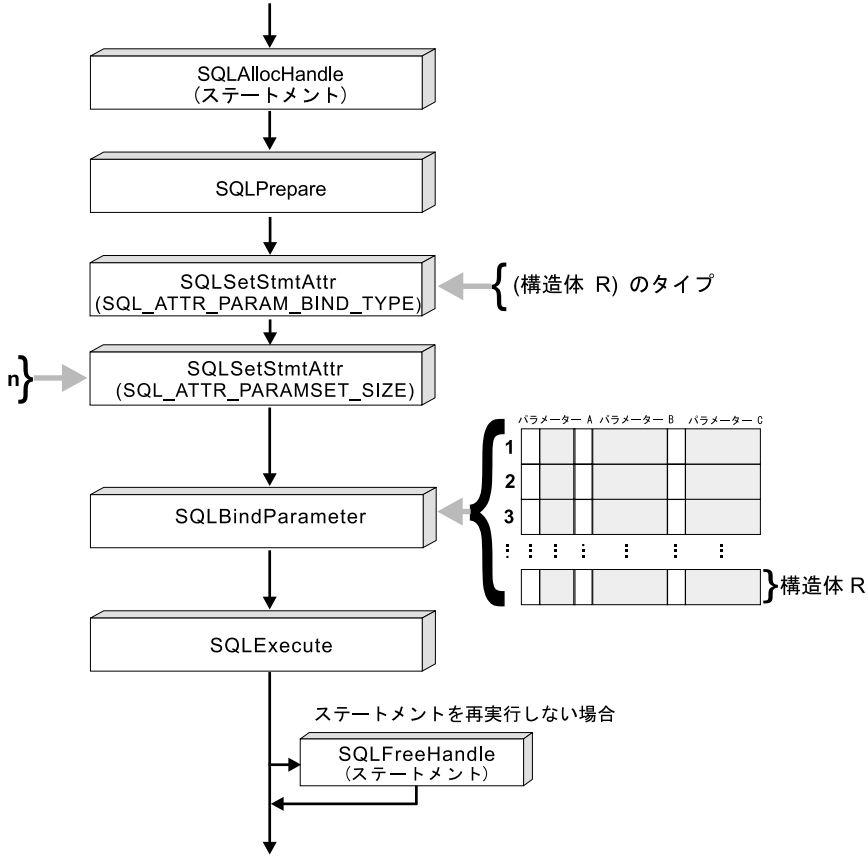


図 11. 行方向配列の挿入

アプリケーションでアクセスできるエラーに関する詳細については、『診断情報の取り出し』を参照してください。

診断情報の取り出し

パラメーター状況の配列 は、 SQLExecute() または SQLExecDirect() 呼び出し後に移植できます。この配列には、パラメーターの各セットの処理に関する情報が収められています。詳細については、ステートメント属性 SQL_ATTR_PARAM_STATUS_PTR または対応する IPD 記述子のヘッダー・フィールド SQL_DESC_ARRAY_STATUS_PTR を参照してください。

ステートメント属性 SQL_ATTR_PARAMS_PROCESSED (または対応する IPD 記述子のヘッダー・フィールド SQL_DESC_ROWS_PROCESSED_PTR) を使用

すると、すでに処理されたパラメーターのセットの数を返すことができます。SQLSetStmtAttr() または SQLSetDescField() の説明の中のこれらの属性の項を参照してください。

アプリケーションがどのパラメーターにエラーがあるかを一度判別したなら、ステートメント属性 SQL_ATTR_PARAM_OPERATION_PTR (または対応する APD 記述子のヘッダー・フィールド SQL_DESC_ARRAY_STATUS_PTR、どちらも値の配列を指す) を使用すると、 SQLExecute() または SQLExecDirect() への 2 番目の呼び出しにおいて、パラメーターのどのセットを無効にするかを制御することができます。 SQLSetStmtAttr() または SQLSetDescField() の説明の中のこれらの属性の項を参照してください。

その他の情報

基礎サポートで複合 SQL が使える環境 (DB2 ユニバーサル・データベース、または DB2 コネクト V 2.3 以降の DRDA 環境) では、ネットワーク・フローをさらに節約します。配列内のすべてのデータは実行要求と一緒に 1 つの流れとしてパッケージされます。 DRDA 環境の場合、基礎の複合 SQL サポートは常に NOT ATOMIC COMPOUND SQL です。このことは、中間の配列要素のいずれかでエラーが検出されても、実行は続行することを意味します。配列操作の後で SQLRowCount() が呼び出される場合、受け取る行カウントは、入力パラメーター値の配列内のすべての要素によって制御される行の集合の数です。

DB2 ユニバーサル・データベースに接続した場合、アプリケーションは ATOMIC または NOT ATOMIC COMPOUND SQL を選択することができます。 ATOMIC SQL (省略時値) にすると、配列のすべての要素は正常に処理されるか、またはまったく処理されません。アプリケーションは、SQLSetStmtAttr() を指定して SQL_ATTR_PARAMOPT_ATOMIC 属性を設定することで COMPOUND SQL タイプを使用するよう選択できます。

注: SQLBindParam() を使用して、配列の記憶場所をパラメーター・マーカーにバインドすることはできません。文字または 2 進入力データの場合、入力配列内の各要素のサイズを指定する方法はありません。

WHERE 文節にパラメーター・マーカーを指定した照会の場合、入力値の配列によっては複数の順次結果セットが生成される可能性があります。SQLMoreResults() を呼び出すと、個々の結果セットを処理し、次の集まりに進むことができます。詳細および例については、610ページの『SQLMoreResults - さらに結果セットがあるかどうかを判別する』を参照してください。

パラメーター・バインドの相対位置

パラメーター・バインドの変更の必要が生じた場合、アプリケーションはもう一度 SQLBindParameter() を呼び出すことができます。これにより、バインドされているパラメーターのバッファー・アドレスと、それに対応する使用中の長さ / 標識バッファー・アドレスを変更します。この変更は、行方向配列の挿入でのみ使用できますが、アプリケーションが個々にまたは配列を使用してパラメーターをバインドするかどうかを決めます。

SQLBindParameter() への複数の呼び出しの代わりに、 DB2 CLI はパラメーター・バインドの相対位置もサポートしています。毎回再バインドするよりも、相対位置を使用すると、 SQLExecute() または SQLExecDirect() への次の呼び出しで使用される新しいバッファー・アドレスおよび長さ / 標識アドレスを指定することができます。

パラメーター・バインドの相対位置を使用するには、アプリケーションは次のステップに従ってください。

1. 通常どおり、SQLBindParameter() を呼び出します。バインドされるパラメーターのバッファー・アドレスと、それに対応する長さ / 標識のバッファー・アドレスの最初のセットは、テンプレートとしての働きをします。そして、アプリケーションは相対位置を使用して、このテンプレートをいろいろな記憶場所に移動します。
2. 通常どおり、 SQLExecute() または SQLExecDirect() を呼び出します。バインドされるアドレス内に保管されている値が使用されます。
3. メモリー相対位置の値を保持する変数を設定します。

   ステートメント属性 SQL_ATTR_PARAM_BIND_OFFSET_PTR は、相対位置が保管されることになる SQLINTEGER バッファーのアドレスを指します。このアドレスは、カーソルがクローズするまで有効である必要があります。

   この、余分のレベルの間接参照によって、単一のメモリー変数を使用するだけで、異なるステートメント・ハンドルにあるパラメーター・バッファーの複数のセットについて、相対位置を保管することができます。アプリケーションは、この 1 つのメモリー変数と、変更されるすべての相対位置だけを設定する必要があります。
4. 相対位置の値 (バイト数) を、前のステップのステートメント属性セットが指し示すメモリー位置に保管します。

   相対位置の値は、常に最初にバインドされている値のメモリー位置に加えられ、この合計が有効なメモリー・アドレスを指すことになります。

5. 再び、 SQLExecute() または SQLExecDirect() を呼び出します。 CLI は上記で指定される相対位置を SQLBindParam() への元の呼び出しで使用される場所に追加して、使用するパラメーターがどのメモリーにあるかを判別します。
6. 必要に応じて上記のステップ 4 および 5 を繰り返します。

詳細については、SQLBindParam() の 289ページの『パラメーター・バインドの相対位置』を参照してください。

配列の入力例

この例では配列の INSERT ステートメントを示します。配列の照会ステートメントの例については、610ページの『SQLMoreResults - さらに結果セットがあるかどうかを判別する』を参照してください。

```
/* ... */
    SQLCHAR * stmt =
    "INSERT INTO CUSTOMER ( Cust_Num, First_Name, Last_Name ) "
    "VALUES (?, ?, ?)" ;
    SQLINTEGER Cust_Num[] = {
        10,  20,  30,  40,  50,  60,  70,  80,  90, 100,
       110, 120, 130, 140, 150, 160, 170, 180, 190, 200,
       210, 220, 230, 240, 250,
    } ;
    SQLCHAR First_Name[][31] = {
       "EVA",     "EILEEN",    "THEODORE", "VINCENZO",  "SEAN",
       "DOLORES", "HEATHER",   "BRUCE",    "ELIZABETH", "MASATOSHI",
       "MARILYN", "JAMES",     "DAVID",    "WILLIAM",   "JENNIFER",
       "JAMES",   "SALVATORE", "DANIEL",   "SYBIL",     "MARIA",
       "ETHEL",   "JOHN",      "PHILIP",   "MAUDE",     "BILL",
    } ;
    SQLCHAR Last_Name[][31] = {
       "SPENSER", "LUCCHESI", "O'CONNELL", "QUINTANA",  "NICHOLLS",
       "ADAMSON", "PIANKA",   "YOSHIMURA", "SCOUTTEN",  "WALKER",
       "BROWN",   "JONES",    "LUTZ",      "JEFFERSON", "MARINO",
       "SMITH",   "JOHNSON",  "PEREZ",     "SCHNEIDER", "PARKER",
       "SMITH",   "SETRIGHT", "MEHTA",     "LEE",       "GOUNOT",
    } ;
/* ... */
    /* Prepare the statement */
    rc = SQLPrepare( hstmt, stmt, SQL_NTS ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    rc = SQLSetStmtAttr( hstmt,
                         SQL_ATTR_PARAMSET_SIZE,
                         ( SQLPOINTER ) row_array_size,
                         0
                       ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    rc = SQLBindParameter( hstmt,
                           1,
                           SQL_PARAM_INPUT,
                           SQL_C_SLONG,
```

```
                        SQL_INTEGER,
                        0,
                        0,
                        Cust_Num,
                        0,
                        NULL
                      ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
rc = SQLBindParameter( hstmt,
                        2,
                        SQL_PARAM_INPUT,
                        SQL_C_CHAR,
                        SQL_CHAR,
                        31,
                        0,
                        First_Name,
                        31,
                        NULL
                      ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
rc = SQLBindParameter( hstmt,
                        3,
                        SQL_PARAM_INPUT,
                        SQL_C_CHAR,
                        SQL_CHAR,
                        31,
                        0,
                        Last_Name,
                        31,
                        NULL
                      ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
rc = SQLExecute( hstmt ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
printf( "Inserted %ld Rows¥n", row_array_size ) ;
```

配列への結果セットの取り出し

アプリケーションが行う最も一般的なタスクの一つに、照会ステートメントを発行してから、SQLBindCol() を使ってバインドされたアプリケーション変数中に結果セットの各行を取り出すことがあります。アプリケーションが結果セットの各列または各行を配列内に保管するよう要求する場合は、個々の取り出しの後に続いてデータのコピー操作を行うか新たに一連の SQLBindCol() 呼び出しを行って、次の取り出しのために新しい記憶域を割り当てなくてはなりません。

もう一つの方法として、アプリケーションがデータの複数行を (行の集まりを呼び出して) 一度に配列内へ取り出して、余分なデータ・コピーまたは余分な SQLBindCol() 呼び出しのオーバーヘッドを取り除くことができます。

注: オーバーヘッドを少なくする 3 番目の方法は、単独でも配列でも使用できますが、バインドの相対位置を指定することです。毎回再バインドするよりも、相対位置を使用すると、 SQLFetch() または SQLFetchScroll() への次回の呼び出しで使用される新しいバッファー・アドレスおよび長さ / 標識アドレスを指定することができます。これは行相対位置のバインドでのみ使用可能であり、103ページの『列バインドの相対位置』で説明しています。

結果セットを配列に取り出す場合、SQLBindCol() を使用して、アプリケーションの配列変数用の記憶域を割り当てることも行います。省略時設定では、行のバインドは列方向です。これは前のセクションに記述されている、SQLBindParameter() を使用して入力パラメーター値の配列をバインドする場合とは対称的です。

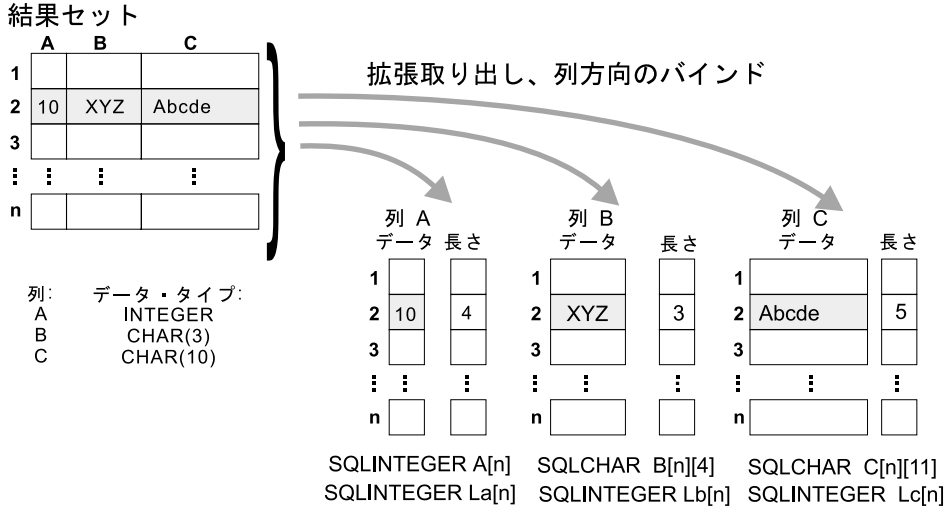


図 12. 列方向バインド

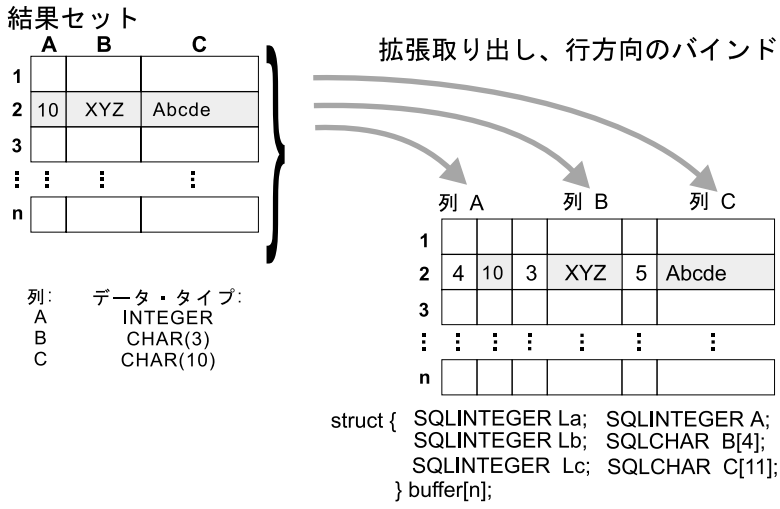


図 13. 行方向バインド

SQLFetchScroll() は、スクロール可能カーソル、つまり結果セット内の任意の位置から上方および下方に移動する機能をサポートします。これは、列方向および行方向のバインドの両方で使用できます。詳細については、75ページの『スクロール可能カーソル』を参照してください。

列方向バインドによる配列データの取り出し

100ページの図12 は、列方向バインドの論理視点です。 103ページの図14 の右側に、列方向取り出しの関数の流れを示してあります。

列方向の配列取り出しを指定するには、アプリケーションで SQL_ATTR_ROW_ARRAY_SIZE 属性を指定して SQLSetStmtAttr() を呼び出し、一度に取り出す行数を示します。 SQL_ATTR_ROW_ARRAY_SIZE 属性の値が 1 よりも大きいと、DB2 CLI は、処理を行う据え置き出力データ・ポインターと長さポインターは、結果セットの列のデータと長さのある 1 つの要素を指すものではなく、データと長さの配列を指すものであることを認識します。

次に、アプリケーションは SQLFetchScroll() を呼び出してデータを取り出します。データを返すとき、 DB2 CLI は SQLBindCol() の最大バッファー・サイズ引き数 (*BufferLength*) を使用し、データの連続行を配列内のどこに保管するのかを判別します。各要素を返すのに使用できるバイト数は、据え置き長さ配列に保管されています。結果セットの行数が

SQL_ATTR_ROW_ARRAY_SIZE 属性値よりも大きい場合に、すべての行を取り出すには、複数回 SQLFetchScroll() を呼び出す必要があります。

行方向バインド・データへの配列データの戻り

アプリケーションは行方向バインド、つまり結果セットの 1 行全体を 1 つの構造に関連付けることも行えます。この場合、行の集まりは構造の配列中に取り出されます。個々の構造には 1 つの行のデータおよび関連付けられた長さフィールドがあります。 101ページの図13 は、行方向バインドを図示しています。

行方向の配列取り出しを行うには、アプリケーションで SQL_ATTR_ROW_ARRAY_SIZE 属性を指定して SQLSetStmtAttr() を呼び出し、一度に取り出す行数を示す必要があります。さらに SQL_ATTR_ROW_BIND_TYPE 属性値を、結果列がバインドされる構造のサイズに設定して、 SQLSetStmtAttr() を呼び出さなくてはなりません。 DB2 CLI は、SQLBindCol() の据え置き出力データ・ポインターを、このような構造の配列の先頭要素の列に関するデータ・フィールド・アドレスとして扱います。据え置き出力長さポインターは、列の関連長さフィールドのアドレスとして扱います。

次に、アプリケーションは SQLFetchScroll() を呼び出してデータを取り出します。データを返すときに、 DB2 CLI は SQL_ATTR_ROW_BIND_TYPE 属性によって設定されている構造サイズを使用して、構造の配列のどこに連続行を保管するかを判別します。

103ページの図14 は、それぞれの方式に必要な関数を示しています。左側は、一度に 1 行が選択され、取り出された行が *n* 行あり、それらが m 個のアプリケーション変数に入れられていることを示しています。右側は、同じ n 行が選択され、取り出されて配列に直接入れられていることを示しています。

- 図には *m* 列のバインドが示されています。つまり、*m* 回の SQLBindCol() 呼び出しがどちらの場合にも必要です。
- *n* 個よりも少ない要素の配列が割り振られている場合は、複数の SQLFetchScroll() 呼び出しが必要になります。

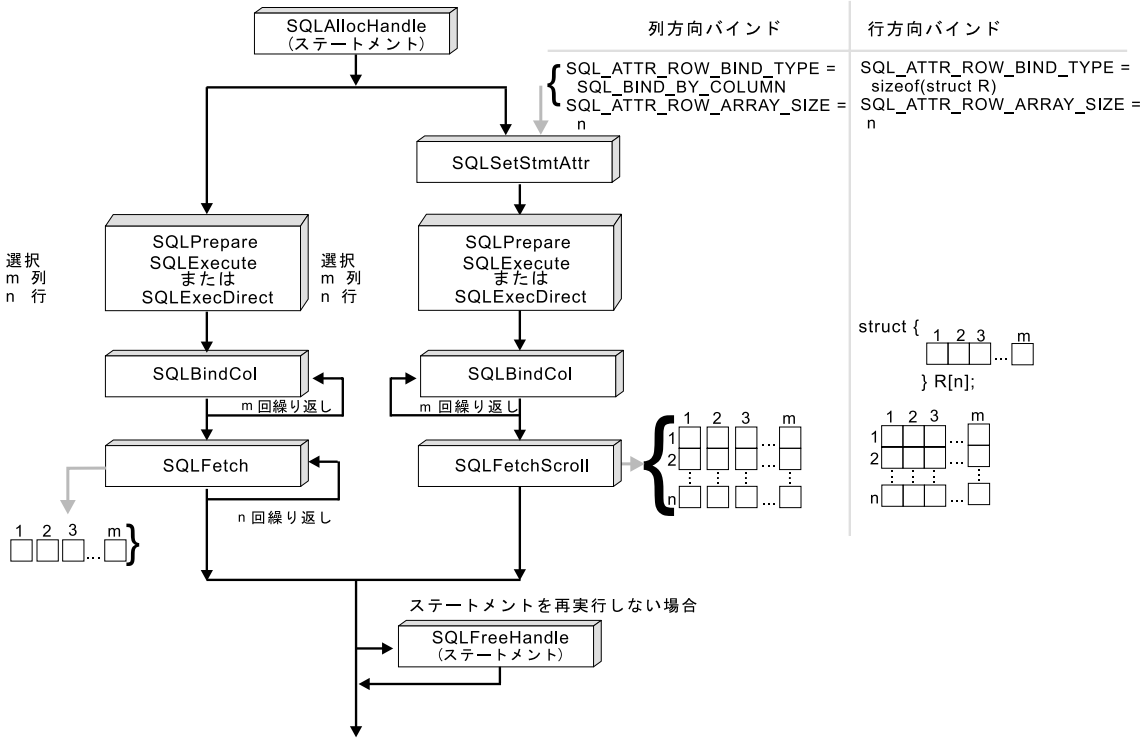


図 14. 配列の取り出し

列バインドの相対位置

バインドの変更が生じた場合 (たとえば、次の取り出しのために)、アプリケーションはもう一度 SQLBindCol() を呼び出すことができます。これによって、バッファー・アドレスおよび使用中の長さ / 標識ポインターが変更されます。

SQLBindCol() への複数の呼び出しの代わりに、 DB2 CLI はパラメーター・バインドの相対位置もサポートしています。毎回再バインドするよりも、相対位置を使用すると、 SQLFetch() または SQLFetchScroll() への次回の呼び出しで使用される新しいバッファー・アドレスおよび長さ / 標識アドレスを指定することができます。これは、行方向バインドでのみ使用できますが、アプリケーションが一度に単一行を取り出すか、または複数行を取り出すかを決めます。

列バインドの相対位置を使用するには、アプリケーションは次のステップに従ってください。

1. 通常どおり、SQLBindCol() を呼び出します。バインドされるデータ・バッファーと、長さ / 標識バッファーのアドレスの最初のセットは、テンプレートとしての働きをします。そして、アプリケーションは相対位置を使用して、このテンプレートをいろいろな記憶場所に移動します。
2. 通常どおり、 SQLFetch() または SQLFetchScroll() を呼び出します。返されるデータは、上記にバインドされる場所に保管されます。
3. メモリー相対位置の値を保持する変数を設定します。

   ステートメント属性 SQL_ATTR_ROW_BIND_OFFSET_PTR は、相対位置が保管されることになる SQLINTEGER バッファーのアドレスを指します。このアドレスは、カーソルがクローズするまで有効である必要があります。

   この、余分のレベルの間接参照によって、単一のメモリー変数を使用するだけで、異なるステートメント・ハンドルにあるバインドの複数のセットについて、相対位置を保管することができます。アプリケーションは、この 1 つのメモリー変数と、変更されるすべての相対位置だけを設定する必要があります。
4. 相対位置の値 (バイト数) を、前のステップのステートメント属性セットが指し示すメモリー位置に保管します。

   相対位置の値は、常に最初にバインドされている値のメモリー位置に加えられ、この合計が有効なメモリー・アドレスを指すことになります。
5. 再び、 SQLFetch() または SQLFetchScroll() を呼び出します。 CLI は上記で指定される相対位置を SQLBindCol() への元の呼び出しで使用される場所に追加して、結果を保管するのがどのメモリーかを判別します。
6. 必要に応じて上記のステップ 4 および 5 を繰り返します。

詳細については、SQLBindCol() の 260ページの『列バインドの相対位置』を参照してください。

列方向、行方向のバインド例

```
/* From the CLI sample PCALL.C */
/* ... */
    /* print result sets, if any */
    do
    {   rc = StmtResultPrint( hstmt1);
    }
    while( SQLMoreResults( hstmt1) == SQL_SUCCESS);
```

記述子の使用

DB2 CLI は、結果セット内の列に関する情報 (データ・タイプ、サイズ、ポインターなど)、および SQL ステートメントのパラメーターを保管します。また、列およびパラメーターに対するアプリケーション・バッファーのバインドも、保管する必要があります。*記述子* は上記情報の論理視点であり、この情報を照会および更新するための 1 つの手段をアプリケーションに提供します。

多くの CLI 関数は記述子を利用しますが、アプリケーション自身は直接、記述子を操作する必要はありません。

たとえば、次のようにします。

- アプリケーションが `SQLBindCol()` を使用して列のデータをバインドする場合、バインドをすべて完全に記述している記述子フィールドを設定します。
- いくつか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。この場合、記述子に直接値をセットする関数 `SQLSetDescField()` を呼び出すことと、 `SQLSetStmtAttr()` を呼び出すこととでは、同じ効果をもたらすことができます。

データベース操作は記述子に対する直接アクセスを必要としてはいませんが、記述子による直接作業が、より効率的であったり、結果としてより簡単なコードとなる場合があります。たとえば、ある表から取り出される行を記述する記述子を使用すると、その後その表の中に挿入される行を記述することが可能になります。

記述子タイプ

記述子には次の 4 つのタイプがあります。

アプリケーション・パラメーター記述子 (APD)
: アプリケーション・バッファー (ポインター、データ・タイプ、位取り、精度、長さ、最大バッファー長など) を記述します。これらのバッファーは、 SQL ステートメントのパラメーターにバインドされています。パラメーターが CALL ステートメントの一部の場合には、それは入力、出力、またはその両方の可能性があります。この情報は、アプリケーションの C データ・タイプを使用して記述されます。

アプリケーション行記述子 (ARD)
: 列にバインドするアプリケーション・バッファーを記述します。アプリケーションは、実装行記述子にあるバッファーからいろいろなデータ・

タイプを指定して、列データのデータ変換を行うことができます。この記述子は、アプリケーションが指定する任意のデータ変換を反映します。

実装パラメーター記述子 (IPD)

SQL ステートメントにあるパラメーターを記述します (SQL タイプ、サイズ、精度など)。

- パラメーターが入力として使用される場合、この記述子は DB2 CLI が何らかの必要な変換を行った後に、データベース・サーバーが受け取る SQL データを記述します。
- パラメーターが出力として使用される場合には、この記述子で、DB2 CLI がアプリケーションの C データ・タイプへの必要な変換を行う前の SQL データを記述します。

実装行記述子 (IRD)

DB2 CLI がアプリケーションの C データ・タイプへの必要な変換を行う前の、結果セットからのデータの行を記述します。

上記に示す 4 つのタイプの記述子の唯一の違いは、それらがどのように使用されるかにあります。記述子の利点の 1 つは、単一の記述子が多くの目的に使用できることです。たとえば、あるステートメントにある行記述子は別のステートメントでパラメーター記述子として使用することができます。

記述子が存在するようになるとすぐに、それはアプリケーション記述子かまたは実装記述子かのどちらかになります。記述子がまだデータベース操作で使用されていない場合でも、こうしたケースがあります。記述子が `SQLAllocHandle()` を使用して、アプリケーションで割り当てられる場合、それはアプリケーション記述子となります。

記述子に保管される値

それぞれの記述子には、ヘッダー・フィールドとレコード・フィールドの両方があります。これらのフィールドが一緒になって、列またはパラメーターを完全に記述します。

ヘッダー・フィールド

それぞれのヘッダー・フィールドは各記述子の中で 1 回だけ出現します。このフィールドの 1 つを変更すると、すべての列またはパラメーターに影響します。

以下のヘッダー・フィールドの大部分は、ステートメント属性に対応しています。 `SQLSetDescField()` を使用して記述子のヘッダー・フィールドを設定することは、 `SQLSetStmtAttr()` を使用して対応するステートメント属性を設定す

るのと同じです。同じことが、SQLGetDescField() または SQLGetStmtAttr() を使用して情報を取り出す場合にもそのまま適用されます。アプリケーションに記述子ハンドルがすでに割り当てられているのでなければ、記述子ハンドルを割り当てて記述子呼び出しを使用するよりも、ステートメント属性呼び出しを使用する方が一層能率的です。

表 7. ヘッダー・フィールド

SQL_DESC_ALLOC_TYPE	SQL_DESC_BIND_TYPE[a]
SQL_DESC_ARRAY_SIZE[a]	SQL_DESC_COUNT
SQL_DESC_ARRAY_STATUS_PTR[a]	SQL_DESC_ROWS_PROCESSED_PTR[a]
SQL_DESC_BIND_OFFSET_PTR[a]	

注:

[a] ステートメント属性に対応するヘッダー・フィールドです。

上記フィールドのそれぞれの詳細については、 SQLSetDescField() の 707ページの『ヘッダー・フィールド』を参照してください。

記述子のヘッダー・フィールド SQL_DESC_COUNT は、情報が入っている最大番号の記述子レコードの、1 を基数とする指標です。 DB2 CLI は、列またはパラメーターがバインドおよびアンバインドされるときに、自動的にこのフィールド (および記述子の物理サイズ) を更新します。記述子が最初に割り当てられるとき、SQL_DESC_COUNT の初期値は 0 です。

記述子レコード

ゼロ個以上の記述子レコードが単一の記述子にあります。新しい列またはパラメーターがバインドされるとき、新しい記述子レコードがその記述子に追加されます。列またはパラメーターがアンバインドされるとき、その記述子レコードは除去されます。

表8 には、記述子レコードにあるフィールドをリストしています。それぞれは 1 つの列またはパラメーターを記述し、各記述子レコード内で 1 回だけ出現します。

表 8. レコード・フィールド

SQL_DESC_AUTO_UNIQUE_VALUE	SQL_DESC_LOCAL_TYPE_NAME
SQL_DESC_BASE_COLUMN_NAME	SQL_DESC_NAME
SQL_DESC_BASE_TABLE_NAME	SQL_DESC_NULLABLE
SQL_DESC_CASE_SENSITIVE	SQL_DESC_OCTET_LENGTH
SQL_DESC_CATALOG_NAME	SQL_DESC_OCTET_LENGTH_PTR
SQL_DESC_CONCISE_TYPE	SQL_DESC_PARAMETER_TYPE
SQL_DESC_DATA_PTR	SQL_DESC_PRECISION
SQL_DESC_DATETIME_INTERVAL_CODE	SQL_DESC_SCALE

表 8. レコード・フィールド (続き)

SQL_DESC_DATETIME_INTERVAL_PRECISION	SQL_DESC_SCHEMA_NAME
SQL_DESC_DISPLAY_SIZE	SQL_DESC_SEARCHABLE
SQL_DESC_FIXED_PREC_SCALE	SQL_DESC_TABLE_NAME
SQL_DESC_INDICATOR_PTR	SQL_DESC_TYPE
SQL_DESC_LABEL	SQL_DESC_TYPE_NAME
SQL_DESC_LENGTH	SQL_DESC_UNNAMED
SQL_DESC_LITERAL_PREFIX	SQL_DESC_UNSIGNED
SQL_DESC_LITERAL_SUFFIX	SQL_DESC_UPDATABLE

上記フィールドのそれぞれの詳細については、 SQLSetDescField() の 714ページの『レコード・フィールド』を参照してください。

据え置きフィールド: 据え置きフィールドは、記述子ヘッダーまたは記述子レコード作成時に作成されます。定義される変数のアドレスは保管されますが、アプリケーションで使用されるのはもっと後です。これらの変数をフィールドに関連付けている時や、CLI がそれらを読み書きしている間は、アプリケーションはこれらの変数を割り当て解除または廃棄してはなりません。

以下の表には、据え置きフィールドとその意味、また NULL ポインターが適用できる箇所を示しています。

表 9. 据え置きフィールド

フィールド	ヌル値の意味
SQL_DESC_DATA_PTR	レコードがアンバインドされています。
SQL_DESC_INDICATOR_PTR	(なし)
SQL_DESC_OCTET_LENGTH_PTR (ARD および APD 専用)	• ARD: 列の長さ情報が返されない。 • APD: パラメーターが文字ストリングの場合、ドライバーは文字列がヌル値であるとみなす。出力パラメーターの場合、このフィールドのヌル値はドライバーが長さ情報を返さないようにします。(SQL_DESC_TYPE フィールドが文字ストリング・パラメーターを指定しない場合は、SQL_DESC_OCTET_LENGTH_PTR フィールドは無視されます。)
SQL_DESC_ARRAY_STATUS_PTR (複数行取り出し専用)	複数行の取り出しで、行単位の診断情報の構成要素を返すのに失敗する。
SQL_DESC_ROWS_PROCESSED_PTR (複数行取り出し専用)	(なし)

バインド済み記述子レコード: 各記述子レコードの SQL_DESC_DATA_PTR フィールドは、パラメーター値 (APD の場合) または列の値 (ARD の場合) を含む変数を指しています。これは、ヌルを省略時値とする据え置きフィールドです。列またはパラメーターが一度バインドされると、それはパラメーターまたは列の値を指します。この時点で、記述子レコードはバインドされたことになります。

アプリケーション・パラメーター記述子 (APD)
: 各バインド済みレコードはバインド済みパラメーターを構成します。アプリケーションはステートメントを実行する前に、 SQL ステートメントにあるそれぞれの入出力パラメーター・マーカーごとに 1 つのパラメーターをバインドする必要があります。

アプリケーション行記述子 (ARD)
: 各バインド済みレコードは、バインド済みの列に関連しています。

整合性検査: 整合性検査は、アプリケーションが APD または ARD の SQL_DESC_DATA_PTR フィールドを設定するたびに、自動的に実行されます。検査では、種々のフィールドが互いに整合していること、および適切なデータ・タイプが指定されていることを確認します。

IPD フィールドの整合性検査を強制させるには、アプリケーションは IPD の SQL_DESC_DATA_PTR フィールドを設定します。この設定は整合性検査を強制する場合にのみ使用されます。値は保管されません。それで、`SQLGetDescField()` または `SQLGetDescRec()` への呼び出しで取り出すことはできません。

整合性検査は、IRD では実行できません。

整合性検査に関する詳細については、 `SQLSetDescRec()` の 730ページの『整合性検査』を参照してください。

記述子の割り当ておよび解放

記述子は次の 2 つの方法のどちらかで割り当てられます。

暗黙的に割り当てられる記述子
: ステートメント・ハンドルが割り当てられると、一連の 4 つの記述子が暗黙的に割り当てられます。ステートメント・ハンドルが解放されると、暗黙的に割り当てられた記述子ハンドルすべてが同様に解放されます。

暗黙的に割り当てられる記述子に対するハンドルを得るには、アプリケーションは、ステートメント・ハンドルおよび次に示す*属性* 値を渡して、 SQLGetStmtAttr() を呼び出します。

- SQL_ATTR_APP_PARAM_DESC (APD)
- SQL_ATTR_APP_ROW_DESC (ARD)
- SQL_ATTR_IMP_PARAM_DESC (IPD)
- SQL_ATTR_IMP_ROW_DESC (IRD)

明示的に割り当てられる記述子

アプリケーションは、明示的にアプリケーション記述子を割り当てることができます。しかし、実装記述子を割り当てることはできません。

接続でのアプリケーション記述子は、データベースに接続されるどんな時にでも明示的に割り当てることができます。このことは、ステートメント・ハンドルおよび次に示す*属性* 値を渡して、 SQLSetStmtAttr() を呼び出すことにより行われます。

- SQL_ATTR_APP_PARAM_DESC (APD)
- SQL_ATTR_APP_ROW_DESC (ARD)

この場合は、暗黙的に割り当てられた記述子よりも、明示的に指定され割り当てられる記述子が使用されます。

明示的に割り当てられる記述子は、複数のステートメントに関連付けることが可能です。

フィールドの初期設定

アプリケーションの行記述子が割り当てられると、そのフィールドは SQLSetDescField() のセクション 699ページの『記述子フィールドの初期設定』に示されている初期値を受け取ります。 SQL_DESC_TYPE フィールドは SQL_DEFAULT にセットされます。それは、アプリケーションへの表示のためのデータベース・データの標準的な取り扱いを提供します。アプリケーションは、記述子レコードのフィールドを設定することにより、データの異なる取り扱いを指定することができます。

SQL_DESC_ARRAY_SIZE ヘッダー・フィールドの初期値は 1 です。複数行の取り出しを可能にするには、アプリケーションはこの値を ARD にセットし、行セット内の行数を指定します。スクロール可能カーソル内の行セットに関する詳細は、 75ページの『スクロール可能カーソル』を参照してください。

IRD のフィールドについては省略時値がありません。これらのフィールドはステートメントの準備または実行時に設定されます。

IPD 内の以下のフィールドは、 SQLPrepare() への呼び出しで自動的に移植されるまでは未定義のままです。

- SQL_DESC_CASE_SENSITIVE
- SQL_DESC_FIXED_PREC_SCALE
- SQL_DESC_TYPE_NAME
- SQL_DESC_DESC_UNSIGNED
- SQL_DESC_LOCAL_TYPE_NAME

IPD の自動移植

アプリケーションが準備済み SQL ステートメントのパラメーターに関する情報を知りたい場合もあります。ある ad-hoc 照会が準備された場合の適切な例を考えてみましょう。アプリケーションは事前に、パラメーターに関することは何もわかりません。アプリケーションが SQL_ATTR_ENABLE_AUTO_IPD ステートメント属性を SQL_TRUE に (SQLSetStmtAttr() を使用して) 設定することで、 IPD の自動移植を可能にすると、 IPD のフィールドはパラメーターを記述するため自動的に移植されます。これには、データ・タイプ、精度、位取りなど (SQLDescribeParam() が返すのと同じ情報) が含まれます。アプリケーションはこの情報を使って、データ変換が必要かどうか、およびどのアプリケーション・バッファーがパラメーターをバインドするのに最も適切かを判別します。

IPD の自動移植には、いくらかのオーバーヘッドを必要とします。この情報が CLI ドライバーにより自動的に集められる必要がない場合は、SQL_ATTR_ENABLE_AUTO_IPD ステートメント属性は SQL_FALSE に設定してください。これは省略時の設定であり、アプリケーションで必要がなくなったときに、この値を返すべきです。

IPD の自動移植が活動中のとき、SQLPrepare() への各呼び出しを使用すると、 IPD のフィールドが更新されることになります。その結果の記述子情報は、次の関数を呼び出すことにより取り出せます。

- SQLGetDescField()
- SQLGetDescRec()
- SQLDescribeParam()

記述子の解放

明示的に割り当てられる記述子

明示的に割り当てられる記述子が解放されると、その解放された記述子が適用されていたすべてのステートメント・ハンドルは、暗黙的に割り当てられていた元の記述子に自動的に戻ります。

明示的に割り当てられている記述子は、次の 2 つの方法のどちらかにより解放されます。

- SQL_HANDLE_DESC の *HandleType* を指定して SQLFreeHandle() を呼び出す。
- 記述子に関連する接続ハンドルを解放する。

暗黙的に割り当てられる記述子

暗黙的に割り当てられている記述子は、次の 2 つの方法のどちらかにより解放されます。

- 接続でオープンしている任意のステートメントまたは記述子を除去する SQLDisconnect() を呼び出す。
- SQL_HANDLE_STMT の *HandleType* を指定して、SQLFreeHandle() を呼び出す。これによって、ステートメント・ハンドルと、ステートメントに関連する暗黙的に割り当てられたすべての記述子を解放します。

暗黙的に割り当てられた記述子は、 SQL_HANDLE_DESC の *HandleType* を指定して SQLFreeHandle() を呼び出すことでは解放できません。

記述子フィールドの入手、設定、およびコピー

以降のセクションでは、記述子ハンドルを使用した記述子の操作について説明します。最終セクションである 115ページの『ハンドルを使用しない記述子のアクセス』では、記述子ハンドルを使わない CLI 関数を呼び出すことにより記述子の値をどのように操作するかを説明します。

明示的に割り当てられる記述子のハンドルは、その記述子を割り当てるためにアプリケーションが SQLAllocHandle() を呼び出す時点で、 *OutputHandlePtr* 引き数に返されます。

暗黙的に割り当てられる記述子のハンドルは、SQL_ATTR_IMP_PARAM_DESC または SQL_ATTR_IMP_ROW_DESC のどちらかを指定して SQLGetStmtAttr() を呼び出すことで得られます。

記述子フィールドの値の取り出し

記述子レコードの単一フィールドを得る方法の詳細については、501ページの『SQLGetDescField - 記述子レコードの単一フィールド設定を入手する』を参照してください。

データ・タイプおよび列またはパラメーター・データの記憶域に影響する複数の記述子フィールドの設定を入手する方法については、 507ページの『SQLGetDescRec - 記述子レコードの複数フィールド設定を入手する』を参照してください。

記述子フィールドの値の設定

このセクションでは記述子ハンドルを使用して記述子フィールドの値を設定する方法を扱っています。また、記述子ハンドルを使用しないでこれらのフィールドの多くを設定することもできます。詳細については、115ページの『ハンドルを使用しない記述子のアクセス』を参照してください。

次の 2 つの方式を使用して、記述子フィールドを設定することができます。それは、一度に 1 つのフィールド、または一度に複数のフィールドの 2 つの方式です。

個々に記述子フィールドを設定する: 中には、読み取り専用の記述子フィールドもありますが、その他のフィールドは関数 SQLSetDescField() を使用して設定できます。以下のものを設定する各フィールドに関する特有の情報については、以下のセクションを参照してください。

- 707ページの『ヘッダー・フィールド』
- 714ページの『レコード・フィールド』

レコードおよびヘッダー・フィールドは、 SQLSetDescField() を使用してそれぞれに設定されます。

ヘッダー・フィールド

SQLSetDescField() への呼び出しでは、設定するヘッダー・フィールドと、レコード番号 0 を渡します。記述子につき 1 つのヘッダー・フィールドしかないので、レコード番号は無視されます。この場合、レコード番号 0 はブックマーク・フィールドを示していません。

レコード・フィールド

SQLSetDescField() への呼び出しでは、設定するレコード・フィールドと、レコード番号 1 またはそれ以上の数値を渡します。あるいは、ブックマーク・フィールドを示す 0 を渡します。

記述子の個々のフィールドを設定するときは、 698ページの『記述子フィールドの設定順序』で定義されているステップに従ってください。いくつかのフィールドを設定すれば、 DB2 CLI は自動的にその他のフィールドを設定できます。整合性検査は、アプリケーションが指定のステップに従った後に行われます。これは、記述子フィールドの値が整合していることを確認します。詳細については、109ページの『整合性検査』を参照してください。

記述子を設定するはずの関数呼び出しが失敗した場合、関数呼び出しが失敗した後は、その記述子フィールドの内容は未定義のままです。

一度に複数の記述子フィールドを設定する: 事前に定義された記述子フィールドのセットを、個々のフィールドを 1 つずつ設定するのではなく、 1 回の呼び出しでまとめて設定することができます。 SQLSetDescRec() は、単一の列またはパラメーターについて、以下のフィールドを設定します。

- 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 も ODBC で定義されていますが、 DB2 CLI ではサポートされていません。)

詳細については、728ページの『SQLSetDescRec - 列またはパラメーター・データに複数の記述子フィールドを設定する』を参照してください。

記述子のコピー

記述子の 1 つの利点は、単一の記述子が多目的に使用できるという点にあります。たとえば、あるステートメント・ハンドルでの ARD を、別のステートメント・ハンドルでの APD として使用できます。

他の例を挙げてみましょう。ここで、アプリケーションが元の記述子のコピーを作ろうとします。そしてあるフィールドを変更します。この場合には、SQLCopyDesc() を使用して別の記述子からの値によって既存の記述子のフィールドを上書きします。複写元記述子および複写先記述子の両方で定義されているフィールドだけが、コピーされます (変更できない SQL_DESC_ALLOC_TYPE フィールドは例外) 。

フィールドはどんなタイプの記述子からもコピーできますが、アプリケーション記述子 (APD や ARD) または IPD に対してだけコピーできます。 IRD にコピーすることはできません。記述子の割り当てタイプは、コピー手順によって変更されることはありません (SQL_DESC_ALLOC_TYPE フィールドは変更できません)。

記述子のコピーについての詳細な説明は、361ページの『SQLCopyDesc - ハンドル間で記述子情報をコピーする』を参照してください。

ハンドルを使用しない記述子のアクセス

記述子に関してこのセクションの初めに述べたように、多くの CLI 関数は、記述子を使用していますが、アプリケーション自身は直接、記述子を操作する必要はありません。その代わりに、アプリケーションは他の関数を実行する場合と同じように、 1 つまたは複数の記述子フィールドを設定または取り出す別個の関数を使用できます。このカテゴリーの CLI 関数は、コンサイス 関数と呼ばれています。 SQLBindCol() は、記述子フィールドを操作するコンサイス関数の一例です。

複数のフィールドを操作することに加えて、コンサイス関数は記述子ハンドルを明示的に指定しないで呼び出されます。それで、アプリケーションは、コンサイス関数を使用するために記述子ハンドルを取り出す必要さえもありません。

以下のタイプのコンサイス関数があります。

- 関数 SQLBindCol() および SQLBindParameter() は、引き数に対応する記述子フィールドを設定することで、列またはパラメーターをバインドします。また、これらの関数は記述子に関連のないその他のタスクも実行します。

  また、必要であれば、アプリケーションは記述子呼び出しを使用して、バインドの個々の細目を直接変更することができます。この場合、記述子ハンドルを取り出し、バインドを変更するために関数 SQLSetDescField() または SQLSetDescRec() を呼び出す必要があります。
- 以下の関数は常に記述子フィールドの値を取り出します。
  - SQLColAttribute()
  - SQLDescribeCol()
  - SQLDescribeParam()
  - SQLNumParams()
  - SQLNumResultCols()
- 関数 SQLSetDescRec() と SQLGetDescRec() は、データ・タイプおよび列またはパラメーター・データの記憶域に影響する複数の記述子フィールドを設定または入手します。 SQLSetDescRec() への単一呼び出しを使用すると、列またはパラメーターのバインドで使用する値を変更することができます。
- 関数 SQLSetStmtAttr() および SQLGetStmtAttr() は、どのステートメント属性が指定されるかに応じて、記述子フィールドを変更または返します。詳細は、106ページの『記述子に保管される値』を参照してください。

記述子のサンプル

```
/* ... */
    SQLCHAR * sqlstmt =
       "SELECT deptname, location from org where division = ? " ;
/* ... */
    /* macro to initalize server, uid and pwd */
    INIT_UID_PWD ;
    /* allocate an environment handle */
    rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv ) ;
    if ( rc != SQL_SUCCESS ) return( terminate( henv, rc ) ) ;
    /* allocate a connect handle, and connect */
    rc = DBconnect( henv, &hdbc ) ;
    if ( rc != SQL_SUCCESS ) return( terminate( henv, rc ) ) ;
    rc = SQLAllocHandle( SQL_HANDLE_STMT, hdbc, &hstmt ) ;
    CHECK_HANDLE( SQL_HANDLE_DBC, hdbc, rc ) ;
    /* Use SQLGetStmtAttr() to get implicit parameter descriptor handle */
    rc = SQLGetStmtAttr ( hstmt,
                          SQL_ATTR_IMP_PARAM_DESC,
                          &hIPDdesc,
                          SQL_IS_POINTER,
                          NULL);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    /* Use SQLGetStmtAttr() to get implicit row descriptor handle */
    rc = SQLGetStmtAttr ( hstmt,
                          SQL_ATTR_IMP_ROW_DESC,
                          &hIRDdesc,
                          SQL_IS_POINTER,
                          NULL);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    /* Call SQLGetDescField() to see how the header field */
    /* SQL_DESC_ALLOC_TYPE is set. */
    rc = SQLGetDescField( hIPDdesc,
                          0, /* ignored for header fields */
                          SQL_DESC_ALLOC_TYPE,
                          &desc_smallint, /* The result */
                          SQL_IS_SMALLINT,
                          NULL ); /* ignored */
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    /* Print the descriptor information */
    printf("The IPD header descriptor field SQL_DESC_ALLOC_TYPE is %s¥n",
           ALLOCTYPES[desc_smallint]);
    /* prepare statement for multiple use */
    rc = SQLPrepare(hstmt, sqlstmt, SQL_NTS);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    /* bind division to parameter marker in sqlstmt */
    rc = SQLBindParameter( hstmt,
                           1,
                           SQL_PARAM_INPUT,
                           SQL_C_CHAR,
                           SQL_CHAR,
                           10,
                           0,
                           division.s,
                           11,
```

```
                      NULL
                     ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
/* bind deptname to first column in the result set */
rc = SQLBindCol(hstmt, 1, SQL_C_CHAR, (SQLPOINTER) deptname.s, 15,
                &deptname.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER) location.s, 14,
                &location.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
/* Call SQLGetDescField() to see how the descriptor record */
/* field SQL_DESC_PARAMETER_TYPE is set */
rc = SQLGetDescField( hIPDdesc,
                      1, /* Look at the parameter */
                      SQL_DESC_PARAMETER_TYPE,
                      &desc_smallint, /* The result */
                      SQL_IS_SMALLINT,
                      NULL ); /* ignored */
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
printf("The IPD record descriptor field SQL_DESC_PARAMETER_TYPE is %s¥n",
       PARAMTYPE[desc_smallint]);
strcpy( division.s,  "Eastern");
rc = SQLExecute(hstmt);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
printf("¥nDepartments in %s Division:¥n", division.s);
printf("Department       Location¥n");
printf("-------------- ------------¥n");
while ( ( rc = SQLFetch( hstmt ) ) == SQL_SUCCESS )
   CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
   printf( "%-14.14s %-13.13s ¥n", deptname.s, location.s ) ;
if ( rc != SQL_NO_DATA_FOUND )
   CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
/* Print out some implementation row descriptor fields */
/* from the last SQLFetch() above */
for (colCount = 1; colCount <=2; colCount++) {
   printf("¥nInformation for column %i¥n",colCount);
   /* Call SQLGetDescField() to see how the descriptor record */
   /* field SQL_DESC_TYPE_NAME is set */
   rc = SQLGetDescField( hIRDdesc,
                colCount,
                SQL_DESC_TYPE_NAME, /* record field */
                desc_char, /* The result */
                25,
                NULL ); /* ignored */
   CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
   printf("  - IRD record descriptor field SQL_DESC_TYPE_NAME is %s¥n",
        desc_char);
   /* Call SQLGetDescField() to see how the descriptor record */
   /* field SQL_DESC_LABEL is set */
   rc = SQLGetDescField( hIRDdesc,
                colCount,
                SQL_DESC_LABEL, /* record field */
                desc_char, /* The result */
                25,
                NULL ); /* ignored */
```

```
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    printf("  - IRD record descriptor field SQL_DESC_LABEL is %s¥n",
         desc_char);
} /* End of the for statement */
```

複合 SQL の使用

複合 SQL を利用すると、複数のステートメントを実行可能な 1 つのブロックとしてグループ化することができます。このステートメントのブロックを入力パラメーター値と共に使って、 1 つの連続ストリームで実行することができ、これにより実行時間およびネットワーク通信量を少なくすることができます。複合 SQL は、INSERT、UPDATE、および DELETE ステートメントの集まりを効果的に実行するのに頻繁に使用されます。

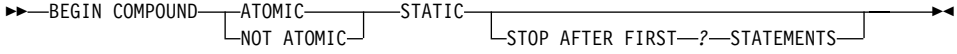
動的に作成される SQL ステートメント (照会ステートメントを除く) は、いずれも複合ステートメントの内部にあるステートメントとして実行することができます。複合 SQL ステートメントの中にあるステートメントは、サブステートメントと呼ばれます。複合 SQL は、サブステートメントが実行される順序を保証しないので、ステートメント間に依存性があってはなりません。

複合 SQL ステートメントはネストすることはできません。複合 SQL ステートメントの許可 ID は、複合 SQL ステートメント内に含まれる個々のサブステートメントのすべてについて適切な許可でなければなりません。

複合 SQL がサポートされるのは、DB2 ユニバーサル・データベースに接続する場合、または DB2 コネクト V 2.3 以降の DRDA 環境にある場合です。

ATOMIC および NOT ATOMIC 複合 SQL

複合 SQL ステートメントのブロックを指定する場合は、 BEGIN COMPOUND ステートメントと END COMPOUND ステートメントでサブステートメントを囲みます。 BEGIN COMPOUND 構文を次に示します。



ATOMIC

複合 SQL ステートメント内のいずれかのサブステートメントが失敗した場合、すべてのサブステートメントによってデータベースに加えられたすべての変更がやり直されることを指定します。 ATOMIC は DRDA 環境ではサポートされていません。

NOT ATOMIC

どのサブステートメントが失敗しても、複合 SQL ステートメントは他のサブステートメントによってデータベースに行われた変更をやり直さないことを指定します。

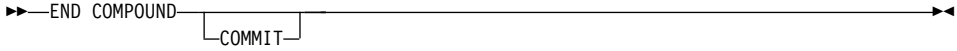
STATIC

すべてのサブステートメントの入力変数が、元の値を保存することを指定します。同じ変数が 2 つ以上のサブステートメントで設定されている場合、複合 SQL ステートメントの後の変数の値は、最後のサブステートメントで設定された値になります。

STOP AFTER FIRST ? STATEMENTS

特定の数のサブステートメントだけを実行するよう指定します。この文節が省略されると、すべてのサブステートメントが実行されます。

END COMPOUND 構文を次に示します。



COMMIT オプションを指定すると、すべてのサブステートメントが正常に実行された場合に、それらがコミットされます。 COMMIT は、複合ステートメントの前にあるステートメントも含めて、現行のトランザクションに適用されます。 COMMIT が指定されており、かつ接続が調整分散接続 (SQL_COORDINATED_TRANS) である場合には、エラーが返されます (SQLSTATE 25000)。

END COMPOUND の後に COMMIT オプションを指定しないと、アプリケーションが自動コミット・モードで操作を行っている場合を除いて、サブステートメントはコミットされません。自動コミット・モードで操作されている場合は、END COMPOUND 時にコミットが出されます。自動コミット・モードの詳細については、25ページの『コミットまたはロールバック』を参照してください。

121ページの図15 は、複合 SQL ステートメントを実行する上で必要な関数呼び出しの一般的な順序を示しています。次の点について注意してください。

- `SQLPrepare()` と `SQLExecute()` を、 `SQLExecDirect()` の代わりに使用することができます。
- BEGIN COMPOUND および END COMPOUND ステートメントは同じステートメント・ハンドルで実行します。

- 個々のサブステートメントには独自のステートメント・ハンドルが必要です。
- すべてのステートメント・ハンドルは同じ接続に属し、同じ分離レベルでなければなりません。
- 必ずしも必要であるというわけではありませんが、可能であればサブステートメントは、 BEGIN COMPOUND ステートメントの前に作成されるようにします。特に DRDA 環境ではネットワーク・フローを減らすことが可能になる場合もあるのでそのようにします。
- END COMPOUND ステートメントが実行されるまで、ステートメント・ハンドルは割り振られている状態でなければなりません。
- 複合サブステートメント用に割り振られたステートメント・ハンドルを使用して呼び出せる関数は、以下のものだけです。
  - SQLAllocHandle()
  - SQLBindParameter()
  - SQLBindFileToParam()
  - SQLParamData()
  - SQLPutData()
  - SQLExecDirect()、SQLPrepare()、SQLExecute()
- SQLTransact() は、同一接続で、または BEGIN および END COMPOUND 間の接続要求で呼び出すことはできません。
- サブステートメントは任意の順序で実行できます。
- SQLRowCount() (または SQLGetSQLCA()) は、BEGIN、 END COMPOUND ステートメントと同じステートメント・ハンドルを使用して呼び出すことができ、影響を受けた行の集合カウントを得られます。

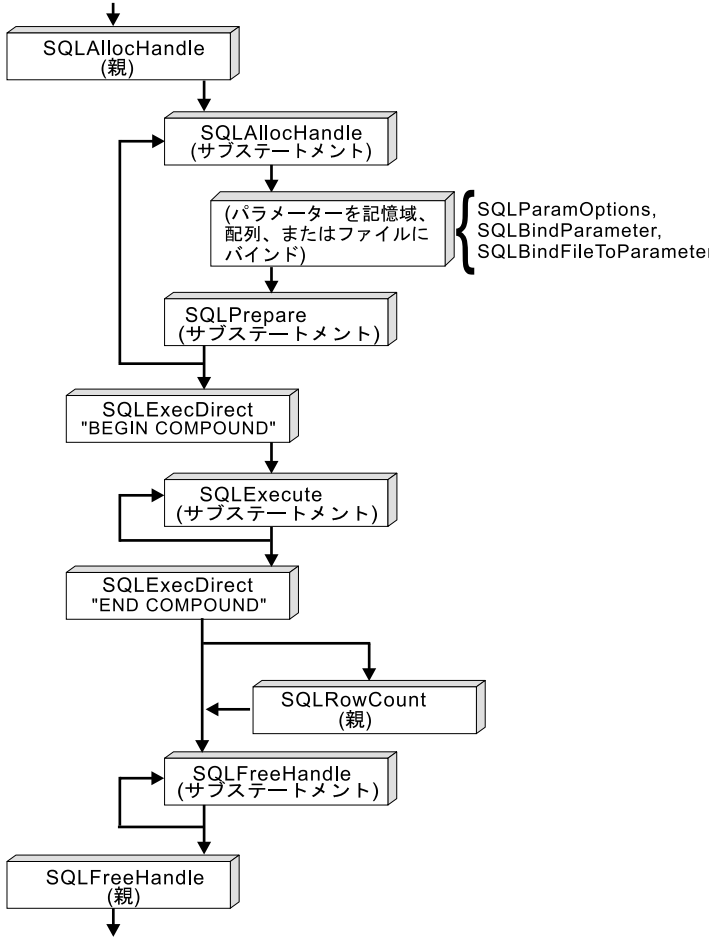


図 15. 複合 SQL

複合 SQL のエラー処理

複合ステートメントが ATOMIC で、 END COMPOUND `SQLExecDirect()` 呼び出しが次のものを返すと、次のようになります。

- SQL_SUCCESS - すべてのサブステートメントは実行され、警告やエラーはありませんでした。
- SQL_SUCCESS_WITH_INFO - すべてのサブステートメントは正常に実行されましたが、 1 つまたは複数の警告がありました。総称診断情報を得る場合は `SQLError()` を呼び出し、複合 SQL ステートメント全体の SQLCA を得る場合は `SQLGetSQLCA()` を呼び出してください。

SQLError() または SQLGetSQLCA() で使用するステートメント・ハンドルは、 BEGIN、END COMPOUND SQL を処理するのに使用したものと同じでなければなりません。

SQLCA 内の情報のほとんどは、データベース・サーバーが最後のサブステートメントを処理したときに設定した値を反映しています。たとえば、SQLCODE および SQLSTATE などです。 1 つまたは複数のエラーが発生し、いずれも重大でなければ、 SQLCA の SQLERRMC フィールドには、これらのエラーのうち最大 7 つに関する情報が入っています。

- SQL_NO_DATA_FOUND - BEGIN、 END COMPOUND は実行されたがすべてのサブステートメントは実行されなかったか、またはどのサブステートメントもいずれかの行に影響を与えませんでした。
- SQL_ERROR - 1 つまたは複数のサブステートメントが失敗しました。すべてのサブステートメントはロールバックされました。

複合ステートメントが NOT ATOMIC で、 END COMPOUND SQLExecDirect() 呼び出しが次のものを返すと、以下のようになります。

- SQL_SUCCESS - すべてのサブステートメントは実行され、エラーはありませんでした。
- SQL_SUCCESS_WITH_INFO - COMPOUND ステートメントは実行されましたが、 1 つまたは複数の警告がありました。 1 つまたは複数のサブステートメントが警告を返しました。警告の情報に関する追加情報を受け取るには、 SQLError() または SQLGetSQLCA() を呼び出してください。
- SQL_NO_DATA_FOUND - BEGIN、 END COMPOUND は実行されたがすべてのサブステートメントは実行されなかったか、またはどのサブステートメントもいずれかの行に影響を与えませんでした。
- SQL_ERROR - COMPOUND ステートメントは失敗しました。少なくとも 1 つのサブステートメントがエラーを返しました。 SQLCA を調べてどのステートメントが失敗したかを判別してください。

注: 複合 SQL 実行後の SQLCA の内容に関する詳細は、*SQL 解説書* を参照してください。

複合 SQL の例

次の例では、新規の AWARDS 表に行を挿入するために、 4 つのサブステートメントからなる複合ステートメントを実行します。

```
/* ... */
    SQLCHAR * stmt[] = {
        "INSERT INTO awards (id, award) "
        "SELECT id, 'Sales Merit' from staff "
```

```
        "WHERE job = 'Sales' AND (comm/100 > years)",
        "INSERT INTO awards (id, award) "
        "SELECT id, 'Clerk Merit' from staff "
        "WHERE job = 'Clerk' AND (comm/50 > years)",
        "INSERT INTO awards (id, award) "
        "SELECT id, 'Best ' concat job FROM STAFF "
        "WHERE comm = (SELECT max(comm) FROM staff WHERE job = 'Clerk')",
        "INSERT INTO awards (id, award) "
        "SELECT id, 'Best ' concat job FROM STAFF "
        "WHERE comm = (SELECT max(comm) FROM STAFF WHERE job = 'Sales')",
    } ;
    SQLINTEGER i ;
/* ... */
    /* Prepare 4 substatements */
    for ( i = 1; i < 4; i++ ) {
        rc = SQLAllocHandle( SQL_HANDLE_STMT, hdbc, &cmhstmt[i] ) ;
        CHECK_HANDLE( SQL_HANDLE_DBC, hdbc, rc ) ;
        rc = SQLPrepare( cmhstmt[i], stmt[i], SQL_NTS ) ;
        CHECK_HANDLE( SQL_HANDLE_STMT, cmhstmt[i], rc ) ;
    }
    rc = SQLExecDirect( hstmt,
                        ( SQLCHAR * ) "BEGIN COMPOUND NOT ATOMIC STATIC",
                        SQL_NTS
                      ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    /* Execute 4 substatements */
    for ( i = 1; i < 4; i++ ) {
        rc = SQLExecute( cmhstmt[i] ) ;
        CHECK_HANDLE( SQL_HANDLE_STMT, cmhstmt[i], rc ) ;
    }
    /* Execute the COMPOUND statement (of 4 sub-statements) */
    printf( "Executing the COMPOUND statement (of 4 sub-statements)¥n" ) ;
    rc = SQLExecDirect( hstmt,
                        ( SQLCHAR * ) "END COMPOUND COMMIT",
                        SQL_NTS
                      ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    rc = SQLFreeHandle( SQL_HANDLE_STMT, hstmt ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    for ( i = 1; i < 4; i++ ) {
        rc = SQLFreeHandle( SQL_HANDLE_STMT, cmhstmt[i] ) ;
        CHECK_HANDLE( SQL_HANDLE_STMT, cmhstmt[i], rc ) ;
    }
```

ラージ・オブジェクトの使用

ラージ・オブジェクト という用語および総称頭字語の *LOB* は、任意のタイプのラージ・オブジェクトを指して呼ぶのに使われます。 3 つの LOB データ・タイプがあります。それは 2 進ラージ・オブジェクト (BLOB)、文字ラージ・オブジェクト (CLOB)、および 2 バイト文字ラージ・オブジェクト (DBCLOB) です。これらの LOB データ・タイプは記号でそれぞれ、SQL_BLOB、SQL_CLOB、SQL_DBCLOB と表されます。 33ページの表2 のリストには 3

つの LOB データ・タイプ、それに対応する記号名、および省略時の C 記号名の項目があります。 SQL データ・タイプ引き数を受け入れたり返したりする DB2 CLI 関数 (SQLBindParameter()、 SQLDescribeCol() など) の場合は、LOB 記号定数を指定したり返したりすることができます。

LOB 値は非常に大きいことがあるので、 SQLGetData() および SQLPutData() による分割の順次方式を使用してデータを転送すると、非常に時間がかかる可能性があります。この種のデータを扱うアプリケーションの場合、普通はセグメント単位で、または直接ファイル入出力を使って転送を行います。

アプリケーションがラージ・オブジェクト値を選択してその部分に関する操作を行う必要があるが、その値全体をデータベース・サーバーからアプリケーションのメモリーへ転送する必要がなかったり、転送したくないような場合がよくあります。このような場合、アプリケーションでラージ・オブジェクト・ロケーター (LOB ロケーター) を使って個々の LOB 値を参照することができます。

LOB ロケーターは 1 つの機構で、アプリケーション・プログラムがラージ・オブジェクト値をランダム・アクセス方法によって効率的に操作できるようにするものです。 LOB ロケーターとは実行時の概念で、持続タイプではなく、データベースに保管もされません。これはトランザクション中に LOB 値を参照するために使用する機構で、作成されたトランザクションを越えて持続することはありません。 3 つの LOB ロケーター・タイプのそれぞれには、独自の C データ・タイプ (SQL_C_BLOB_LOCATOR、SQL_C_CLOB_LOCATOR、SQL_C_DBCLOB_LOCATOR) があります。これらのタイプを使用すると、データベース・サーバーとの間で LOB ロケーター値を転送できるようになります。

LOB ロケーターは、1 つの LOB 値を表す単純なトークン値です。ロケーターは、レコード内のフィールドを参照するためではなく、ラージ・オブジェクト *値* を参照するために作成されます。行に保管されている元の LOB 値に有効なロケーターについては、実行できる操作はありません。アプリケーションは、LOB ロケーターをアプリケーション変数中に (SQLBindCol() または SQLGetData() 関数を使用して) 取り出してから、そのロケーターによって関連付けられている LOB 値に次の DB2 CLI 関数を適用することができます。

SQLGetLength()
LOB ロケーターによって表されるストリングの長さを入手します。

SQLGetPosition()
LOB ロケーターによって表されているソース・ストリング内の探索ストリングの位置を入手します。探索ストリングを LOB ロケーターによって表すこともできます。

次のことを行うと、ロケーターが暗黙割り振りされます。

- バインドされた LOB 列を適切な C ロケーター・タイプに取り出します。
- SQLGetSubString() を呼び出して、サブストリングをロケーターとして取り出すよう指定します。
- バインドされていない LOB 列について SQLGetData() を呼び出して、適切な C ロケーター・タイプを指定します。ロケーター C タイプは LOB 列タイプと一致していなければなりません。一致していないとエラーが発生します。

LOB ロケーターも、サーバーの表のある列のデータを (同じまたは異なる表の) 別の列に移動するときに、そのデータを一度アプリケーション・メモリーに取り出してからサーバーに送り返す必要がなく、便利な方法です。たとえば、次の INSERT ステートメントは、ロケーターによって表される 2 つの LOB 値が連結された 1 つの LOB 値を挿入します。

```
INSERT INTO lobtable values (CAST ? AS CLOB(4k) || CAST ? AS CLOB(5k))
```

FREE LOCATOR ステートメントを実行して、トランザクションの終了前にロケーターを明示的に解放することができます。構文を次に示します。

このステートメントは動的に準備することはできませんが、 DB2 CLI はこれを SQLPrepare() および SQLExecDirect() にとって有効なステートメントとして受け入れます。アプリケーションは、 SQL データ・タイプ引き数を適切な SQL および C 記号データ・タイプ (33ページの表2 を参照) に設定して、SQLBindParameter() を使用します。

もう一つの方法として、アプリケーションで LOB 列値全体が必要な場合に、LOB に関する直接ファイル入出力を要求することができます。データベースの照会、更新、および挿入には、 1 つ 1 つの LOB 列値をファイルとの間でやりとりすることが含まれています。 DB2 CLI LOB ファイル・アクセス関数には、以下の 2 つがあります。

SQLBindFileToCol()
結果セット内の LOB 列をファイル名にバインド (関連付け) します。

SQLBindFileToParam()
LOB パラメーター・マーカーをファイル名にバインド (関連付け) します。

ファイル名は、ファイルの完全パス名 (これをお勧めします) か、相対ファイル名のいずれかです。相対ファイル名が指定されると、クライアント・プロセスの (操作環境の) 現行パスにその名前が追加されます。実行または取り出しの際に、ファイルとの間のデータ転送は、バインド済みアプリケーション変数の場合と同様に行われます。これら 2 つの関数に関連づけられているファイル・オプション引き数は、転送時にファイルを処理する方法を指定します。

SQLBindFileToParam() を使用する方が、 SQLPutData() を使用してデータ・セグメントを順次入力するよりも効率的です。 SQLPutData() の場合は入力セグメントを一時ファイルへ完全に挿入してから、 SQLBindFileToParam() 手法を使って LOB データ値をサーバーへ送信するからです。アプリケーションで SQLPutData() を使用する代わりに SQLBindFileToParam() を活用することをお勧めします。

SQL_LONGVARCHAR を使用して文字ラージ・オブジェクト・データを参照し、 SQL_LONGVARBINARY を使用して 2 進ラージ・オブジェクト・データを参照する総称 ODBC アプリケーションを作成する場合の詳細については、 835ページの『付録C. DB2 CLI と ODBC』を参照してください。

現在すべての DB2 サーバーでラージ・オブジェクトがサポートされている訳ではありません。いずれかの LOB 関数が現行のサーバーでサポートされているかどうかを判別するには、該当する関数名の引き数値を指定して SQLGetFunctions() を呼び出してください。

127ページの図16 は、文字 LOB (CLOB) の取り出しを示しています。

- 図の左側は、ロケーターを使用して、 CLOB 全体をアプリケーション・バッファーへ転送せずに CLOB から文字ストリングを抽出することを示しています。

  LOB ロケーターが取り出され、次いでこのロケーターがサブストリングの CLOB を探索するための入力パラメーターとして使用されて、サブストリングが検索されます。
- 右側は、CLOB が直接ファイル内に取り出される様子を示しています。

  ファイルはまず CLOB 列にバインドされ、行が取り出されると、 CLOB 値全体が直接ファイルに転送されます。

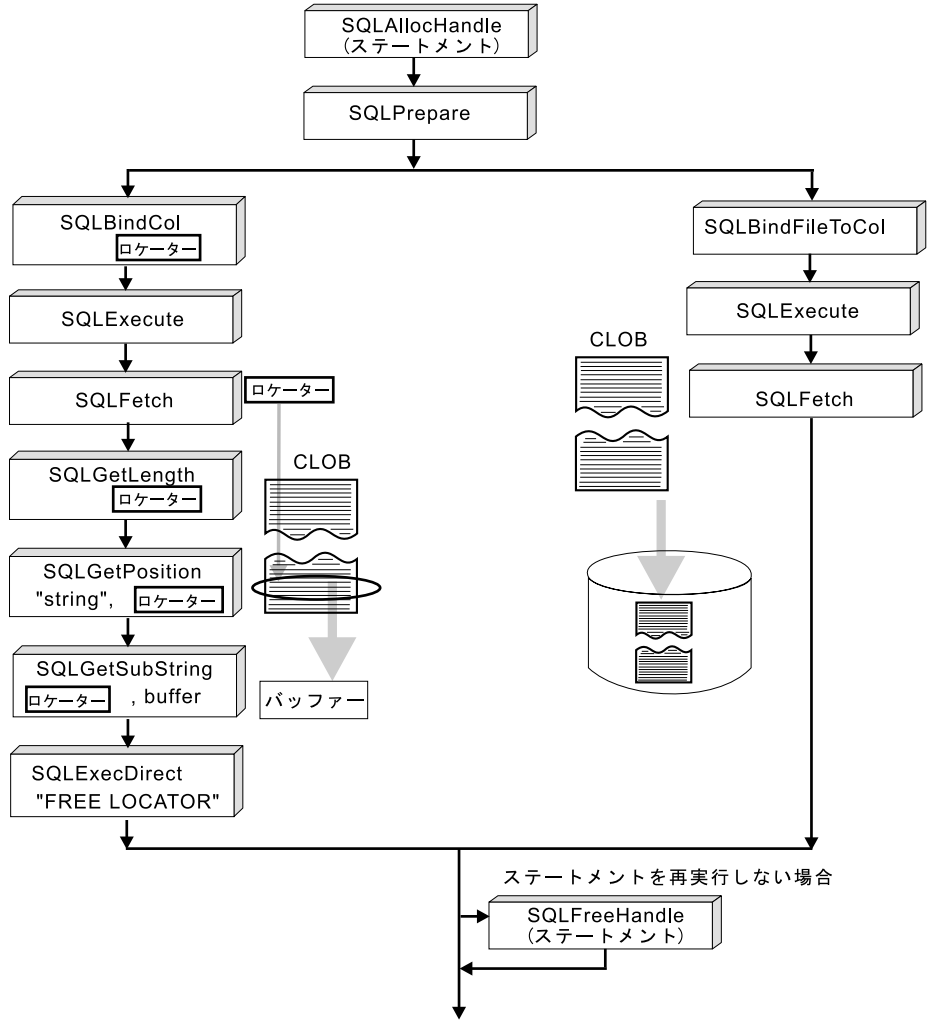


図 16. CLOB データの取り出し

LOB の例

次の例では、EMP_RESUME 表の Resume CLOB 列から ″Interests″ セクションを抽出します。サブストリングだけがアプリケーションに転送されます。

```
/* From the CLI sample dtlob.c */
/* ... */
    /* get the starting postion of the CLOB piece of data */
    sqlrc = SQLGetPosition( hstmtLocUse,
```

```
SQL_C_CLOB_LOCATOR,
clobLoc,
0,
( SQLCHAR * ) "Interests",
strlen( "Interests"),
1,
&clobPiecePos,
&ind ) ;
```

ODBC アプリケーションでの LOB の使用

既存の ODBC アプリケーションは、 DB2 の BLOB および CLOB データ・タイプの代わりに SQL_LONGVARCHAR および SQL_LONGVARBINARY を使用します。 LONGDATACOMPAT キーワードを初期設定ファイルに設定するか、または SQLSetConnectAttr() を使用して SQL_ATTR_LONGDATA_COMPAT 接続属性を設定することにより、 DB2 CLI は ODBC 長形式データ・タイプを DB2 LOB データ・タイプにマッピングします。

このマッピングが有効になると、次のようになります。

- SQL_LONGVARCHAR、SQL_LONGVARBINARY または SQL_LONGVARGRAPHIC を指定して SQLGetTypeInfo() を呼び出すと、CLOB、BLOB、および DBCLOB 特性が返されます。
- CLOB、BLOB、または DBCLOB データ・タイプの記述であれば、以下の関数は SQL_LONGVARCHAR、SQL_LONGVARBINARY または SQL_LONGVARGRAPHIC を返します。
  - SQLColumns()
  - SQLSpecialColumns()
  - SQLDescribeCol()
  - SQLColAttribute()
  - SQLProcedureColumns()
- LONG VARCHAR および LONG VARCHAR FOR BIT DATA は、引き続き SQL_LONGVARCHAR および SQL_LONGVARBINARY として記述されます。

SQL_ATTR_LONGDATA_COMPAT の省略時設定は、SQL_LD_COMPAT_NO です。マッピングは有効ではありません。

詳細については、174ページの『構成キーワード』および 663ページの『SQLSetConnectAttr - 接続属性を設定する』を参照してください。

マッピングが有効になると、ODBC アプリケーションは SQLGetData()、SQLPutData()、および関連関数を使用して LOB データを取り出すことができます。データの分割挿入や分割取り出しの詳細については、87ページの『長形式データの分割送信 / 取り出し』を参照してください。

注: DB2 CLI は、LOB データを分けて挿入するときに一時ファイルを使用します。データが元々ファイルにある場合は、SQLBindFileToParam() を使用して、一時ファイルを使用しないようにすることができます。SQLGetFunctions() を呼び出して、 SQLBindFileToParam() のサポートがあるかどうかを照会してください。

ユーザー定義タイプ (UDT) の使用

32ページの『データ・タイプとデータ変換』に定義されている SQL データ・タイプ (基本 SQL データ・タイプといいます) に加えて、新しい特殊タイプ (distinct type) をユーザー側で定義することもできます。これらのユーザー定義タイプ (UDT) は、内部表記を既存のタイプと共用しますが、既存タイプとは独立していて、ほとんどの操作で互換性のないタイプであるとみなされます。この UDT は、CREATE DISTINCT TYPE SQL ステートメントを使用して作成します。

UDT は、オブジェクト指向プログラミングで必要な強力型指定制御を行うのに役立ち、特殊タイプで明示定義された関数や演算子だけをそのインスタンスに確実に適用できるようにします。アプリケーションは、アプリケーション変数については引き続き C データ型で処理するので、 SQL ステートメントを組み立てる場合に限り UDT タイプを考慮する必要があります。

これは次のことを意味します。

- 組み込みタイプに適用される SQL から C データ・タイプ変換規則が、すべて UDT に適用されます。
- UDT は、組み込みタイプと同じ省略時 C タイプになります。
- SQLDescribeCol() は組み込みタイプ情報を返します。ユーザー定義のタイプ名を得るには、 SQL_DESC_DISTINCT_TYPE に設定された入力記述子タイプを指定して、 SQLColAttribute() を呼び出します。
- パラメーター・マーカーが含まれる SQL 述部は、明示的に UDT へキャストされなければなりません。アプリケーションは組み込みタイプしか処理できないので、これは必須です。パラメーターを使って操作を実行する前に、これを C 組み込みタイプから UDT へキャストしなければなりません。こ

のことを行わないと、ステートメント作成時にエラーが起きてしまいます。詳細については、831ページの『述部内でのユーザー定義タイプ』を参照してください。

ユーザー定義タイプ (UDT) の詳細な規則と説明については、 *SQL 解説書* を参照してください。

ユーザー定義タイプの例

この例では、複数の UDT および UDF が定義されていることを示し、 UDT 列がある複数の表も示します。 UDT 列のある表へ行を挿入する例については、98ページの『配列の入力例』を参照してください。

```
/* ... */
    /* Initialize SQL statement strings */
    SQLCHAR * stmt[] = {
        "CREATE DISTINCT TYPE CNUM AS INTEGER WITH COMPARISONS",
        "CREATE DISTINCT TYPE PUNIT AS CHAR(2) WITH COMPARISONS",
        "CREATE DISTINCT TYPE UPRICE AS DECIMAL(10, 2) "
        "WITH COMPARISONS",
        "CREATE DISTINCT TYPE PRICE AS DECIMAL(10, 2) "
        "WITH COMPARISONS",
        "CREATE FUNCTION PRICE( CHAR(12), PUNIT, char(16) ) "
        "returns char(12) "
        "NOT FENCED EXTERNAL NAME 'order!price' "
        "NOT VARIANT NO SQL LANGUAGE C PARAMETER STYLE DB2SQL "
        "NO EXTERNAL ACTION",
        "CREATE DISTINCT TYPE PNUM AS INTEGER WITH COMPARISONS",
        "CREATE FUNCTION ¥"+¥"(PNUM, INTEGER) RETURNS PNUM "
        "source sysibm.¥"+¥"(integer, integer)",
        "CREATE FUNCTION MAX(PNUM) RETURNS PNUM "
        "source max(integer)",
        "CREATE DISTINCT TYPE ONUM AS INTEGER WITH COMPARISONS",
        "CREATE TABLE CUSTOMER ( "
        "Cust_Num     CNUM NOT NULL, "
        "First_Name   CHAR(30) NOT NULL, "
        "Last_Name    CHAR(30) NOT NULL, "
        "Street       CHAR(128) WITH DEFAULT, "
        "City         CHAR(30) WITH DEFAULT, "
        "Prov_State   CHAR(30) WITH DEFAULT, "
        "PZ_Code      CHAR(9) WITH DEFAULT, "
        "Country      CHAR(30) WITH DEFAULT, "
        "Phone_Num    CHAR(20) WITH DEFAULT, "
        "PRIMARY KEY  (Cust_Num) )",
        "CREATE TABLE PRODUCT ( "
        "Prod_Num     PNUM NOT NULL, "
        "Description  VARCHAR(256) NOT NULL, "
        "Price        DECIMAL(10,2) WITH DEFAULT , "
        "Units        PUNIT NOT NULL, "
        "Combo        CHAR(1) WITH DEFAULT, "
        "PRIMARY KEY (Prod_Num), "
        "CHECK (Units in (PUNIT('m'), PUNIT('l'), PUNIT('g'), PUNIT('kg'), "
        "PUNIT(' ')))  )",
```

```
        "CREATE TABLE PROD_PARTS ( "
        "Prod_Num     PNUM NOT NULL, "
        "Part_Num     PNUM NOT NULL, "
        "Quantity     DECIMAL(14,7), "
        "PRIMARY KEY (Prod_Num, Part_Num), "
        "FOREIGN KEY (Prod_Num) REFERENCES Product, "
        "FOREIGN KEY (Part_Num) REFERENCES Product, "
        "CHECK (Prod_Num <> Part_Num) )",
        "CREATE TABLE ORD_CUST ( "
        "Ord_Num      ONUM NOT NULL, "
        "Cust_Num     CNUM NOT NULL, "
        "Ord_Date     DATE NOT NULL, "
        "PRIMARY KEY (Ord_Num), "
        "FOREIGN KEY (Cust_Num) REFERENCES Customer )",
        "CREATE TABLE ORD_LINE ( "
        "Ord_Num      ONUM NOT NULL, "
        "Prod_Num     PNUM NOT NULL, "
        "Quantity     DECIMAL(14,7), "
        "PRIMARY KEY (Ord_Num, Prod_Num), "
        "FOREIGN KEY (Prod_Num) REFERENCES Product, "
        "FOREIGN KEY (Ord_Num) REFERENCES Ord_Cust )",
        ( char * ) 0,
    } ;
/* ... */
    /* Execute Direct statements */
    i = 0 ;
    while ( stmt[i] != ( char * ) 0 ) {
       printf( ">Executing Statement %ld¥n", ( i + 1 ) ) ;
       rc = SQLExecDirect( hstmt, stmt[i], SQL_NTS ) ;
       CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
       i++ ;
    }
```

ストアード・プロシージャーの使用

アプリケーションは 2 つの部分で稼働するよう設計することができます。つまりクライアントで 1 つ、サーバーでもう 1 つです。ストアード・プロシージャーは、アプリケーションと同じトランザクション内のデータベースで稼働する部分です。ストアード・プロシージャーは、組み込み SQL で、または DB2 CLI 関数を使用して作成することができます (138ページの『CLI でのストアード・プロシージャーの作成』を参照)。一般に、ストアード・プロシージャーには次の利点があります。

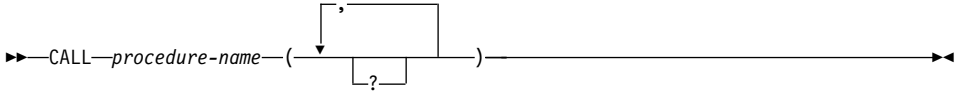
- 照会の長い順序列の中間結果の一部として、大量のデータがネットワークに転送されないようにします。
- クライアント・データベース・アプリケーションをクライアント / サーバーの部分に配置します。

さらに、組み込み静的 SQL で作成されたストアード・プロシージャーには、次の利点があります。

- パフォーマンス - 静的 SQL はプリコンパイル時に準備され、アクセス・プラン (パッケージ) 生成による実行時オーバーヘッドがありません。
- カプセル化 (情報隠蔽) - ユーザーはデータベース・オブジェクトにアクセスするためにその詳細について知る必要はありません。静的 SQL は、このカプセル化を強化するのに役立ちます。
- 機密保護 - ユーザーのアクセス特権はストアード・プロシージャーに関連づけられたパッケージ内でカプセル化されるので、各データベース・オブジェクトへの明示的なアクセス権を付与する必要はありません。たとえば、ユーザーが表から選択する特権を持っていないときに、その表からデータを選択するストアード・プロシージャーへの実行アクセスを付与してもらうことができます。

ストアード・プロシージャーの呼び出し

ストアード・プロシージャーを DB2 CLI アプリケーションから呼び出すには、次の CALL ステートメント構文を `SQLExecDirect()` に渡すか、または `SQLPrepare()` と `SQLExecute()` にこの順番で渡します。



注:

CALL ステートメントは動的には作成できませんが、 DB2 CLI は動的に作成できる場合と同様に CALL ステートメントを受け入れます。

ストアード・プロシージャーは、 153ページの『ストアード・プロシージャー呼び出し構文』にある ODBC ベンダー・エスケープ・シーケンスを使用して呼び出すこともできます。

procedure-name

ストアード・プロシージャー名を指定します。次のいずれかの形式にすることができます。

- *procedure-name*

  実行するプロシージャーの名前 (拡張子なし)。起動されるプロシージャーは、次のようにして判別されます。

  1. *procedure-name* は、ストアード・プロシージャーのライブラリーおよびそのライブラリー内の関数名として両方で使用されます。たとえば、*procedure-name* が `proclib` である場合、 DB2 サー

バーは proclib 名のストアード・プロシージャーのライブラリーをロードして、それからそのライブラリー内の関数ルーチン proclib() を実行します。

2. そのライブラリーまたは関数を見いだすことができなければ、*procedure-name* を使用して定義済みプロシージャーを検索し、合致するプロシージャーがないかどうかを調べます。合致するプロシージャーは以下のステップを使用して判別します。
   a. カタログ (SYSCAT.PROCEDURES) に入っているプロシージャーを検出します。そこでは、PROCNAME が、指定される *procedure-name* と合致します。 PROCSCHEMA は関数パス内のスキーマ名です。
   b. 次に、 CALL ステートメントで指定される引き数番号と同じパラメーター番号を持たないプロシージャーのうちのどれか 1 つを除去します。
   c. 残っているプロシージャーで、関数パスに最も早くからあるものを選択します。
   d. ステップ 2 の後、残っているプロシージャーがない場合には、エラー (SQLSTATE 42884) が返されます。

   プロシージャーが一度選択されると、DB2 は外部名で定義されるプロシージャーを起動します。

- *procedure-name!func-name*

  感嘆符を使用して、識別されるライブラリー名を procedure-name で指定し、実行される関数を func-name で指定します。この方法によって、類似する関数ルーチンを同じストアード・プロシージャー・ライブラリーに入れることができます。

- /u/db2user/procedure-name!func-name

  ストアード・プロシージャー・ライブラリーの名前が、全パス名として指定されます。実行される関数は func-name で指定します。

CALL ステートメントおよびストアード・プロシージャーの使用法に関する詳細については、 *SQL 解説書* およびアプリケーション開発の手引き を参照してください。

サーバーが DB2 ユニバーサル・データベース バージョン 2.1 以降、または DB2 (MVS/ESA 版) V4.1 以降の場合、 SQLProcedures() を呼び出して、データベースで利用できるストアード・プロシージャーをリストすることができます。

注: DB2 ユニバーサル・データベースの場合、 SQLProcedures() がすべてのプロシージャーを返さないこともありますが、アプリケーションは有効なプロシージャーであれば、 SQLProcedures() によって返されるかどうかにかかわらず使用することができます。詳細については、『ストアード・プロシージャーの登録』および 649ページの『SQLProcedures - プロシージャー名のリストを入手する』を参照してください。

CALL ステートメント構文図内の ? は、ストアード・プロシージャーの引き数に対応するパラメーター・マーカーを示しています。すべての引き数はパラメーター・マーカーを使用して渡さなければなりません。リテラル、NULL キーワード、および特殊レジスターは使用できません。しかし、ベンダーのエスケープ呼び出しステートメントを使用する場合、つまり呼び出しステートメントが中括弧 '{...}' で囲まれている場合には、リテラルを使用することができます。

CALL ステートメント内のパラメーター・マーカーは、 SQLBindParameter() によってアプリケーション変数にバインドされます。ストアード・プロシージャーの引き数は入力と出力の両方に使用できますが、クライアントとサーバーの間で不必要なデータが送信されないようにするために、アプリケーションで SQLBindParameter() の入力引き数のパラメーター・タイプを SQL_PARAM_INPUT に、出力引き数のパラメーター・タイプを SQL_PARAM_OUTPUT に指定する必要があります。入出力の両方の引き数には、パラメーター・タイプ SQL_PARAM_INPUT_OUTPUT があります。

サーバーが DB2 ユニバーサル・データベース バージョン 2.1 以降、または DB2 (MVS/ESA 版) V4.1 以降の場合、アプリケーションが SQLProcedureColumns() を呼び出して、プロシージャー呼び出しのパラメーターのタイプを判別します。詳細については、下記の『ストアード・プロシージャーの登録』および 640ページの『SQLProcedureColumns - プロシージャーの入力 / 出力情報を入手する』を参照してください。

ストアード・プロシージャーの登録

DB2 ユニバーサル・データベースの場合、ストアード・プロシージャーが (SYSCAT.PROCEDURES および SYSCAT.PROCPARMS 内にある) サーバーに登録してからでなければ、 SQLProcedures() および SQLProcedureColumns() を呼び出すことはできません。登録されていない場合、上記 2 つのカタログ関数呼び出しは空の結果セットを返します。ストアード・プロシージャーのサーバーでの登録の詳細については、 897ページの『付録H. ストアード・プロシージャー登録の疑似カタログ表』を参照してください。

ストアード・プロシージャーが DB2 (MVS/ESA 版) V4.1 以降のサーバーにある場合は、ストアード・プロシージャーの名前を SYSIBM.SYSPROCEDURES カタログ表に定義しなければなりません。 DB2 ユニバーサル・データベースで使用する疑似カタログ表は、 DB2 (MVS/ESA 版) SYSIBM.SYSPROCEDURES カタログ表から派生し拡張したものです。

ストアード・プロシージャーが DB2 ユニバーサル・データベース AS/400 用 V3.1 のサーバーにある場合は、アプリケーションは事前にそのプロシージャーの実際のパスおよび名前を認識していなければなりません。ストアード・プロシージャーまたはその引き数リストに関する情報を検索するための実際または疑似のカタログ表はありません。

ストアード・プロシージャー引き数の処理 (SQLDA)

多くの点で、ストアード・プロシージャーはその他のアプリケーションと似ていますが、 CLI (および組み込み SQL) で書かれたストアード・プロシージャーでは、ストアード・プロシージャー引き数が入っている SQLDA 構造に対して特別な考慮を払う必要があります。 SQLDA 構造については、*SQL 解説書* にその詳細が説明されています。

重要なこととして、SQLDA 構造内に格納されているすべてのデータは、 SQL データ・タイプとして格納されていること、そしてそれに応じた仕方でストアード・プロシージャーによって扱われなければならないことを理解しておかなければなりません。たとえば、

- ストリングは、ヌル終了しません。
- CHAR タイプには、ブランクが埋め込まれます。
- VARCHAR および LONG VARCHAR タイプには、定義された (最大の) 長さ、そして最初の 2 バイト (SQLCHAR 構造) に格納されている実際の長さの両方があります。
- DECIMAL (または NUMERIC) タイプは、パック 10 進数形式で格納されます。
- LOB または UDF タイプによって、 *doubled-SQLDA* の送信が引き起こされます。

推奨されている方法は、ストアード・プロシージャー向けの方法です。 SQLDA を解釈し、すべての入力引き数を項目上のホスト言語変数に、またホスト言語変数から出口上の SQLDA へ移動します。これにより、 SQLDA 特定コードをストアード・プロシージャー内でローカライズすることができます。

ストアード・プロシージャーから結果セットを返す

DB2 CLI には、ストアード・プロシージャーが、 1 つまたは複数の (おのおのが 1 つの照会に関連づけられている) カーソルがオープンされ、かつストアード・プロシージャーが終了してもカーソルがオープンされたままであるようにコーディングされている場合に、ストアード・プロシージャー呼び出しから 1 つまたは複数の結果セットを取り出すことができる機能が備わっています。2 つ以上のカーソルがオープンされたままであると、複数の結果セットが返されます。

CLI アプリケーション内での処理

DB2 CLI アプリケーションは、ストアード・プロシージャーを実行してカーソルをオープンした後、以下の方法で結果セットを検索することができます。

- ストアード・プロシージャーを呼び出す前に、ステートメント・ハンドルと関連したオープン・カーソルがないことを確認してください。
- 現時点までのストアード・プロシージャーに対する呼び出しでオープンしたカーソルが、すべてクローズしていることを確認してください。
- ストアード・プロシージャーを呼び出します。
- ストアード・プロシージャー CALL (呼び出し) ステートメントを実行することにより、結果セットに関連するカーソルを効率的にオープンすることができます。
- ストアード・プロシージャーによって返された出力パラメーターをすべて調べてください。たとえば、生成された結果セットを出力パラメーターで正確に示すようにプロシージャーが設計されています。
- すると、DB2 CLI アプリケーションは、通常の照会を処理するためにアプリケーションが利用できるすべての通常の関数を使用できます。アプリケーションが結果セットの特質や返される列数を知らない場合は、`SQLNumResultCols()`、 `SQLDescribeCol()` または `SQLColAttribute()` を呼び出します。そして次に、`SQLBindCol()`、`SQLFetch()`、および `SQLGetData()` の許される任意の組み合わせを使用して、結果セットのデータを得るよう指定することができます。
- `SQLFetch()` が SQL_NO_DATA_FOUND を返すか、またはアプリケーションが現行の結果セットの処理を終了すると、アプリケーションは `SQLMoreResults()` を呼び出して、検索すべき結果セットが他にないかどうかを判別することができます。 `SQLMoreResults()` を呼び出すと、現行のカーソルはクローズし、ストアード・プロシージャーによってオープンされている次のカーソルへと処理が進みます。
- 別の結果セットがあると `SQLMoreResults()` は success を返し、結果セットがなければ SQL_NO_DATA_FOUND が返されます。

- 結果セットは、アプリケーションによって順次に処理されます。

結果セットを返すためのストアード・プロシージャーのプログラミング

DB2 ユニバーサル・データベースのストアード・プロシージャーは、 1 つまたは複数の結果セットを CLI アプリケーションに返すために、以下の要件を満たす必要があります。

- ストアード・プロシージャーは FENCED モードで実行しなければなりません。このモードで実行していなければ、結果セットは返されず、エラーは生成されません。分離および非分離のストアード・プロシージャーに関する詳細については、 *アプリケーション開発の手引き* を参照してください。
- ストアード・プロシージャーは、結果セット上でカーソルを宣言し、結果セット上でカーソルをオープンし (すなわち、照会を実行する)、そしてストアード・プロシージャーが終了してもカーソルをオープンしたままにしておくことによって、結果セットが返されるように指示します。
- カーソルがオープンされたままであれば、結果セットはアプリケーションに返されます。
- 複数のカーソルがオープンされたままであれば、結果セットはストアード・プロシージャー内でカーソルがオープンされた順番に返されます。
- ストアード・プロシージャーが現行トランザクションをコミットすると、WITH HOLD 文節を使用して宣言されていないすべてのカーソルがクローズされます。
- ストアード・プロシージャーが現行トランザクションをロールバックすると、すべてのカーソルはクローズされます。
- ストアード・プロシージャーが SQL_CLOSE で `SQLFreeStmt()` を呼び出す場合、現行の結果セットのカーソルはクローズし、行はフラッシュされます。このことは、この同じストアード・プロシージャー呼び出しによって生成された他の結果セットと関連する他のカーソルすべてにも当てはまることに注意してください。
- まだ読み取られていない行だけが、元に返されます。たとえば、あるカーソルの結果セットには 500 行あり、ストアード・プロシージャー終了時までにストアード・プロシージャーが読み取ったのはそのうちの 150 行であったという場合、 151 行～ 500 行まではストアード・プロシージャーに返されます。これは、ストアード・プロシージャーが最初の数行に限ってフィルターをかけて、それらをアプリケーションに返さないことにしているときには役立ちます。

結果セットの戻りと照会ステートメントの実行の違い

一般には、結果セットを返すストアード・プロシージャーの呼び出しは、照会ステートメントの実行と同等です。以下の制約事項が適用されます。

- 静的照会ステートメントについては SQLDescribeCol() または SQLColAttribute() のどちらでも、列名は返されません。この場合、列の元の位置が代わりに返されます。
- LOB データ・タイプの長さの値は、常に最大長に設定されているか、LOBMAXCOLUMNSIZE キーワードが指定されていればその値に設定されます。
- 結果セットは、すべて読み取り専用です。
- カーソルはスクロール可能カーソルとしては使用できません。
- スキーマ関数 (SQLTables() など) を使用して結果セットを返すことはできません。スキーマ関数をストアード・プロシージャー内で使用すると、関連するステートメント・ハンドルのカーソルすべてを返す前にクローズしなければなりません。クローズしなければ、関係ない結果セットが返されることがあります。
- 照会を準備すると、実行する前に結果セットの列情報が使用できるようになります。ストアード・プロシージャーを準備すると、 CALL ステートメントを実行するまで結果セットの列情報は使用できません。

CLI でのストアード・プロシージャーの作成

組み込み SQL のストアード・プロシージャーには多くの利点がありますが、既存の DB2 CLI アプリケーションを所有しているアプリケーション開発者はアプリケーションの構成要素を移動してサーバー上で実行したい場合があります。アプリケーションのコードおよび論理に加えなければならない変更を最小限にするために、これらの構成要素を DB2 CLI を使って作成したストアード・プロシージャーとして組み込むことができます。

DB2 CLI 接続に関連するすべての内部情報は接続ハンドルによって参照され、ストアード・プロシージャーはクライアント・アプリケーションと同じ接続およびトランザクションで実行されるので、 DB2 CLI を用いて作成されるストアード・プロシージャーは、ヌル SQLConnect() の呼び出しを行って、接続ハンドルをクライアント・アプリケーションの基礎接続に関連付ける必要があります。ヌル 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。もちろん、SQLConnect() 呼び出しを完全に行えるためには、環境および接続ハンドルがすでに割り振られている必要があります。

注: 組み込み SQL を使用して作成されたストアード・プロシージャーは、DATETIME ISO オプションを使用してプリコンパイルし、 DB2 CLI が日時の値を正確に処理できるようにしなければなりません。

ストアード・プロシージャーの例

以下に示すのは、ストアード・プロシージャー、およびストアード・プロシージャーの呼び出しの例です。 (次の例は入力例です。出力例は outcli2.c、outsrv2.c のサンプルを参照してください。)

また、 DB2 には複数行の結果セットを返すストアード・プロシージャーを例示するいろいろなプログラム例があります (mrsp で始まる以下のプログラム例を参照してください。 mrspcli.c、mrspcli2.c、mrspcli3.sqc、clicall.c、mrspsrv.c および mrspsrv2.sqc)。

```
/* ... */
/************************************************************************
*
* PURPOSE: This sample program demonstrates stored procedures,
*          using CLI.  It is rewrite of the inpsrv.sqc embedded SQL
*          stored procedure.
*
*          There are two parts to this program:
*              - the inpcli2 executable (placed on the client)
*              - the inpsrv2 library (placed on the server)
*          CLI stored procedures can be called by either CLI or embbeded
*          applications.
*
*          The inpsrv function will take the information
*          received in the SQLDA to create a table and insert the
*          names of the presidents.
*
*          Refer to the inpcli2.c program for more details on how
*          this program is invoked as the inpsrv2 function
*          in the inpsrv2 library by the EXEC SQL CALL statement.
*
*          The SQL CALL statement will pass in 2 identical SQLDA
*          structures for input and output because all parameters
*          on the CALL statement are assummed to have both the
*          input and output attributes.  However, only changes
*          make to the data and indicator fields in the output SQLDA
*          will be returned to the client program.
*
* NOTE:    One technique to minimize network flow is to set the
*          variables that returns no output to null on the server program
*          before returning to the client program.
*          This can be achieved by setting the value -128 to the
*          indicator value associated with the data.
*
*          The sqleproc API will call the inpsrv routine stored
*          in the inpsrv library.
*
```

```
*          The inpsrv routine will take the information received
*          and create a table called "Presidents" in the "sample"
*          database.  It will then place the values it received in
*          the input SQLDA into the "Presidents" table.
*
******************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sqlda.h>
#include <sqlcli1.h>
#include "samputil.h"          /* Header file for CLI sample code */
int SQL_API_FN inpsrv2( struct sqlchar * input_data,
                        struct sqlda   * input_sqlda,
                        struct sqlda   * output_sqlda,
                        struct sqlca   * ca
                      ) {
   /* Declare a local SQLCA */
   struct sqlca sqlca ;
   SQLCHAR table_stmt[80] = "CREATE TABLE " ;
   SQLCHAR insert_stmt[80] = "INSERT INTO " ;
   SQLCHAR insert_data[21] ;
   SQLINTEGER insert_data_ind ;
   /* Delare Miscellanous Variables */
   int cntr ;
   char * table_name ;
   short table_name_length ;
   char * data_item[3] ;
   short data_item_length[3] ;
   int num_of_data = 0 ;
   /* Delare CLI Variables */
   SQLHANDLE henv, hdbc, hstmt ;
   SQLRETURN rc ;
   /*-----------------------------------------------------------------*/
   /* Assign the data from the SQLDA to local variables so that we    */
   /* don't have to refer to the SQLDA structure further.  This will  */
   /* provide better portability to other platforms such as DB2 MVS   */
   /* where they receive the parameter list differently.              */
   /* Note: Strings are not null-terminated in the SQLDA.             */
   /*-----------------------------------------------------------------*/
   table_name = input_sqlda->sqlvar[0].sqldata ;
   table_name_length = input_sqlda->sqlvar[0].sqllen ;
   num_of_data = input_sqlda->sqld - 1 ;
   for ( cntr = 0; cntr < num_of_data; cntr++ ) {
       data_item[cntr] = input_sqlda->sqlvar[cntr+1].sqldata ;
       data_item_length[cntr] = input_sqlda->sqlvar[cntr+1].sqllen ;
   }
   /*-----------------------------------------------------------------*/
   /* Setup CLI required environment                                  */
   /*-----------------------------------------------------------------*/
   SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv ) ;
   SQLAllocHandle( SQL_HANDLE_DBC, henv, &hdbc ) ;
   /*-----------------------------------------------------------------*/
   /* Issue NULL Connect, since in CLI we need a statement handle     */
   /* and thus a connection handle and environment handle.            */
```

```
/* A connection is not established, rather the current             */
/* connection from the calling application is used                 */
/*-----------------------------------------------------------------*/
SQLConnect( hdbc, NULL, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS ) ;
SQLAllocHandle( SQL_HANDLE_STMT, hdbc, &hstmt ) ;
/*-----------------------------------------------------------------*/
/* Create President Table                                          */
/* - For simplicity, we'll ignore any errors from the              */
/*   CREATE TABLE so that you can run this program even when the   */
/*   table already exists due to a previous run.                   */
/*-----------------------------------------------------------------*/
strncat( ( char * ) table_stmt,
         ( char * ) table_name,
         table_name_length
       ) ;
strcat( ( char * ) table_stmt, " (name CHAR(20))" ) ;
SQLExecDirect( hstmt, table_stmt, SQL_NTS ) ;
SQLFreeStmt( hstmt, SQL_RESET_PARAMS ) ;
/*-----------------------------------------------------------------*/
/* Generate and execute a PREPARE for an INSERT statement, and     */
/* then insert the three presidents.                               */
/*-----------------------------------------------------------------*/
strncat( ( char * ) insert_stmt,
         ( char * ) table_name,
         table_name_length
       ) ;
strcat( ( char * ) insert_stmt, "  VALUES (?)" ) ;
if ( SQLPrepare(hstmt, insert_stmt, SQL_NTS) != SQL_SUCCESS ) goto ext ;
/* Bind insert_data to parameter marker */
SQLBindParameter( hstmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_CHAR,
                  SQL_CHAR,
                  20,
                  0,
                  insert_data,
                  21,
                  &insert_data_ind
                ) ;
for ( cntr = 0; cntr < num_of_data; cntr++ ) {
    strncpy( ( char * ) insert_data,
             ( char * ) data_item[cntr],
             data_item_length[cntr]) ;
    insert_data_ind = data_item_length[cntr] ;
    if ( SQLExecute( hstmt ) != SQL_SUCCESS ) goto ext ;
}
/*-----------------------------------------------------------------*/
/* Return to caller                                                */
/*   -  Copy the SQLCA                                             */
/*   -  Update the output SQLDA.  Since there's no output to       */
/*      return, we are setting the indicator values to -128 to     */
/*      return only a null value.                                  */
/*   -  Commit or Rollback the inserts.                            */
/*-----------------------------------------------------------------*/
```

```
ext:
    rc = SQLGetSQLCA( henv, hdbc, hstmt, &sqlca ) ;
    if ( rc != SQL_SUCCESS ) printf( "RC = %d¥n", rc ) ;
    memcpy( ca, &sqlca, sizeof( sqlca ) ) ;
    if ( output_sqlda != NULL ) {
       for ( cntr = 0; cntr < output_sqlda->sqld; cntr++ ) {
           if ( output_sqlda->sqlvar[cntr].sqlind != NULL )
              *( output_sqlda->sqlvar[cntr].sqlind ) = -128 ;
       }
    }
    rc = SQLFreeHandle( SQL_HANDLE_STMT, hstmt ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    rc = SQLEndTran( SQL_HANDLE_DBC, hdbc, SQL_COMMIT ) ;
    CHECK_HANDLE( SQL_HANDLE_DBC, hdbc, rc ) ;
    printf( ">Disconnecting .....¥n" ) ;
    rc = SQLDisconnect( hdbc ) ;
    CHECK_HANDLE( SQL_HANDLE_DBC, hdbc, rc ) ;
    rc = SQLFreeHandle( SQL_HANDLE_DBC, hdbc ) ;
    CHECK_HANDLE( SQL_HANDLE_DBC, hdbc, rc ) ;
    rc = SQLFreeHandle( SQL_HANDLE_ENV,  henv ) ;
    if ( rc != SQL_SUCCESS )
        return( terminate( henv, rc ) ) ;
    return( SQL_SUCCESS ) ;
}
/* ... */
    SQLCHAR * stmt = "CALL inpsrv2(?, ?, ?, ?)" ;
/* ... */
    rc = SQLPrepare( hstmt, stmt, SQL_NTS ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    rc = SQLBindParameter( hstmt,
                           1,
                           SQL_PARAM_INPUT,
                           SQL_C_CHAR,
                           SQL_CHAR,
                           9,
                           0,
                           Tab_Name,
                           10,
                           NULL
                         ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    rc = SQLBindParameter( hstmt,
                           2,
                           SQL_PARAM_INPUT,
                           SQL_C_CHAR,
                           SQL_CHAR,
                           10,
                           0,
                           Pres_Name[0],
                           11,
                           NULL
                         ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    rc = SQLBindParameter( hstmt,
                           3,
```

```
                        SQL_PARAM_INPUT,
                        SQL_C_CHAR,
                        SQL_CHAR,
                        10,
                        0,
                        Pres_Name[1],
                        11,
                        NULL
                      ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
rc = SQLBindParameter( hstmt,
                        4,
                        SQL_PARAM_INPUT,
                        SQL_C_CHAR,
                        SQL_CHAR,
                        10,
                        0,
                        Pres_Name[2],
                        11,
                        NULL
                      ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
rc = SQLExecute( hstmt ) ;
/* Ignore Warnings */
if ( rc != SQL_SUCCESS_WITH_INFO )
   CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
```

組み込み SQL と DB2 CLI の混合

アプリケーションが、組み込み静的 SQL と組み合わせて DB2 CLI を使用することは可能であり、かつ望ましいことです。アプリケーション開発者が、DB2 CLI カタログ関数の利点を活用し、パフォーマンスが重要になるアプリケーション処理の部分を最大化したいというシナリオを考えてみてください。DB2 CLI と組み込み SQL を混合して使用するには、アプリケーションが次の規則に従っていなければなりません。

- 接続管理およびトランザクション管理はすべて、 DB2 CLI または組み込み SQL のいずれかを使用して完全に実行する必要があります。 DB2 CLI アプリケーションがすべての接続およびコミット / ロールバックを実行し、組み込み SQL を使用して作成された関数を呼び出すか、組み込み SQL アプリケーションがすべての接続およびコミット / ロールバックを実行し、ヌル接続を使用する DB2 CLI 内で作成された関数を呼び出すかのいずれかを行います (ヌル接続の詳細は、 138ページの『CLI でのストアード・プロシージャーの作成』を参照してください)。
- 同一ステートメントでは、照会ステートメント処理が DB2 CLI および組み込み SQL のインターフェースの両方に関係していてはなりませんし、またそうすることはできません。たとえば、アプリケーションが組み込み SQL

ルーチンでカーソルをオープンしてから、 DB2 CLI SQLFetch() 関数を呼び出して行データを取り出すことはできません。

DB2 CLI では複数接続ができるので、組み込み SQL で作成されたルーチンに対する関数呼び出しを行う前に、 SQLSetConnection() 関数を呼び出さなくてはなりません。このことを行うと、アプリケーションは、組み込み SQL ルーチンが処理を実行する必要のある接続を明示指定することができます。

DB2 CLI アプリケーションがマルチスレッドにされ、また組み込み SQL 呼び出しまたは DB2 の API 呼び出しを作成する場合、各スレッドは 1 つの DB2 コンテキストを持つ必要があります。詳細な説明については、50ページの『マルチスレッドのアプリケーション作成』を参照してください。

組み込み SQL と DB2 CLI の混合の例

次の例では、2 つのデータ・ソースに接続し、 DB2 CLI を使用して組み込み SQL と動的 SQL の両方を実行しているアプリケーションを示しています。

```
/* ... */
    /* allocate an environment handle   */
    SQLAllocEnv(&henv);
    /* Connect to first data source */
    DBconnect(henv, &hdbc[0]);
    /* Connect to second data source */
    DBconnect(henv, &hdbc[1]);
    /*********   Start Processing Step  ************************/
    /* NOTE: at this point there are two active connections */

    /* set current connection to the first database */
    if ( (rc = SQLSetConnection(hdbc[0])) != SQL_SUCCESS )
         printf("Error setting connection 1¥n");
    /* call function that contains embedded SQL */
    if ((rc = Create_Tab() ) != 0)
         printf("Error Creating Table on 1st connection, RC=%d¥n", rc);
   /*  Commit transation on connection 1 */
    SQLTransact(henv, hdbc[0], SQL_COMMIT);
    /* set current connection to the second database */
    if ( (rc = SQLSetConnection(hdbc[1])) != SQL_SUCCESS )
         printf("Error setting connection 2¥n");
    /* call function that contains embedded SQL */
    if ((rc = Create_Tab() ) != 0)
         printf("Error Creating Table on 2nd connection, RC=%d¥n", rc);
   /*  Commit transation on connection 2 */
    SQLTransact(henv, hdbc[1], SQL_COMMIT);
    /* Pause to allow the existance of the tables to be verified. */
    printf("Tables created, hit Return to continue¥n");
    getchar();
    SQLSetConnection(hdbc[0]);
    if (( rc = Drop_Tab() ) != 0)
        printf("Error dropping Table on 1st connection, RC=%d¥n", rc);
   /*  Commit transation on connection 1 */
```

```
    SQLTransact(henv, hdbc[0], SQL_COMMIT);
    SQLSetConnection(hdbc[1]);
    if (( rc = Drop_Tab() ) != 0)
        printf("Error dropping Table on 2nd connection, RC=%d¥n", rc);
   /*  Commit transation on connection 2 */
    SQLTransact(henv, hdbc[1], SQL_COMMIT);
    printf("Tables dropped¥n");
    /*********   End Processing Step  **************************/
/* ... */
/*************   Embedded SQL Functions  ******************************
** This would normally be a seperate file to avoid having to           *
** keep precompiling the embedded file in order to compile the DB2 CLI *
** section.                                                            *
************************************************************************/
#include "sql.h"
#include "sqlenv.h"
EXEC SQL INCLUDE SQLCA;
int
Create_Tab( )
{

   EXEC SQL CREATE TABLE mixedup
            (ID INTEGER, NAME CHAR(10));
   return( SQLCODE);
}
int
Drop_Tab( )
{
   EXEC SQL DROP TABLE mixedup;
   return( SQLCODE);
}
```

CLI の非同期実行

DB2 CLI は、非同期に関数のサブセットを実行可能です。DB2 CLI ドライバーは、関数を呼び出した後アプリケーションに制御を戻しますが、その関数が実行を完了する前にこのことが行われます。実行が完了するまでは、関数は呼び出されるたびに SQL_STILL_EXECUTING を戻します。完了の時点では、別の値 (たとえば、SQL_SUCCESS) を戻します。

非同期実行は、単一スレッドのオペレーティング・システムでのみ有益です。マルチスレッドのオペレーティング・システムで実行するアプリケーションは、その代わりに、分離スレッドで関数を実行すべきです。

非同期実行は、通常要求をサーバーに送信し、そして応答を待つ関数で可能となります。非同期に実行している関数は、待つのではなく、アプリケーションに制御を戻します。その後、アプリケーションは他のタスクを実行したり、オペレーティング・システムに制御を戻すこともできます。そして、

SQL_STILL_EXECUTING 以外の戻りコードが返されるまで、割り込みを用いてその関数を繰り返しポーリングします。

典型的な非同期アプリケーション

非同期に関数を実行する各アプリケーションは、通常の CLI ステップに加えて以下のステップをその順序に従って完了する必要があります。

1. 環境のセットアップ

関数を非同期に確実に呼び出すには、アプリケーションは SQL_ASYNC_MODE のオプションを指定した SQLGetInfo() を呼び出します。

```
/* See what type of Asynchronous support is available. */
rc = SQLGetInfo( hdbc, /* Connection handle */
                 SQL_ASYNC_MODE, /* Query the support available */
                 &ubuffer, /* Store the result in this variable */
                 4,
                 &outlen);
```

SQLGetInfo() への呼び出しでは、以下の値のうちの 1 つが戻されます。

SQL_AM_STATEMENT - ステートメント・レベル

非同期実行をステートメント・レベルでオン / オフできるということを示します。

ステートメント・レベルの非同期実行は、ステートメント属性 SQL_ATTR_ASYNC_ENABLE を使用して設定されます。アプリケーションは、多くて 1 つの活動状態の関数を任意の 1 つの接続において非同期モードで実行させることができます。 SQLSetStmtAttr() を使用して、SQL_ASYNC_ENABLE_ON に設定してください。

```
/* Set statement level asynchronous execution on */
rc = SQLSetStmtAttr( hstmt, /* Statement handle */
                     SQL_ATTR_ASYNC_ENABLE,
                     (SQLPOINTER) SQL_ASYNC_ENABLE_ON,
                     0);
```

SQL_AM_CONNECTION - 接続レベル

DB2 ユニバーサル・データベースは SQL_AM_STATEMENT をサポートしますが、その他のデータ・ソースにより SQL_AM_CONNECTION が戻される可能性があります。接続におけるすべてのステートメントが、同じ方法で実行されることを示します。

接続レベルの非同期実行は、接続属性 SQL_ATTR_ASYNC_ENABLE を使用して設定されます。 SQLSetConnectAttr() を使用して、SQL_ASYNC_ENABLE_ON に設定してください。

すでに割り当てられているすべてのステートメントは、この接続にこれから割り当てられるステートメント・ハンドルと同様に、非同期実行で使用可能です。

SQL_AM_NONE - 非同期実行がサポートされない

これは次の 2 つのいずれかの理由で戻されます。

1. データ・ソースそのものが、非同期実行をサポートしない。
2. DB2 CLI/ODBC の構成キーワード ASYNCENABLE が、非同期実行できないように明確に設定されている。詳細については、177ページの『ASYNCENABLE』を参照してください。

どちらの場合でも、関数は同期的に実行されます。アプリケーションが、 SQLSetStmtAttr() または SQLSetConnectAttr() を呼び出し、非同期実行をオンにする場合、 SQLSTATE として 01S02 (オプション値が変更された) が戻されます。

2. 非同期実行をサポートする関数の呼び出し

アプリケーションが非同期に実行される関数を呼び出すとき、次の 2 つの事柄のいずれかが生じます。

- 関数が非同期実行することで利点とならない場合、DB2 CLI は同期的に関数を実行し、かつ通常の戻りコード (SQL_STILL_EXECUTING 以外) を戻すようにします。

  この場合、アプリケーションは非同期モードが使用可能にされていないかのように実行します。
- DB2 CLI は最小限度の処理 (たとえば、引き数を検査してエラーを調べる) を行い、それから、サーバーにステートメントを渡します。このクィック処理がいったん完了したら、 SQL_STILL_EXECUTING の戻りコードをアプリケーションに戻します。

SQLSetStmtAttr() 関数の SQL_ATTR_ASYNC_ENABLE ステートメント属性を検査し、非同期に実行される関数のリストがあるかどうかを調べます。

以下の例では、両方の考えられる結果を考慮に入れた共通の while ループを例示しています。

```
while ( (rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS) ) == SQL_STILL_EXECUTING)
{
    /* Other processing can be performed here, between each call to
     * see if SQLExecDirect() has finished running asynchronously.
     * This section will never run if CLI runs the function
     * synchronously.
```

```
    */
}
/* The application continues at this point when SQLExecDirect() */
/* has finished running. */
```

3. 他の関数呼び出し間の非同期関数のポーリング

アプリケーションは、最初に関数を呼び出すために使用したのと同じ引き数を指定して繰り返し呼び出すことで、関数が終了したかどうかを判別します。戻りコード SQL_STILL_EXECUTING は、関数がまだ終了していないことを表し、その他の値はすでに終了したことを示します。 SQL_STILL_EXECUTING 以外の値は、関数が同期的に実行した場合に戻されるのと同じ戻りコードです。

非同期実行の間に呼び出せる関数

以下の関数は、ある関数が非同期に実行されている間に呼び出し可能です。他のすべての関数は、SQLSTATE として HY010 (関数順序エラー) を戻します。

- 同じ接続内の他のステートメント上のすべての関数
- 非同期ステートメントに関連する関数以外の接続上のすべての関数
- 停止するための非同期ステートメント上にある SQLCancel() (149ページの『5. 非同期関数呼び出しの取り消し』を参照)。
- レコードの診断フィールドではなく、ヘッダーの診断フィールドを得るための非同期ステートメント上にある SQLGetDiagField() および SQLGetDiagRec() (『4. 実行中の診断情報』を参照)。
- ステートメント・ハンドルを割り振るため、非同期ステートメントに関連した接続上にある SQLAllocHandle()。

4. 実行中の診断情報

SQLGetDiagField() が非同期関数を実行しているステートメント・ハンドル上に呼び出されるとき、以下の値が戻されます。

- SQL_DIAG_CURSOR_ROW_COUNT、SQL_DIAG_DYNAMIC_FUNCTION、SQL_DIAG_DYNAMIC_FUNCTION_CODE、および SQL_DIAG_ROW_COUNT ヘッダー・フィールドの値は未定義。
- SQL_DIAG_NUMBER ヘッダー・フィールドは 0 を戻す。
- SQL_DIAG_RETURN_CODE ヘッダー・フィールドは、SQL_STILL_EXECUTING を戻す。
- すべてのレコード・フィールドは SQL_NO_DATA を戻す。

SQLGetDiagRec() は、非同期関数を実行しているステートメント・ハンドル上に呼び出されるとき、常に SQL_NO_DATA を戻します。

5. 非同期関数呼び出しの取り消し

アプリケーションは、 SQLCancel() を呼び出して非同期に実行しているすべての関数の取り消し要求を発行することができます。しかし、場合によってはこの要求が実行されないこともあります (たとえば、関数がすでに終了している場合)。

SQLCancel() 呼び出しからの戻りコードは、取り消し要求が受信されたかどうかを示します。非同期関数の実行が停止したかどうかを示すのではありません。

関数が取り消されたかどうかを知らせる唯一の方法は、元の引き数を使用して、再度関数を呼び出すことです。

- 取り消しが正常に終了した場合、関数は SQL_ERROR と、SQLSTATE の HY008 (操作が取り消された) を戻します。
- 取り消しが正常に終了しなかった場合、関数は SQL_ERROR 以外の値を戻すか、または SQL_ERROR と HY008 以外の SQLSTATE を戻すかのどちらかです。

サンプルの非同期アプリケーション

以下の CLI サンプル async.c は、非同期に SQLExecDirect() を実行する簡単なアプリケーションを例示しています。これは、CLI のサンプル・プログラム fetch.c に基づいています。

```
/* ... */
    /* Make the result from SQLGetInfo() more meaningful by mapping */
    /* the returned value to the string. */
    static char ASYNCMODE[][19] =  {  "SQL_AM_NONE",
                                      "SQL_AM_CONNECTION",
                                      "SQL_AM_STATEMENT" };
/* ... */
     * See what type of Asynchronous support is available,
     * and whether or not the CLI/ODBC configuration keyword ASYNCENABLE
     * is set on or off.
     */
    rc = SQLGetInfo( hdbc, /* Connection handle */
                     SQL_ASYNC_MODE, /* Query the support available */
                     &ubuffer, /* Store the result in this variable */
                     4,
                     &outlen);
    CHECK_STMT(hstmt, rc);
    printf("SQL_ASYNC_MODE value from SQLGetInfo() is %s.¥n¥n",
            ASYNCMODE[ubuffer]);
    if (ubuffer == SQL_AM_NONE ) { /* Async not supported */
```

```
      printf("Asynchronous execution is not supported by this datasource¥n");
      printf("or has been turned off by the CLI/ODBC configuration keyword¥n");
      printf("ASYNCENABLE. The application will continue, but¥n");
      printf("SQLExecDirect() will not be run asynchronously.¥n¥n");
      /* There is no need to set the SQLSetStmtAttr() option */
   } else {
      /* Set statement level asynchronous execution on */
      rc = SQLSetStmtAttr(
               hstmt,
               SQL_ATTR_ASYNC_ENABLE,
               (SQLPOINTER) SQL_ASYNC_ENABLE_ON,
               0);
      CHECK_STMT(hstmt, rc);
   }
   /* The while loop is new for the asynchronous sample, the */
   /* SQLExecDirect() call remains the same. */
   while ((rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS) ) == SQL_STILL_EXECUTING) {
      printf("   ...SQLExecDirect() still executing asynchronously...¥n");
      /* Other processing can be performed here, between each call
       * to see if SQLExecDirect() has finished running asynchronously.
       * This section will never run if CLI runs the function
       * synchronously.
       */
   }
   CHECK_STMT(hstmt, rc);
   rc = SQLBindCol(hstmt, 1, SQL_C_CHAR, (SQLPOINTER) deptname.s, 15,
                   &deptname.ind);
   CHECK_STMT(hstmt, rc);
   rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER) location.s, 15,
                   &location.ind);
   CHECK_STMT(hstmt, rc);
   printf("Departments in Eastern division:¥n");
   printf("DEPTNAME       Location¥n");
   printf("-------------- ------------¥n");
   while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
       printf("%-14.14s %-14.14s ¥n", deptname.s, location.s);
   }
   if (rc != SQL_NO_DATA_FOUND)
       check_error(henv, hdbc, hstmt, rc, __LINE__, __FILE__);
   rc = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
   CHECK_STMT(hstmt, rc);
   rc = SQLEndTran(SQL_HANDLE_ENV, henv, SQL_COMMIT);
   CHECK_DBC(hdbc, rc);
   printf("Disconnecting .....¥n");
   rc = SQLDisconnect(hdbc);
   CHECK_DBC(hdbc, rc);
   rc = SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
   CHECK_DBC(hdbc, rc);
   rc = SQLFreeHandle(SQL_HANDLE_ENV, henv);
   if (rc != SQL_SUCCESS)
       return (terminate(henv, rc));
}                              /* end main */
```

ベンダー・エスケープ文節の使用

X/Open SQL CAE 仕様では、**エスケープ文節**を、『ベンダー固有の SQL 拡張機能を、標準化された SQL の枠組みの中で実装するための構文上の機構』として定義しています。 DB2 CLI と ODBC の両方とも、X/Open で定義されているベンダー・エスケープ文節をサポートしています。

現在では、エスケープ文節は、SQL 拡張を定義するために ODBC によって広く使用されています。 DB2 CLI は、ODBC 拡張を正しい DB2 構文に変換します。 SQLNativeSql() 関数を使用して、その結果の構文を表示することができます。

アプリケーションが DB2 データ・ソースだけにアクセスする場合は、エスケープ文節を使用する必要はありません。アプリケーションが同じサポートを備えている他のデータ・ソースにアクセスしようとする際に、別の構文を使用していれば、エスケープ文節を使うとアプリケーションの可搬性が高くなります。

DB2 CLI は、エスケープ文節に標準構文と短縮構文の両方を使用してきましたが、標準構文は (DB2 CLI はサポートはしていますが) 使用すべきでないものとされています。標準構文を使用したエスケープ文節は、次の形式を取っていました。

```
--(*vendor(vendor-identifier),
      product(product-identifier) extended SQL text*)--
```

アプリケーションでは、現在では短縮構文 (以下に説明しています) だけを使用してください。最新の ODBC 標準に付いていくためです。

エスケープ文節の構文

エスケープ文節の定義の形式は次のとおりです。

```
{ extended SQL text }
```

これによって、以下の SQL 拡張子を定義します。

- 拡張された日付、時刻、タイム・スタンプのデータ
- 外部結合
- LIKE 述部
- ストアード・プロシージャーの呼び出し
- 拡張されたスカラー関数
  - 数値関数
  - ストリング関数

– システム関数

ODBC 日付、時刻、タイム・スタンプのデータ

日付、時刻、およびタイム・スタンプのデータの ODBC エスケープ文節は、次のとおりです。

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

d は、 *value* が *yyyy-mm-dd* 形式の日付であることを示します。
t は、 *value* が *hh:mm:ss* 形式の時刻であることを示します。
ts は、 *value* が *yyyy-mm-dd hh:mm:ss[.f...]* 形式のタイム・スタンプであることを示します。

たとえば、次のステートメントを使用して、 **EMPLOYEE** 表に対する照会を発行することができます。

```
SELECT * FROM EMPLOYEE WHERE HIREDATE={d '1994-03-29'}
```

DB2 CLI は、上記ステートメントを DB2 形式に変換します。 SQLNativeSql() を使用して、変換されたステートメントを返すことができます。

日付、時刻、およびタイム・スタンプのリテラルの ODBC エスケープ文節は、 C データ・タイプの SQL_C_CHAR を指定した入力パラメーターで使用することができます。

ODBC 外部結合構文

外部結合の ODBC エスケープ文節は、次のとおりです。

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

outer join は次のとおりです。

```
table-name  {LEFT | RIGHT | FULL} OUTER JOIN
            {table-name | outer-join}
            ON search-condition
```

たとえば、DB2 CLI が次のステートメントを変換する場合を考慮します。

```
SELECT * FROM {oj T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C3}
    WHERE T1.C2>20
```

これは IBM の形式に変換され、その形式は SQL92 外部結合構文に対応します。

```
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C3 WHERE T1.C2>20
```

注: すべての DB2 サーバーで外部結合がサポートされているわけではありません。現行サーバーが外部結合をサポートしているかどうかを判別するには、 SQL_SQL92_RELATIONAL_JOIN_OPERATORS および SQL_OJ_CAPABILITIES オプションを指定して、SQLGetInfo() を呼び出します。

LIKE 述部エスケープ文節

SQL LIKE 述部では、メタキャラクター % がゼロ個以上の任意の文字に相当し、メタキャラクター _ が任意の 1 文字に相当します。 ESCAPE 文節を利用すると、実際のパーセント文字および下線文字を含む値に相当するようにパターンの定義を行うことができ、この場合はその文字の前にエスケープ文字を入れます。 LIKE 述部のエスケープ文字を定義するのに ODBC が使用するエスケープ文節は、次のとおりです。

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

escape-character は、エスケープ文節の使用の基準となる DB2 規則でサポートされている任意の文字です。

さまざまなベンダー DBMS 製品間の可搬性には関係のないアプリケーションの場合、 ESCAPE 文節を直接そのデータ・ソースへ渡す必要があります。特定の DB2 データ・ソースで LIKE 述部エスケープ文字がサポートされる時点を判別するには、アプリケーションで SQL_LIKE_ESCAPE_CLAUSE 情報タイプを指定して SQLGetInfo() を呼び出してください。

ストアード・プロシージャー呼び出し構文

ストアード・プロシージャーを呼び出す場合の ODBC エスケープ文節は、次のとおりです。

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

- *procedure-name* は、データ・ソースに保管されているプロシージャーの名前を指定します。
- *parameter* は、プロシージャー・パラメーターを指定します。

プロシージャーにはゼロ個以上のパラメーターがあります。 (角大括弧 ([ ]) は、オプションの引き数を表します。)

ODBC は、任意指定パラメーター **?=** がプロシージャーの戻り値を表すように指定します。戻り値があれば、SQLBindParameter() によって定義される最初のパラメーター・マーカーによって指定された場所に保管されます。 **?=** がエスケープ文節にあると、 DB2 CLI は SQLCODE をプロシージャーの戻り値と

して戻します。 **?=** がない場合、アプリケーションは SQLGetSQLCA() 関数を使用して SQLCA を検索することができます。 ODBC とは異なり、DB2 CLI はプロシージャー引き数としてリテラルをサポートしてはいないため、パラメーター・マーカーを使用します。

ストアード・プロシージャーに関する詳細については、 131ページの『ストアード・プロシージャーの使用』またはアプリケーション*開発の手引き* を参照してください。

たとえば、DB2 CLI が次のステートメントを変換する場合を考慮します。

```
{CALL NETB94(?,?,?)}
```

これは、次の内部 CALL ステートメント形式に変換されます。

```
CALL NEBT94(?, ?, ?)
```

ODBC スカラー関数

ストリングの長さ、サブストリング、またはトリムなどのスカラー関数を、結果セットの列や、結果セットの行を制限する列で使用することができます。スカラー関数の ODBC エスケープ文節は次のとおりです。

{fn *scalar-function***}**

ここで、 *scalar-function* は 841ページの『付録D. 拡張スカラー関数』にリストされている関数です。

たとえば、DB2 CLI が次のステートメントを変換する場合を考慮します。

```
SELECT {fn CONCAT(FIRSTNAME,LASTNAME)} FROM EMPLOYEE
```

これを次のように変換します。

```
SELECT FIRSTNAME CONCAT LASTNAME FROM EMPLOYEE
```

SQLNativeSql() を呼び出して、変換された SQL ステートメントを得ることができます。

どのスカラー関数が、特定の接続ハンドルで参照される現行サーバーによってサポートされているかを判別するには、 SQLGetInfo() を、SQL_NUMERIC_FUNCTIONS、SQL_STRING_FUNCTIONS、SQL_SYSTEM_FUNCTIONS、および SQL_TIMEDATE_FUNCTIONS オプションを指定して呼び出してください。

第4章 CLI の構成およびサンプル・アプリケーションの実行

DB2 CLI の実行時環境はどの DB2 クライアント・アプリケーション・イネーブラー製品にも含まれています。各プラットフォームの開発サポートは、対応する DB2 のソフトウェア開発キット (DB2 SDK) で提供されています。これは、別売の DB2 アプリケーション開発キット製品の一部です。

たとえば、OS/2 アプリケーションは、DB2 アプリケーション開発クライアントを使用して開発され、 DB2 クライアント・アプリケーション・イネーブラー (OS/2 版) を使用して DB2 サーバーに対して実行することができます。

DB2 CLI 実行時環境の設定

DB2 CLI アプリケーション用の実行時サポートは、 DB2 クライアント・アプリケーション・イネーブラーおよび DB2 アプリケーション開発クライアントを含むすべての DB2 ユニバーサル・データベース製品に備えられています。このセクションでは、必要とされる一般的なセットアップについて説明します。関連情報として、158ページの『CLI/ODBC アクセスのためのプラットフォーム固有の詳細情報』も参照してください。

DB2 CLI アプリケーションが正常に DB2 データベースにアクセスするためには、次のことが必要です。

1. データベース (データベースがリモートの場合はノード) をカタログ化しなければなりません。プラットフォームにはコマンド行プロセッサーまたは (適用できれば) DB2 管理ツールのいずれかを使用してください。
2. DB2 CLI バインド・ファイルをデータベースにバインドしなければなりません。

   ユーザーに適切な権限があると、データベースに最初にアクセスする際に DB2 CLI は自動バインドします。データベース管理者は、最初の接続を実行するか、またはファイルを明示的にバインドする必要があります。詳細については、166ページの『DB2 CLI/ODBC ドライバーをデータベースにバインドする方法』を参照してください。
3. 任意選択で、DB2 CLI/ODBC の構成キーワードを設定できます。

   ご使用のプラットフォームで使用可能なツールを使用して行う方法の詳細については 158ページの『CLI/ODBC アクセスのためのプラットフォーム固

有の詳細情報』を、さらに手動で行う場合の詳細は、167ページの『CLI/ODBC 構成キーワードの設定方法』を参照してください。

CLI/ODBC プログラムの実行

DB2 コール・レベル・インターフェース (CLI) 実行時環境および DB2 CLI/ODBC ドライバーは、インストール時にオプションの構成要素として DB2 クライアントに組み込まれます。

このサポートを使用することにより、ODBC および DB2 CLI API を使用して開発されるアプリケーションは、任意の DB2 サーバーで使用できるようになります。 DB2 CLI アプリケーション開発サポートは、 DB2 サーバーと一緒にパッケージされている DB2 アプリケーション開発クライアントで提供されています。

DB2 CLI または ODBC アプリケーションを DB2 にアクセスさせるには、まず DB2 CLI パッケージをサーバーにバインドする必要があります。ユーザーがパッケージのバインドに必要な権限を持っていれば、バインド処理は初回の接続時に自動的に行われますが、管理者が最初に、サーバーにアクセスするそれぞれのプラットフォームのそれぞれのバージョンとのバインド操作を実行するようお勧めします。 具体的な説明については、166ページの『DB2 CLI/ODBC ドライバーをデータベースにバインドする方法』を参照してください。

DB2 CLI と ODBC アプリケーションに DB2 データベースへのアクセス権を与えるには、 クライアント・システム上で、以下のような一般的な手順が必要です。これらの手順は、有効なユーザー ID とパスワードを使用して正常に DB2 に接続したことを前提としています。プラットフォームによって異なりますが、これらのステップの多くは自動処理で行われます。すべての説明を含む詳細については、ご使用のプラットフォームを特に扱っているセクションを参照してください。

ステップ 1. クライアント構成アシスタント (CCA) を使って、データベースを追加 (クライアントとサーバーのマシンが別々な場合) し、そのインスタンスとデータベースがコントロール・センターに認識されるようにしてから、そのシステムにインスタンスとデータベースを追加します。 このプログラムにアクセスできない場合、コマンド行プロセッサーで **catalog** コマンドを使用できます。

ステップ 2. DB2 CLI/ODBC ドライバーは、 Windows プラットフォームにおける DB2 クライアントのインストールでは、オプションの構成要素です。この時点でこれが選択されていることを確認してくだ

さい。 OS/2 では、**「ODBC ドライバーのインストール (Install ODBC Driver)」**アイコンを使って、 DB2 CLI/ODBC ドライバーと ODBC ドライバー・マネージャーの両方をインストールする必要があります。 UNIX プラットフォームでは、 DB2 CLI/ODBC ドライバーはクライアントと共に自動的にインストールされます。

ステップ 3. ODBC から DB2 データベースにアクセスするには、次のようにします。

a. ODBC ドライバー・マネージャー (Microsoft または他社) は、すでにインストールされていなければなりません (これは、 32 ビット Windows システム上では、DB2 のインストール時に省略時で完了)。

b. DB2 データベースは、ODBC データ・ソースとして登録されている必要があります。 ODBC ドライバー・マネージャーは、DB2 カタログ情報を読み取らないで、代わりに独自のデータ・ソースのリストを参照します。

c. DB2 表の索引が固有でない場合には、 ODBC アプリケーションの多くは読み取り専用としてオープンします。 ODBC アプリケーションによって更新される、それぞれの DB2 表の固有索引が作成されていなければなりません。 *SQL 解説書* の **CREATE INDEX** ステートメントを参照してください。コントロール・センターを使用して、表の設定値を変更してから、「**基本キー (Primary Key)**」タブをクリックし、 1 つ以上の列を使用可能な列リストから基本キー列リストに移動します。基本キーの一部として選択する任意の列は、NOT NULL に定義してください。

ステップ 4. 必要な場合には、各種の CLI/ODBC 構成キーワードを変更して、DB2 CLI/ODBC およびこれを使用するアプリケーションの動作を修正できます。

上記のステップに従って ODBC サポートをインストールし、 DB2 データベースを ODBC データ・ソースとして追加すると、 ODBC アプリケーションはこの時点でその製品にアクセスできるようになります。

プラットフォーム特有の指示に従うと、以下のトピックに関するより多くの詳細情報を得ることができます。

- 166ページの『DB2 CLI/ODBC ドライバーをデータベースにバインドする方法』
- 167ページの『CLI/ODBC 構成キーワードの設定方法』

- 168ページの『db2cli.ini の構成』

CLI/ODBC アクセスのためのプラットフォーム固有の詳細情報

DB2 CLI と ODBC アプリケーションに DB2 へのアクセス権を与える方法についてのプラットフォーム固有の詳細情報は、以下のカテゴリーに分類されます。

- 『CLI/ODBC を使用した DB2 への Windows 32 ビット オペレーティング・システム クライアント・アクセス』
- 161ページの『CLI/ODBC を使用した DB2 への OS/2 クライアント・アクセス』
- 163ページの『CLI/ODBC を使用した DB2 への UNIX クライアント・アクセス』

CLI/ODBC を使用した DB2 への Windows 32 ビット オペレーティング・システム クライアント・アクセス

Windows クライアントから DB2 CLI および ODBC アプリケーションが正常に DB2 データベースにアクセスできるようにするには、クライアント・システムで以下のステップを実行します。

ステップ 1. DB2 データベース (データベースがリモートの場合はノード) をカタログ化しなければなりません。この処理を実行するには、CCA (またはコマンド行プロセッサー) を使用してください。

詳細については、 CCA のオンライン・ヘルプ (またはコマンド *解説書* の **CATALOG DATABASE** および **CATALOG NODE** コマンド) を参照してください。

ステップ 2. Microsoft ODBC ドライバー・マネージャーおよび DB2 CLI/ODBC ドライバーがインストールされたことを確認します。Windows 32 ビット オペレーティング・システムでは、インストール時に ODBC 構成要素が手操作で選択解除されない限り、これらの製品は両方とも DB2 と一緒にインストールされます。Microsoft ODBC Driver Manager の新しいバージョンが見つかった場合、DB2 はそれを上書きしません。

両方がマシン上に存在することを確認するには、次のように実行します。

a. コントロール パネルで「Microsoft ODBC データ・ソース (Microsoft ODBC Data Sources)」アイコンを開始するか、コマンド行から **odbcad32.exe** コマンドを実行します。

b. **「ドライバー (Drivers)」**タブをクリックします。

c. リストに「IBM DB2 ODBC ドライバー」が表示されているかどうかを検証します。

Microsoft ODBC ドライバー・マネージャーまたは IBM DB2 CLI/ODBC ドライバーがインストールされていない場合には、Windows 32 ビット オペレーティング・システムに DB2 インストールを再実行し、ODBC 構成要素を選択します。

ステップ 3. DB2 データベースをデータ・ソース として ODBC ドライバー・マネージャーに登録します。 Windows 32 ビット オペレーティング・システムでは、 データ・ソースをシステムのすべてのユーザーが使用できるようにするか (システム・データ・ソース)、現在のユーザーにのみ使用可能にすることができます (ユーザー・データ・ソース)。以下に示す方式のいずれかを使用して、データ・ソースを追加してください。

- CCA を使用して、
  a. データ・ソースとして追加したい DB2 データベース別名を選択します。
  b. **「プロパティー (Properties)」**押しボタンをクリックします。「データベース・プロパティー (Database Properties)」ウィンドウがオープンします。
  c. **「ODBC 用にこのデータベースを登録 (Register this database for ODBC)」**チェック・ボックスを選択します。
  d. Windows 32 ビット オペレーティング・システムでは、ラジオ・ボタンを使用してデータ・ソースをユーザーまたはシステム・データ・ソースとして追加することができます。
- **Microsoft32 ビット ODBC 管理ツール**を使用すると、コントロール パネルのアイコンからまたはコマンド行から **odbcad32.exe** を実行することでアクセスが可能となります。次のようにしてください。
  a. Windows 32 ビット オペレーティング・システムでは、デフォルトでユーザー・データ・ソースのリストが表示されます。システム・データ・ソースを追加する場合は、**「システム DSN (System DSN)」**ボタンまたは**「システム DSN (System DSN)」**タブをクリックします (プラットフォームによって異なります)。
  b. **「追加」**押しボタンをクリックします。
  c. リストにある IBM DB2 ODBC ドライバーをダブルクリックします。

d. 追加する DB2 データベースを選択し、それから「**OK**」をクリックします。

- Windows 32 ビット オペレーティング・システムでは、 DB2 データベースをデータ・ソースとして ODBC ドライバー・マネージャーに登録するために、コマンド行プロセッサー内で発行できるコマンドがあります。 管理者がコマンド行プロセッサー・スクリプトを作成すると、必要なデータベースを登録することができます。作成したら、 ODBC を介して DB2 データベースへのアクセスが必要なすべてのマシンでこのスクリプトを実行します。

  コマンド解説書 には、CATALOG コマンドについての詳細が含まれます。

  ```
  CATALOG [ user | system ] ODBC DATA SOURCE
  ```

ステップ 4. CCA を使用した DB2 CLI/ODBC ドライバーの構成: (任意選択)

a. 構成したい DB2 データベース別名を選択します。

b. **「プロパティー (Properties)」**押しボタンをクリックします。「データベース・プロパティー (Database Properties)」ウィンドウがオープンします。

c. **「設定 (Settings)」**押しボタンをクリックします。「CLI/ODBC 設定 (CLI/ODBC Settings)」ウィンドウがオープンします。

d. **「拡張 (Advanced)」**押しボタンをクリックします。オープンするウィンドウで構成キーワードを設定できます。これらのキーワードは、データベースの*別名* に関連付けられており、そのデータベースにアクセスするすべての DB2 CLI/ODBC アプリケーションに影響します。オンライン・ヘルプでは、すべてのキーワードについて説明しています。それは、174ページの『構成キーワード』で説明されています。

   このファイル (db2cli.ini) を手動で編集することについては、168ページの『db2cli.ini の構成』を参照してください。

ステップ 5. ODBC アクセスをインストール (上記で説明したとおりに) し終われば、 ODBC アプリケーションを使用して DB2 データにアクセスできます。 ODBC アプリケーションを開始して、「オープン (Open)」ウィンドウに進んでください。 **「ODBC データベース (ODBC databases)」**ファイル・タイプを選択します。 ODBC データ・ソースとして追加した DB2 データベースを、リストから

選択できるようになります。 固有索引が入っていない限り、多くの ODBC アプリケーションでは、表は読み取り専用でオープンされます。

CLI/ODBC を使用した DB2 への OS/2 クライアント・アクセス

OS/2 クライアントから DB2 CLI および ODBC アプリケーションが正常に DB2 データベースにアクセスできるようにするには、クライアント・システムで以下のステップを実行します。

1. DB2 データベース (データベースがリモートの場合はノード) をカタログ化しなければなりません。この処理を実行するには、CCA (またはコマンド行プロセッサー) を使用してください。

   詳細については、CCA のオンライン・ヘルプを参照してください。 (または*コマンド解説書* の **CATALOG DATABASE** および **CATALOG NODE** コマンド) を参照してください。 (または*コマンド解説書* の **CATALOG DATABASE** および **CATALOG NODE** コマンド) を参照してください。

2. ODBC アプリケーションを使用して DB2 データにアクセスする場合は、以下のステップを実行してください。 (CLI アプリケーションだけを使用している場合は、このステップを飛ばして次のステップに進んでください。)

   a. ODBC ドライバー・マネージャーがインストールされていることを検査します。 ODBC ドライバー・マネージャーが DB2 にインストールされていない場合、 ODBC アプリケーションに付属のドライバー・マネージャーを使用することをお勧めします。また、 DB2 CLI/ODBC ドライバーがインストールされていることを確認してください。

      1) ODBC 管理ツールを、これに付属の文書の説明にしたがって実行します。これは、通常次の 2 つの方法のどちらかで実行します。

         - OS/2 で**「ODBC」**フォルダーをダブルクリックしてから、**「ODBC アドミニストレーター (ODBC Administrator)」**アイコンをダブルクリックする。
         - コマンド行から **odbcadm.exe** を実行する。

         「データ・ソース (Data Sources)」ウィンドウがオープンします。

      2) **「ドライバー (Drivers)」**押しボタンをクリックします。「ドライバー (Drivers)」ウィンドウがオープンします。

      3) リストに「IBM DB2 ODBC ドライバー」が表示されているかどうかを検証します。

      ODBC ドライバー・マネージャーがインストールされていない場合、ご使用の ODBC アプリケーションに付いているインストールの指示に従ってください。 IBM DB2 CLI/ODBC ドライバーがインストールされて

いない場合には、 DB2 フォルダーで**「ODBC ドライバーのインストール (Install ODBC Driver)」**アイコンをダブルクリックして、DB2 CLI/ODBC ドライバーをインストールします。

b. DB2 データベースをデータ・ソース として ODBC ドライバー・マネージャーに登録します。

- CCA を使用して、
  1) データ・ソースとして追加したい DB2 データベース別名を選択します。
  2) **「プロパティー (Properties)」**押しボタンをクリックします。
  3) **「ODBC 用にこのデータベースを登録 (Register this database for ODBC)」**チェック・ボックスを選択します。
- ODBC ドライバー・マネージャーを使用して、
  1) 本書で説明されているとおり、ODBC ドライバー・マネージャーを実行します。これは、通常次の 2 つの方法のどちらかで実行します。
     - OS/2 で**「ODBC」**フォルダーをダブルクリックしてから、**「ODBC アドミニストレーター (ODBC Administrator)」**アイコンをダブルクリックする。
     - コマンド行から **odbcadm.exe** を実行する。
  2) 「データ・ソース (Data Sources)」ウィンドウから**「追加 (Add)」**押しボタンをクリックします。「データ・ソースの追加 (Add Data Source)」ウィンドウがオープンします。
  3) リストにある IBM DB2 ODBC DRIVER をダブルクリックします。
  4) 追加する DB2 データベースを選択し、それから「**OK**」をクリックします。

3. CCA を使用した DB2 CLI/ODBC ドライバーの構成: (任意選択)
   a. 構成したい DB2 データベース別名を選択します。
   b. **「プロパティー (Properties)」**押しボタンをクリックします。「データベース・プロパティー (Database Properties)」ウィンドウがオープンします。
   c. **「設定 (Settings)」**押しボタンをクリックします。「CLI/ODBC 設定 (CLI/ODBC Settings)」ウィンドウがオープンします。
   d. **「拡張 (Advanced)」**押しボタンをクリックします。オープンするウィンドウで構成キーワードを設定できます。これらのキーワードは、データベースの*別名* に関連付けられており、そのデータベースにアクセスす

るすべての DB2 CLI/ODBC アプリケーションに影響します。 オンライン・ヘルプでは、すべてのキーワードについて説明しています。それは、174ページの『構成キーワード』で説明されています。

このファイル (db2cli.ini) を手動で編集することについては、 168ページの『db2cli.ini の構成』を参照してください。

4. ODBC アクセスをインストール (上記で説明したとおりに) し終われば、ODBC アプリケーションを使用して DB2 データにアクセスできます。ODBC アプリケーションを開始して、「オープン (Open)」ウィンドウに進んでください。 **「ODBC データベース (ODBC databases)」**ファイル・タイプを選択します。 ODBC データ・ソースとして追加した DB2 データベースを、リストから選択できるようになります。 固有索引が入っていない限り、多くの ODBC アプリケーションでは、表は読み取り専用でオープンされます。

CLI/ODBC を使用した DB2 への UNIX クライアント・アクセス

DB2 CLI および ODBC アプリケーションが、 UNIX クライアントから DB2 データベースに正常にアクセスできるようにするには、クライアント・システムで以下のステップを実行してください。

1. DB2 データベース (データベースがリモートの場合はノード) をカタログ化しなければなりません。この処理を実行するには、コマンド行プロセッサーを使用してください。

   詳細については、 *コマンド解説書* の **CATALOG DATABASE** および **CATALOG NODE** コマンドを参照してください。

2. DB2 CLI/ODBC ドライバーは DB2 クライアント・インストールの際にインストールされます。この時点でこれが選択されていることを確認してください。

3. ODBC アプリケーションを使用して DB2 データにアクセスする場合は、以下のステップを実行してください。 (CLI アプリケーションだけを使用している場合は、このステップを飛ばして次のステップに進んでください。)

   a. ODBC アプリケーションを使用する場合は、 ODBC ドライバー・マネージャーがインストールされていて、 ODBC を使用するそれぞれのユーザーに ODBC へのアクセス権があることを確認する必要があります。 DB2 が ODBC ドライバー・マネージャーをインストールしていない場合、 ODBC クライアント・アプリケーションまたは ODBC SDK に付属の ODBC ドライバー・マネージャーを使用して、このアプリケーションで DB2 データにアクセスできるようにしなければなりません。

b. ドライバー・マネージャーは、次の 2 つの初期設定ファイルを使用します。

odbcinst.ini インストールされるデータベース・ドライバーが示されている ODBC ドライバー・マネージャーの構成ファイル。 ODBC を使用するそれぞれのユーザーには、このファイルへのアクセス権がなければなりません。

.odbc.ini エンド・ユーザーのデータ・ソース構成。このファイルのコピーを、各ユーザー ID が自分のホーム・ディレクトリーに持つことになります。このファイルはドットで開始することに注意してください。

odbcinst.ini の設定

このファイルの設定は、マシンにあるすべての ODBC ドライバーに影響を与えます。

ASCII エディターを使用して、このファイルを更新してください。 このファイルには [IBM DB2 ODBC DRIVER] というスタンザ (セクション) がなければならず、ここに DB2 ODBC ドライバーへのフルパスを示す ″Driver″ で始まる行を指定します。 DB2 ODBC ドライバーの名前は、AIX では db2.o であり、その他の UNIX プラットフォームでは libdb2 に、プラットフォーム別のファイル拡張子を付けたものになります。 (Solaris 実行環境版では libdb2.so です。) たとえば、AIX 上では、エンド・ユーザーのホーム・ディレクトリーが /u/thisuser/ で、sqllib ディレクトリーがそこにインストールされている場合は、以下の入力が正しい入力になります。

```
    [IBM DB2 ODBC DRIVER]
Driver=/u/thisuser/sqllib/lib/db2.o
```

.odbc.ini の設定

このファイルの設定は、マシンの特定のユーザーに関連付けられたもので、ユーザーごとに異なる .odbc.ini ファイルがあります。

.odbc.ini ファイルがエンド・ユーザーのホーム・ディレクトリーに存在しなければなりません (ファイル名の先頭にドットが付くことに注意してください)。 ASCII エディターを使用してこのファイルを更新し、適切なデータ・ソース構成情報を反映させます。 DB2 データベースを ODBC データ・ソースとして登録するには、それぞれの DB2 データベースごとに 1 つのスタンザ (セクション) が必要です。

.odbc.ini ファイルには、以下の行が含まれていなければなりません。

- [ODBC Data Source] スタンザには、

  ```
  SAMPLE=IBM DB2 ODBC DRIVER
  ```

  IBM DB2 ODBC DRIVER を使用した、 SAMPLE というデータ・ソースがあることを示しています。

- [SAMPLE] スタンザには (たとえば、AIX 上で)、

  ```
  [SAMPLE]
  Driver=/u/thisuser/sqllib/lib/libdb2.a
  Description=Sample DB2 ODBC Database
  ```

  SAMPLE データベースが /u/thisuser ディレクトリーにある DB2 インスタンスの一部であることを示しています。

- [ODBC] スタンザには、

  ```
  InstallDir=/u/thisuser/sqllib/odbclib
  ```

  /u/thisuser/sqllib/odbclib を、 ODBC がインストールされている場所として扱う必要があることを示しています。

- InstallDir が ODBC ドライバー・マネージャーの位置を正確に指していることを確認してください。

  たとえば、 ODBC ドライバー・マネージャーが /opt/odbc にインストールされた場合、 [ODBC] スタンザは以下のようになります。

  ```
  [ODBC]
  Trace=0
  TraceFile=odbctrace.out
  InstallDir=/opt/odbc
  ```

詳しくは 170ページの『ODBC.INI の構成方法』を参照してください。

いったん .ini ファイルを設定すると、ODBC アプリケーションを実行して、 DB2 データベースにアクセスできるようになります。 追加ヘルプと追加情報については、ODBC アプリケーションに添付されている文書を参照してください。

4. DB2 CLI/ODBC ドライバーを構成します (任意選択)。

   このファイルにはさまざまなキーワードおよび値が入っており、これらを使用して、 DB2 CLI/ODBC およびそれを使用するアプリケーションの動作を修正することができます。キーワードはデータベースの*別名* に関連付けられており、そのデータベースにアクセスするすべての DB2 CLI/ODBC アプリケーションに影響します。

このファイル (db2cli.ini) を手動で編集することについては、 168ページの『db2cli.ini の構成』を参照してください。特定のキーワードについては、174ページの『構成キーワード』を参照してください。

詳細な構成情報

158ページの『CLI/ODBC アクセスのためのプラットフォーム固有の詳細情報』では、必要な情報がすべて提供されています。以下の追加情報は、DB2 ツール・サポートが使用できない場合や、さらに詳細な情報を必要とする管理者にとって役立ちます。

- 『DB2 CLI/ODBC ドライバーをデータベースにバインドする方法』
- 167ページの『CLI/ODBC 構成キーワードの設定方法』
- 168ページの『db2cli.ini の構成』

DB2 CLI/ODBC ドライバーをデータベースにバインドする方法

CLI/ODBC ドライバーは、適切な特権または許可のあるデータベースへ最初に接続したときに、自動バインドされます。管理者は、初回接続を実行したり、必要なファイルを明示的にバインドできます。

表 10. DB2 CLI バインド・ファイルおよびパッケージ名

バインド・ファイル名	パッケージ名	DB2 ユニバーサル・データベースで必要	DRDA サーバーで必要
db2clish.bnd	SQLLFyxx	あり	あり
db2clisn.bnd	SQLLCyxx	あり	あり
db2clibh.bnd	SQLLDyxx	あり	あり
db2clihn.bnd	SQLLEyxx	あり	あり
db2cliws.bnd	SQLL65zz	バージョン 2 以降	なし
db2clims.bnd	SQLL75zz	なし	DB2 (MVS/ESA 版)
db2clivm.bnd	SQLL85zz	なし	SQL/DS
db2cliv1.bnd	SQLLB5zz	バージョン 1 のみ	なし
db2cliv2.bnd	SQLL95zz	バージョン 2 以降	なし
db2clias.bnd	SQLLA5zz	なし	DB2 ユニバーサル・データベース (AS/400 版)

表 10. DB2 CLI バインド・ファイルおよびパッケージ名 (続き)

バインド・ファイル名	パッケージ名	DB2 ユニバーサル・データベースで必要	DRDA サーバーで必要

注:

- 'xx' は、00〜FF の 16 進値です。
- 'y' の範囲は、0〜4 です。
- 'zz' は、それぞれのプラットフォームで固有の値です。

以前のバージョンの DB2 サーバーでは、すべてのバインド・ファイルが必要なわけではなく、バインド時にエラーが返されます。

db2cli.lst ファイルには、 DB2 CLI が DB2 バージョン 2 以降のサーバーに接続する場合に必要なバインド・ファイルの名前が入っています (db2clixx.bnd、xx は cs、rr、rs、ur、ws、および v2)。 db2cli1.lst ファイルには、 DB2 CLI が DB2 バージョン 1 以降のサーバーに接続する場合に必要なバインド・ファイルの名前が入っています (db2clixx.bnd。 xx は cs、rr、ur、および v1)。

DRDA サーバーの場合、

- ddcsvm.lst、ddcsmvs.lst、ddcsvse.lst、または ddcs400.lst の各バインド・リスト・ファイルのうちいずれか 1 つを使用してください。
- 174ページの『構成キーワード』の SYSSCHEMA キーワードを参照してください。
- 必要なバインド・オプションの詳細については、 *概説およびインストール* または *DB2 コネクト エンタープライズ・エディション (OS/2 および Windows 版) 概説およびインストール* を参照してください。

CLI/ODBC 構成キーワードの設定方法

CCA または DB2 クライアント・セットアップ管理ツール (プラットフォームに適用できる場合) を使用するか、または db2cli.ini ファイルを手動で編集することによって、 DB2 CLI の構成をさらに行うことができます。

このファイルには、DB2 CLI とこの製品を使うアプリケーションの動作を修正する場合に使用できる、さまざまなキーワードと値が入っています。キーワードは、データベース*別名* と関連しており、そのデータベースにアクセスするすべての DB2 CLI および ODBC アプリケーションに影響を与えます。

省略時設定では、CLI/ODBC 構成キーワード・ファイルは、 Intel プラットフォームの sqllib ディレクトリー、および UNIX プラットフォームで CLI/ODBC アプリケーションを実行しているデータベース・インスタンスの sqllib/cfg ディレクトリーにあります。

環境変数 *DB2CLIINIPATH* を使って、省略時値をオーバーライドし、異なるファイル位置を指定することもできます。

構成キーワードを使用すると、以下のことが可能になります。

- データ・ソース名、ユーザー名、およびパスワードなどの一般的な機能を構成する。
- パフォーマンスに影響を及ぼすオプションを設定する。
- ワイルドカード文字などの照会パラメーターを指示する。
- さまざまな ODBC アプリケーション用にパッチまたは作業環境を設定する。
- コード・ページと IBM グラフィック・データ・タイプなどの接続に関連したその他の機能を設定する。

すべてのキーワードの完全な説明および使用法については、 174ページの『構成キーワード』を参照してください。

db2cli.ini の構成: db2cli.ini 初期設定ファイルは、 DB2 CLI 構成オプション用の値を保管している ASCII ファイルです。作業を開始する助けとして、サンプル・ファイルが提供されます。 各キーワードの詳細については、174ページの『構成キーワード』を参照してください。

ご使用のプラットフォームでこのファイルを修正する方法については、158ページの『CLI/ODBC アクセスのためのプラットフォーム固有の詳細情報』を参照してください。

ファイル内には、ユーザーが構成を希望するデータベース (データ・ソース) ごとに 1 つのセクションがあります。必要であれば、DB2 へのすべての接続に影響を与える共通セクションもあります。

COMMON セクションには、 DB2 CLI/ODBC ドライバーを介した DB2 へのすべての接続に適用するキーワードのみ含まれています。以下のキーワードが組み込まれています。

- DISABLEMULTITHREAD
- TRACE
- TRACEFILENAME

- TRACEFLUSH
- TRACEPATHNAME

他のすべてのキーワードはデータベース特定のセクションに置かれ、以下の箇所で説明されています。

db2cli.ini ファイルの COMMON セクションは、次の語で始まります。

```
[COMMON]
```

共通キーワードを設定する前に、クライアントからのすべての DB2 CLI/ODBC 接続にこの設定が与える影響を評価するのは重要なことです。たとえば、TRACE などのキーワードは、 DB2 に接続している DB2 CLI/ODBC アプリケーションのうち 1 つだけを障害追及しようとしている場合でも、 DB2 に接続しているすべての DB2 CLI/ODBC アプリケーションに関する情報をそのクライアントで生成します。

それぞれのデータベースの特定のセクションは、必ず大括弧で囲まれたデータベース別名の名前で始まります。

[*database alias***]**

これを***セクション見出し*** と呼びます。

パラメーターを設定するには、キーワードとその関連キーワード値を次の形式で指定します。

KeywordName =*keywordValue*

- 各データベースのすべてのキーワードとその関連値は、そのデータベースのセクション見出しの下になければなりません。
- 各セクションのキーワード設定値は、そのセクション見出しで指定されたデータベース別名だけに適用されます。
- キーワードは大文字小文字の区別はありません。しかし、その値が文字ベースのものであれば値にその区別がある場合もあります。
- 各キーワードに関連する構文については、 174ページの『DB2 CLI/ODBC 構成キーワードのリスト』を参照してください。
- .INI ファイルにデータベースがない場合、これらのキーワードの省略時値が有効になっています。
- 新しい行の先頭位置にセミコロンを入れると、注釈行になります。
- ブランク行は許可されています。
- 1 つのキーワードに重複項目があると、最初の項目が使用されます (警告は与えられません)。

2 つのデータベース別名セクションがある .INI サンプル・ファイルを次に示します。

```
; This is a comment line.
[MYDB22]
AUTOCOMMIT=0
TABLETYPE="'TABLE','SYSTEM TABLE'"
; This is another comment line.
[MYDB2MVS]
DBNAME=SAAID
TABLETYPE="'TABLE'"
SCHEMALIST="'USER1',CURRENT SQLID,'USER2'"
```

db2cli.ini ファイルは手動で編集できますが、ご使用のプラットフォームで使用可能であれば CCA を使用するようお勧めします。 手作業で db2cli.ini ファイルを編集する場合、最後の項目の後にブランク行を追加してください。

ODBC.INI の構成方法

Microsoft の 16 ビット ODBC ドライバー・マネージャーおよびすべての非 Microsoft ODBC ドライバー・マネージャーは、使用可能なドライバーとデータ・ソースに関する情報を記録する場合に、 odbc.ini ファイルを使用します。 UNIX プラットフォーム上の ODBC ドライバー・マネージャーも、 odbcinst.ini ファイルを使用します。 ほとんどのプラットフォームでは、必要なファイルはこれらのツールで自動更新されますが、 UNIX プラットフォームで ODBC を使用するユーザーは手動で編集する必要があります。 odbc.ini (および必要な odbcinst.ini) ファイルは、以下の位置に入れられています。

UNIX ODBC アプリケーションを実行しているユーザー ID のホーム・ディレクトリー (UNIX では、 odbc.ini ファイル名には先頭にドットが付いて .odbc.ini になります。)

また、手動でこのファイルを修正することもできます。このファイルの既存の項目を変更することはしないでください。このファイルを手動で編集するには、以下の手順を実行してください。

ステップ 1. odbc.ini を編集するには、ASCII エディターを使用します。

次は odbc.ini ファイルの例です。

```
[ODBC Data Sources]
MS Access Databases=Access Data (*.mdb)
[MS Access Databases]
Driver=D:¥WINDOWS¥SYSTEM¥simba.dll
FileType=RedISAM
SingleUser=False
UseSystemDB=False
```

[ODBC Data Sources] セクションには、利用可能な各データ・ソースの名前および関連ドライバーの記述がリストしてあります。

[ODBC Data Sources] セクションにリストされているデータ・ソースごとに、そのデータ・ソースに関する追加情報をリストするセクションがあります。これらは *Data Source Specification* セクションと呼ばれます。

ステップ 2. [ODBC DATA SOURCE] 項目の下に、次の行を追加します。

```
database_alias=IBM DB2 ODBC DRIVER
```

database_alias は、データベース・ディレクトリーにカタログされているデータベースの別名です (コマンド行プロセッサーの CONNECT TO ステートメントで使用されるデータベース名)。

ステップ 3. Data Source Specification セクションに新しい項目を追加し、データ・ソースをドライバーと関連付けます。

```
[database_alias]
Driver=x:¥windows¥system¥db2cliw.dll
```

ここで、各パラメーターは以下のとおりです。

- *database_alias* は、データベース・ディレクトリーにカタログされているデータベースの別名で、データ・ソース仕様セクションの下にリストされます。
- *x:* は、Windows オペレーティング・システムがインストールされているドライブです。

次は、IBM データ・ソース項目を追加したファイルの例です。

```
[ODBC Data Sources]
MS Access Databases=Access Data (*.mdb)
SAMPLE=IBM DB2 ODBC DRIVER
[MS Access Databases]
Driver=D:¥WINDOWS¥SYSTEM¥simba.dll
FileType=RedISAM
SingleUser=False
UseSystemDB=False
[SAMPLE]
Driver=D:¥WINDOWS¥SYSTEM¥db2cliw.dll
Description=Sample DB2 Client/Server database
```

.ini ファイルの UNIX での構成

163ページの『CLI/ODBC を使用した DB2 への UNIX クライアント・アクセス』では、 odbc.ini および odbcinst.ini ファイルの両方を更新する方法に関する詳細な手順を説明しています。

アプリケーション開発環境

DB2 アプリケーション開発クライアント製品をインストールすると、DB2 CLI アプリケーション開発サポートが得られます。 DB2 アプリケーション開発クライアントでは、他の DB2 クライアントと同じ実行時セットアップが必要です。 DB2 CLI 開発環境を使用する前に、ご使用の環境が正しくセットアップされていることを、下記のステップに従って検査してください。

- コマンド `DB2` を発行してコマンド行プロセッサーを始動する。または、プラットフォームで使用可能であればコマンド・センターを用いてください。
- 次のコマンドを指定して、カタログ・データベースをリストする。
  `LIST DATABASE DIRECTORY`
- 次のコマンドを指定して、データベース *database_name* に接続する。
  `CONNECT TO database_name USER userid USING password`

カタログされているデータベースがない場合、または接続が失敗する場合は、環境の構成に関する情報について、*概説およびインストール* を参照してください。

接続を確認してから、サンプル・アプリケーションのコンパイルに進んでください。

注: DB2 CLI サンプル・ディレクトリーには「対話式 CLI」または `db2cli` と呼ばれるアプレットもあります。 CLI 関数呼び出しの設計およびプロトタイプ作成に使用するプログラマー用ユーティリティーの詳細については、同じディレクトリーにある `INTCLI.DOC` ファイルを参照してください。

サンプル・アプリケーションのコンパイル

DB2 CLI には、以下の位置にある各種のサンプル・アプリケーションが含まれます。

- OS/2 および Windows 32 ビット・オペレーティング・システム:
  `%DB2PATH%¥samples¥cli` (`%DB2PATH%` は、 DB2 がインストールされている位置を表す変数)
- UNIX: `$HOME/sqllib/samples` (`$HOME` は、インスタンスの所有者のホーム・ディレクトリー)

同一ディレクトリーの README ファイルには、各サンプルが解説付きでリストされ、提供されている makefile および作成ファイルを使用して、サンプルを構築する方法を説明しています。 DB2 CLI のサンプルはすべて 907ページの『付録J. CLI サンプル・コード』にリストされています。

アプリケーション構築の手引き では、 DB2 がサポートするすべてのオペレーティング・システム上で DB2 CLI アプリケーションおよびストアード・プロシージャーをコンパイル、リンク、および実行するための情報を提供します。この資料のセクション『DB2 コール・レベル・インターフェース (CLI) アプリケーション』で、詳細にわたる説明および例を参照してください。

以前のバージョンの DB2 CLI からアプリケーションを移行している場合には、付加的な情報として 822ページの『非互換性』を参照してください。

注: 同名のファイルが複数ある場合には (たとえば、 sql.h と sqlext.h は複数の ODBC SDK 環境に組み込まれています)、コンパイラーが組み込み (ヘッダー) ファイルを探索する順序が重要であることがあります。

DB2 CLI アプリケーションのみを構築している場合は、それ以外のアプリケーションの前に DB2 組み込みパスを加えてください。

ODBC アプリケーションを構築している場合、正しい組み込みファイルを使用することができるように、構築環境をさらにカスタマイズすることが必要になる可能性があります。詳細については、ODBC SDK の資料を参照してください。

コンパイルおよびリンク・オプション

特定のプラットフォーム用サンプルのコンパイルおよびリンクに関する詳細は、 *アプリケーション構築の手引き* を参照してください。 make ファイルおよび構築スクリプトには、プラットフォームおよびサポートされるコンパイラーに対する正しいオプションが備えられています。これらのファイルは、sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内にあります。

注: プラットフォームまたはコンパイラーに特定のリンク・オプションには、一部の CLI アプリケーションに必須のものもあります。詳細な説明については、*アプリケーション構築の手引き* を参照してください。

DB2 CLI/ODBC 構成キーワードのリスト

ご使用のプラットフォームの DB2 CLI/ODBC 構成キーワードの設定方法に関する固有の情報については、 158ページの『CLI/ODBC アクセスのためのプラットフォーム固有の詳細情報』の最後のステップを参照してください。
db2cli.ini ファイルの位置および形式に関する詳細については、 167ページの『CLI/ODBC 構成キーワードの設定方法』を参照してください。

構成キーワード

キーワードは、176ページの『APPENDAPINAME』で開始するアルファベット順にリストされています。またカテゴリーごとに分けられています。これらのカテゴリーのそれぞれは、クライアント構成アシスタント (UNIX プラットフォームでは利用不能) からアクセス可能な「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックで別個のタブとして表されています。

カテゴリー別構成キーワード

CLI/ODBC 設定の汎用構成キーワード: 汎用キーワード。

- 192ページの『DBALIAS』
- 212ページの『PWD』
- 229ページの『UID』

互換性構成キーワード: オプションの**互換性**セットを使用して、DB2 の動作を定義します。それらのオプションを設定すると、他のアプリケーションと DB2 との互換性の確認が可能です。

- 195ページの『DEFERREDPREPARE』
- 196ページの『DISABLEMULTITHREAD』
- 197ページの『EARLYCLOSE』

データ・タイプの構成キーワード: このオプションの**データ・タイプ**・セットを使用して、 DB2 が種々のデータ・タイプをどのように報告し、取り扱うかを定義します。

- 178ページの『BITDATA』
- 199ページの『GRAPHIC』
- 204ページの『LOBMAXCOLUMNSIZE』
- 204ページの『LONGDATACOMPAT』

エンタープライズの構成キーワード: オプションの**エンタープライズ**・セットを使用して、大容量のデータベースへの接続効率を最大にします。

- 180ページの『CLISCHEMA』
- 181ページの『CONNECTNODE』

- 183ページの『CURRENTPACKAGESET』
- 185ページの『CURRENTSCHEMA』
- 185ページの『CURRENTSQLID』
- 188ページの『DB2CONNECTVERSION』
- 193ページの『DBNAME』
- 198ページの『GRANTEELIST』
- 198ページの『GRANTORLIST』
- 214ページの『SCHEMALIST』
- 220ページの『SYSSCHEMA』
- 221ページの『TABLETYPE』

環境の構成キーワード: オプションの**環境**セットを使用して、サーバーおよびクライアント・マシンでの種々のファイル位置を定義します。
- 179ページの『CLIPKG』
- 182ページの『CURRENTFUNCTIONPATH』
- 194ページの『DEFAULTPROCLIBRARY』
- 213ページの『QUERYTIMEOUTINTERVAL』
- 222ページの『TEMPDIR』

ファイル DSN 構成キーワード: オプションの**ファイル DSN** セットを使用して、ファイル DSN 接続の TCP/IP 設定を定義します。
- 187ページの『DATABASE』
- 200ページの『HOSTNAME』
- 212ページの『PROTOCOL』
- 215ページの『SERVICENAME』

最適化構成キーワード: オプションの**最適化**セットを使用して、 CLI/ODBC ドライバーとそのサーバー間のネットワーク・フローの速度を上げ、量を減らします。
- 184ページの『CURRENTREFRESHAGE』
- 189ページの『DB2DEGREE』
- 189ページの『DB2ESTIMATE』
- 190ページの『DB2EXPLAIN』
- 192ページの『DB2OPTIMIZATION』
- 203ページの『KEEPSTATEMENT』
- 208ページの『OPTIMIZEFORNROWS』
- 208ページの『OPTIMIZESQLCOLUMNS』
- 229ページの『UNDERSCORE』

サービス構成キーワード: オプションの**サービス**・セットを使用して、CLI/ODBC 接続で生じる問題のトラブルシューティングを助けます。また、プ

ログラマーがいくつかのオプションを使用すると、 CLI プログラムがどのようにサーバーへの呼び出しに変換されるかを一層よく理解できます。

- 『APPENDAPINAME』
- 201ページの『IGNOREWARNINGS』
- 201ページの『IGNOREWARNLIST』
- 209ページの『PATCH1』
- 210ページの『PATCH2』
- 211ページの『POPUPMESSAGE』
- 215ページの『SQLSTATEFILTER』
- 223ページの『TRACE』
- 224ページの『TRACECOMM』
- 225ページの『TRACEFILENAME』
- 226ページの『TRACEFLUSH』
- 227ページの『TRACEPATHNAME』
- 230ページの『WARNINGLIST』

静的 SQL 構成キーワード: オプションの**静的 SQL** セットは、 CLI/ODBC アプリケーションで静的 SQL ステートメントを実行する際に使用されます。

- 216ページの『STATICCAPFILE』
- 217ページの『STATICLOGFILE』
- 217ページの『STATICMODE』
- 218ページの『STATICPACKAGE』

トランザクションの構成キーワード: オプションの**トランザクション・**セットを使用して、アプリケーションで使用される SQL ステートメントを制御し速度を上げます。

- 177ページの『ASYNCENABLE』
- 182ページの『CONNECTTYPE』
- 186ページの『CURSORHOLD』
- 202ページの『KEEPCONNECT』
- 205ページの『MAXCONN』
- 206ページの『MODE』
- 207ページの『MULTICONNECT』
- 219ページの『SYNCPOINT』
- 228ページの『TXNISOLATION』

APPENDAPINAME

キーワードの説明:

エラーを生成した CLI/ODBC 関数名をエラー・メッセージに追加します。

db2cli.ini キーワード構文:

APPENDAPINAME = **0** | **1**

省略時設定値:

DB2 CLI 関数名を表示しない。

DB2 CLI/ODBC 設定タブ:

サービス

使用上の注意:

エラーを生成した DB2 CLI 関数 (API) 名が、 SQLGetDiagRec() または SQLError() を使用して取り出されたエラー・メッセージに追加されます。関数名は、中括弧 { } で囲まれます。

たとえば、

```
[IBM][CLI Driver]" CLIxxxx: < text >
SQLSTATE=XXXXX {SQLGetData}"
```

0 = DB2 CLI 関数名を追加しない (省略時値)

1 = DB2 CLI 関数名を追加する

このキーワードはデバッグにのみ使用できます。

ASYNCENABLE

キーワードの説明:

照会を非同期に実行する機能を使用可能または使用不能にします。

db2cli.ini キーワード構文:

ASYNCENABLE = **1** | **0**

省略時設定値:

照会を非同期に実行します。

DB2 CLI/ODBC 設定タブ:

トランザクション

等価なステートメント属性:

SQL_ATTR_ASYNC_ENABLE

使用上の注意:

このオプションを使用すると、照会を非同期に実行する機能を使用可能または使用不能にすることができます。ただし、実際の使用は、この機能を利用でき

るようにプログラミングされたアプリケーションに限られます。 使用可能にするとアプリケーションが正常に機能しない場合にのみ、このオプションを使用不能にしてください。このオプションは、 db2cli.ini ファイル内のデータ・ソース・セクションにあります。

1 = 照会を非同期に実行する (省略時値)
0 = 照会を非同期には実行しない

注: CLI/ODBC ドライバーは、非同期 ODBC をサポートしていない旧バージョンの DB2 のときと同じように動作します。

BITDATA

キーワードの説明:

2 進データ・タイプを 2 進データ・タイプまたは文字データ・タイプのどちらで報告するか指定します。

db2cli.ini キーワード構文:

BITDATA = **1** | **0**

省略時設定値:

FOR BIT DATA と BLOB データ・タイプを文字データ・タイプとして報告します。

DB2 CLI/ODBC 設定タブ:

データ・タイプ

使用上の注意:

このオプションを使用すると、ODBC データ・タイプ (SQL_BINARY、SQL_VARBINARY、SQL_LONGVARBINARY、および SQL_BLOB) を 2 進データ・タイプとして報告するかどうか指定できます。 IBM DBMS は 2 進データ・タイプのある列をサポートしており、 CHAR、VARCHAR、および LONG VARCHAR 列に FOR BIT DATA 属性を定義します。 DB2 ユニバーサル・データベースは、BLOB データ・タイプでも 2 進データをサポートしています (この場合、 2 進データは CLOB データ・タイプにマップされます)。

DB2 バージョン 1 アプリケーションを使用している場合、このアプリケーションは (LONG) (VAR) CHAR データを SQL_C_CHAR バッファーに取り込むため、ユーザー側でこのオプションを設定しておくことが必要になります。 DB2 バージョン 1 では、データはそのまま SQL_C_CHAR バッファーに移されますが、 DB2 バージョン 2 以降では、データは 16 進数ニブルの ASCII 表示に変換されます。

FOR BIT DATA または BLOB として定義された列すべてに文字データしか入っておらず、かつアプリケーションが 2 進データ列を表示できない場合は、BITDATA = 0 にしか設定できません。

1 = FOR BIT DATA および BLOB データ・タイプを 2 進データ・タイプとして報告します (省略時値)。

0 = FOR BIT DATA と BLOB データ・タイプを文字データ・タイプとして報告します。

CLIPKG

キーワードの説明:

生成されるラージ・パッケージの数

db2cli.ini キーワード構文:

CLIPKG = **3** | **4** | ... | **30**

省略時設定値:

3

DB2 CLI/ODBC 設定タブ:

このキーワードは「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックを使用して設定することはできません。直接このキーワードを使用するように db2cli.ini ファイルを修正しなければなりません。

使用上の注意:

値が 3〜30 の整数に指定されていない場合には、省略時値が使用されます。このとき、エラーまたは警告は出されません。

このキーワードは、CLI/ODBC アプリケーションでの SQL セクションの数を増やすのに使用されます。 これが使用される場合には、管理者は CLIPKG バインド・オプションを使用して、必要な CLI バインド・ファイルを明示的にバインドしなければなりません。また、サーバー (UNIX または Intel プラットフォームで DB2 UDB V6.1 以降) にある db2cli.ini ファイルを CLIPKG と同じ値で更新しなければなりません。

この設定は、ラージ・パッケージ (364 個のセクションを含む) にのみ適用されます。 スモール・パッケージ (64 個のセクションを含む) の数は 3 個であり、変更できません。

パッケージはデータベースでスペースをとるため、増やすセクションの数は、ご使用のアプリケーションを実行できるだけの数にとどめるようお勧めします。

CLISCHEMA

キーワードの説明:

使用する DB2 ODBC カタログ視点を設定します。

db2cli.ini キーワード構文:

CLISCHEMA = *ODBC カタログ視点*

省略時設定値:

None - ODBC カタログ視点は使用されない

DB2 CLI/ODBC 設定タブ:

このキーワードは「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックを使用して設定することはできません。直接このキーワードを使用するように db2cli.ini ファイルを修正しなければなりません。

参照項目:

220ページの『SYSSCHEMA』

等価な接続属性:

SQL_ATTR_CLISCHEMA

使用上の注意:

DB2 ODBC カタログは、 DB2 コネクトを介してホスト DBMS に接続する ODBC アプリケーション内の表のリストを呼び出すスキーマ呼び出しのパフォーマンスを向上させるために設計されました。

ホスト DBMS 上で作成および保守される DB2 ODBC カタログには、実際の DB2 カタログで定義されているオブジェクトを表す行が含まれますが、それらの行には ODBC 操作をサポートするために必要な列だけが存在します。 DB2 ODBC カタログ内の表は事前に結合されていて、 ODBC アプリケーションへの高速なアクセスをサポートするように特に索引付けされています。

システム管理者は、それぞれに特定のユーザー・グループが必要とする行だけを含む複数の DB2 ODBC カタログ視点を作成できます。各ユーザーは、使用したい DB2 ODBC カタログ視点を (このキーワードを設定することにより) 選択できます。

CLISCHEMA 設定の使用は、ODBC アプリケーションからはまったく見えません。このオプションは任意の ODBC アプリケーションで使用できます。

このキーワードには SYSSCHEMA キーワードと類似の効果がありますが、(適用できる場合) CLISCHEMA を代わりに使用してください。

CLISCHEMA はデータ・アクセスを効率的に改善します。 SYSSCHEMA と共に使用するユーザー定義の表は DB2 カタログの表のミラー・イメージなので、 ODBC ドライバーはやはり複数の表の行を結合して ODBC ユーザーに必要な情報を生成しなければなりません。 CLISCHEMA を使用しても、カタログ表との競合が少なくなります。

CONNECTNODE

キーワードの説明:

接続を作成するノードを指定します。

db2cli.ini キーワード構文:

CONNECTNODE = **1 から 999 までの整数値** |
SQL_CONN_CATALOG_NODE

省略時設定値:

マシン上のポート 0 に定義された論理ノードが使用されます。

DB2 CLI/ODBC 設定タブ:

このキーワードは「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックを使用して設定することはできません。直接このキーワードを使用するように db2cli.ini ファイルを修正しなければなりません。

適用可能な条件:

複数ノードのエンタープライズ拡張エディション・データベース・サーバーに接続しています。

等価な接続属性:

SQL_ATTR_CONNECT_NODE

使用上の注意:

接続したい DB2 エンタープライズ拡張エディション・データベース区画サーバーの宛先論理ノードを指定するために使用します。 このキーワード (または属性設定値) は、環境変数 DB2NODE の値を指定変更します。 以下の値に設定できます。

- 0 から 999 までの整数
- SQL_CONN_CATALOG_NODE

この変数が設定されていない場合、省略時の宛先論理ノードとしてマシン上のポート 0 に定義された論理ノードが使用されます。

CONNECTTYPE

キーワードの説明:

リモートまたは分散作業単位

db2cli.ini キーワード構文:

CONNECTTYPE = **1** | **2**

省略時設定値:

リモート作業単位

DB2 CLI/ODBC 設定タブ:

トランザクション

参照項目:

219ページの『SYNCPOINT』

等価な接続属性:

SQL_ATTR_CONNECTTYPE

使用上の注意:

このオプションを使用すると、省略時接続タイプが指定できます。

1 = リモート作業単位。複数の並行接続 (各接続には独自のコミット効力範囲がある)。並行トランザクションは整合するように調整されません。 (省略時値)

2 = 分散作業単位。複数のデータベースは同一の分散作業単位の一部となる整合接続。この設定は、SYNCPOINT 設定値と組み合わされて、トランザクション・マネージャーを使用するかどうかの判別に利用されます。

CURRENTFUNCTIONPATH

キーワードの説明:

動的 SQL ステートメント内の関数参照とデータ・タイプ参照を解決するためにスキーマを使用するよう指定します。

db2cli.ini キーワード構文:

CURRENTFUNCTIONPATH = *current_function_path*

省略時設定値:

下記の説明を参照してください。

DB2 CLI/ODBC 設定タブ:

環境

使用上の注意:

このキーワードは、動的 SQL ステートメント内で使用されている関数参照とデータ・タイプ参照を解決するために用いるパスを定義します。キーワードには 1 つまたは複数のスキーマ名をリストにして含めることができますが、スキーマ名はコンマで区切り、二重引用符で囲みます。

省略時値は ″SYSIBM″,″SYSFUN″,X です。ここで、X は二重引用符で区切られた USER 特殊レジスターの値です。スキーマ SYSIBM を指定する必要はありません。このスキーマが関数パスに組み込まれていなくても、暗黙的に最初のスキーマとみなされます。

このキーワードは、現行ユーザーのスキーマ以外のスキーマ名に定義されている可能性のある非修飾関数参照を解決するプロセスの一部として使用されます。スキーマ名の順序によって、解決する関数名の順序が決まります。関数の解決に関する詳細については、 *SQL 解説書* を参照してください。

CURRENTPACKAGESET

キーワードの説明:

各接続後に ″SET CURRENT PACKAGESET schema″ を発行します。

db2cli.ini キーワード構文:

CURRENTPACKAGESET = スキーマ名

省略時設定値:

文節は追加されません。

DB2 CLI/ODBC 設定タブ:

このキーワードは「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックを使用して設定することはできません。直接このキーワードを使用するように db2cli.ini ファイルを修正しなければなりません。

等価な接続属性:

SQL_ATTR_CURRENT_PACKAGE_SET

使用上の注意:

このオプションを使用すると、データベースに接続するたびに、 ″SET CURRENT PACKAGESET schema″ コマンドを発行します。省略時設定では、この文節は追加されません。

このステートメントはスキーマ名 (コレクション識別子) を設定しますが、このスキーマ名を使用して以後の SQL ステートメントに使用するパッケージが選択されます。

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。このオプションを使用することにより、これらのステートメントを実行する際に使用できる特権を制御できます。

- 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際に使用するスキーマを選択します。
- スキーマ中のオブジェクトが必要な特権を持っていることを確認してから、状況に応じて再バインドします。
- このスキーマに CURRENTPACKAGESET オプションを設定します。

こうして、CLI/ODBC から出される SQL ステートメントは指定したスキーマの下で実行し、スキーマに定義された特権を使用します。

SET CURRENT PACKAGESET コマンドに関する詳細については、 *SQL 解説書* を参照してください。

CURRENTREFRESHAGE

キーワードの説明:

CURRENT REFRESH AGE 特殊レジスターの値を設定します。

db2cli.ini キーワード構文:

CURRENTREFRESHAGE = **0** | **ANY** | **数値定数**

省略時設定値:

0 - REFRESH DEFERRED に定義された要約表は照会の処理を最適化するためには使用されません。

DB2 CLI/ODBC 設定タブ:

このキーワードは「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックを使用して設定することはできません。直接このキーワードを使用するように db2cli.ini ファイルを修正しなければなりません。

使用上の注意:

要約表および SET CURRENT REFRESH AGE ステートメントについての情報は、 *SQL 解説書* を参照してください。

このキーワードには、以下のいずれかの値を設定しなければなりません。

- 0 - REFRESH DEFERRED に定義された要約表は照会の処理を最適化するためには使用されないことを示します (省略時値)。

- 99999999999999 - REFRESH DEFERRED または REFRESH IMMEDIATE に定義された要約表が、照会の処理を最適化するために使用されることがあることを示します。この値は、9999 年 99 月 99 日 99 時間 99 分および 99 秒を表します。
- ANY - これは 99999999999999 の短縮形です。

CURRENTSCHEMA

キーワードの説明:

接続が成功したときに SET CURRENT SCHEMA ステートメントで使用されるスキーマを指定します。

db2cli.ini キーワード構文:

CURRENTSCHEMA = *スキーマ名*

省略時設定値:

ステートメントは発行されません。

DB2 CLI/ODBC 設定タブ:

このキーワードは「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックを使用して設定することはできません。直接このキーワードを使用するように db2cli.ini ファイルを修正しなければなりません。

使用上の注意:

このオプションが設定されていると、接続が成功したときに、 SET CURRENT SCHEMA ステートメントが DBMS に送信されます。こうすることにより、エンド・ユーザーまたはアプリケーションが SQL オブジェクトをスキーマ名を使わずに指定できるようになります。

SET CURRENT SCHEMA ステートメントの詳細については、 *SQL 解説書* を参照してください。

CURRENTSQLID

キーワードの説明:

接続が成功したときに DBMS に送信される SET CURRENT SQLID ステートメントで使われる ID を指定します。

db2cli.ini キーワード構文:

CURRENTSQLID = *current_sqlid*

省略時設定値:

ステートメントは発行されません。

DB2 CLI/ODBC 設定タブ:
エンタープライズ

適用可能な条件:
SET CURRENT SQLID がサポートされている DB2 DBMS (DB2 (MVS/ESA 版) など) に接続中であること。

使用上の注意:

このオプションが設定されていると、接続が成功したときに、 SET CURRENT SQLID ステートメントが DBMS に送信されます。こうすることにより、エンド・ユーザーとアプリケーションが SQL オブジェクトをスキーマ名を使わずに指定できるようになります。

CURSORHOLD

キーワードの説明:
オープン・カーソル上でトランザクション完了の影響。

db2cli.ini キーワード構文:
CURSORHOLD = **1** | **0**

省略時設定値:
選択 − カーソルは破棄されません。

DB2 CLI/ODBC 設定タブ:
トランザクション

等価なステートメント属性:
SQL_ATTR_CURSOR_HOLD

使用上の注意:

このオプションは、オープン・カーソル上でトランザクション完了の影響を制御します。

1 = カーソル保留。トランザクションがコミットされても、カーソルは破棄されません (省略時値)。

0 = カーソル保留なし。トランザクションがコミットされると、カーソルは破棄されます。

注: トランザクションがロールバックされた場合は、常にカーソルは破棄されます。

SQL_CURSOR_COMMIT_BEHAVIOR または SQL_CURSOR_ROLLBACK_BEHAVIOR とともに呼び出された場合、このオプションは SQLGetInfo() によって戻された結果に影響します。カーソルの with hold がサポートされていない DB2 (VSE および VM 版) に接続している場合は、 CURSORHOLD の値は無視されます。

このオプションを使用してパフォーマンスを調整することができます。 使用しているアプリケーションが以下のような場合に、このオプションをカーソル保留なし (0) に設定できます。

1. SQLGetInfo() によって戻される SQL_CURSOR_COMMIT_BEHAVIOR または SQL_CURSOR_ROLLBACK_BEHAVIOR 情報に依存する動作がない場合、かつ
2. カーソルを保存して、トランザクションからトランザクションへと次々に渡す必要がない場合。

トランザクション終了後に資源を保持する必要がなくなるので、 DBMS の操作はより効率的になります。

DATABASE

キーワードの説明:

ファイル DSN を使用する際に接続されるサーバー上のデータベース。

db2cli.ini キーワード構文:

DATABASE = データベース名

省略時設定値:

なし

DB2 CLI/ODBC 設定タブ:

このキーワードは「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックを使用して設定することはできません。直接このキーワードを使用するように db2cli.ini ファイルを修正しなければなりません。

適用可能な条件:

PROTOCOL が TCPIP に設定されていること。

参照項目:

200ページの『HOSTNAME』, 212ページの『PROTOCOL』, 215ページの『SERVICENAME』

使用上の注意:

ファイル DSN を使用している場合には、このオプションを使用して、接続先のサーバーのデータベースを指定しなければなりません。この値は、クライアントに指定されるデータベース別名とは関係ありません。これは、サーバー自体のデータベース名に設定されなければなりません。

この設定は、TCPIP に PROTOCOL オプションが設定されている場合にのみ適用されます。

DB2CONNECTVERSION

キーワードの説明:

使用している DB2 コネクトまたは DB2 DDCS ゲートウェイのバージョンを指定します。

db2cli.ini キーワード構文:

DB2CONNECTVERSION = *ゲートウェイのバージョン*

省略時設定値:

5

DB2 CLI/ODBC 設定タブ:

このキーワードは「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックを使用して設定することはできません。直接このキーワードを使用するように db2cli.ini ファイルを修正しなければなりません。

適用可能な条件:

DB2 コネクトまたは DB2 DDCS ゲートウェイを介してデータ・ソースに接続中であること。

使用上の注意:

このオプションは、どのバージョンの DB2 コネクトまたは DB2 DDCS ゲートウェイを使用しているかを DB2 CLI ドライバーに示すために使用します。CLI ドライバーはこの情報を使用して、データ・ソースとの対話を最大限に活用します (たとえば、複数の結果セットを戻すストアード・プロシージャーのサポート)。

5 = バージョン 5 の DB2 コネクト・ゲートウェイを使用していることを示します (省略時値)。

2 = バージョン 2 の DB2 DDCS ゲートウェイを使用していることを示します。

DB2DEGREE

キーワードの説明:

SQL ステートメント実行時の並列性の程度を設定します。

db2cli.ini キーワード構文:

DB2DEGREE = **0** | **1 ～ 32767 の整数値** | **ANY**

省略時設定値:

SET CURRENT DEGREE ステートメントは発行しません。

DB2 CLI/ODBC 設定タブ:

最適化

適用可能な条件:

クラスター・データベース・システムに接続中であること。

使用上の注意:

このオプションは、DB2 バージョン 5.2 以降のサーバーにのみ適用できます。 0 (省略時値) 以外の値を指定した場合、 DB2 CLI は正常に接続された後に次の SQL ステートメントを発行します。

```
SET CURRENT DEGREE value
```

このステートメントは SQL ステートメント実行時の並列性の程度を設定します。 ANY を指定した場合は、データベース・マネージャーが並列性の程度を決定します。

SET CURRENT DEGREE ステートメントの詳細については、 *SQL 解説書* を参照してください。

DB2ESTIMATE

キーワードの説明:

SQL 照会ステートメント準備後に CLI 最適化プログラム見積もりを表示する際のしきい値。

db2cli.ini キーワード構文:

DB2ESTIMATE = **0** | **値の大きい正数**

省略時設定値:

見積もりは戻されません。

DB2 CLI/ODBC 設定タブ:

最適化

適用可能な条件:

GUI アプリケーションが DB2 バージョン 2 以降のサーバーにアクセス中であること。

等価な接続属性:

SQL_ATTR_DB2ESTIMATE

使用上の注意:

このオプションは、DB2 最適化プログラムから戻された見積もりを報告するため、 DB2 CLI が SQL 照会ステートメントの準備の終わりにダイアログ・ボックスを表示するかどうかを決めます。

0 = 見積もりは戻されません (省略時値)。

値の大きい整数 = この値よりしきい値が大きいときに、 DB2 CLI は見積もりを報告するウィンドウを表示します。この値は、PREPARE に関連付けられている SQLCA の SQLERRD(4) フィールドと比較されます。SQLERRD(4) の値が DB2ESTIMATE より大きい場合に、見積ウィンドウが表示されます。

図形ウィンドウには最適化プログラムの見積もりが表示されますが、そこには押しボタンがあって、この照会をさらに継続して実行するかまたは取り消すかをユーザーが選択できるようになっています。

DB2ESTIMATE の推奨値は 60000 です。

このオプションは、DB2 バージョン 2 以降のデータベースへの接続中にのみ関係します。ウィンドウが表示されるには、アプリケーションにグラフィカル・インターフェースがなければなりません。

このオプションを使用すると、 DB2 CLI/ODBC オプション DEFERREDPREPARE はオフとみなされます。

DB2EXPLAIN

キーワードの説明:

Explain スナップショットまたは Explain 表情報 (あるいはその両方) をサーバーから生成するかどうかを判別します。

db2cli.ini キーワード構文:

DB2EXPLAIN = **0** | **1** | **2** | **3**

省略時設定値:

Explain スナップショットも Explain 表情報もサーバーから生成しません。

DB2 CLI/ODBC 設定タブ:

最適化

等価な接続属性:

SQL_ATTR_DB2EXPLAIN

使用上の注意:

このキーワードは、Explain スナップショットまたは Explain 表情報 (あるいはその両方) がサーバーから生成されるようにするかどうかを判別します。

0 = 両方ともオフ (省略時値)

'SET CURRENT EXPLAIN SNAPSHOT=NO' および 'SET CURRENT EXPLAIN MODE=NO' ステートメントがサーバーに送信され、 Explain スナップショット機能と Explain 表情報のキャプチャー機能が両方とも使用不能にされます。

1 = Explain スナップショット機能のみオン

'SET CURRENT EXPLAIN SNAPSHOT=YES' と 'SET CURRENT EXPLAIN MODE=NO' ステートメントがサーバーに送信され、 Explain スナップショット機能は使用可能にされますが、 Explain 表情報キャプチャー機能は使用不能にされます。

2 = Explain 表情報キャプチャー機能のみオン

'SET CURRENT EXPLAIN MODE=YES' と 'SET CURRENT EXPLAIN SNAPSHOT=NO' がサーバーに送信され、 Explain 表情報キャプチャー機能は使用可能にされますが、 Explain スナップショット機能は使用不能にされます。

3 = 両方ともオン

'SET CURRENT EXPLAIN MODE=YES' と 'SET CURRENT EXPLAIN SNAPSHOT=YES' がサーバーに送信され、 Explain スナップショット機能と Explain 表情報キャプチャー機能の両方が使用可能にされます。

Explain 情報は Explain 表に挿入されますが、 Explain 情報が生成される前に Explain 表が作成されていなければなりません。これらの表に関する詳細については、 *SQL 解説書* を参照してください。

現行の許可 ID は、Explain 表に対して INSERT 特権を持っていなければなりません。

オプション 1 は DB2 共通サーバー バージョン 2.1.0 以降のデータベースに接続中にのみ有効です。オプション 2 と 3 は DB2 共通サーバー バージョン 2.1.1 以降のデータベースに接続中にのみ有効です。

DB2OPTIMIZATION

キーワードの説明:

照会最適化レベルを設定します。

db2cli.ini キーワード構文:

DB2OPTIMIZATION = *0 ～ 9 の整数値*

省略時設定値:

SET CURRENT QUERY OPTIMIZATION ステートメントは発行されません。

DB2 CLI/ODBC 設定タブ:

最適化

適用可能な条件:

DB2 バージョン 2 以降のサーバーに接続中であること。

使用上の注意:

このオプションを設定すると、DB2 CLI は正常に接続した後に次の SQL ステートメントを発行します。

```
SET CURRENT QUERY OPTIMIZATION positive number
```

このステートメントは、最適化プログラムが SQL 照会を動作させる照会最適化レベルを指定します。許される最適化レベルについては、 *SQL 解説書* を参照してください。

DBALIAS

キーワードの説明:

8 文字を超えるデータ・ソース名を使用可能にします。

db2cli.ini キーワード構文:

DBALIAS = *データベース別名*

省略時設定値:

DB2 データベース別名を ODBC データ・ソース名として使用します。

DB2 CLI/ODBC 設定タブ:

CLI/ODBC 設定一般

使用上の注意:

このキーワードを使用すると、 8 文字を超える単一バイト文字をデータ・ソース名に使用できるようになります。データ・ソース名 (DSN) は db2cli.ini ファイル (プラットフォーム上にある ASCII ファイル) のセクション見出しを表す、大括弧で囲まれた名前です。通常、セクション見出しはデータベース別名で、最大で 8 バイトの長さです。 データ・ソースを表すのにより長くて分かりやすい名前を使いたいという場合は、セクション見出しに長い名前を配置し、 CATALOG コマンドでこのキーワード値をデータベース別名に設定することができます。以下は、この例です。

```
; The much longer name maps to an 8 single byte character dbalias
[MyMeaningfulName]
DBALIAS=DB2DBT10
```

エンド・ユーザーは接続するデータ・ソースの名前を [MyMeaningfulName] に指定できますが、実際のデータベース別名は DB2DBT10 です。

16 ビットの Windows ODBC 環境では、 ODBC.INI ファイルの [ODBC DATA SOURCES] 項目の下にある次の行を長い別名 (*dbname*) で更新することも必要です。

```
< alias >=IBM DB2 ODBC DRIVER
```

DBNAME

キーワードの説明:

データベース名を指定して、アプリケーションが MVS 表情報を照会するのに要する時間を短縮します。

db2cli.ini キーワード構文:

DBNAME = *dbname*

省略時設定値:

DBNAME 列をフィルターにかけません。

DB2 CLI/ODBC 設定タブ:

エンタープライズ

適用可能な条件:

DB2 (MVS/ESA 版) に接続していること。

参照項目:

214ページの『SCHEMALIST』, 221ページの『TABLETYPE』

使用上の注意:

このオプションは、DB2 (MVS/ESA 版) に接続する時にのみ、また (基本) 表のカタログ情報がアプリケーションによって要求された場合にのみ、使用されます。 DB2 (MVS/ESA 版) サブシステムにある表の数が非常に多い場合、*dbname* を指定して、アプリケーションが表情報を照会するために要する時間を短縮し、アプリケーションにリストされる表の数を少なくすることができます。

このオプションが設定されている場合、ステートメント IN DATABASE *dbname* は、 CREATE TABLE などの種々のステートメントに追加されます。

この値は、DB2 (MVS/ESA 版) システム・カタログ表の DBNAME 列にマップされます。値を指定しない場合、または TABLETYPE で視点、シノニム、システム表、または別名も指定されている場合は、表情報だけが制限され、視点、別名、およびシノニムは DBNAME によっては制限されません。
SCHEMALIST や TABLETYPE と組み合わせて使用し、情報を戻す表の数を一層制限することができます。

DEFAULTPROCLIBRARY

キーワードの説明:

省略時ストアード・プロシージャー・ライブラリーを設定します。

db2cli.ini キーワード構文:

DEFAULTPROCLIBRARY = < 全パス名 >

省略時設定値:

省略時ストアード・プロシージャーをストアード・プロシージャー呼び出しに追加しません。

DB2 CLI/ODBC 設定タブ:

環境

適用可能な条件:

アプリケーションがストアード・プロシージャー・カタログ表を使用していないこと。

使用上の注意:

このオプションは一時的にしか使用できません。その代わりストアード・プロシージャー・カタログ表を使用してください。詳しくは、*SQL 解説書* を参照してください。

このオプションによって指されるライブラリーは、まだ明示的にはライブラリーを指定していないストアード・プロシージャー呼び出しすべてで使用できま

す。サーバー・マシン上にロケーションを指定するので、クライアントではなくオペレーティング・システムのパス形式を使用しなければなりません。詳しくは、 *SQL 解説書* の CALL ステートメントを参照してください。

たとえば、ストアード・プロシージャーがサーバー上のライブラリー・ファイル d:¥terry¥proclib¥comstor にある場合は、 DEFAULTPROCLIBRARY を d:¥terry¥proclib¥comstor に設定してから、ストアード・プロシージャー *func* をライブラリーを指定せずに呼び出します。 その結果の SQL ステートメントは次のようになります。

```
CALL d:¥terry¥proclib¥comstor!func
```

DEFERREDPREPARE

キーワードの説明:

PREPARE 要求を付随する実行要求と結合させることにより、ネットワーク・フローを最小にします。

db2cli.ini キーワード構文:

DEFERREDPREPARE = **0** | **1**

省略時設定値:

準備 (PREPARE) 要求は、実行要求が送信されるまで遅延されます。

DB2 CLI/ODBC 設定タブ:

互換性

適用不能な条件:

DB2ESTIMATE が設定されていること。

等価なステートメント属性:

SQL_ATTR_DEFERRED_PREPARE

使用上の注意:

付随する実行要求が発行されるまで、PREPARE 要求の送信は遅延されます。それから、 2 つの要求は、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に、 (2 つではなく) 1 つのコマンド/応答フローに結合されます。

省略時設定の動作は DB2 バージョン 2 から変更されました。現在は据え置き準備が省略時値になっており、必要なら明示的にオフにしなければなりません。

- 0 = 据え置き準備を使用不能にします。 PREPARE 要求は、その要求が発行され次第実行されます。

- 1 (省略時値) = 据え置き準備を使用可能にします。付随する実行要求が発行されるまで、PREPARE 要求の実行は遅延されます。

  ターゲットの DB2 共通サーバー・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は、その接続に対してはクライアントが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。このことは、これらの値を使用して SQL ステートメントを継続するかどうかを決めているユーザーにとって重要となります。

このオプションは、 CLI/ODBC オプション DB2ESTIMATE がゼロ以外の値に設定されていれば、オフになります。

DISABLEMULTITHREAD

キーワードの説明:

マルチスレッド化を使用不能にします。

db2cli.ini キーワード構文:

DISABLEMULTITHREAD = **0** | **1**

省略時設定値:

マルチスレッド化を使用可能にします。

DB2 CLI/ODBC 設定タブ:

互換性

使用上の注意:

CLI/ODBC ドライバーには、複数の並行スレッドをサポートする能力があります。

このオプションを使用して、マルチスレッド・サポートを使用可能または使用不能にします。

0 = マルチスレッド化は使用可能にされます (省略時値)。
1 = マルチスレッド化を使用不能にします。

マルチスレッド化を使用不能にすると、すべてのスレッドへの呼び出し全部は処理レベルで逐次化されます。 DB2 バージョン 2 で動作の逐次化が必要なマルチスレッド・アプリケーションには、この設定を使用します。

(このオプションは初期設定ファイルの Common (共通) セクションにあるので、 DB2 への接続すべてに適用されます。)

EARLYCLOSE

キーワードの説明:

結果セットの終わりを検出したときに、接続に関連付けられているカーソルを DB2 サーバーの方から早めにクローズしますか？

db2cli.ini キーワード構文:

EARLYCLOSE = **1** | **0**

省略時設定値:

EARLYCLOSE 動作はオンです。

DB2 CLI/ODBC 設定タブ:

互換性

等価なステートメント属性:

SQL_ATTR_EARLYCLOSE

使用上の注意:

このオプションは、最終レコードがクライアントに送信された時点で、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するかどうかを指定します。

0 = サーバーの一時カーソルを早めにはクローズしません。

1 = サーバーの一時カーソルを早めにクローズします (省略時値)。

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。

– ステートメントのブロッキングが適格でない場合
– カーソル・タイプが SQL_CURSOR_FORWARD_ONLY 以外である場合

注: このオプションはいつでも設定できますが、実際に使用されるオプション値は、ステートメントを実行する (カーソルをオープンする) ときに存在しているオプション値です。

GRANTEELIST

キーワードの説明:

アプリケーションが表または列の特権のリストを入手したときに、戻される情報の量を少なくします。

db2cli.ini キーワード構文:

GRANTEELIST = ″ *'userID1', 'userID2',... 'userIDn'* ″

省略時設定値:

結果をフィルターにかけません。

DB2 CLI/ODBC 設定タブ:

エンタープライズ

参照項目:

『GRANTORLIST』

使用上の注意:

このオプションは、アプリケーションがデータベース内の表の特権または表内の列の特権のリストを入手したときに、戻される情報の量を少なくするために使用できます。指定した許可 ID のリストをフィルターとして使用します。指定した ID *に対して* 付与された特権を持つのは、戻された表や列だけです。

このオプションは付与された特権を持つ 1 つまたは複数の許可 ID のリストに対して設定しますが、それぞれの ID は単一引用符で囲み、コンマで区切ります。ストリング全体を二重引用符で囲むことも必要です。たとえば、

```
GRANTEELIST=" 'USER1', 'USER2', 'USER8' "
```

上記の例では、アプリケーションが特定の表に対する特権のリストを入手した場合、 USER1、USER2、または USER8 *に対して* 付与されている特権を持つ列だけが戻されます。

GRANTORLIST

キーワードの説明:

アプリケーションが表または列の特権のリストを入手したときに、戻される情報の量を少なくします。

db2cli.ini キーワード構文:

GRANTORLIST = ″ *'userID1', 'userID2',... 'userIDn'* ″

省略時設定値:

結果をフィルターにかけません。

DB2 CLI/ODBC 設定タブ:
エンタープライズ

参照項目:
198ページの『GRANTEELIST』

使用上の注意:

このオプションは、アプリケーションがデータベース内の表の特権または表内の列の特権のリストを入手したときに、戻される情報の量を少なくするために使用できます。指定した許可 ID のリストをフィルターとして使用します。指定した ID によって 付与された特権を持つのは、戻された表や列だけです。

このオプションは付与された特権を持つ 1 つまたは複数の許可 ID のリストに対して設定しますが、それぞれの ID は単一引用符で囲み、コンマで区切ります。ストリング全体を二重引用符で囲むことも必要です。たとえば、

```
GRANTORLIST=" 'USER1', 'USER2', 'USER8' "
```

上記の例では、アプリケーションが特定の表に対する特権のリストを入手した場合、 USER1、USER2、または USER8 によって 付与されている特権を持つ列だけが戻されます。

GRAPHIC

キーワードの説明:
DB2 CLI が IBM GRAPHIC (2 バイト文字サポート) を、サポートされているデータ・タイプの 1 つとして報告するかどうかを制御します。

db2cli.ini キーワード構文:
GRAPHIC = **0** | **1** | **2** | **3**

省略時設定値:
GRAPHIC はサポートされているデータ・タイプとして報告されません。

DB2 CLI/ODBC 設定タブ:
データ・タイプ

使用上の注意:

このオプションは、2 つの関連する情報をアプリケーションから戻す方法を制御します。

- SQLGetTypeInfo() が呼び出されたときに、 DB2 CLI が IBM GRAPHIC (2 バイト文字サポート) を、サポートされているデータ・タイプの 1 つとして報告するかどうか。 SQLGetTypeInfo() は、現行接続で DB2 データベースによってサポートされているデータ・タイプをリストします。
- 図形列の長さを報告するためにどの単位を使用するか。このことは、出力引き数、または結果セットの一部のいずれかで長さ / 精度を返す、すべての DB2 CLI/ODBC 関数に適用されます。
  0 = IBM GRAPHIC データ・タイプをサポートされたタイプとしては報告しません。図形列の長さは DBCS の文字数として戻されます。 (省略時値)
  1 = IBM GRAPHIC データ・タイプをサポートされたタイプとして報告します。図形列の長さは DBCS の文字数として戻されます。
  2 = IBM GRAPHIC データ・タイプをサポートされたタイプとしては報告しません。図形列の長さはバイト数として戻されます。 (これは、**Microsoft Access** 1.1-J** および **Microsoft Query**-J** で必要です。)
  3 = 設定値 1 と 2 の組み合わせ。 IBM GRAPHIC データ・タイプはサポートされたデータ・タイプとして報告されます。図形列の長さはバイト数として戻されます。

省略時設定では、多くの場合市販のアプリケーションがこのデータ・タイプを認識せず、適切な処理を行えないため、GRAPHIC は戻されません。

HOSTNAME

キーワードの説明:

ファイル DSN で使用するサーバー・システムのホスト名または IP アドレス。

db2cli.ini キーワード構文:

HOSTNAME = *ホスト名* | *IP アドレス*

省略時設定値:

なし

DB2 CLI/ODBC 設定タブ:

ファイル DSN

適用可能な条件:

PROTOCOL が TCPIP に設定されていること。

参照項目:

212ページの『PROTOCOL』, 215ページの『SERVICENAME』

使用上の注意:

このオプションと SERVICENAME オプションを組み合わせて使用することにより、このクライアント・マシンから DB2 を実行しているサーバーへの TCP/IP 接続に必要な属性を指定できます。これら 2 つの値は、TCPIP に PROTOCOL オプションが設定されている場合にのみ適用されます。

サーバー・システムのホスト名または IP アドレスのいずれかを指定します。

IGNOREWARNINGS

キーワードの説明:

警告を無視します。

db2cli.ini キーワード構文:

IGNOREWARNINGS = **0** | **1**

省略時設定値:

警告は通常通りに戻されます。

DB2 CLI/ODBC 設定タブ:

サービス

参照項目:

230ページの『WARNINGLIST』, 『IGNOREWARNLIST』

使用上の注意:

ごくまれに、アプリケーションが警告メッセージを適正に処理しないことがあります。このオプションを使用することにより、データベース・マネージャーからの警告がアプリケーションに渡されないように指定することができます。

0 = 通常通りに警告を報告します (省略時値)。

1 = データベース・マネージャーの警告は無視され、SQL_SUCCESS が戻されます。 DB2 CLI/ODBC ドライバーからの警告は通常の操作に必要なので、そのまま戻されます。

このオプションは単独でも使用できますが、 WARNINGLIST CLI/ODBC 構成キーワードと組み合わせて使用することもできます。

IGNOREWARNLIST

キーワードの説明:

指定した sqlstate を無視します。

db2cli.ini キーワード構文:

IGNOREWARNLIST = 『'sqlstate1', 'sqlstate2', ...』

省略時設定値:

警告は通常通りに戻されます。

DB2 CLI/ODBC 設定タブ:

このキーワードは「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックを使用して設定することはできません。直接このキーワードを使用するように db2cli.ini ファイルを修正しなければなりません。

参照項目:

230ページの『WARNINGLIST』, 201ページの『IGNOREWARNINGS』

使用上の注意:

まれに、アプリケーションが一部の警告メッセージを適切に処理しないものの、すべての警告メッセージを無視したくはないという場合があります。 このキーワードを使用することにより、どの警告をアプリケーションに渡さないようにするかを指定できます。 すべてのデータベース・マネージャー警告を無視する場合は、 IGNOREWARNINGS キーワードを使用してください。

sqlstate が IGNOREWARNLIST と WARNINGLIST との両方に含まれる場合、それらはすべて無視されます。

各 sqlstate は大文字にし、単一引用符で囲み、各値をコンマで区切らなければなりません。 ストリング全体を二重引用符で囲むことも必要です。 たとえば、

```
IGNOREWARNLIST="'01000', '01004','01504'"
```

KEEPCONNECT

キーワードの説明:

キャッシュする接続の数。

db2cli.ini キーワード構文:

KEEPCONNECT = **0** | **正の整数**

省略時設定値:

接続をキャッシュしません。

DB2 CLI/ODBC 設定タブ:

トランザクション

使用上の注意:

0 = データベース接続をキャッシュしません (省略時値)。

このオプションをゼロより大きい値に設定すると、同じ接続情報を使用して同じデータベースとの間で定期的に接続と切断を繰り返すアプリケーションの処理速度は向上します。

一回ごとに接続のクローズとオープンを繰り返すのではなく、 CLI/ODBC ドライバーは接続をオープンにしたままで、接続情報をキャッシュします。次に同じデータベースへの接続要求が生じたときには、既存の接続が使用されます。このようにして、最初の接続のクローズと次の接続の再オープンに要する時間、資源、およびネットワーク・フローを節約することができます。

このオプションに設定した値で、キャッシュするデータベース接続の数が指定されます。最大値はシステム資源の数までに限定されていますが、通常はこの動作の利点を活用するアプリケーションに 1 または 2 の値を設定すれば十分です。

KEEPSTATEMENT

キーワードの説明:

キャッシュするステートメント・ハンドルの数。

db2cli.ini キーワード構文:

KEEPSTATEMENT = **5** | **正の整数**

省略時設定値:

5 つのステートメント・ハンドルをキャッシュします。

DB2 CLI/ODBC 設定タブ:

最適化

使用上の注意:

省略時設定では、5 つのステートメント・ハンドルに必要なメモリーがキャッシュされます。ステートメント・ハンドルがクローズされると、そのハンドルに使用されていたメモリーは割り振り解除されますが、次のステートメント・ハンドルが割り振られるとまた使用されます。

このオプションに設定した値で、キャッシュするステートメント・ハンドルの数が決まります。この値を 4 以下に設定し、ステートメント・キャッシュに使用するメモリーの量を明示的に減らすことができます。また値を 6 以上に増やして、アプリケーションのパフォーマンスを向上させ、多数のステートメント・セットをオープン、クローズ、そして再オープンするようにもできます。

キャッシュされるステートメント・ハンドルの最大数は、システム資源によって決まります。

LOBMAXCOLUMNSIZE

キーワードの説明:

LOB データ・タイプの省略時 COLUMN_SIZE を一時変更します。

db2cli.ini キーワード構文:

LOBMAXCOLUMNSIZE = *ゼロより大きい整数*

省略時設定値:

2 ギガバイト (DBCLOB の場合は 1 ギガバイト)

DB2 CLI/ODBC 設定タブ:

データ・タイプ

適用可能な条件:

LONGDATACOMPAT オプションを使用していること。

参照項目:

『LONGDATACOMPAT』

使用上の注意:

このオプションを指定すると、`SQLGetTypeInfo()` によって戻され、SQL_CLOB、SQL_BLOB、および SQL_DBCLOB SQL データ・タイプ用に指定される COLUMN_SIZE 列の値を、 2 ギガバイト (DBCLOB の場合は 1 ギガバイト) に一時変更します。それ以後に LOB 列を含む CREATE TABLE ステートメントでは、省略時値ではなくここに設定した列サイズが使用されます。

LONGDATACOMPAT

キーワードの説明:

LOB を長形式データ・タイプまたはラージ・オブジェクト・タイプとして報告します。

db2cli.ini キーワード構文:

LONGDATACOMPAT = **0** | **1**

省略時設定値:

LOB データ・タイプをラージ・オブジェクト・タイプとして参照します。

DB2 CLI/ODBC 設定タブ:
データ・タイプ

参照項目:
204ページの『LOBMAXCOLUMNSIZE』

等価な接続属性:
SQL_ATTR_LONGDATA_COMPAT

使用上の注意:

このオプションは、ラージ・オブジェクト (LOB) 列を含むデータベースを処理する際にアプリケーションが予期しているデータ・タイプが何かを DB2 CLI に指示します。

データベースのデータ・タイプ	ラージ・オブジェクト (0 : 省略時値)	長形式データ・タイプ (1)
CLOB	SQL_CLOB	SQL_LONGVARCHAR
BLOB	SQL_BLOB	SQL_LONGVARBINARY
DBCLOB	SQL_DBCLOB	SQL_LONGVARGRAPHIC

このオプションは、ラージ・オブジェクト・データ・タイプを処理できない ODBC アプリケーションを実行する際には便利です。

このオプションに DB2 CLI/ODBC オプションの LOBMAXCOLUMNSIZE を組み合わせて使用することにより、データに対して宣言される省略時サイズを減らすことができます。

MAXCONN

キーワードの説明:
各アプリケーションに許可される最大接続数。

db2cli.ini キーワード構文:
MAXCONN = **0** | **正数**

省略時設定値:
システム資源が許可する最大接続数。

DB2 CLI/ODBC 設定タブ:
トランザクション

等価な接続属性:
SQL_ATTR_MAXCONN

使用上の注意:

このオプションは、各 CLI/ODBC アプリケーションに許可される最大接続数を指定するときに使用します。このオプションは、各アプリケーションがオープンする最大接続数を管理者が制限したい場合に、その最大数の管理プログラムの代わりに使用できます。値 0 は、*制限なし* を表すのに使用することができます。つまり、アプリケーションはシステム資源で許可される範囲で最大数の接続をオープンすることができます。

OS/2 および WIN32 プラットフォーム (Windows NT および Windows 95) では、 NetBIOS プロトコルを使用している場合、この値がアプリケーションによって並行してセットアップされる接続 (NetBIOS セッション) の数に対応します。 OS/2 NetBIOS の値の範囲は、1 から 254 です。 0 (省略時値) を指定すると、5 つの *予約済み* 接続となります。*予約済み NetBIOS セッション* は他のアプリケーションからは使用できません。このオプションで指定された接続数は、 DB2 NetBIOS プロトコルがリモート・サーバーへの接続に使用するアダプターすべてに適用されます (アダプター番号は NetBIOS ノードのノード・ディレクトリーに指定されます)。

MODE

キーワードの説明:

省略時接続モード。

db2cli.ini キーワード構文:

MODE = **SHARE** | **EXCLUSIVE**

省略時設定値:

SHARE

DB2 CLI/ODBC 設定タブ:

トランザクション

適用不能な条件:

DRDA に接続中であること。

使用上の注意:

CONNECT モードを SHARE または EXCLUSIVE に設定します。接続時にアプリケーションによって設定されたモードがあれば、この値は無視されます。省略時値は SHARE です。

注: EXCLUSIVE は DRDA 接続には許可されていません。 CONNECT ステートメントの詳細については、 *SQL 解説書* を参照してください。

MULTICONNECT

キーワードの説明:

SQLConnect() 要求が物理データベース接続にマップされる方法。

db2cli.ini キーワード構文:

MULTICONNECT = **0** | **1**

省略時設定値:

アプリケーションが SQLConnect() 要求を出すたびに、物理データベース接続が確立されます。

DB2 CLI/ODBC 設定タブ:

トランザクション

使用上の注意:

このオプションは、 SQLConnect() 要求が物理データベース接続にマップされる方法を指定するときに使用します。

1 = アプリケーションが SQLConnect() 要求を出すたびに、物理データベース接続が確立されます。

0 = 複数の接続が 1 つの物理接続にマップされ、1 つの接続が使用されます。− アプリケーションのすべての接続は 1 つの物理接続にマップされます。これは以下の場合に役立つことがあります。

- ODBC アプリケーションが非常に多くの接続を使用しているために、ファイル・ハンドルを使い尽くしてしまった場合
- アプリケーションがデータベースからデータだけを読み取る場合
- アプリケーションが自動コミットを (一部の場合に) 使用する場合
- アプリケーションが 1 つの接続で複数のステートメントを使用する代わりに複数の接続をオープンする場合、複数の接続を使用すると、接続の相互間でロッキング競合が生じることがあります。

MULTICONNECT が 0 に設定されている場合、キーワード DISABLEMULTITHREAD を使用してマルチスレッド化を使用不可にしなければなりません。

注: MULTICONNECT の設定が解除されると、ステートメントはすべて同じ接続で、したがって同じトランザクションで実行されます。これは、ロールバックを実行すると、すべての接続のすべてのステートメントをロールバ

ックしてしまうということです。ロールバックを実行する前に、アプリケーションが MULTICONNECT オフで動作するように設計されていることを確認してください。そのように設計されていないと、アプリケーションが異常な動作をする場合があります。

(このオプションは初期設定ファイルの Common (共通) セクションにあるので、 DB2 への接続すべてに適用されます。)

OPTIMIZEFORNROWS

キーワードの説明:

SELECT ステートメントすべてに ″OPTIMIZE FOR n ROWS″ 文節を追加します。

db2cli.ini キーワード構文:

OPTIMIZEFORNROWS = *整数*

省略時設定値:

文節は追加されません。

DB2 CLI/ODBC 設定タブ:

最適化

等価なステートメント属性:

SQL_ATTR_OPTIMIZE_FOR_NROWS

使用上の注意:

このオプションは、SELECT ステートメントすべてに ″OPTIMIZE FOR n ROWS″ 文節を追加します (ここで、 n は 1 以上の整数です)。 n を 0 (省略時値) に設定すると、この文節は追加されません。

OPTIMIZE FOR n ROWS 文節の影響に関する詳細については、 *管理の手引き* を参照してください。

OPTIMIZESQLCOLUMNS

キーワードの説明:

明示的なスキーマおよび表名を使用して `SQLColumns()` 呼び出しを最適化します。

db2cli.ini キーワード構文:

OPTIMIZESQLCOLUMNS = **0** | **1**

省略時設定値:

0 - すべての列情報が戻されます

DB2 CLI/ODBC 設定タブ:

このキーワードは「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックを使用して設定することはできません。直接このキーワードを使用するように db2cli.ini ファイルを修正しなければなりません。

等価な接続属性:

SQL_ATTR_OPTIMIZE_SQLCOLUMNS

使用上の注意:

OPTIMIZESQLCOLUMNS がオン (1 に設定) の場合、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 % (すべての列) が列名に指定された場合、 SQLColumns() へのすべて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、 COLUMN_DEF 情報 (列に関する省略時ストリングを含む) は戻されません。 AS/400 データベースに接続しているとき、 SQLColumns() から列に関して戻される、データ・タイプが NUMERIC の情報は不正確なものとなります。アプリケーションがその情報を必要としない場合、最適化をオンにしてパフォーマンスを向上させることができます。

アプリケーションが COLUMN_DEF 情報を必要とする場合、OPTIMIZESQLCOLUMNS を 0 に設定してください。これが省略時値です。

PATCH1

キーワードの説明:

ODBC アプリケーションで起きることが分かっている問題に一時修正処置を適用します。

db2cli.ini キーワード構文:

PATCH1 = **{ 0 | 1 | 2 | 4 | 8 | 16 | ... }**

省略時設定値:

一時修正処置を適用しません。

DB2 CLI/ODBC 設定タブ:

サービス

参照項目:

210ページの『PATCH2』

使用上の注意:

このキーワードは、 ODBC アプリケーションで起きることが分かっている問題に一時修正処置を適用する際に使用します。指定できる値は、なし、1 つ、または複数の一時修正処置です。このキーワードに指定したパッチの値に、同時に設定した PATCH2 の値を組み合わせて使用することができます。

「DB2 CLI/ODBC 設定 (DB2 CLI/ODBC Settings)」ノートブックを使用して、使用するパッチを 1 つまたは複数選択できます。 db2cli.ini ファイルにある値を設定し、かつ複数のパッチ値を使用したい場合は、ただその値もキーワード値に追加すれば十分です。 たとえば、パッチ 1、4、および 8 を使用したい場合は、 PATCH1=13 と指定します。

0 = 一時修正処置は使用しません (省略時値)

「DB2 CLI/ODBC 設定 (DB2 CLI/ODBC Settings)」ノートブックには、値のリストがあります。このリストの値を更新する方法については、「DB2」フォルダーの「サービス (Service)」フォルダーを選択してください。同じ情報が、 README ファイルにも記載されています (ご使用のプラットフォームに現行パッチ値がない場合は、 README ファイルにも該当する項目はありません)。

PATCH2

キーワードの説明:

CLI/ODBC アプリケーションで起きることが分かっている問題に一時修正処置を適用します。

db2cli.ini キーワード構文:

PATCH2 = *"パッチ値 1, パッチ値 2, パッチ値 3, ..."*

省略時設定値:

一時修正処置を適用しません。

DB2 CLI/ODBC 設定タブ:

このキーワードは「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックを使用して設定することはできません。直接このキーワードを使用するように db2cli.ini ファイルを修正しなければなりません。

参照項目:

209ページの『PATCH1』

使用上の注意:

このキーワードは、 CLI/ODBC アプリケーションで起きることが分かっている問題に一時修正処置を適用する際に使用します。指定できる値は、なし、1

つ、または複数の一時修正処置です。このキーワードに指定したパッチの値に、同時に設定した PATCH1 の値を組み合わせて使用することができます。

複数のパッチを指定する場合、パッチ値は 1 つのストリング内にコンマで区切って指定します (PATCH1 とは異なります。 PATCH1 ではすべての値を足した合計を使用します)。

0 = 一時修正処置は使用しません (省略時値)

PATCH2 値に 3、4 および 8 を設定するには、次のようにします。

```
PATCH2="3, 4, 8"
```

PATCH2 値は、 README ファイルに記載されています (ご使用のプラットフォームに現行パッチ値がない場合は、 README ファイルにも該当する項目はありません)。

POPUPMESSAGE

キーワードの説明:

CLI/ODBC がエラーを生成するたびにメッセージ・ボックスを表示します。

db2cli.ini キーワード構文:

POPUPMESSAGE = **0** | **1**

省略時設定値:

メッセージ・ボックスを表示しません。

DB2 CLI/ODBC 設定タブ:

サービス

適用可能な条件:

OS/2 または Windows アプリケーションを実行中であること。

参照項目:

215ページの『SQLSTATEFILTER』

使用上の注意:

DB2 CLI が SQLGetDiagRec() または SQLError() を使用して検索可能なエラーを生成するたびに、メッセージ・ボックスを表示します。ユーザーにメッセージを報告しないアプリケーションのデバッグには便利です。

0 = メッセージ・ボックスを表示しません (省略時値)

1 = メッセージ・ボックスを表示します

PROTOCOL

キーワードの説明:

ファイル DSN に使用される通信プロトコル。

db2cli.ini キーワード構文:

PROTOCOL = **TCPIP**

省略時設定値:

なし

DB2 CLI/ODBC 設定タブ:

ファイル DSN

参照項目:

200ページの『HOSTNAME』, 215ページの『SERVICENAME』

使用上の注意:

ファイル DSN を使用する際にサポートされるプロトコルは TCP/IP だけです。オプションをストリング TCPIP (スラッシュは入れない) に設定してください。

このオプションが設定される際には、以下のオプションも設定されていなければなりません。

- 187ページの『DATABASE』
- 215ページの『SERVICENAME』
- 200ページの『HOSTNAME』

PWD

キーワードの説明:

省略時パスワードを定義します。

db2cli.ini キーワード構文:

PWD = *パスワード*

省略時設定値:

なし

DB2 CLI/ODBC 設定タブ:

CLI/ODBC 設定一般

使用上の注意:

このパスワード 値は、接続時にアプリケーションによってパスワードが提供されなかった場合に使用されます。

平文で保管されるので、パスワード情報は保護されません。

QUERYTIMEOUTINTERVAL

キーワードの説明:

照会タイムアウトの検査の間の遅延 (秒単位)

db2cli.ini キーワード構文:

QUERYTIMEOUTINTERVAL = **0** | **正の整数**

省略時設定値:

5 秒

DB2 CLI/ODBC 設定タブ:

このキーワードは「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックを使用して設定することはできません。直接このキーワードを使用するように db2cli.ini ファイルを修正しなければなりません。

等価なステートメント属性:

SQL_ATTR_QUERY_TIMEOUT

使用上の注意:

アプリケーションは、SQLSetStmtAttr() 関数を使用して SQL_ATTR_QUERY_TIMEOUT ステートメント属性を設定できます。これは、アプリケーションに戻る前に、 SQL ステートメントが実行されるまで待機する秒数を指定します。

QUERYTIMEOUTINTERVAL 構成キーワードは、照会が完了したかどうかを調べる検査と検査の間に CLI ドライバーが待機する時間を指定するために使用されます。

たとえば、SQL_ATTR_QUERY_TIMEOUT が 25 秒 (25 秒待機後にタイムアウト)、 QUERYTIMEOUTINTERVAL が 10 秒 (10 ごとに照会を実行) に設定されているとします。照会は、30 秒経過しないとタイムアウトになりません (25 秒経過した後に実行される最初の検査)。

SQL_ATTR_QUERY_TIMEOUT で設定されている値が小さすぎて、照会がタイムアウトにならない場合があります。アプリケーションを修正できない場合 (たとえば、第三者の ODBC アプリケーションであるため)、

QUERYTIMEOUTINTERVAL を 0 に設定することができます。こうすると、CLI ドライバーは SQL_ATTR_QUERY_TIMEOUT 設定を無視します。

(このオプションは初期設定ファイルの Common (共通) セクションにあるので、 DB2 への接続すべてに適用されます。)

SCHEMALIST

キーワードの説明:

表情報を照会するために使用されるスキーマを制限します。

db2cli.ini キーワード構文:

SCHEMALIST = ″ 'スキーマ *1*', 'スキーマ *2*',... 'スキーマ *N*' ″

省略時設定値:

なし

DB2 CLI/ODBC 設定タブ:

エンタープライズ

使用上の注意:

SCHEMALIST は、 DBMS 内のすべての表をリストするアプリケーションの省略時値をより制限の強いものにすることによって、パフォーマンスを向上させます。

データベースに定義された表の数が非常に多い場合、スキーマ・リストを指定して、アプリケーションが表情報を照会するために要する時間を短縮し、アプリケーションにリストされる表の数を少なくすることができます。スキーマ名は大文字小文字を区別され、また各スキーマ名をコンマで区切り、二重引用符で囲む必要があります。ストリング全体を二重引用符で囲むことも必要です。たとえば、

```
SCHEMALIST="'USER1','USER2','USER3'"
```

DB2 (MVS/ESA 版) の場合、CURRENT SQLID もこのリストに加えることができますが、単一引用符は使用しません。たとえば次のようにします。

```
SCHEMALIST="'USER1',CURRENT SQLID,'USER3'"
```

ストリングの最大長は 256 文字です。

このオプションを DBNAME および TABLETYPE と組み合わせて使用し、情報を戻す表の数を一層制限することができます。

SERVICENAME

キーワードの説明:

ファイル DSN で使用されるサーバー・システムのサービス名またはポート番号。

db2cli.ini キーワード構文:

SERVICENAME = *サービス名* | *ポート番号*

省略時設定値:

なし

DB2 CLI/ODBC 設定タブ:

ファイル DSN

適用可能な条件:

PROTOCOL が TCPIP に設定されていること。

参照項目:

212ページの『PROTOCOL』, 200ページの『HOSTNAME』

使用上の注意:

このオプションと HOSTNAME オプションを組み合わせて使用することにより、このクライアント・マシンから DB2 を実行しているサーバーへの TCP/IP 接続に必要な属性を指定できます。これら 2 つの値は、TCPIP に PROTOCOL オプションが設定されている場合にのみ適用されます。

サーバー・システムのサービス名またはポート番号のいずれかを指定します。

SQLSTATEFILTER

キーワードの説明:

定義された SQLSTATES のエラー・メッセージを表示しません。

db2cli.ini キーワード構文:

SQLSTATEFILTER = ″ '*XXXXX*', '*YYYYY*', ... ″

省略時設定値:

なし

DB2 CLI/ODBC 設定タブ:

サービス

適用可能な条件:

POPUPMESSAGE オプションがオンになっていること。

参照項目:

211ページの『POPUPMESSAGE』

使用上の注意:

POPUPMESSAGE オプションと組み合わせて使用します。 こうすることにより、定義された状態に関連付けられたエラーを DB2 CLI が表示することはありません。

各 SQLSTATE は大文字にし、単一引用符で囲み、各値をコンマで区切らなければなりません。ストリング全体を二重引用符で囲むことも必要です。たとえば、

```
SQLSTATEFILTER=" 'HY1090', '01504', '01508' "
```

STATICCAPFILE

キーワードの説明:

キャプチャー・ファイル名、および任意選択でこれを保管するディレクトリーを指定します。

db2cli.ini キーワード構文:

STATICCAPFILE = < **完全ファイル名** >

省略時設定値:

なし。キャプチャー・ファイル名を指定する必要があります。

DB2 CLI/ODBC 設定タブ:

静的 SQL

適用可能な条件:

STATICMODE が Capture または Match に設定されていること。

参照項目:

217ページの『STATICLOGFILE』, 217ページの『STATICMODE』, 218ページの『STATICPACKAGE』

使用上の注意:

このキーワードは、キャプチャー・ファイル名、および任意選択でこれが保管されるディレクトリーを指定するのに使用されます。

CLI/ODBC アプリケーションを静的 SQL として実行することについては、STATICMODE キーワードを参照してください。

STATICLOGFILE

キーワードの説明:

静的プロファイル作成ログ・ファイル名、および任意選択でこれを保管するディレクトリーを指定します。

db2cli.ini キーワード構文:

STATICLOGFILE = < **完全ファイル名** >

省略時設定値:

静的プロファイル作成ログが作成されません。ファイル名にパス名が指定されていない場合には、現行のパスが使用されます。

DB2 CLI/ODBC 設定タブ:

静的 SQL

適用可能な条件:

STATICMODE が Capture または Match に設定されていること。

参照項目:

216ページの『STATICCAPFILE』, 『STATICMODE』, 218ページの『STATICPACKAGE』

使用上の注意:

このキーワードは、静的プロファイル作成ログ・ファイル名、および任意選択でこれが保管されるディレクトリーを指定するのに使用されます。

CLI/ODBC アプリケーションを静的 SQL として実行することについては、STATICMODE キーワードを参照してください。

STATICMODE

キーワードの説明:

CLI/ODBC アプリケーションが SQL を取り込むかどうか、またはこの DSN に静的 SQL パッケージを使用するかどうかを指定します。

db2cli.ini キーワード構文:

STATICMODE = **DISABLED** | **CAPTURE** | **MATCH**

省略時設定値:

0 使用不可 - SQL ステートメントは取り込まれず、静的 SQL パッケージは使用されません。

DB2 CLI/ODBC 設定タブ:

静的 SQL

参照項目:

216ページの『STATICCAPFILE』, 『STATICPACKAGE』, 217ページの『STATICLOGFILE』

使用上の注意:

このオプションを使用すると、この DSN について CLI/ODBC アプリケーションにより発行される SQL をどのように処理するかを指定することができます。

- DISABLED = 静的モードは使用不可。特に処理は実行されません。CLI/ODBC ステートメントは動的 SQL として実行されますが、変更はありません。これが省略時値です。
- CAPTURE = キャプチャー・モード。 CLI/ODBC ステートメントを動的 SQL として実行します。 SQL ステートメントが正常に実行されると、これらはファイル (キャプチャー・ファイルと呼ばれる) に取り込まれて、後で DB2CAP コマンドによりバインドされます。
- MATCH = 突き合わせモード。 STATICCAPFILE に指定されたキャプチャー・ファイルに、一致したステートメントが見つかった場合には、CLI/ODBC ステートメントを静的 SQL ステートメントとして実行します。キャプチャー・ファイルはまず、DB2CAP コマンドによりバインドしなければなりません。詳しくは、コマンド解説を参照してください。

CLI/ODBC アプリケーションを静的 SQL として実行することについての詳細は、リリース情報と、インターネットの http://www.ibm.com/software/data/db2/udb/staticcli を参照してください。

STATICPACKAGE

キーワードの説明:

静的プロファイル作成機能と共に使用するパッケージを指定します。

db2cli.ini キーワード構文:

STATICPACKAGE = *collection_id.package_name*

省略時設定値:

なし。パッケージ名を指定する必要があります。

DB2 CLI/ODBC 設定タブ:

静的 SQL

適用可能な条件:

STATICMODE が CAPTURE に設定されていること。

参照項目:

216ページの『STATICCAPFILE』，217ページの『STATICMODE』，217ページの『STATICLOGFILE』

使用上の注意:

このキーワードは、アプリケーションが突き合わせモードで実行されるときに使用されるパッケージを指定するのに使用されます。まず、突き合わせモードを使用してキャプチャー・ファイルを作成する必要があります。

指定されているパッケージ名の先頭の 7 文字のみ使用されます。それぞれの分離レベルを示すために、1 バイトの接尾部が追加されます。次のようになります。

- 0 = 非コミット読み取り (UR)
- 1 = カーソル固定 (CS)
- 2 = 読み取り固定 (RS)
- 3 = 反復可能読み取り (RR)
- 4 = コミットなし (NC)

CLI/ODBC アプリケーションを静的 SQL として実行することについては、STATICMODE キーワードを参照してください。

SYNCPOINT

キーワードの説明:

複数のデータベース (DUOW) 接続の間でコミットとロールバックを整合させる方法。

db2cli.ini キーワード構文:

SYNCPOINT = **1** | **2**

省略時設定値:

1 フェーズ・コミット。

DB2 CLI/ODBC 設定タブ:

トランザクション

適用可能な条件:

省略時接続タイプが調整接続 (CONNECTTYPE=2) に設定されていること。

参照項目:

182ページの『CONNECTTYPE』

等価な接続属性:

SQL_ATTR_SYNC_POINT

使用上の注意:

複数のデータベース (DUOW) 接続の間でコミットとロールバックを整合させる方法を指定するときに、このオプションを使用します。このオプションは、省略時接続タイプが「調整済み接続 (Coordinated Connections)」(CONNECTTYPE = 2) に設定されている場合にのみ有効です。

- 1 = ONEPHASE (省略時値)

  2 フェーズ・コミットを実行するためにトランザクション・マネージャーは使用しませんが、各データベースが行った作業を 1 つの多重データベース・トランザクションでコミットするには 1 フェーズ・コミットが使用されます。
- 2 = TWOPHASE

  2 フェーズ・コミットをサポートする複数のデータベースの間で 2 フェーズ・コミットを調整するには、トランザクション・マネージャーが必要です。

SYSSCHEMA

キーワードの説明:

SYSIBM (または SYSTEM、QSYS2) の代わりに探索する代替スキーマを指定します。

db2cli.ini キーワード構文:

SYSSCHEMA = *sysschema*

省略時設定値:

代替スキーマを指定しません。

DB2 CLI/ODBC 設定タブ:

エンタープライズ

使用上の注意:

このオプションは、 DB2 CLI や ODBC カタログ関数呼び出しが発行されてシステム・カタログ情報を取得するときに、 SYSIBM (または SYSTEM、QSYS2) の代わりに探索する代替スキーマを指定します。

このスキーマ名を使用して、次の各システム・カタログ表の行のサブセットから成る視点の集まりを、システム管理者が定義することができます。

DB2 ユニバーサル・データベース	DB2 (MVS/ESA 版)	DB2 (VSE および VM 版)	OS/400	DB2 ユニバーサル・データベース (AS/400 版)
SYSTABLES	SYSTABLES	SYSCATALOG	SYSTABLES	SYSTABLES
SYSCOLUMNS	SYSCOLUMNS	SYSCOLUMNS	SYSCOLUMNS	SYSCOLUMNS
SYSINDEXES	SYSINDEXES	SYSINDEXES	SYSINDEXES	SYSINDEXES
SYSTABAUTH	SYSTABAUTH	SYSTABAUTH		SYSCST
SYSRELS	SYSRELS	SYSKEYCOLS		SYSKEYCST
SYSDATATYPES	SYSSYNONYMS	SYSSYNONYMS		SYSCSTCOL
SYSPROCEDURES	SYSKEYS	SYSKEYS		SYSKEYS
SYSPROCPARMS	SYSCOLAUTH	SYSCOLAUTH		SYSREFCST
	SYSFOREIGNKEYS			
	SYSPROCEDURES **1**			
	SYSDATABASE			

1 DB2 (MVS/ESA 版) 4.1 のみ

たとえば、システム・カタログ表の視点のセットが ACME スキーマにある場合、 SYSIBM.SYSTABLES の視点は ACME.SYSTABLES なので、 SYSSCHEMA を ACME に設定しなければなりません。

定義および使用するシステム・カタログ表の視点の数を限定すると、アプリケーションによってリストされる表の数も少なくてすみ、結果として表情報の照会にアプリケーションが要する時間を短縮することができます。

値を指定しない場合、省略時値は以下のとおりです。

- DB2 ユニバーサル・データベースでは SYSCAT または SYSIBM
- バージョン 2.1 より前の DB2 (共通サーバー用)、DB2 (MVS/ESA 版)、および OS/400 では SYSIBM
- DB2 (VSE および VM 版) では SYSTEM
- DB2 ユニバーサル・データベース (AS/400 版) では QSYS2

このキーワードを SCHEMALIST および TABLETYPE (DB2 (MVS/ESA 版) では DBNAME も) と組み合わせて使用し、情報を戻す表の数をさらに制限することができます。

TABLETYPE

キーワードの説明:

表情報を照会したときに戻される TABLETYPES の省略時リストを定義します。

db2cli.ini キーワード構文:

TABLETYPE = ″ *'TABLE'* | *,'ALIAS'* | *,'VIEW'* | , *'INOPERATIVE VIEW'* | , *'SYSTEM TABLE'* | *,'SYNONYM'* ″

省略時設定値:

TABLETYPES の省略時リストは定義しません。

DB2 CLI/ODBC 設定タブ:

エンタープライズ

使用上の注意:

データベースに定義された表の数が非常に多い場合、TABLETYPE ストリングを指定して、アプリケーションが表情報を照会するために要する時間を短縮し、アプリケーションにリストされる表の数を少なくすることができます。

値はいくつでも指定できます。各タイプは大文字で表記し、単一引用符で囲み、コンマで区切る必要があります。ストリング全体を二重引用符で囲むことも必要です。たとえば、

```
TABLETYPE="'TABLE','VIEW'"
```

このオプションを DBNAME および SCHEMALIST と組み合わせて使用し、情報を戻す表の数を一層制限することができます。

TABLETYPE は、データベース内の表、視点、別名、およびシノニムのリストを取り出す DB2 CLI 関数に省略時値を提供するときに使用します。アプリケーションが関数呼び出しに表タイプを指定しておらず、かつこのキーワードも指定していない場合は、すべての表タイプに関する情報が戻されます。アプリケーションが関数を呼び出す際に *tabletype* に値を設定した場合には、その引き数値がこのキーワード値を指定変更します。

TABLETYPE に TABLE 以外の値が含まれている場合、 DBNAME キーワード設定値を使用して情報を特定の DB2 (MVS/ESA 版) データベースに制限することはできません。

TEMPDIR

キーワードの説明:

LOB フィールドに関連付けられている一時ファイル用に使用されるディレクトリーを定義します。

db2cli.ini キーワード構文:

TEMPDIR = < 全パス名 >

省略時設定値:

システム一時ディレクトリーを使用します。

DB2 CLI/ODBC 設定タブ:

環境

使用上の注意:

ラージ・オブジェクト (CLOBS、BLOBS など) を処理する場合、情報を保管する一時ファイルがクライアント・マシン上に作成されることがよくあります。このオプションを使用すると、この一時ファイルの場所を指定することができます。何も指定しないと、システム一時ディレクトリーが使用されます。

このキーワードは、db2cli.ini ファイル内のデータ・ソース・セクションにあり、次の構文を取ります。

- TempDir= F:¥DB2TEMP

ラージ・オブジェクトにアクセスされたとき、指定したディレクトリーに一時ファイルが作成できない場合には、 SQLSTATE の HY507 が戻されます。

TRACE

キーワードの説明:

DB2 CLI/ODBC トレース機能をオンにします。

db2cli.ini キーワード構文:

TRACE = **0** | **1**

省略時設定値:

トレース情報をキャプチャーしません。

DB2 CLI/ODBC 設定タブ:

サービス

参照項目:

225ページの『TRACEFILENAME』, 226ページの『TRACEFLUSH』, 227ページの『TRACEPATHNAME』

等価な接続属性:

SQL_ATTR_TRACE

使用上の注意:

このオプションをオン (1) にすると、CLI/ODBC トレース・レコードが、TRACEFILENAME 構成パラメーターに指定されたファイル、または TRACEPATHNAME 構成パラメーターに指定されたサブディレクトリーのファイルに追加されます。

たとえば、トレースが入力されるたびにディスクに書き込まれた CLI/ODBC トレース・ファイルをセットアップするには、次のように指定します。

```
[COMMON]
TRACE=1
TRACEFILENAME=E:¥TRACES¥CLI¥MONDAY.CLI
TRACEFLUSH=1
```

(このオプションは初期設定ファイルの Common (共通) セクションにあるので、 DB2 への接続すべてに適用されます。)

TRACECOMM

キーワードの説明:

各ネットワーク要求に関する情報トレース・ファイルに含めます。

db2cli.ini キーワード構文:

TRACECOMM = **0** | **1**

省略時設定値:

0 - 取り込まれるネットワーク要求の情報はありません。

DB2 CLI/ODBC 設定タブ:

このキーワードは「CLI/ODBC 設定 (CLI/ODBC Settings)」ノートブックを使用して設定することはできません。直接このキーワードを使用するように db2cli.ini ファイルを修正しなければなりません。

適用可能な条件:

CLI/ODBC TRACE オプションがオンになっていること。

参照項目:

223ページの『TRACE』，225ページの『TRACEFILENAME』，227ページの『TRACEPATHNAME』，226ページの『TRACEFLUSH』

使用上の注意:

TRACECOMM がオン (1) に設定されているとき、各ネットワーク要求に関する情報はトレース・ファイルに含められます。

このオプションは、TRACE CLI/ODBC オプションがオンになっている場合にのみ使用できます。例については、TRACE を参照してください。

(このオプションは初期設定ファイルの Common (共通) セクションにあるので、 DB2 への接続すべてに適用されます。)

TRACEFILENAME

キーワードの説明:

DB2 CLI/ODBC トレース情報を保管するために使用するファイル。

db2cli.ini キーワード構文:

TRACEFILENAME = < **完全ファイル名** >

省略時設定値:

なし

DB2 CLI/ODBC 設定タブ:

サービス

適用可能な条件:

TRACE がオンになっていること。

参照項目:

223ページの『TRACE』，226ページの『TRACEFLUSH』，227ページの『TRACEPATHNAME』

等価な接続属性:

SQL_ATTR_TRACEFILE

使用上の注意:

指定したファイルが存在していない場合には、そのファイルが作成されます。ファイルが存在している場合は、そのファイルの末尾に新しいトレース情報が追加されます。

指定したファイル名が無効である場合、またはファイルを作成できないかファイルへの書き込みができない場合は、トレースは実行されず、エラー・メッセージも戻されません。

このオプションは、TRACE オプションがオンになっている場合にのみ使用できます。 このオプションを CLI/ODBC 構成ユーティリティーに設定すると、このオプションは自動的に実行されます。

さまざまなトレース設定値の使用の例については、TRACE オプションを参照してください。このオプションが設定されている場合は、TRACEPATHNAME オプションが無視されます。

DB2 CLI トレースは、デバッグ用に限って使用してください。これらのオプションを使用すると CLI/ODBC の処理速度が低下し、しばらくそのままにしておくとトレース情報が非常に大きく膨れ上がります。

(このオプションは初期設定ファイルの Common (共通) セクションにあるので、 DB2 への接続すべてに適用されます。)

TRACEFLUSH

キーワードの説明:

CLI/ODBC トレース入力後に毎回、ディスクへの書き込みが強制的に実行されます。

db2cli.ini キーワード構文:

TRACEFLUSH = **0** | **1**

省略時設定値:

毎回の入力後には書き込みません。

DB2 CLI/ODBC 設定タブ:

サービス

適用可能な条件:

CLI/ODBC TRACE オプションがオンになっていること。

参照項目:

223ページの『TRACE』, 225ページの『TRACEFILENAME』, 227ページの『TRACEPATHNAME』

使用上の注意:

このオプションをオン (TRACEFLUSH = 1) に設定すると、トレース入力後に毎回、ディスクへの書き込みが強制的に実行されます。この場合、トレース処理の速度は低下しますが、アプリケーションが次のステートメントに移って実行を継続する前に、入力ごとにディスクに確実に書き込まれます。

このオプションは、TRACE CLI/ODBC オプションがオンになっている場合にのみ使用できます。例については、TRACE オプションを参照してください。

(このオプションは初期設定ファイルの Common (共通) セクションにあるので、 DB2 への接続すべてに適用されます。)

TRACEPATHNAME

キーワードの説明:

個別の DB2 CLI/ODBC トレース・ファイルを保管するために使用されるサブディレクトリー。

db2cli.ini キーワード構文:

TRACEPATHNAME = < **完全サブディレクトリー名** >

省略時設定値:

なし

DB2 CLI/ODBC 設定タブ:

サービス

適用可能な条件:

TRACE オプションがオンになっていること。

適用不能な条件:

TRACEFILENAME オプションがオンになっていること。

参照項目:

223ページの『TRACE』，225ページの『TRACEFILENAME』，226ページの『TRACEFLUSH』

使用上の注意:

同一の DLL または共用ライブラリーを使用するスレッドやプロセスにはそれぞれ、個別の DB2 CLI/ODBC トレース・ファイルが作成され、指定されたディレクトリーに保管されています。

指定したサブディレクトリーが無効であったり、サブディレクトリーへの書き込みができない場合、トレースは実行されず、エラー・メッセージも戻されません。

このオプションは、TRACE オプションがオンになっている場合にのみ使用できます。このオプションを CLI/ODBC 構成ユーティリティーに設定すると、このオプションは自動的に実行されます。

さまざまなトレース設定値の使用の例については、TRACE オプションを参照してください。 DB2 CLI/ODBC オプション TRACEFILENAME を使用している場合、このオプションは無視されます。

DB2 CLI トレースは、デバッグ用に限って使用してください。これらのオプションを使用すると CLI/ODBC の処理速度が低下し、しばらくそのままにしておくとトレース情報が非常に大きく膨れ上がります。

(このオプションは初期設定ファイルの Common (共通) セクションにあるので、 DB2 への接続すべてに適用されます。)

TXNISOLATION

キーワードの説明:

省略時分離レベルを設定します。

db2cli.ini キーワード構文:

TXNISOLATION = **1** | **2** | **4** | **8** | **32**

省略時設定値:

読み取りコミット済み (カーソル固定)

DB2 CLI/ODBC 設定タブ:

トランザクション

適用可能な条件:

省略時分離レベルが使用されている。アプリケーションが特定の分離レベルを設定している場合、このキーワードには効果がありません。

等価なステートメント属性:

SQL_ATTR_TXN_ISOLATION

使用上の注意:

分離レベルを以下のように設定します。

1 = 読み取り未コミット (非コミット読み取り)
2 = 読み取りコミット済み (カーソル固定) (省略時値)
4 = 反復可能読み取り (読み取り固定)
8 = 逐次化可能 (反復可能読み取り)
32 = (コミットなし、DB2 (AS/400 版) ; 自動コミットに類似したもの)

括弧で囲んだ語は、IBM 用語で等価の SQL92 分離レベルに対応したものです。コミットなし (*no commit*) は SQL92 分離レベルではなく、 DB2 ユニバーサル・データベース (AS/400 版) でのみサポートされています。分離レベルの詳細については、*SQL 解説書* を参照してください。

このキーワードが適用されるのは、省略時の分離レベルが使用されている場合だけです。 アプリケーションが特定の分離レベルを設定している場合、このキーワードには効果がありません。

UID

キーワードの説明:
省略時ユーザー ID を定義します。

db2cli.ini キーワード構文:
UID = ユーザー *ID*

省略時設定値:
なし

DB2 CLI/ODBC 設定タブ:
CLI/ODBC 設定一般

使用上の注意:

指定されたユーザー *ID* 値は、接続時にユーザー ID がアプリケーションによって提供されなかった場合に使用されます。

UNDERSCORE

キーワードの説明:
下線文字 ″_″ をワイルドカード文字として使用するかどうかを指定します。

db2cli.ini キーワード構文:
UNDERSCORE = **1** | **0**

省略時設定値:
″_″ はワイルドカードの役割を果たします。

DB2 CLI/ODBC 設定タブ:
最適化

使用上の注意:

このオプションを使用することにより、下線文字 ″_″ をワイルドカード文字 (文字なしを含め、任意の 1 文字にマッチングする) として使用するか、または下線文字そのものとして使用するかどうかを指定できます。このオプションは、探索パターン文字列を受け入れるカタログ関数呼び出しにのみ影響します。

- 1 = ″_″ はワイルドカードの役割を果たします (省略時値)

  下線文字はワイルドカード文字 (任意の 1 文字または文字なしにマッチングする) とみなされます。たとえば、2 つの表を次のように定義した場合を考えてみます。

  ```
  CREATE TABLE "OWNER"."KEY_WORDS" (COL1 INT)
  CREATE TABLE "OWNER"."KEYWORDS" (COL1 INT)
  ```

  表名探索パターン引き数に ″KEY_WORDS″ を指定した場合、表情報を戻す DB2 CLI カタログ関数呼び出し (SQLTables()) は上記の両方を戻します。
- 0 = ″_″ は下線文字そのものとみなされます。

  下線は下線そのものとして処理されます。 2 つの表が上記の例のように定義されている場合、表名探索パターン引き数に ″KEY_WORDS″ を指定した場合、 (SQLTables()) は ″KEY_WORDS″ だけを戻します。

  このキーワードを 0 に設定すると、データベースのオブジェクト名 (所有者、表、列) に下線が含まれている場合に、パフォーマンスが向上します。

注: このキーワードはバージョン 2.1 より前の DB2 共通サーバー版にしか効果がありません。それ以降のバージョンおよびその他のすべての DB2 サーバーには、LIKE 述部の ESCAPE 文節を使用できます。 ESCAPE 文節の詳細については、*SQL 解説書* を参照してください。

WARNINGLIST

キーワードの説明:

警告に格下げするエラーを指定します。

db2cli.ini キーワード構文:

WARNINGLIST = ″ '**xxxxx**', '**yyyyy**', ...″

省略時設定値:

どの SQLSTATE も格下げしません。

DB2 CLI/ODBC 設定タブ:

サービス

参照項目:

201ページの『IGNOREWARNLIST』, 201ページの『IGNOREWARNINGS』

使用上の注意:

エラーとして戻される SQLSTATE をいくつでも、警告に格下げできます。それぞれは大文字で表記し、単一引用符で囲み、コンマで区切る必要があります。ストリング全体を二重引用符で囲むことも必要です。たとえば、

```
WARNINGLIST=" '01S02', 'HY090' "
```

このオプションは IGNOREWARNINGS CLI/ODBC 構成キーワードと組み合わせて使用できます。 IGNOREWARNINGS もオンに設定している場合は、エラーを警告に格下げしても報告されることはありません。

第5章 DB2 CLI 関数

このセクションでは、各 DB2 CLI 関数をアルファベット順に説明します。さらに、235ページの『DB2 CLI 関数の要約』にはカテゴリー別の関数表があります。それぞれの説明には、以下の項目があります。

- 状況
- 目的
- 構文
- 引き数
- 使用法
- 戻りコード
- 診断
- 制限
- 例
- 参照

次に、それぞれの項について説明します。

DB2 CLI バージョン 5 以降におけるこの関数の状況

この項は、バージョン 5 で新しい関数で置き換えられたバージョン 2 関数だけに含まれます。

この項では、どんな新規関数を使用するのか、および旧関数の代わりに新規関数を使用する方法について説明します。

目的 この項では、関数の機能に関する簡単な概要を示します。また、説明されている関数を呼び出す前後にいずれかの関数を呼び出さなければならないかどうかも示します。

各関数には、次の表のような、その関数がどの仕様または標準に準拠するかを示す表もあります。最初の列は、DB2 CLI のどのレベルから関数が準拠しているかを示し、 2 番目の列は、ODBC のどのバージョン (1.0、2.0、または 3.0) から準拠しているかを示します。最後の列は、関数が ISO CLI 規格に含まれているかどうかを示します。

注: この表は関数のサポートを示しています。関数の中にはすべての仕様または規格が適用されない一連のオプションを使用するものがあります。制限の項には、重要な相違点が示されています。

表 11. 関数仕様表の例

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

構文 この項には、汎用 'C' プロトタイプが含まれています。汎用プロトタイプは、Windows を含むすべての環境に使用されます。

注: ポインターであるすべての関数実引き数は、マクロ FAR を使用して定義されますが、このマクロは Windows を除くすべてのプラットフォームで除外定義 (ブランクに設定) されています。 Windows では、ポインター引き数を far ポインターとして定義する際に FAR が用いられます。

引き数 この項では、各関数実引き数と、そのデータ・タイプ、説明、その引き数が入力引き数と出力引き数のどちらであるかをリストしています。

SQLGetInfo() と SQLBindParameter() だけが、入力および出力の両方のパラメーターを持っています。

関数の中には、*据え置き* または*バインド済み* 引き数という入力引き数または出力引き数が含まれているものがあります。

これらの引き数はアプリケーションによって割り当てられたバッファーを指すポインターであり、 SQL ステートメント内のパラメーターか、結果セット内の列に関連付けられ (またはバインドされ) ます。関数で指定したデータ域は、後で DB2 CLI にアクセスされます。 DB2 CLI にアクセスされる時点で、これらの据え置きデータ域がまだ有効であることが重要です。

使用法 この項は、関数の使用法に関する情報と特別な考慮事項を示します。起こる可能性のあるエラー状態についてはここでは説明せず、診断の項にリストしています。

戻りコード

この項では、戻される可能性のあるすべての関数戻りコードをリストします。 SQL_ERROR または SQL_SUCCESS_WITH_INFO が返された場合は、 SQLError() を呼び出してエラー情報を得ることができます。

戻りコードの詳細については、29ページの『診断』を参照してください。

診断 この項には、 DB2 CLI から明示的に返された SQLSTATE (DBMS に生成された SQLSTATE も返されることがある) をリストした表があり、エラーの原因が示されています。この値は、関数が SQL_ERROR または SQL_SUCCESS_WITH_INFO を返した後に SQLError() を呼び出すと得られます。

診断の詳細については、29ページの『診断』を参照してください。

制限 この項は、アプリケーションに影響を与える可能性がある DB2 CLI と ODBC の間の相違点または制限を示します。

例 この項には、汎用データ・タイプ定義を使用したときの、コード断片、または関数の使用法を示すコード断片への参照のいずれかが含まれています。すべてのコード断片に使用される完全なソースは、`sqllib/samples/cli` (または `sqllib¥samples¥cli`) ディレクトリー内で使用可能です。組み込まれている例のリストについては、907ページの『付録J. CLI サンプル・コード』を参照してください。

DB2 CLI 環境のセットアップとサンプル・アプリケーションのアクセスに関する詳細は、 155ページの『第4章 CLI の構成およびサンプル・アプリケーションの実行』を参照してください。

参照 この項は、関連する DB2 CLI 関数をリストします。

DB2 CLI 関数の要約

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。詳しくは 822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

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 12. 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				
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。
SQLAllocConnect	Depr	95	V 1.1	接続ハンドルを獲得します。
SQLAllocHandle()	コア	95	V 5	ハンドルを獲得します。

表 12. 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
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、または任意選択でドライバー・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実装されているため、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。
SQLSetConnection()	なし	SQL3	V 2.1	現行の活動状態の接続を設定します。この関数は、複数の並行接続がある 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()	なし	あり	V 5.2	DATALINK 値を作成します。

表 12. 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
SQLGetDataLinkAttr()	なし	あり	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。
SQLSetConnectOption()	レベル 1	あり	V 2.1 [a]	接続オプションを設定します。
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				

表 12. 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
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()	なし	なし	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()	なし	なし	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記憶域を割り当てます (ODBC 2.0)。
SQLSetParam()	Depr	なし	V 1.1	SQL ステートメントのパラメーター用の記憶域を割り当てます (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。
要求の実行依頼				
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。

表 12. 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
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	あり	V 5	結果セット内の列の属性を記述します。
SQLColAttributes()	Depr	あり	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	結果列のための記憶域を割り当て、データ・タイプを指定します。

表 12. 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
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。
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。
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。

表 12. 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
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入手します。
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()	レベル 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()	レベル 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。

表 12. 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
SQLPrimaryKeys()	レベル 1	SQL3	V 2.1	表の基本キーを構成する列名のリストを返します。
SQLProcedureColumns()	レベル 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()	レベル 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()	レベル 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	ハンドル資源を解放します。
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、保留中の結果を廃棄し、任意選択で、ステートメント・ハンドルに関連したすべての資源を解放します。
SQLCancel()	コア	95	V 1.1	SQL ステートメントを取り消します。

表 12. 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
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。
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 クライアント・アプリケーション・イネーブラー (DOS 版) バージョン 1.2 製品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 ドライバー・マネージャーによって実装されています。

SQLAllocConnect - 接続ハンドルを割り振る

使用すべきでない関数

注:

ODBC バージョン 3 では、SQLAllocConnect() を使用すべきではなく、その代わりに SQLAllocHandle() を使用します。詳細については、246ページの『SQLAllocHandle - ハンドルを割り振る』を参照してください。

DB2 CLI のこのバージョンでは、引き続き SQLAllocConnect() をサポートしますが、 DB2 CLI プログラムでは最初から SQLAllocHandle() の使用をお勧めします。そうすれば最新の標準に適合します。 上記の関数と、その他の使用すべきでない関数の詳細については、
822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

新しい関数への移行

たとえば、次の旧ステートメント

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

は、新しい関数を使用して、以下のように書き換えます。

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

SQLAllocEnv - 環境ハンドルを割り振る

使用法

注:

ODBC バージョン 3 では、SQLAllocEnv を使用すべきではなく、その代わりに SQLAllocHandle() を使用します。詳細については、246ページの『SQLAllocHandle - ハンドルを割り振る』を参照してください。

DB2 CLI のこのバージョンでは、引き続き SQLAllocEnv をサポートしますが、 DB2 CLI プログラムでは最初から SQLAllocHandle() の使用をお勧めします。そうすれば最新の標準に適合します。 上記の関数と、その他の使用すべきでない関数の詳細については、
822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

新しい関数への移行

たとえば、次の旧ステートメント

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

は、新しい関数を使用して、以下のように書き換えます。

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

SQLAllocHandle - ハンドルを割り振る

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ります。

注: この関数は、ハンドルを割り振るための汎用関数で、使用すべきでないバージョン 2 の関数 SQLAllocConnect()、 SQLAllocEnv()、および SQLAllocStmt() にとって代わるものです。

構文

```
SQLRETURN   SQLAllocHandle   (SQLSMALLINT       HandleType,
                              SQLHANDLE         InputHandle,
                              SQLHANDLE         *OutputHandlePtr);
```

関数引き数

表 13. 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、以下に示すように、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。

複数の環境、接続、またはステートメント・ハンドルを、アプリケーションによって一度に割り振ることができます。

アプリケーションが、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対して、 SQLAllocHandle() と *OutputHandlePtr を呼び出す場合、DB2 CLI はハンドルに関連した情報を上書きします。 DB2 CLI は、*OutputHandlePtr に入力されたハンドルがすでに使用中であるかどうかを調べませんし、そのハンドルに上書きする前にハンドルの以前の内容を調べることもしません。

マルチ・スレッドをサポートするオペレーティング・システムでは、アプリケーションは、異なるスレッド上で同じ環境、接続、ステートメント、または記述子ハンドルを使用できます。 DB2 CLI は、すべてのハンドルおよび関数呼び出しについてスレッド・セーフのアクセスを提供します。アプリケーションの作成するスレッドが DB2 CLI 資源の使用を調整しない場合は、アプリケーション自体が予期しない動作を経験するかもしれません。詳細については、50ページの『マルチスレッドのアプリケーション作成』を参照してください。

環境ハンドルの割り振り

環境ハンドルは、有効な接続ハンドルおよび活動状態の接続ハンドルのようなグローバル情報へのアクセスを提供します。環境ハンドルを要求するには、アプリケーションが、SQL_HANDLE_ENV の HandleType および SQL_NULL_HANDLE の HandleType を用いて SQLAllocHandle() を呼び出します。 DB2 CLI は、環境ハンドルを割り振って、関連したハンドルの値を *OutputHandlePtr 引き数に渡します。アプリケーションは、*OutputHandle 値を、環境ハンドル引き数を必要とする後続のすべての呼び出しに渡します。

DB2 CLI が SQL_HANDLE_ENV の *HandleType* を使用して SQLAllocHandle() 関数を処理するとき、 DB2 CLI は db2cli.ini ファイルの [COMMON] セクション内の **Trace** キーワードを調べます。 1 に設定してある場合、DB2 CLI は現行のアプリケーションをトレースすることができます。トレース・フラグが設定されている場合、トレースは、最初の環境ハンドルが割り当てられる時に開始し、最後の環境ハンドルが解放される時に終了します。詳細については、223ページの『TRACE』を参照してください。

環境ハンドルを割り振った後、アプリケーションは、環境ハンドル上で SQLSetEnvAttr() を呼び出して、 SQL_ATTR_ODBC_VERSION 環境属性を設定しなければなりません。アプリケーションが ODBC アプリケーションとして実行され、そして環境に接続ハンドルを割り振るために SQLAllocHandle() が呼び出される前にこの属性が設定されない場合は、接続を割り振る呼び出しが SQLSTATE HY010 (関数の順序エラーです。) を返します。

接続ハンドルの割り振り

接続ハンドルは、接続上の有効なステートメントおよび記述子ハンドルなどの情報や、トランザクションが現在オープンしているかどうかといった情報を使用できるようにします。接続ハンドルを要求するには、アプリケーションが、SQL_HANDLE_DBC の HandleType を用いて SQLAllocHandle() を呼び出します。 InputHandle 引き数は、ハンドルを割り振った SQLAllocHandle() への呼び出しによって返された環境ハンドルに設定されます。 DB2 CLI は、接続ハンドルを割り振って、関連したハンドルの値を *OutputHandlePtr に戻します。アプリケーションは、接続ハンドルを必要とするすべての後続の呼び出しで *OutputHandlePtr を渡します。

その環境で接続ハンドルを割り振るために SQLAllocHandle() が呼び出される前に、 SQL_ATTR_ODBC_VERSION 環境属性が設定されていない場合は、接続を割り振る呼び出しは、アプリケーションが ODBC ドライバー・マネージャーを使用中のときは、 SQLSTATE HY010 (関数の順序エラーです。) を返すことになります。

ステートメント・ハンドルの割り振り

ステートメント・ハンドルは、エラー・メッセージ、カーソル名のようなステートメント情報、および SQL ステートメント処理の状況情報を使用できるようにします。ステートメント・ハンドルを要求するには、アプリケーションがデータ・ソースに接続して、 SQL ステートメントの実行前に、SQLAllocHandle() を呼び出します。この呼び出しで、HandleType を SQL_HANDLE_STMT に設定し、 InputHandle を接続ハンドルに設定してください。その接続ハンドルは、ハンドルを割り振る SQLAllocHandle() への呼び出しによって返されたものです。 DB2 CLI は、ステートメント・ハンドルを割り振り、指定した接続とステートメント・ハンドルを関連付け、そして関連付けたハンドルの値を *OutputHandlePtr に戻します。アプリケーションは、ステートメント・ハンドルを必要とするすべての後続の呼び出しで *OutputHandlePtr 値を渡します。

ステートメント・ハンドルが割り振られるとき、DB2 CLI は、自動的に 1 組の 4 つの記述子を割り振り、その記述子用ハンドルを SQL_ATTR_APP_ROW_DESC、SQL_ATTR_APP_PARAM_DESC、SQL_ATTR_IMP_ROW_DESC、SQL_ATTR_IMP_PARAM_DESC ステートメント属性に割り当てます。自動的に割り振られたものの代わりに明示的に割り当てられたアプリケーション記述子を使用するには、以下の『記述子ハンドルの割り振り』の項を参照してください。

記述子ハンドルの割り振り

アプリケーションが SQL_HANDLE_DESC の HandleType を用いて SQLAllocHandle() を呼び出すと、 DB2 CLI は、アプリケーション記述子を明示的に割り振ります。アプリケーションは、自動的に割り振られたアプリケーション記述子の代わりに、 SQL_ATTR_APP_ROW_DESC または SQL_ATTR_APP_PARAM_DESC 属性付きの SQLSetStmtAttr() を呼び出すことにより、明示的に割り振られたアプリケーション記述子を使用することができます。実装記述子は、明示的に割り振ることができず、また SQLSetStmtAttr() 関数の中に指定することもできません。

明示的に割り振られる記述子は、(自動的に割り振られた記述子がそうであるように) ステートメント・ハンドルよりも、接続ハンドルに関連があります。記述子は、アプリケーションが実際にデータベースに接続中のときにだけ、接続ハンドルに関連付けることができます。明示的に割り振られた記述子は、接続ハンドルに関連付けられるので、アプリケーションは、割り振られた記述子を接続内の複数のステートメントに明示的に関連付けることができます。他方、自動的に割り振られたアプリケーション記述子は、複数のステートメント・ハンドルに関連付けることができません。明示的に割り振られた記述子ハンドルは、 SQL_HANDLE_DESC の HandleType を用いて SQLFreeHandle() を呼び出すことにより、アプリケーションによって明示的に解放するか、または切断の際に接続ハンドルが解放されるときに、暗黙的に解放することができます。

明示的に割り振られたアプリケーション記述子がステートメントに関連付けられるときに、もはや使用されていない自動的に割り振られた記述子は、まだ接続ハンドルに関連付けられています。明示的に割り振られた記述子が解放されるとき、自動的に割り振られた記述子は、再びステートメントと関連付けられます (そのステートメント用の SQL_ATTR_APP_ROW_DESC または SQL_ATTR_APP_PARAM_DESC 属性は、自動的に割り振られた記述子ハンドルに設定されます)。このことは、明示的に割り振られた記述子に接続で関連付けられたすべてのステートメントにあてはまります。各ステートメントの元の自動的に割り振られた記述子ハンドルは、再びそのステートメントと関連付けられます。

記述子が最初に使用されるとき、その SQL_DESC_TYPE フィールドの初期値は SQL_C_DEFAULT です。 DATA_PTR、INDICATOR_PTR、および OCTET_LENGTH_PTR は、すべてヌル・ポインターに初期設定されます。その他のフィールドの初期値については、696ページの『SQLSetDescField - 記述子レコードの単一フィールドを設定する』を参照してください。

詳細については、105ページの『記述子の使用』を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_INVALID_HANDLE
- SQL_ERROR

環境ハンドル以外のハンドルを割り振るとき、 SQLAllocHandle() が SQL_ERROR を返すならば、出力引き数が NULL ポインターでなければ、HandleType の値によって、 OutputHandlePtr を SQL_NULL_HENV、SQL_NULL_HDBC、SQL_NULL_HSTMT、または SQL_NULL_HDESC に設定します。アプリケーションは、次に、 InputHandle 引き数内のハンドルに関連付けられる診断データ構造体から追加情報を入手することができます。

環境ハンドルの割り振りエラー

診断

表 14. SQLAllocHandle SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
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* 引き数によって指定される接続は、オープンしていませんでした。 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	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	DB2 CLI は、指定したハンドル用のメモリーを割り振りできませんでした。

表 14. SQLAllocHandle SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数の順序エラーです。	*HandleType* 引き数は SQL_HANDLE_DBC でしたが、SQLSetEnvAttr() は、 SQL_ODBC_VERSION 環境属性の設定用に呼び出されていません。
HY013	予期しないメモリーのハンドル・エラーが起きました。	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	もはやハンドルはありません。	*HandleType* 引き数によって指定されるハンドルのタイプについて割り振ることのできるハンドルの数の限界に達しました。
HY092	オプション・タイプが範囲外です。	*HandleType* 引き数は、以下のものではありませんでした。 • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
HYC00	ドライバーが機能していません。	*HandleType* 引き数は SQL_HANDLE_DESC でしたが、 DB2 CLI ドライバーは、バージョン 2 またはそれ以前のものでした。

制約

なし。

例

```
/* From the CLI sample utilcli.c */
/* ... */
    /* allocate an environment handle */
    sqlrc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, pHenv ) ;
    if ( sqlrc != SQL_SUCCESS )
    {   printf( "¥n--ERROR while allocating the environment handle.¥n" ) ;
        printf( "  sqlrc             = %d¥n", sqlrc);
        printf( "  line              = %d¥n", __LINE__);
        printf( "  file              = %s¥n", __FILE__);
        return( 1 ) ;
    }

/* ... */
    /* allocate a database connection handle */
    sqlrc = SQLAllocHandle( SQL_HANDLE_DBC, *pHenv, pHdbc ) ;
    HANDLE_CHECK( SQL_HANDLE_ENV, *pHenv, sqlrc, pHenv, pHdbc ) ;
```

参照

- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 409ページの『SQLExecute - ステートメントの実行』
- 470ページの『SQLFreeHandle - ハンドル資源を解放する』
- 630ページの『SQLPrepare - ステートメントを準備する』
- 663ページの『SQLSetConnectAttr - 接続属性を設定する』
- 501ページの『SQLGetDescField - 記述子レコードの単一フィールド設定を入手する』
- 734ページの『SQLSetEnvAttr - 環境属性を設定する』
- 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』

SQLAllocStmt - ステートメント・ハンドルを割り振る

使用すべきでない関数

注:

ODBC バージョン 3 では、SQLAllocStmt() を使用すべきでなく、その代わりに SQLAllocHandle() を使用します。詳細については、246ページの『SQLAllocHandle - ハンドルを割り振る』を参照してください。

DB2 CLI のこのバージョンでは、引き続き SQLAllocStmt() をサポートしていますが、 DB2 CLI プログラムでは最初から SQLAllocHandle() の使用をお勧めします。そうすれば最新の標準に適合します。 上記の関数と、その他の使用すべきでない関数の詳細については、
822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

新しい関数への移行

たとえば、次の旧ステートメント

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

は、新しい関数を使用して、以下のように書き換えます。

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

SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする

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とアプリケーション変数の配列 (記憶域バッファー)。この場合、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)。この場合、SQLFetch() が呼び出されると、データ自体ではなく 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。

一般に、SQLPrepare()、 SQLExecDirect() またはスキーマ関数の 1 つがこの関数の前に呼び出され、その後で SQLFetch() または SQLFetchScroll() 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,
                              SQLUSMALLINT      ColumnNumber,
                              SQLSMALLINT       TargetType,
                              SQLPOINTER        TargetValuePtr,
                              SQLINTEGER        BufferLength,
                              SQLINTEGER   *FAR StrLen_or_IndPtr);
```

関数引き数

表 15. 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
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は、列番号は 1 から始まります (SQL_ATTR_USE_BOOKMARKS ステートメント属性を SQL_UB_OFF に設定します)。 • ブックマークを使用している場合は、列番号は 0 から始まります (ステートメント属性を SQL_UB_ON に設定します)。

表 15. SQLBindCol 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*TargetType*	入力	結果セット内の列番号 *ColumnNumber* 用の C データ・タイプ。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_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_DEFAULT を指定するとデータが省略時の C データ・タイプに転送されます。詳細については 33ページの表2 を参照してください。

表 15. 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 SQLFetchScroll() を使用する場合はバッファーの配列) を指すポインター。 SQLSetPos に対する *Operation* 引き数が SQL_REFRESH のときに、このバッファーを使用してデータが返されます。 SQLSetPos の *Operation* 引き数が SQL_UPDATE に設定されているときに、このバッファーを使用してデータが取り出されます。 TargetValuePtr がヌルの場合は、列はアンバインドされます。
SQLINTEGER	*BufferLength*	入力	列データまたは LOB ロケーターを保管するために使用可能な *TargetValuePtr* バッファーのサイズ (バイト)。 *TargetType* が 2 進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、あるいは *TargetType* が SQL_C_DEFAULT である場合は、BufferLength が 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。それ以外の場合は、この引き数は無視されます。

表 15. 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。 SQLSetPos に対する *Operation* 引き数が SQL_REFRESH のときに、このバッファーを使用してデータが返されます。 SQLSetPos の *Operation* 引き数が SQL_UPDATE に設定されているときに、このバッファーを使用してデータが取り出されます。 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固有であるか、または NULL でなければなりません。 SQL_COLUMN_IGNORE の値は、SQLBulkOperations() と共に使用するために設定することもできます。詳細については、306ページの『SQLBulkOperations - 行のセットの追加、更新、削除、または取り出し』を参照してください。 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。

- この関数の場合は、 *TargetValuePtr* および *StrLen_or_Ind* の両方が据え置きの出力です。このことは、結果セット列が取り出されるまでは、そのポインターが指す記憶場所が更新されていないことを意味します。したがって、そのポインターで参照される記憶場所は、 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、 *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・タイプについてパフォーマンスを最適化することができます。詳細については、後述します。

使用法

アプリケーションは、データまたは (LOB 列の場合は任意選択で) LOB ロケーターのどちらかを検索するために必要な結果セット内の列ごとに 1 回ずつ SQLBindCol() を呼び出します。結果セットは、SQLExecute()、SQLExecDirect()、 SQLGetTypeInfo()、またはカタログ関数のいずれかを呼び出すと生成されます。 SQLFetch() を呼び出すときは、そのバインドされた 列のそれぞれにあるデータは、割り当てた場所に置かれます (ポインター *TargetValuePtr* および *StrLen_or_Ind* で指される場所に)。 *TargetType* が LOB ロケーターの場合は、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 列内のすべてのデータ値を参照します。

SQLFetch() および SQLFetchScroll() を使用すると、結果セットから複数行を配列内に取り出すことができます。この場合、*TargetValuePtr* は配列を参照します。詳細については、99ページの『配列への結果セットの取り出し』および446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』を参照してください。

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は、列番号は 1 から始まります (SQL_ATTR_USE_BOOKMARKS ステートメント属性を SQL_UB_OFF に設定します)。
- ブックマークを使用している場合は、列番号は 0 から始まります (ステートメント属性を SQL_UB_ON に設定します)。

ブックマークを使用しようとする場合は、最初に SQL_ATTR_USE_BOOKMARKS ステートメント属性を SQL_UB_ON に設定しなければなりません。

結果セットの列番号を判別するには、*DescType* 引き数を SQL_COLUMN_COUNT に設定して SQLNumResultCols() または SQLColAttribute() を呼び出します。

アプリケーションは、最初に SQLDescribeCol() または SQLColAttribute() を呼び出すと、列の属性 (データ・タイプやデータ長など) を照会することができます。次にこの情報を使用して正しいデータ・タイプと長さで記憶場所を割り振って、別のデータ・タイプへのデータ変換を指示するか、 LOB データ・タイプの場合にロケーターを返すこともできます。省略時のタイプとサポートされる変換の詳細については、 32ページの『データ・タイプとデータ変換』を参照してください。

アプリケーションは、すべての列をバインドするとは限らないことを選択したり、またはどの列もバインドしないことを選択することもできます。また、どの列にあるデータでも、バインドされている列を現在行のために取り出してから、SQLGetData() を使用して取り出すことができます。一般には、SQLBindCol() の方が SQLGetData() よりも効果的です。ある関数を別の関数に優先して使用すべき時についての説明は、 811ページの『付録A. プログラミングのヒントと提案』を参照してください。

これ以降の取り出しで、アプリケーションは SQLBindCol() を呼び出して、これらの列の結び付きを変更したり、すでにアンバインドされている列をバインドすることができます。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。

列バインドの相対位置

SQLBindCol() への複数の呼び出しの代わりに、 DB2 CLI はパラメーター・バインドの相対位置もサポートしています。毎回再バインドするよりも、相対位置を使用すると、SQLFetch() または SQLFetchScroll() への次回の呼び出しで使用される新しいバッファー・アドレスおよび長さ / 標識アドレスを指定することができます。これは、行方向バインドでのみ使用できますが、アプリケーションが一度に単一行を取り出すか、または複数行を取り出すかを決めます。

相対位置を使用するのに必要なステップのリストについては、 103ページの『列バインドの相対位置』を参照してください。

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記憶域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、アプリケーションは最大長のバインドされた列が必要とする記憶域を割り当てる必要があります。そうしないと、データが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。詳細については、32ページの『データ・タイプとデータ変換』を参照してください。

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 (ヌル終止符を足す) 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 バイトのヌル終止符でヌル終了します。*TargetType* が SQL_C_CHAR である場合は、データのヌル終了はありません。いずれの場合も、*TargetValuePtr* バッファーの長さ (*BufferLength*) はバイト単位であり、そのため 2 の倍数になります。強制的に DB2 CLI に図形ストリングをヌル終了させることもできます。 174ページの『構成キーワード』の PATCH1 キーワードを参照してください。

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 (ヌル不可) であり、かつ
- ストリング切り捨てが起きているとき。

LOB ロケーターは、一般に他の任意のデータ・タイプとして処理できますが、次のような重要な相違点があります。

- ロケーターがサーバーで生成されるのは、行が取り出され、かつ LOB ロケーター C データ・タイプが `SQLBindCol()` に指定されている、または

SQLGetSubString() が呼び出されて別の LOB の一部にロケーターを定義している場合です。アプリケーションに転送されるのはロケーターだけです。

- ロケーターの値は、現行トランザクション内だけで有効です。 LOB を取り出すために使用するカーソルに WITH HOLD 属性があるとしても、ロケーター値を保管したり、現行のトランザクションを越えてロケーター値を使用したりすることはできません。
- FREE LOCATOR ステートメントを使用して、トランザクションの終了前にロケーターを解放することもできます。
- ロケーターが受信されると、アプリケーションは SQLGetSubString() を使用して、 LOB 値の一部を受信するか、またはサブストリングを表す別のロケーターを生成することができます。ロケーターの値は、パラメーター・マーカーの入力としても使用できます (SQLBindParameter() を使用)。

  LOB ロケーターは、データベース位置を指すポインターではなく、 LOB 値への参照、つまり LOB 値のスナップショットです。カーソルの現在位置と LOB 値が抽出された行との間には、何の関連もありません。このことは、カーソルが異なる行へ移動した後でも、 LOB ロケーター (および LOB ロケーターが表す値) が、まだ参照できることを意味します。
- SQLGetPosition() と SQLGetLength() は、サブストリングを定義する際に SQLGetSubString() とともに使用することができます。

結果セット内の特定の LOB 列の場合、以下の対象をバインドすることができます。
- 全 LOB データ値を保持する記憶域バッファー、
- LOB ロケーター、または
- LOB ファイル参照 (SQLBindFileToCol() を使用)。

最新の列バインド関数呼び出しは、有効なバインドのタイプを判別します。 **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。
1. SQLGetStmtAttr() を呼び出して、ARD ハンドルを獲得します。
2. 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。
3. 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 および SQL_DESC_CONCISE_TYPE を *TargetType* の値に設定します。ただし、 *TargetType* が日時または間隔サブタイプのコンサイス識別子の 1 つである場合は例外です。その場合は、SQL_DESC_TYPE を SQL_DATETIME または SQL_INTERVAL にそれぞれ設定し、 SQL_DESC_CONCISE_TYPE をコンサイス識別子に設定し、さらに SQL_DESC_DATETIME_INTERVAL_CODE を対応する日時または間隔サブタイプに設定します。
   - TargetType に適するように、1 つまたは複数の SQL_DESC_LENGTH、SQL_DESC_PRECISION、SQL_DESC_SCALE、および SQL_DESC_DATETIME_INTERVAL_PRECISION 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_Ind* 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_Ind* の値に設定します (以下の節を参照してください)。

StrLen_or_Ind 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ヌル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_Ind* 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、ヌル値を検出した場合に取り出しを失敗させてしまうと、SQL_NULL_DATA を返す方法がなくなってしまいます。

SQLBindCol() への呼び出しが失敗したときは、それが ARD に設定する記述子の内容フィールドは未定義で、 ARD の SQL_DESC_COUNT フィールドの値は変更されません。

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_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。しかし、アプリケーションが SQL_DEFAULT の *TargetType* を指定すれば、 SQLBindCol() は、メタデータへの変更か、異なる列へのコードの適用のどちらかの方法により、アプリケーションが意図した列から、異なるデータ・タイプの列へと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 16. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。

表 16. SQLBindCol SQLSTATEs (続き)

SQLSTATE	説明	解説
HY002	列の番号が無効です。	引き数 *ColumnNumber* に指定された値は、0 より小さい値でした。 引き数 *ColumnNumber* 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。
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	関数の順序エラーです。	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 SQL の操作中に、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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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	ドライバーが機能していません。	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。

制約

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。アプリケーションが LOB ロケーター C データ・タイプを指定しようとする場合、 SQLSTATE **HY**C00 が返されます。

例

```
/* From the CLI sample utilcli.c */
/* ... */

       /* bind columns to program vars, converting all types to CHAR */
        sqlrc = SQLBindCol( hstmt,
                    ( SQLSMALLINT ) ( i + 1 ),
                    SQL_C_CHAR,
```

```
                outData[i].buff,
                outData[i].buffLen,
                &outData[i].len ) ;
    STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参照

- 267ページの『SQLBindFileToCol - LOB 列に LOB ファイル参照をバインドする』
- 434ページの『SQLFetch - 次の行の取り出し』
- 446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』

SQLBindFileToCol - LOB 列に LOB ファイル参照をバインドする

目的

仕様:	DB2 CLI 2.1		

SQLBindFileToCol() 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る (バインドする) ときに使用します。この処理により、ステートメント・ハンドル用に各行が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、取り出し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      *FAR FileName,
                              SQLSMALLINT  *FAR FileNameLength,
                              SQLUINTEGER  *FAR FileOptions,
                              SQLSMALLINT       MaxFileNameLength,
                              SQLINTEGER   *FAR StringLength,
                              SQLINTEGER   *FAR IndicatorValue);
```

関数引き数

表 17. 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。

表 17. SQLBindFileToCol 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLPOINTER	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。このポインターをヌル (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) を入れる場所を指すポインター。このポインターがヌル (NULL) の場合は、 SQL_NTS の長さが想定されます。 ファイル名の長さの最大値は 255 です。

表 17. SQLBindFileToCol 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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) を入れる場所を指すポインター。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、省略時値はありません。
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* 配列内の各要素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) を入れる場所を指すポインター。ポインターがヌル (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) を入れる場所を指すポインター。

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、データ変換およびヌル終了の追加をせずに、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* 配列内の各要素の長さを指定し、 DB2 CLI が *FileName* 配列内の各要素の場所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要素の配列を指します。

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 18. SQLBindFileToCol 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。

表 18. SQLBindFileToCol SQLSTATE (続き)

SQLSTATE	説明	解説
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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY002	列の番号が無効です。	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。
HY009	引き数値が無効です。	*FileName*、*StringLength*、または *FileOptions* は NULL ポインターです。
HY010	関数の順序エラーです。	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 SQL の操作中に、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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY090	ストリングまたはバッファー長が無効です。	引き数 *MaxFileNameLength* に指定された値は、 0 より小さい値でした。
HYC00	ドライバーが機能していません。	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。

制約

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

```
/* From the CLI sample dtlob.c */
/* ... */

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

参照

- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 434ページの『SQLFetch - 次の行の取り出し』
- 446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』
- 273ページの『SQLBindFileToParam - LOB パラメーターに LOB ファイル参照をバインドする』

SQLBindFileToParam - LOB パラメーターに LOB ファイル参照をバインドする

目的

仕様:	DB2 CLI 2.1		

SQLBindFileToParam() 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る (バインドする) ときに使用されます。この処理により、このステートメントが続けて実行されたときに、その列のデータを LOB 列に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。 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,
                              SQLUSMALLINT      TargetType,
                              SQLSMALLINT       DataType,
                              SQLCHAR       *FAR FileName,
                              SQLSMALLINT   *FAR FileNameLength,
                              SQLUINTEGER   *FAR FileOptions,
                              SQLSMALLINT       MaxFileNameLength,
                              SQLINTEGER    *FAR IndicatorValue);
```

関数引き数

表 19. SQLBindFileToParam 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	StatementHandle	入力 (据え置き)	ステートメント・ハンドル。
SQLUSMALLINT	TargetType	入力 (据え置き)	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。

表 19. SQLBindFileToParam 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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。 この引き数をヌル (NULL) にすることはできません。
SQLSMALLINT *	FileNameLength	入力 (据え置き)	次回の `SQLExecute()` または `SQLExecDirect()` のときに *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) を入れる場所を指すポインター。 このポインターがヌル (NULL) の場合は、 SQL_NTS の長さが想定されます。 ファイル名の長さの最大値は 255 です。

表 19. SQLBindFileToParam 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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) を入れる場所を指すポインター。この場所は、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。) このポインターをヌル (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* 配列内の各要素の長さになります。
SQLINTEGER *	IndicatorValue	出力 (据え置き)	標識値 (または標識値の配列) が入る場所を指すポインターで、パラメーターのデータ値がヌル (NULL) になる場合は、 SQL_NULL_DATA に設定されます。データ値がヌルでない場合は、データ値を 0 にしなければなりません (または、ポインターをヌルに設定できます)。

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* 配列内の各要素の長さを指定し、 DB2 CLI が *FileName* 配列内の各要素の場所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 20. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY004	SQL のデータ・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。
HY009	引き数値が無効です。	*FileName*、*FileOptions*、*FileNameLength* は、NULL ポインターです。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。

表 20. SQLBindFileToParam SQLSTATE (続き)

SQLSTATE	説明	解説
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY090	ストリングまたはバッファー長が無効です。	引き数 *MaxFileNameLength* に指定された値は、0 より小さい値でした。
HY093	パラメーターの数値が無効です。	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	ドライバーが機能していません。	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。

制約

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

```
/* From the CLI sample dtlob.c */
/* ... */
    /* bind the file-parameter */
    rc = SQLBindFileToParam(hstmt, 3, SQL_BLOB, fileName, &fileNameLength,
                            &fileOption, 14, &fileInd);
```

参照

- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』
- 409ページの『SQLExecute - ステートメントの実行』
- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 626ページの『SQLParamOptions - パラメーターに入力配列を指定する』

SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする

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とアプリケーション変数の配列 (記憶域バッファー)。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ができます。

この関数は、パラメーターを入力または出力 (あるいはその両方) できるストアード・プロシージャー CALL ステートメントのパラメーターに、アプリケーション記憶域をバインドするときにも使用しなければなりません。この関数は、基本的には SQLSetParam() の拡張です。

構文

```
SQLRETURN  SQL_API 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 *FAR   StrLen_or_IndPtr);/* pcbValue */
```

関数引き数

表 21. 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 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き、パラメーターの実際のデータ値がサーバーへ送信されます。 *ParameterValuePtr* バッファーは、有効な入力データ値を含んでいなければなりません。 *StrLen_or_IndPtr* バッファーは、対応する長さの値または SQL_NTS、SQL_NULL_DATA、もしくは (その値が `SQLParamData()` および `SQLPutData()` を介して送られる場合は) SQL_DATA_AT_EXEC を含んでいなければなりません。 • SQL_PARAM_INPUT_OUTPUT: パラメーター・マーカーは、呼び出された (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き、パラメーターの実際のデータ値がサーバーへ送信されます。 *ParameterValuePtr* バッファーは、有効な入力データ値を含んでいなければなりません。 *StrLen_or_IndPtr* バッファーは、対応する長さの値または SQL_NTS、 SQL_NULL_DATA、もしくは (その値が `SQLParamData()` および `SQLPutData()` を介して送らされる場合は) SQL_DATA_AT_EXEC を含んでいなければなりません。 • SQL_PARAM_OUTPUT: パラメーター・マーカーは、呼び出された (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 21. 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_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_DEFAULT を指定すると、データが省略時の 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 21. SQLBindParameter 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	ParameterType	入力	パラメーターの 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_BIGINT • SQL_BINARY • SQL_BLOB • SQL_BLOB_LOCATOR • SQL_CHAR • SQL_CLOB • SQL_CLOB_LOCATOR • SQL_DBCLOB • SQL_DBCLOB_LOCATOR • SQL_DECIMAL • SQL_DOUBLE • SQL_FLOAT • SQL_GRAPHIC • SQL_INTEGER • SQL_LONGVARBINARY • SQL_LONGVARCHAR • SQL_LONGVARGRAPHIC • SQL_NUMERIC • SQL_REAL • SQL_SMALLINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_VARBINARY • SQL_VARCHAR • SQL_VARGRAPHIC **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。
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。 • 2 進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。 列サイズが事前に分からない場合、アプリケーションはこの値を 0 に設定することができます。詳細については、288ページの『列サイズが事前に分からないとき』 を参照してください。

表 21. SQLBindParameter 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	DecimalDigits	入力	*ParameterType* が SQL_DECIMAL または SQL_NUMERIC である場合、相当するパラメーターの位取り。 *ParameterType* が SQL_TYPE_TIMESTAMP である場合は、これはタイム・スタンプの文字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 の位取りは 3 になります)。 上記以外の *ParameterType* 値の場合は、*DecimalDigits* は無視されます。
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 を含む場合は、*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 21. 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 2 進データの場合、 *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要素として扱われる場合)、または *ParameterValuePtr* 配列内の個々の要素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 2 進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 (単一要素である場合) または *ParameterValuePtr* 配列内の各要素の長さ (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 2 進出力データを切り捨てるかどうかを判別します。 • 文字データの場合、戻りに使用できるバイト数が *BufferLength* 以上であれば、 *ParameterValuePtr* 内のデータは *BufferLength-1* バイトに切り捨てられ、ヌル終了します (ヌル終了がオフになっていない場合)。 • 2 進データの場合、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 21. 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) 入れる場所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、この記憶場所に SQL_NULL_DATA を入れる必要があります。 *ValueType* が SQL_C_CHAR である場合、この記憶場所には、*ParameterValuePtr* に保管されているデータの正確な長さか、または *ParameterValuePtr* の内容がヌル終了の場合は SQL_NTS が入っていなければなりません。 *ValueType* が (明示的に、または SQL_C_DEFAULT を使用して暗黙的に) 文字データを示し、かつこのポインターが NULL に設定されている場合、アプリケーションが常に *ParameterValuePtr* にヌル終了のストリングを提供すると想定されています。また、このことは、このパラメーター・マーカーがヌル値ではありえないことをも暗黙指定しています。 *ParameterType* が図形データ・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。一般に図形データ・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。
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)	`SQLParamOptions()` を使用して各パラメーターに複数の値を指定する場合、 *StrLen_or_IndPtr* は SQLINTEGER 値の配列 (各要素が対応する *ParameterValuePtr* 要素 (ヌル終了を除く) 内のバイト数となり得る)、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、出力パラメーターまたはストアード・プロシージャー呼び出し (CALL) 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 (ヌル終了文字を除く)。 • 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

パラメーター・マーカーは ″?″ 文字で表され、アプリケーションに提供された値がステートメントの実行時に置き換えられるステートメント内の位置を指示するのに使用されます。この値は、次のものから得られます。

- アプリケーション変数。

  SQLBindParameter() (または SQLSetParam()) は、アプリケーション記憶域をパラメーター・マーカーにバインドするのに使用されます。

- データベース・サーバーからの LOB 値 (LOB ロケーターを指定します) 。

  SQLBindParameter() (または SQLSetParam()) は、 LOB ロケーターをパラメーター・マーカーにバインドするのに使用されます。 LOB 値自体はデータベース・サーバーで得られるため、 LOB ロケーターだけがデータベース・サーバーとアプリケーションの間で転送されます。

  アプリケーションは、SQLGetSubString()、SQLGetPosition()、または SQLGetLength() のどれかでロケーターを使用することができます。 SQLGetSubString() は、別のロケーターか、またはデータ自体を返すことができます。すべてのロケーターは、そのロケーターを作成したトランザクションの終了まで (カーソルが別の行へ移動した場合も含む)、または FREE LOCATOR ステートメントで解放されるまで有効です。

- LOB 値を含むファイル (アプリケーション環境内)。

  LOB パラメーター・マーカーにファイルをバインドするには、 SQLBindFileToParam() を使用します。 SQLExecDirect() を実行すると、 DB2 CLI はファイルの内容をデータベース・サーバーに直接転送します。

アプリケーションは、SQL ステートメントを実行する前に、その SQL ステートメント内の各パラメーター・マーカーに変数をバインドする必要があります。この関数の場合、 *ParameterValuePtr* および *StrLen_or_IndPtr* が据え置き引き数であり、ステートメント実行時には記憶場所が有効であり、かつ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、またはこれらの記憶場所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。

SQLBindParameter() (または SQLSetParam()) は、結果セットの列が知られている場合は、SQLPrepare() の前に呼び出すことができます。それ以外の場合は、ステートメントが作成された後で結果セットの属性を獲得できます。

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

この関数によってバインドされるすべてのパラメーターは、以下のいずれかのときまで有効です。

- SQLFreeStmt() に SQL_RESET_PARAMS オプションを指定して呼び出すとき、または
- SQLFreeHandle() を *HandleType* に SQL_HANDLE_STMT を指定して呼び出すとき、または
- 同じパラメーター *ParameterNumber* の番号で SQLBindParameter() を再度呼び出すとき。

SQL ステートメントを実行し、結果を処理した後、アプリケーションはステートメント・ハンドルを再利用して別の SQL ステートメントを実行したい場合があります。パラメーター・マーカーの仕様 (パラメーターの数、長さ、またはタイプ) が異なる場合、パラメーターのバインドをリセットまたはクリアするには、 SQL_RESET_PARAMS を指定して SQLFreeStmt() を呼び出す必要があります。

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。

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、SQL_DATA_AT_EXEC 値を *StrLen_or_IndPtr* バッファーに置くことによって、 DB2 CLI に実行時データ・パラメーターについて通知します。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。

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形式のエラーは SQLExecute() または SQLExecDirect() を呼び出すまで検出も報告もされません。

SQLBindParameter() は、以下のことを行う方法を提供することにより、実質的には SQLSetParam() 関数の機能を拡張したものとなっています。

- 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。
- SQLParamOptions() が SQLBindParameter() と組み合わせて使用されたときに、入力パラメーター値の配列を指定します。単一の要素アプリケーション

変数をストアード・プロシージャー呼び出し (CALL) ステートメントの一部ではないパラメーター・マーカーにバインドする際には、 SQLSetParam() を使用することができます。

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。すべてのプロシージャー引き数が入出力であることを DB2 ストアード・プロシージャー引き数規則で一般に暗黙指定している場合でも、アプリケーション・プログラマーはこれまでどおり SQLBindParameter() に関する入力や出力の性質をさらに正確に指定して、さらに厳格なコーディング・スタイルに従うことができます。

- 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、記憶場所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、これらの記憶場所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* が SQL_PARAM_OUTPUT または SQL_PARAM_INPUT_OUTPUT に設定されている場合、 ParameterValuePtr および StrLen_or_IndPtr バッファー位置は、 CALL ステートメントが実行されるまで有効でなければなりません。

文字データと 2 進 C データの場合、 *BufferLength* 引き数は *ParameterValuePtr* バッファーが単一要素である場合にそのバッファーの長さを指定します。一方、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、 *BufferLength* は、ヌル終了を含む *ParameterValuePtr* 配列内の*個々の* 要素の長さになります。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* 配列内の値の位置を判別します。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。

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、SQL_DATA_AT_EXEC 値を *StrLen_or_IndPtr* バッファーに置くことによって、 DB2 CLI に実行時データ・パラメーターについて通知します。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場合、 *ParameterValuePtr* バッファーが *StrLen_or_IndPtr* バッファーの後のメモリーに連続的に置かれると、 DB2 CLI ではある程度パフォーマンスを拡張できます。たとえば、

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

パラメーターは、ファイルまたは記憶場所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は、有効なバインドを判別します。 **列サイズが事前に分からないとき**

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として 0 を指定できます。 (*ColumnSize* を 0 に設定する)。

以前のリリースでは、 *ColumnSize* が 0 に設定されていると DB2 CLI は列のデータ・タイプに可能な最大サイズを使用しました。場合によっては、これにより不必要に大きなメモリー・ブロックが割り振られる結果となりました。バージョン 6 で、この動作は変更されました。

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、2 進ストリング、またはラージ・オブジェクトである場合、 *ColumnSize* を 0 に設定すると別の意味を持ちます。

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。つまり、ストアード・プロシージャーは *BufferLength* バイトを超えるデータを戻すことができないということです。それを超えるデータを戻した場合、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。つまり、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ということ、そして戻されるデータの最大サイズは *BufferLength* バイトであるということです。

ColumnSize を 0 に設定する必要がなければ、そうすることは勧められていません。それにより、DB2 CLI は実行時にデータの長さを不必要にチェックすることになるからです。

パラメーター・バインドの相対位置

パラメーター・バインドの変更の必要が生じた場合、アプリケーションはもう一度 `SQLBindParameter()` を呼び出すことができます。これにより、バインドされているパラメーターのバッファー・アドレスと、それに対応する使用中の長さ / 標識バッファー・アドレスを変更します。

`SQLBindParameter()` への複数の呼び出しの代わりに、 DB2 CLI はパラメーター・バインドの相対位置もサポートしています。毎回再バインドするよりも、相対位置を使用すると、 `SQLExecute()` または `SQLExecDirect()` への次の呼び出しで使用される新しいバッファー・アドレスおよび長さ / 標識アドレスを指定することができます。これは、列方向配列の挿入では使用できませんが、アプリケーションが個々にまたは配列を使用してパラメーターをバインドするかどうかを決めます。

相対位置を使用するのに必要なステップのリストについては、 97ページの『パラメーター・バインドの相対位置』を参照してください。

記述子

パラメーターのバインド方法は、APD および IPD のフィールドによって決定されます。 SQLBindParameter 内の引き数を使用して、その記述子フィールドを設定します。そのフィールドは、また、SQLSetDescField 関数によっても設定できます。 SQLBindParameter は、使用効率が良く、アプリケーションが SQLBindParameter を呼び出すのに記述子ハンドルを獲得する必要はありません。

注: 1 つのステートメントについての SQLBindParameter() の呼び出しは、他のステートメントに影響することがあり得ます。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。 SQLBindParameter() は APD のフィールドを修正するため、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その修正が適用されます。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。

1. SQLGetStmtAttr() を呼び出して、APD ハンドルを獲得します。
2. SQLGetDescField() を呼び出して、 APD の SQL_DESC_COUNT フィールドを獲得します。 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。
3. 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 が日時または間隔サブタイプのコンサイス識別子の 1 つである場合は例外です。その場合は、SQL_DESC_TYPE を SQL_DATETIME または SQL_INTERVAL にそれぞれ設定し、 SQL_DESC_CONCISE_TYPE をコンサイス識別子に設定し、そして SQL_DESC_DATETIME_INTERVAL_CODE を対応日時または間隔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、標識情報およびパラメーター値の長さの両方を指定します。

4. SQLGetStmtAttr() を呼び出して、IPD ハンドルを獲得します。
5. SQLGetDescField() を呼び出して、IPD の SQL_DESC_COUNT フィールドを獲得します。 ColumnNumber 引き数の値が SQL_DESC_COUNT の値を超える場合は、 SQLSetDescField フィールドを呼び出して SQL_DESC_COUNT の値を ColumnNumber に増やします。
6. 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 が日時または間隔サブタイプのコンサイス識別子の 1 つである場合は例外です。その場合は、SQL_DESC_TYPE を SQL_DATETIME または SQL_INTERVAL にそれぞれ設定し、 SQL_DESC_CONCISE_TYPE をコンサイス識別子に設定し、そして SQL_DESC_DATETIME_INTERVAL_CODE を対応する日時または間隔サブコードに設定します。
   - ParameterType に適するように、1 つまたは複数の SQL_DESC_LENGTH、SQL_DESC_PRECISION、および SQL_DESC_DATETIME_INTERVAL_PRECISION を設定します。
   - SQL_DESC_SCALE を DecimalDigits の値に設定します。

SQLBindParameter() への呼び出しが失敗したときは、それが 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 22. SQLBindParameter SQLSTATE

SQLSTATE	説明	解説
07006	変換が無効です。	*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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
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 のデータ・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	引き数値が無効です。	引き数 *ParameterValuePtr* は NULL ポインターで、引き数 *StrLen_or_IndPtr* は NULL ポインターであり、*InputOutputType* は SQL_PARAM_OUTPUT ではありません。
HY010	関数の順序エラーです。	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・パラメーターに関するデータは送られませんでした。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY021	不整合な記述子情報	整合性検査時に検査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長が無効です。	引き数 *BufferLength* に指定された値は、0 より小さい値でした。
HY093	パラメーターの数値が無効です。	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。

表 22. SQLBindParameter SQLSTATE (続き)

SQLSTATE	説明	解説
HY094	位取り値が無効です。	*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	精度値が無効です。	*ParameterType* に指定された値が SQL_DECIMAL または SQL_NUMERIC のどちらかで、 *ParamDef* に指定された値が 1 より小さい値でした。
HY105	パラメーター・タイプが無効です。	*InputOutputType* が SQL_PARAM_INPUT、SQL_PARAM_OUTPUT、または SQL_PARAM_INPUT_OUTPUT のいずれでもありません。
HYC00	ドライバーが機能していません。	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。

制約

DB2 CLI v5 と ODBC 2.0 では、この関数は `SQLSetParam()` に代わるものとして用いられます。

StrLen_or_IndPtr の新しい値 SQL_DEFAULT_PARAM が ODBC 2.0 に導入され、プロシージャーで使用する値を、アプリケーションから送信された値ではなく、パラメーターの省略時値にするように指定できるようになりました。DB2 ストアード・プロシージャーの引き数には省略時値の概念がないため、*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 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、 SQLGetInfo() に SQL_NEED_LONG_DATA_LEN オプションを指定して、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、この情報が SQLPutData() に必要ないことを示す場合、'N' を戻します。

例

以下に、さまざまなデータ・タイプをバインドして、一組のパラメーターに結合する例を示します。追加の例については、139ページの『ストアード・プロシージャーの例』を参照してください。

```
/* From the CLI sample TBREAD.C */
/* ... */
    /* bind divisionParam to the statement */
    printf("    Bind divisionParam to the statement¥n");
    printf("        %s¥n", stmt);
    sqlrc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
                             SQL_CHAR, 15, 0, divisionParam, 15, NULL);
```

参照

- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 409ページの『SQLExecute - ステートメントの実行』
- 622ページの『SQLParamData - データ値が必要な次のパラメーターを入手する』
- 626ページの『SQLParamOptions - パラメーターに入力配列を指定する』
- 655ページの『SQLPutData - パラメーターのデータ値を渡す』

SQLBrowseConnect - データ・ソースへの接続に必要な属性を入手する

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。

構文

```
SQLRETURN   SQLBrowseConnect (SQLHDBC           ConnectionHandle,
                              SQLCHAR           *InConnectionString,
                              SQLSMALLINT       StringLength1,
                              SQLCHAR           *OutConnectionString,
                              SQLSMALLINT       BufferLength,
                              SQLSMALLINT       *StringLength2Ptr);
```

関数引き数

表 23. 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	*szConnStrIn	入力	要求接続ストリングをブラウズします (296ページの『InConnectionString 引き数』を参照)。
SQLSMALLINT	cbConnStrIn	入力	*InConnectionString の長さ。
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 (296ページの『OutConnectionString 引き数』を参照)。
SQLINTEGER	BufferLength	入力	*OutConnectionString バッファーの長さ。
SQLSMALLINT	*StringLength2Ptr	出力	*OutConnectionString 内に返すために使用できる総バイト数 (ヌル終了バイトを除く) 。戻りに使用できるバイト数が BufferLength より大きい場合、 *OutConnectionString 内の接続ストリングは、 BufferLength からヌル終了文字分を差し引いた長さに切り捨てられます。

使用法

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、

- 文字ストリングの文字数は 0 以上です。
- 識別子の文字数は 1 以上です。
- 属性キーワードは大文字小文字の区別がありません。
- 属性値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べきです。システム情報の文法上の理由から、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、中括弧が DRIVER キーワードの前後に必要です。

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、そのキーワードの最初のオカレンスに関連した値を使用します。**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に含まれている場合は、 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、次の構文になります。

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 ::= identifer[: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、
- 文字ストリングおよびローカライズ文字ストリングの文字数は 0 以上です。
- 識別子およびローカライズ識別子の文字数は、1 以上です。属性キーワードは、大文字小文字の区別がありません。
- 属性値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、ローカライズ識別子、および属性値は避けるべきです。システム情報の文法上の理由から、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、以下のセマンティクス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、属性は任意選択であり、 `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 ユニバーサル・データベースに接続するときは、**DSN**、**UID**、および **PWD** だけが必要になります。その他のキーワードは、指定することができますが、接続には影響ありません。
- ODBC 属性キーワードおよびドライバー定義の属性キーワードには、日本語化されたキーワードまたは使いやすくされたキーワードが含まれています。これは、ダイアログ・ボックス内のラベルとしてアプリケーションで使用できます。しかし、ブラウズ要求ストリングを DB2 CLI に渡すときは、**UID**、**PWD**、または識別子単独を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関する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が対象になり得ます。

- 属性値が単一の疑問符 (?) 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() への各呼び出しは、接続処理の次のレベルを満足させるのに必要な情報だけを返します。 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、状況情報を接続ハンドルに関連付けます。

SQLBrowseConnect の使用

SQLBrowseConnect() は、割り当てられた接続を必要とします。SQLBrowseConnect() が SQL_ERROR を返すとき、未解決の接続は終了し、その接続は未接続の状態に戻されます。

SQLBrowseConnect() が接続に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、ブラウズ結果の接続ストリング内で属性の継承レベルおよび属性値を返します。 DB2 CLI は、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、以前のブラウズ結果の接続ストリングの内容を、現在のブラウズ要求接続ストリングを構築する際には使用できないことにご注意ください。すなわち、アプリケーションは、前のレベルで設定された属性セットについて異なる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。接続ストリングは、SQL_DRIVER_NOPROMPT オプションを指定した SQLDriverConnect() 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() への別の呼び出しに使用することはできません。しかし、SQLBrowseConnect() が再度呼び出された場合は、呼び出しのシーケンス全体が繰り返されることになります。

また、SQLBrowseConnect() は、ブラウズ処理の間に、回復可能な非致命的エラーが起きる場合、SQL_NEED_DATA を返します。そのようなエラーには、たとえば、アプリケーションによって提供された無効なパスワードや、アプリケーションによって提供された無効な属性キーワードがあります。
SQL_NEED_DATA が返されて、ブラウズ結果の接続ストリングが未変更の場合、エラーが発生し、アプリケーションは、 SQLGetDiagRec() を呼び出してブラウズ時のエラーについて SQLSTATE を返します。これにより、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、 SQLDisconnect() を呼び出すことによりいつでもブラウズ処理を終了させることができます。 DB2 CLI は、未解決の接続を終了させて、接続を非接続状態へ戻します。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NEED_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 24. SQLBrowseConnect SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
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 **StringLength2Ptr* 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 を戻します。)
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。)

表 24. SQLBrowseConnect SQLSTATE (続き)

SQLSTATE	説明	解説
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 を戻します。)
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、処理系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。
28000	許可指定が無効です。	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、ユーザー識別子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	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY090	ストリングまたはバッファー長が無効です。	引き数 *StringLength1* 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 *BufferLength* に指定された値は、0 より小さい値でした。

制約

なし。

例

該当するサンプルの一覧については、 sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内の README ファイルを参照してください。

参照

- 246ページの『SQLAllocHandle - ハンドルを割り振る』
- 356ページの『SQLConnect - データ・ソースに接続する』
- 379ページの『SQLDisconnect - データ・サーバーからの切断』
- 382ページの『SQLDriverConnect - データ・ソースに (拡張) 接続する』
- 470ページの『SQLFreeHandle - ハンドル資源を解放する』

SQLBuildDataLink - 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,
                             SQLCHAR     FAR   *LinkType,
                             SQLINTEGER        LinkTypeLength,
                             SQLCHAR     FAR   *DataLocation,
                             SQLINTEGER        DataLocationLength,
                             SQLCHAR     FAR   *Comment,
                             SQLINTEGER        CommentLength,
                             SQLCHAR     FAR   *DataLinkValue,
                             SQLINTEGER        BufferLength,
                             SQLINTEGER  FAR   *StringLengthPtr);
```

関数引き数

表 25. SQLBuildDataLink 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	StatementHandle	入力	診断レポートのためだけに使用されます。
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 *	コメント	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。

表 25. SQLBuildDataLink 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER	*StringLengthPtr	出力	**DataLinkValue* 内に戻すために使用できる総バイト数 (ヌル終止符を除く) を戻すバッファーを指すポインター。*DataLinkValue* がヌル・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 からヌル終了文字の長さを引いた長さよりも大きい場合、 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。ヌル終止符を含むストリングの最大長は、*BufferLength* バイトになります。

データ・リンクの詳細については、*管理の手引き: 計画* 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 26. SQLBuildDataLink() SQLSTATE

SQLSTATE	説明	解説
01000	警告	特定の SQLSTATE がなかった場合のエラーが発生しました。 `SQLGetDiagRec()` により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
01004	データは切り捨てられました。	**DataLinkValue* に戻されるデータは、 *BufferLength* からヌル終止符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 を戻します。)

表 26. SQLBuildDataLink() SQLSTATE (続き)

SQLSTATE	説明	解説
HY000	一般エラー。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリー割り振り失敗。	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を割り当てられませんでした。
HY090	ストリングまたはバッファー長が無効です。	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。

制約

なし。

例

```
/* ... */
void getattr( SQLHSTMT hStmt,
              SQLSMALLINT AttrType,
              SQLCHAR* DataLink,
              SQLCHAR* Attribute,
              SQLINTEGER BufferLength)
{
    SQLINTEGER  StringLength ;
    SQLRETURN   rc ;
    rc = SQLGetDataLinkAttr(
                               hStmt,
                               AttrType,
                               DataLink,
                               strlen( (char *)DataLink),
                               Attribute,
                               BufferLength,
                              &StringLength
                             ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hStmt, rc ) ;
    printf("Attribute #%d) >%s>¥n", AttrType, Attribute) ;
    return ;
}
/* ... */
    SQLCHAR szCreate[] = "CREATE TABLE DL_SAMPL "
                                          "( "
                                          "DL1 DATALINK "
                                             "LINKTYPE URL "
                                             "FILE LINK CONTROL "
                                             "INTEGRITY ALL "
                                             "READ PERMISSION DB "
                                             "WRITE PERMISSION BLOCKED "
```

```
                                        "RECOVERY NO "
                                        "ON UNLINK RESTORE "
                                      ")";
    SQLCHAR szInsert[] = "INSERT INTO DL_SAMPL VALUES (?)" ;
    SQLCHAR szFileLink[] =
         "http://fearless.torolab.ibm.com/nfsdlink/rpomeroy/test_1.jpg" ;
    SQLCHAR szComment[] = "My First Datalink" ;
    SQLCHAR szSelect[] = "SELECT * FROM DL_SAMPL" ;
    SQLCHAR szDrop[] = "DROP TABLE DL_SAMPL" ;
    SQLCHAR szDLCol[254] ;
    SQLCHAR szBuffer[254] ;
    SQLSMALLINT cCol ;
    SQLCHAR szColName[33] ;
    SQLSMALLINT fSqlType ;
    SQLUINTEGER cbColDef ;
    SQLSMALLINT ibScale ;
    SQLSMALLINT fNullable ;
    SQLINTEGER  siLength = SQL_NTS ;
/* ... */
    /* Build Datalink */
    rc = SQLBuildDataLink( hstmt,
                           (SQLCHAR *)"URL",
                           strlen("URL"),
                           szFileLink,
                           strlen((char *)szFileLink),
                           szComment,
                           strlen((char *)szComment),
                           szDLCol,
                           sizeof(szDLCol),
                          &siLength
                         );
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    /*  Set input parameter.  */
    rc = SQLBindParameter(
                           hstmt,
                           1,
                           SQL_PARAM_INPUT,
                           SQL_C_DATALINK,
                           SQL_DATALINK,
                           sizeof(szDLCol),
                           0,
                           (SQLPOINTER)szDLCol,
                           sizeof(szDLCol),
                           NULL
                         ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
/* ... */
}                               /* end main */
```

参照

- 497ページの『SQLGetDataLinkAttr - データ・リンク属性値を入手する』

SQLBulkOperations - 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 27. 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。
SQLSMALLINT	操作	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視点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。詳細については、254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』を参照してください。

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。

バルク挿入の実行

SQLBulkOperations() を使用してデータを挿入するため、アプリケーションは以下の一連の手順を実行します。

1. 結果セットを戻す照会を実行する。
2. SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性を挿入したい行数に設定する。

3. SQLBindCol() を呼び出して、挿入したいデータをバインドする。データは SQL_ATTR_ROW_ARRAY_SIZE 値に等しいサイズの配列にバインドされます。

   注: SQL_ATTR_ROW_STATUS_PTR ステートメント属性が示す配列のサイズは SQL_ATTR_ROW_ARRAY_SIZE と等しいか、または SQL_ATTR_ROW_STATUS_PTR がヌル・ポインターでなければなりません。

4. 挿入を実行するために、 SQLBulkOperations(*StatementHandle*, SQL_ADD) を呼び出す。
5. 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た場合、そのアプリケーションはこの配列を検査して操作の結果を知ることができます。

アプリケーションが *Operation* 引き数に SQL_ADD を指定して SQLBulkOperations() を呼び出す前に列 0 をバインドした場合、 CLI はブックマークのあるバインドされた列 0 バッファーを新しく挿入された行によって更新します。そのために、アプリケーションはステートメントの実行前に SQL_ATTR_USE_BOOKMARKS ステートメント属性を SQL_UB_VARIABLE に設定していなければなりません。

長いデータは SQLParamData() および SQLPutData() への呼び出しを行う SQLBulkOperations() によって、複数の部分として追加できます。詳細については、このセクションの『長いデータを使用してバルク挿入およびバルク更新を行う』を参照してください。

アプリケーションは SQLBulkOperations() を呼び出す前に、 SQLFetch() または SQLFetchScroll() を呼び出す必要があります。

重複した列を含むカーソル上で *Operation* 引き数に SQL_ADD を指定して SQLBulkOperations() を呼び出した場合、エラーが戻されます。

ブックマークを使用してバルク更新を実行する

SQLBulkOperations() と共にブックマークを使用してバルク更新を実行するため、アプリケーションは以下の一連の手順を実行します。

1. SQL_ATTR_USE_BOOKMARKS ステートメント属性を SQL_UB_VARIABLE に設定する。
2. 結果セットを戻す照会を実行する。

3. SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性を更新したい行数に設定する。

   SQLBindCol() を呼び出して、更新したいデータをバインドする。データは SQL_ATTR_ROW_ARRAY_SIZE 値に等しいサイズの配列にバインドされます。さらに、SQLBindCol() を呼び出して列 0 (ブックマークの列) をバインドします。
4. 更新したい行のブックマークを列 0 にバインドされた配列にコピーする。
5. バインドされたバッファー内のデータを更新する。

   注: SQL_ATTR_ROW_STATUS_PTR ステートメント属性が示す配列のサイズは SQL_ATTR_ROW_ARRAY_SIZE と等しいか、または SQL_ATTR_ROW_STATUS_PTR がヌル・ポインターでなければなりません。
6. SQLBulkOperations()(*StatementHandle*, SQL_UPDATE_BY_BOOKMARK) を呼び出す。

注: 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た場合、そのアプリケーションはこの配列を検査して操作の結果を知ることができます。

オプションとして、SQLBulkOperations(*StatementHandle*, SQL_FETCH_BY_BOOKMARK) を呼び出してデータをバインドされたアプリケーション・バッファーに取り出し、更新が行われたことを確認します。

データが更新されている場合、CLI は適切な行の行状況配列の値を SQL_ROW_UPDATED に変更します。

SQLParamData() および SQLPutData() への呼び出しを行うことによって、SQLBulkOperations() で実行されるバルク更新には長いデータを含めることができます。詳細については、このセクションの『長いデータを使用してバルク挿入およびバルク更新を行う』を参照してください。

DB2 CLI 内のブックマークは複数のカーソルにまたがることができません。つまり、アプリケーションは直前のカーソルから保存したブックマークを使用することができません。ブックマークによる更新を行う前に SQLFetch() または SQLFetchScroll() を使用して、ブックマークを取得する必要があります。

重複した列を含むカーソル上で *Operation* 引き数に SQL_UPDATE_BY_BOOKMARK を指定して SQLBulkOperations() を呼び出した場合、エラーが戻されます。

ブックマークを使用してバルク取り出しを実行する

SQLBulkOperations() と共にブックマークを使用してバルク取り出しを実行するため、アプリケーションは以下の一連の手順を実行します。

1. SQL_ATTR_USE_BOOKMARKS ステートメント属性を SQL_UB_VARIABLE に設定する。
2. 結果セットを戻す照会を実行する。
3. SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性を取り出したい行数に設定する。
4. SQLBindCol() を呼び出して、取り出したいデータをバインドする。データは SQL_ATTR_ROW_ARRAY_SIZE 値に等しいサイズの配列にバインドされます。さらに、SQLBindCol() を呼び出して列 0 (ブックマークの列) をバインドします。
5. 取り出したい行のブックマークを列 0 にバインドされた配列にコピーする。 (これにより、アプリケーションがブックマークをすでに別個に取得していると想定されます。)

   注: SQL_ATTR_ROW_STATUS_PTR ステートメント属性が示す配列のサイズは SQL_ATTR_ROW_ARRAY_SIZE と等しいか、または SQL_ATTR_ROW_STATUS_PTR ステートメント属性がヌル・ポインターでなければなりません。
6. SQLBulkOperations(*StatementHandle*, SQL_FETCH_BY_BOOKMARK) を呼び出す。
7. 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た場合、そのアプリケーションはこの配列を検査して操作の結果を知ることができます。

DB2 CLI 内のブックマークは複数のカーソルにまたがることができません。つまり、アプリケーションは直前のカーソルから保存したブックマークを使用することができません。ブックマークによる更新を行う前に SQLFetch() または SQLFetchScroll() を使用して、ブックマークを取得する必要があります。

ブックマークを使用してバルク削除を実行する

SQLBulkOperations() と共にブックマークを使用してバルク削除を実行するため、アプリケーションは以下の一連の手順を実行します。

1. SQL_ATTR_USE_BOOKMARKS ステートメント属性を SQL_UB_VARIABLE に設定する。
2. 結果セットを戻す照会を実行する。

3. SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性を削除したい行数に設定する。
4. SQLBindCol() を呼び出して列 0 (ブックマークの列) をバインドする。
5. 削除したい行のブックマークを列 0 にバインドされた配列にコピーする。

   注: SQL_ATTR_ROW_STATUS_PTR ステートメント属性が示す配列のサイズは SQL_ATTR_ROW_ARRAY_SIZE と等しいか、または SQL_ATTR_ROW_STATUS_PTR ステートメント属性がヌル・ポインターでなければなりません。
6. SQLBulkOperations(*StatementHandle*, SQL_DELETE_BY_BOOKMARK) を呼び出す。
7. 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た場合、そのアプリケーションはこの配列を検査して操作の結果を知ることができます。

DB2 CLI 内のブックマークは複数のカーソルにまたがることができません。つまり、アプリケーションは直前のカーソルから保存したブックマークを使用することができません。ブックマークによる更新を行う前に SQLFetch() または SQLFetchScroll() を使用して、ブックマークを取得する必要があります。

長いデータを使用してバルク挿入およびバルク更新を行う

SQLBulkOperations() を呼び出して実行するバルク挿入およびバルク更新では、長いデータを使用できます。アプリケーションが長いデータを挿入または更新するためには、前述の『バルク挿入を実行する』および『ブックマークを使用してバルク更新を実行する』の項で説明した手順に加えて、以下の手順を実行します。

1. SQLBindCol() を使用してデータをバインドするとき、アプリケーションは列番号などのアプリケーション定義値を **TargetValuePtr* バッファーの data-at-execution 列に入れます。後にその値を使用して列を識別できます。

   アプリケーションは、 SQL_LEN_DATA_AT_EXEC(*length*) マクロの結果を **StrLen_or_IndPtr* バッファーに入れます。列の SQL データ・タイプが SQL_LONGVARBINARY、SQL_LONGVARCHAR、または長い、データ・ソースに特定のデータ・タイプであり、 CLI が SQL_NEED_LONG_DATA_LEN 情報タイプとして「Y」を SQLGetInfo() に戻す場合、 *length* はパラメーターに送るデータのバイト数です。その他の場合、負でない値を指定して、その値は無視されます。

2. SQLBulkOperations() が呼び出されたとき、 data-at-execution 列が存在すれば、関数は SQL_NEED_DATA を戻して下記のステップ 3 に進みます。(data-at-execution 列が存在しなければ、処理は完了します。)
3. アプリケーションは SQLParamData() を呼び出して、最初に処理する data-at-execution 列の **TargetValuePtr* バッファーのアドレスを検索します。 SQLParamData() は SQL_NEED_DATA を戻します。アプリケーションは、**TargetValuePtr* バッファーからアプリケーション定義の値を検索します。

   注: data-at-execution パラメーターは data-at-execution 列と類似していますが、 SQLParamData() によって戻される値はそれぞれ異なります。

   Data-at-execution 列は、 SQLBulkOperations() によって行が更新または挿入されたときにデータが SQLPutData() と共に送られる行セット内の列です。それらは SQLBindCol() にバインドされます。SQLParamData() によって戻される値は、処理中の **TargetValuePtr* バッファー内の行のアドレスです。
4. アプリケーションは SQLPutData() を 1 回以上呼び出して、列のデータを送ります。すべてのデータ値を SQLPutData() で指定された **TargetValuePtr* バッファーに戻すことができない場合、複数の呼び出しが必要です。同じ列に対して SQLPutData() を複数回呼び出すことが許可されるのは、文字 C データを文字、バイナリー、またはデータ・ソースに特定のデータ・タイプの列に送るとき、またはバイナリー C データを文字、バイナリー、またはデータ・ソースに特定のデータ・タイプの列に送るときだけです。
5. アプリケーションは再び SQLParamData() を呼び出して、すべてのデータが列に送られたことを知らせます。
   - さらに他の data-at-execution 列がある場合、 SQLParamData() は次に処理する data-at-execution 列の SQL_NEED_DATA および *TargetValuePtr* バッファーのアドレスを戻します。アプリケーションは上記のステップ 4 および 5 を繰り返します。
   - さらに他の data-at-execution 列が存在しなければ、処理は完了します。ステートメントが正常に実行された場合、 SQLParamData() は SQL_SUCCESS または SQL_SUCCESS_WITH_INFO を戻します。実行が失敗した場合、SQL_ERROR を戻します。この時点で、 SQLParamData() は SQLBulkOperations() が戻すことのできる SQLSTATE を戻します。

SQLBulkOperations() が SQL_NEED_DATA を戻した後でデータがすべての data-at-execution 列に送られる前に、操作が取り消されるか SQLParamData() ま

たは SQLPutData() でエラーが生じた場合、アプリケーションがステートメントまたはステートメントに関連した接続で呼び出せるのは SQLCancel()、SQLGetDiagField()、SQLGetDiagRec()、SQLGetFunctions()、SQLParamData()、または SQLPutData() だけです。そのステートメントで、またはそのステートメントに関連した接続で他の関数を呼び出すと、その関数は SQL_ERROR および SQLSTATE HY010 (関数順序エラー) を戻します。

CLI が data-at-execution 列のためにデータをまだ必要としているときにアプリケーションが SQLCancel() を呼び出すと、 CLI は操作を取り消します。その後、アプリケーションは SQLBulkOperations() を再び呼び出せます。取り消しによってカーソル状態または現行カーソル位置が影響を受けることはありません。

行状況の配列

SQLBulkOperations() を呼び出した後、行状況配列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。この配列内の状況値は以下の関数を呼び出した後に設定されます。

- SQLFetch()
- SQLFetchScroll()
- SQLSetPos()
- SQLBulkOperations()

SQLFetch() または SQLFetchScroll() が SQLBulkOperations() よりも前に呼び出されていない場合、この配列は最初 SQLBulkOperations() への呼び出しによって移植されます。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。行状況配列の要素数は、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。) 行状況配列については、434ページの『SQLFetch - 次の行の取り出し』を参照してください。

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 28. SQLBulkOperations SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
01004	切り捨てられたデータ。	*Operation* 引き数は SQL_FETCH_BY_BOOKMARK 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 2 進データは、非ブランク文字または非ヌルの 2 進データの短縮形となります。
01S07	無効な変換。	*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る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す。時刻、タイム・スタンプ、およびインターバル C 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。) (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
07006	制限付きデータ・タイプ属性違反。	*Operation* 引き数は SQL_FETCH_BY_BOOKMARK であり、結果セットにある列のデータ値を、 `SQLBindCol()` への呼び出しで *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	記述子索引が無効です。	引き数 *Operation* は SQL_ADD であり、列は結果セット内の列数よりも大きい列番号にバインドされました。
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、読取専用であるか、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	文字列データの右を切り捨てました。	結果セット内の列への文字または 2 進値の割り当てが、非ブランク文字 (文字の場合) または非ヌル文字 (2 進数の場合) またはバイトに切り捨てられました。

表 28. 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、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_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	無効な日時形式。	*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 / 時フィールドの桁あふれ。	*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。

表 28. SQLBulkOperations SQLSTATE (続き)

SQLSTATE	説明	解説
22015	インターバル・フィールドの桁あふれ。	*Operation* 引き数は SQL_ADD または SQL_UPDATE_BY_BOOKMARK で、厳密な数値またはインターバル C タイプをインターバル SQL データ・タイプに割り当てた結果、有効数字が失われました。 *Operation* 引き数は SQL_ADD または SQL_UPDATE_BY_BOOKMARK で、インターバル SQL タイプに割り当てるとき、インターバル SQL タイプ内に C タイプの値を表すものがありませんでした。 *Operation* 引き数は SQL_FETCH_BY_BOOKMARK で、厳密な数値またはインターバル SQL タイプからインターバル C タイプに割り当てた結果、先頭フィールドの有効数字が失われました。 *Operation* 引き数は SQL_FETCH_BY_BOOKMARK で、インターバル C タイプに割り当てるとき、インターバル C タイプ内に SQL タイプの値を表すものがありませんでした。
22018	キャスト指定の文字値が無効。	*Operation* 引き数は SQL_FETCH_BY_BOOKMARK で、 C タイプは厳密またはおおよその数値、日時、またはインターバル・データ・タイプです。列にある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。 引き数 *Operation* は SQL_ADD または SQL_UPDATE_BY_BOOKMARK です。 SQL タイプは厳密またはおおよその数値、日時、またはインターバル・データ・タイプです。 C タイプは SQL_C_CHAR です。列にある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。
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、省略時値がありません。 *Operation* 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、列には省略時値がありませんでした。

表 28. SQLBulkOperations SQLSTATE (続き)

SQLSTATE	説明	解説
24000	無効なカーソル状態。	*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	逐次化障害。	トランザクションは、別のトランザクションとの資源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、挿入または更新が視点付き表または WITH CHECK OPTION を指定して作成した視点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 つ以上の行が視点付き表に含まれなくなります。
HY000	汎用エラー	処理系固有の SQLSTATE が存在しなかったり、定義されなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
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ために必要なメモリーを割り当てられませんでした。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、 SQLCancel() が複数スレッドの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。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。

表 28. SQLBulkOperations SQLSTATE (続き)

SQLSTATE	説明	解説
HY011	操作はこの時点では無効です。	SQL_ATTR_ROW_STATUS_PTR ステートメント属性は、SQLFetch() または SQLFetchScroll()、および SQLBulkOperations に対する呼び出しの間に設定されました。
HY013	予期しないメモリー処理エラー。	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長が無効。	*Operation* 引き数は SQL_ADD または SQL_UPDATE_BY_BOOKMARK であり、データ値はヌル・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、データ値はヌル・ポインターでなく、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_BOOKMARK ステートメント属性は SQL_UB_VARIABLE に設定され、列 0 は長さがこの結果セットのブックマークの最大長に等しくないバッファーにバインドされました。 (この長さは IRD の SQL_DESC_OCTET_LENGTH フィールドに示されていて、SQLDescribeCol()、 SQLColAttribute()、または SQLGetDescField() を呼び出すことによって取得できます。)

表 28. SQLBulkOperations SQLSTATE (続き)

SQLSTATE	説明	解説
HY092	無効な属性識別子。	*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	任意選択機能は実装されませんでした。	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() を通じて設定します。
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。

制約

なし。

例

該当するサンプルの一覧については、 sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内の README ファイルを参照してください。

参照

- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 321ページの『SQLCancel - ステートメントを取り消す』
- 446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』

- 501ページの『SQLGetDescField - 記述子レコードの単一フィールド設定を入手する』
- 507ページの『SQLGetDescRec - 記述子レコードの複数フィールド設定を入手する』
- 696ページの『SQLSetDescField - 記述子レコードの単一フィールドを設定する』
- 728ページの『SQLSetDescRec - 列またはパラメーター・データに複数の記述子フィールドを設定する』
- 744ページの『SQLSetPos - 行セット (Rowset) でカーソル位置を設定する』
- 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』

SQLCancel - ステートメントを取り消す

目的

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLCancel() を使用して、 87ページの『長形式データの分割送信 / 取り出し』で説明している*実行時データ* 順序列を永久終了することができます。

マルチ・スレッド・アプリケーションでは、SQLCancel() は元の要求を取り消し、 **HY**008 の SQLSTATE を返します。

構文

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

関数引き数

表 29. 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 87ページの『長形式データの分割送信 / 取り出し』に記述されている実行時データ・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。

DB2 CLI バージョン 2 で、または SQL_ATTR_ODBC_VERSION 環境属性が SQL_OV_ODBC2 に設定されている場合に、ステートメントについて何も処理が行われていないときにアプリケーションが SQLCancel() を呼び出すと、SQLCancel() は、 SQL_CLOSE オプションを選択した SQLFreeStmt() と同じ効果を持ちます。このことは、DB2 CLI バージョン 5 の場合、または SQL_ATTR_ODBC_VERSION 環境属性が SQL_OV_ODBC5 に設定されている場合には、該当しません。ステートメントについて何も処理が行われていないときに、 SQLCancel() への呼び出しがなされると、 SQL_CLOSE オプションを指定したときの SQLFreeStmt() と同じ取り扱いにはならず、何の効果もあり

ません。カーソルをクローズするには、アプリケーションが SQLCancel() を呼び出すのではなく、 SQLFreeStmt() を使用する必要があります。

非同期処理の取り消し

アプリケーションは、関数を非同期に呼び出した後に、関数を繰り返し呼び出して、処理を完了したかどうかを判別します。関数がまだ処理中の場合は、SQL_STILL_EXECUTING を返します。関数が処理を完了した場合は、異なるコードを返します。

SQL_STILL_EXECUTING を返した関数に任意の呼び出しをした後で、アプリケーションは、 SQLCancel() を呼び出して、関数を取り消すことができます。取り消し要求が成功したときは、SQL_SUCCESS が返されます。このメッセージは、関数が実際に取り消されたことを示すものではなく、取り消し要求が処理されたことを示すものです。アプリケーションは引き続き、戻りコードが SQL_STILL_EXECUTING でなくなるまで、元の関数を呼び出す必要があります。関数が正常に取り消された場合は、戻りコードは、 SQL_ERROR および SQLSTATE HY008 (操作が取り消されました。) になります。関数が通常の処理を終了した場合、戻りコードは、関数が成功すれば、SQL_SUCCESS または SQL_SUCCESS_WITH_INFO になり、関数が失敗すれば、SQL_ERROR および HY008 (操作が取り消されました。) 以外の SQLSTATE になります。

非同期処理の詳細については、145ページの『CLI の非同期実行』を参照してください。

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、ステートメントについて非同期に実行中の関数を取り消すことができます。関数を取り消すには、アプリケーションは、ターゲット関数で使用されたのと同様 (ただし異なるスレッドで)、同じステートメント・ハンドルを指定して SQLCancel() を呼び出します。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。非同期に実行している関数の取り消しに関して、 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。元の関数が取り消される場合、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、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場合、実行が成功して SQL_SUCCESS を返す一方で、取り消しも成功することが可能です。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。

スレッド化の詳細については、50ページの『マルチスレッドのアプリケーション作成』を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_INVALID_HANDLE
- SQL_ERROR

診断

表 30. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY018	サーバーが取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・エラーが起きました。	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。

制約

なし。

例

```
/* From the CLI sample dtlob.c */
/* ... */

        /* cancel the DATA AT EXEC state for hstmt */
            sqlrc = SQLCancel(hstmt);
            STMT_HANDLE_CHECK( hstmt, sqlrc);
        }
```

参照

- 655ページの『SQLPutData - パラメーターのデータ値を渡す』
- 622ページの『SQLParamData - データ値が必要な次のパラメーターを入手する』

SQLCloseCursor - カーソルをクローズして保留の結果を廃棄する

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、保留の結果を廃棄します。

構文

```
SQLRETURN   SQLCloseCursor   (SQLHSTMT          StatementHandle);
```

関数引き数

表 31. 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

アプリケーションが SQLCloseCursor() を呼び出した後で、アプリケーションは、 **SELECT** ステートメントを同じかまたは異なるパラメーター値で再実行することにより、カーソルを再オープンすることができます。

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、例外としては、SQL_CLOSE を指定した SQLFreeStmt() では、カーソルがステートメントにオープンしておらず、 SQLCloseCursor() が SQLSTATE 24000 (無効なカーソル状態) を返すときは、アプリケーションに対して効果がないことです。

読み取りロックの解放

接続属性 SQL_ATTR_CLOSE_BEHAVIOR は、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 CLI に試行させるかどうかを指示するために使用できます。

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、そのカーソルに保持されていたすべての読み取りロックがあれば、その解放を試行します。読み取りロックは、IS、S、および U 表ロックと、S、NS、および U 行ロックです。

接続属性については、663ページの『SQLSetConnectAttr - 接続属性を設定する』、特に 671ページの『SQL_ATTR_CLOSE_BEHAVIOR 接続属性』を参照してください。読み取りロックの解放については、以下の資料を参照してください。

- *SQL 解説書* の『CLOSE ステートメント』の『WITH RELEASE』
- *管理の手引き* の『CLOSE CURSOR WITH RELEASE』セクション

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 32. 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 を戻します。)
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	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY010	関数の順序エラーです。	*StatementHandle* の非同期実行関数が呼び出され、この関数が呼び出された時点でまだ実行中でした。 *StatementHandle* のために SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が戻されました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。

制約

なし。

例

```
/* From the CLI sample dtlob.c */
/* ... */
    /* bind the file-parameter
    /* close the cursor */
    sqlrc = SQLCloseCursor( hstmt ) ;
    STMT_HANDLE_CHECK( hstmt, sqlrc);

/* ... */

    /* close the cursor */
    sqlrc = SQLCloseCursor( hstmt ) ;
    STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参照

- 321ページの『SQLCancel - ステートメントを取り消す』
- 470ページの『SQLFreeHandle - ハンドル資源を解放する』
- 610ページの『SQLMoreResults - さらに結果セットがあるかどうかを判別する』

SQLColAttribute - 列属性を返す

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。

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 33. 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。 列 0 は、このステートメントで指定できますが、SQL_DESC_TYPE と SQL_DESC_OCTET_LENGTH を除いてすべての値が、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 (330ページの表 34 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。

表 33. SQLColAttribute 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER	BufferLength	入力	フィールドが文字ストリングの場合、*CharacterAttributePtr* バッファーの長さ。それ以外の場合は、このフィールドは無視されます。
SQLSMALLINT	*StringLengthPtr	出力	**CharacterAttributePtr* に戻すために使用可能な総バイト数 (文字データのヌル終了バイトを除く) 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* バイトからヌル終了文字分の長さを差し引いたバイトに切り捨てられ、DB2 CLI によってヌル終了します。 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。その他のすべての形式の情報は、**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 34. SQLColAttribute 引き数

FieldIdentifier	情報の戻り先	説明
SQL_COLUMN_AUTO_INCREMENT	Numeric AttributePtr	SQL_DESC_AUTO_UNIQUE_VALUE に変更されました。[a]
SQL_COLUMN_CASE_SENSITIVE	Numeric AttributePtr	SQL_DESC_CASE_SENSITIVE に変更されました。[a]
SQL_COLUMN_CATALOG_NAME	Character AttributePtr	SQL_DESC_CATALOG_NAME に変更されました。[a]
SQL_COLUMN_COUNT	Numeric AttributePtr	SQL_DESC_COUNT に変更されました。[a]
SQL_COLUMN_DISPLAY_SIZE	Numeric AttributePtr	SQL_DESC_DISPLAY_SIZE に変更されました。[a]
SQL_COLUMN_LABEL	Character AttributePtr	SQL_DESC_LABEL に変更されました。[a]
SQL_COLUMN_DISTINCT_TYPE	Character AttributePtr	SQL_DESC_DISTINCT_TYPE に変更されました。[a]
SQL_COLUMN_LENGTH	Numeric AttributePtr	SQL_DESC_OCTET_LENGTH に変更されました。[a]
SQL_COLUMN_MONEY	Numeric AttributePtr	SQL_DESC_FIXED_PREC_SCALE に変更されました。[a]
SQL_COLUMN_NAME	Character AttributePtr	SQL_DESC_NAME に変更されました。[a]
SQL_COLUMN_NULLABLE	Numeric AttributePtr	SQL_DESC_NULLABLE に変更されました。[a]
SQL_COLUMN_OWNER_NAME	Character AttributePtr	SQL_DESC_SCHEMA_NAME に変更されました。[a]

表 34. *SQLColAttribute 引き数 (続き)*

FieldIdentifier	情報の戻り先	説明
SQL_COLUMN_PRECISION	Numeric AttributePtr	SQL_DESC_PRECISION に変更されました。[a]
SQL_COLUMN_QUALIFIER_NAME	Character AttributePtr	SQL_DESC_CATALOG_NAME に変更されました。[a]
SQL_COLUMN_SCALE	Numeric AttributePtr	SQL_DESC_SCALE に変更されました。[a]
SQL_COLUMN_SEARCHABLE	Numeric AttributePtr	SQL_DESC_SEARCHABLE に変更されました。[a]
SQL_COLUMN_TABLE_NAME	Character AttributePtr	SQL_DESC_TABLE_NAME に変更されました。[a]
SQL_COLUMN_TYPE	Numeric AttributePtr	SQL_DESC_TYPE に変更されました。[a]
SQL_COLUMN_TYPE_NAME	Character AttributePtr	SQL_DESC_TYPE_NAME に変更されました。[a]
SQL_COLUMN_UNSIGNED	Numeric AttributePtr	SQL_DESC_UNSIGNED に変更されました。[a]
SQL_COLUMN_UPDATABLE	Numeric AttributePtr	SQL_DESC_UPDATABLE に変更されました。[a]
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。 SQL_FALSE は、すべての DB2 SQL データ・タイプについて *NumericAttributePtr* に返されます。
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、読取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不能である場合、この変数には空ストリングが入ります。

表 34. 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図形データ・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	列を含む表のカタログが *CharacterAttributePtr* に返されます。 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)	Character 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	文字形式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。 列タイプのおのおのの表示サイズについては、878ページの表197 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。

表 34. SQLColAttribute 引き数 (続き)

FieldIdentifier	情報の戻り先	説明
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位取り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位取りを持っていない場合です。 SQL_FALSE は、すべての DB2 SQL データ・タイプについて *NumericAttributePtr* に返されます。
SQL_DESC_LABEL (DB2 CLI v2)	Character AttributePtr	列ラベルが、*CharacterAttributePtr* に返されます。列にラベルがない場合、列名または列式が返されます。列にラベルがなく、名前もない場合は、空ストリングが返されます。
SQL_DESC_LENGTH (DB2 CLI v2)	Numeric AttributePtr	文字ストリングまたは 2 進データ・タイプの最大文字長または実際の文字長のどちらかである数値。これは、固定長データ・タイプの最大文字長であるか、可変長データ・タイプの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ヌル終了バイトが除かれています。 この情報は、IRD の SQL_DESC_LENGTH レコード・フィールドから返されます。
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不能であるデータ・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不能であるデータ・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。

表 34. SQLColAttribute 引き数 (続き)

FieldIdentifier	情報の戻り先	説明
SQL_DESC_LOCAL_TYPE_NAME (DB2 CLI v5)	Character AttributePtr	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、通常はサーバーの省略時文字セットです。
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。 列名値は環境属性 SQL_ATTR_USE_LIGHT _OUTPUT_SQLDA の影響を受ける場合があります。詳細については、 734ページの『SQLSetEnvAttr - 環境属性を設定する』を参照してください。
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。

表 34. SQLColAttribute 引き数 (続き)

FieldIdentifier	情報の戻り先	説明
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・タイプが正確な数値データ・タイプである場合は、このフィールドには 10 の値が入ります。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。 SQL データ・タイプの個々の長さについては、878ページの表196 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 2 進ストリング (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 2 進ストリング (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 34. 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、列が保持できる文字 の最大数を示します。 列がグラフィック SQL データ・タイプの場合、*NumericAttributePtr* に返される精度は、列が保持できる 2 バイト文字 の最大数を指定します。 SQL データ・タイプの個々の精度については、 875 ページの表194 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	列の位取りの属性が返されます。 SQL データ・タイプの個々の位取りについては、877ページの表195 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。

表 34. SQLColAttribute 引き数 (続き)

FieldIdentifier	情報の戻り先	説明
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	列を含む表の名前が、*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_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が、33ページの表2 にリストされています。 *ColumnNumber* が 0 に等しいときは、可変長ブックマークについて SQL_BINARY が返され、固定長ブックマークについて 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、たとえば、SQL_DATETIME を返します。 この情報は、IRD の SQL_DESC_TYPE レコード・フィールドから返されます。
SQL_DESC_TYPE_NAME (DB2 CLI v2)	Character AttributePtr	列のタイプ (SQL ステートメントに入力したとおりのもの) が、 *CharacterAttributePtr* に返されます。 各データ・タイプについては、 32ページの『データ・タイプとデータ変換』の表にある TYPE_NAME 属性を参照してください。

表 34. SQLColAttribute 引き数 (続き)

FieldIdentifier	情報の戻り先	説明
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。
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。 • すべての DB2 SQL データ・タイプについて、SQL_ATTR_READWRITE_UNKNOWN が *NumericAttributePtr* に返されます。 • 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。 DB2 CLI は返しませんが、ODBC は以下の値も定義しています。 • SQL_DESC_UPDATABLE • SQL_UPDT_WRITE

注:

a DB2 バージョン 5 では、バージョン 2 *FieldIdentifier* 値が変更されています。バージョン 2 の古い *FieldIdentifier* は、今回のバージョンでもやはりサポートされていますが、新しい *FieldIdentifier* を使用することをお勧めします。

この関数は、SQLDescribeCol() の代替として拡張性のあるものです。 SQLDescribeCol() は、ANSI-89 SQL に基づく記述子情報の固定セットを返します。 SQLColAttribute() は、ANSI SQL-92 および DBMS ベンダーの拡張機能で使用可能な、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 35. SQLColAttribute SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
01004	データが切り捨てられました。	バッファー **CharacterAttributePtr* は、ストリング値全部を返せるほど十分に大きくなかったので、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 を戻します。)
07005	ステートメントが結果セットを返しませんでした。	StatementHandle に関連したステートメントが結果セットを返しませんでした。記述する列がありませんでした。
07009	記述子索引が無効です。	*ColumnNumber* に指定された値が 0 と同等であり、SQL_ATTR_USE_BOOKMARKS ステートメント属性が SQL_UB_OFF でした。引き数 *ColumnNumber* に指定された値は、0 より小さい値でした。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	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY008	操作が取り消されました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。

表 35. SQLColAttribute SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数の順序エラーです。	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。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。 *StatementHandle* のために SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が戻されました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、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長が無効です。	引き数 *BufferLength* に指定された値は、0 より小さい値でした。
HY091	記述子タイプが範囲外です。	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、処理系定義の値でもありませんでした。
HYC00	ドライバーが機能していません。	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。

データ・ソースが StatementHandle に関連した SQL ステートメントを評価する時期に依存して、 SQLPrepare() の後、SQLExecute() の前に呼び出されたときに、 SQLColAttribute() は、 SQLPrepare() または SQLExecute() によって返される任意の 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すべきです。

制約

なし。

例

```
/* From the CLI sample utilcli.c */
/* ... */
        /* get display size for column */
        sqlrc = SQLColAttribute( hstmt,
                        ( SQLSMALLINT ) ( i + 1 ),
                        SQL_DESC_DISPLAY_SIZE,
                        NULL,
                        0,
```

```
                    NULL,
                    &colDataDisplaySize ) ;
    STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参照

- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 321ページの『SQLCancel - ステートメントを取り消す』
- 369ページの『SQLDescribeCol - 列の属性のセットを返す』
- 434ページの『SQLFetch - 次の行の取り出し』
- 446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』

SQLColAttributes - 列属性を入手する

使用すべきでない関数

注:

ODBC バージョン 3 では、SQLColAttributes() を使用すべきではなく、その代わりに SQLColAttribute() を使用します。詳細については、328ページの『SQLColAttribute - 列属性を返す』を参照してください。

DB2 CLI のこのバージョンは、引き続き SQLColAttributes() をサポートしていますが、 DB2 CLI プログラムでは、最新の標準に適合するように SQLColAttribute() を最初から使用するようお勧めします。 上記の関数と、その他の使用すべきでない関数の詳細については、822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

新しい関数への移行

バージョン 2 の関数 SQLColAttributes() を用いて使用されるすべてのフィールド識別子は、 SQLColAttribute() を用いて使用するよう変更されました。旧フィールド識別子は、その代替値とともに、330ページの表34 にリストされています。

SQLSTATE 07002

SQLSTATE **07**002 を返すすべての関数は、また、アプリケーションが SQLSetColAttributes() を使用して DB2 CLI に結果セットの記述子情報を通知する場合に、その状態を返すことができます。しかし、結果セット内のすべての列について記述子情報を提供するわけではありません。 **07**002 を返す DB2 CLI 関数のリストについては、 855ページの『SQLState 相互参照表』を参照してください。

SQLColumnPrivileges - 表の列に関連した特権を入手する

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。

構文

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

関数引き数

表 36. 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。これは、ヌル・ポインターまたは長さゼロのストリングでなければなりません。
SQLSMALLINT	NameLength1	入力	*CatalogName* の長さ。これは、0 に設定する必要があります。
SQLCHAR *	SchemaName	入力	表名のスキーマ修飾子。
SQLSMALLINT	NameLength2	入力	*SchemaName* の長さ。
SQLCHAR *	TableName	入力	表名。
SQLSMALLINT	NameLength3	入力	*TableName* の長さ。
SQLCHAR *	ColumnName	入力	列名で結果セットを修飾するためのパターン値 を入れられるバッファー。
SQLSMALLINT	NameLength4	入力	*ColumnName* の長さ。

使用法

結果は、345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_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ことになります。

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。

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 文字よりも少ないので、アプリケーションは、出力バッファー用に 128 文字 (およびヌル終止符) を常にとっておくか、あるいは、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 列の実際の長さをそれぞれ判別するために、SQL_MAX_CATALOG_NAME_LEN、 SQL_MAX_SCHEMA_NAME_LEN、SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を使用して、選択的に SQLGetInfo() を呼び出すようにすることができます。

ColumnName 引き数は探索パターンを受け入れることに留意してください。有効な探索パターンの詳細については、73ページの『カタログ関数での入力引き数』を参照してください。

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 で返される列

列 1 TABLE_CAT (VARCHAR(128) データ・タイプ)
これは常に NULL です。

列 2 TABLE_SCHEM (VARCHAR(128) データ・タイプ)
TABLE_NAME を含むスキーマの名前。

列 3 TABLE_NAME (VARCHAR(128) 非 NULL データ・タイプ)
表または視点の名前。

列 4 COLUMN_NAME (VARCHAR(128) 非 NULL データ・タイプ)
指定された表または視点の列の名前。

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

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

列 7 PRIVILEGE (VARCHAR(128) データ・タイプ)
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 ユニバーサル・データベース、DB2 (MVS/ESA 版) および DB2 (VSE および VM 版) 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 ユニバーサル・データベース、DB2 (MVS/ESA 版)、DB2 (VSE および VM 版) 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 37. 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	逐次化障害	トランザクションは、別のトランザクションとの資源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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、`SQLCancel()` が複数スレッドの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。
HY010	関数順序エラー	非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。 `SQLExecute()`、`SQLExecDirect()`、または `SQLSetPos()` が *StatementHandle* で呼び出され、 SQL_NEED_DATA を戻しました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、この関数が呼び出されました。
HY009	引き数値が無効です。	*TableName* が NULL です。
HY014	もはやハンドルはありません。	DB2 CLI は、内部資源が原因でハンドルを割り当てることができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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 37. SQLColumnPrivileges SQLSTATE (続き)

SQLSTATE	説明	解説
HYC00	ドライバーが機能していません。	DB2 CLI は、表名の修飾子としてカタログ 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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

なし。

例

```
/* From the CLI sample TBINFO.C */
/* ... */
    /* call SQLColumnPrivileges */
    printf("¥n    Call SQLColumnPrivileges for:¥n");
    printf("        tbSchema = %s¥n", tbSchema);
    printf("        tbName = %s¥n", tbName);
    sqlrc = SQLColumnPrivileges( hstmt, NULL, 0,
                                 tbSchema, SQL_NTS,
                                 tbName, SQL_NTS,
                                 colNamePattern, SQL_NTS);
```

参照

- 348ページの『SQLColumns - 表の列の情報を入手する』
- 800ページの『SQLTables - 表情報の入手』

SQLColumns - 表の列の情報を入手する

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。

構文

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

関数引き数

表 38. 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	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。カタログは、3 部分から成る表名の最初の部分です。 これは、NULL ポインターまたは長さゼロのストリングでなければなりません。
SQLSMALLINT	NameLength1	入力	*CatalogName* の長さ。これは、0 に設定する必要があります。
SQLCHAR *	SchemaName	入力	スキーマ名で結果セットを修飾するためのパターン*値* を入れられるバッファー。
SQLSMALLINT	NameLength2	入力	*SchemaName* の長さ。
SQLCHAR *	TableName	入力	表名で結果セットを修飾するためのパターン*値* を入れられるバッファー。

表 38. SQLColumns 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	NameLength3	入力	*TableName* の長さ。
SQLCHAR *	ColumnName	入力	列名で結果セットを修飾するためのパターン値 を入れられるバッファー。
SQLSMALLINT	NameLength4	入力	*ColumnName* の長さ。

使用法

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。通常の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ことになります。

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。 350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。有効な探索パターンについては、73ページの『カタログ関数での入力引き数』を参照してください。

この関数は、結果セット内の列についての情報を返しません。 SQLDescribeCol() または SQLColAttribute() を代わりに使用する必要があります。

SQL_ATTR_LONGDATA_COMPAT 属性が、SQLSetConnectAttr() の呼び出しによって、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_LD_COMPAT_YES に設定される場合、LOB データ・タイプは、SQL_LONGVARCHAR、SQL_LONGVARBINARY、または SQL_LONGVARGRAPHIC として報告されます。

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 文字未満なので、アプリケーションは、出力バッファー用に 128 文字 (およびヌル終止符) を常

にとっておくか、あるいは、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_OWNER_SCHEMA_LEN、 SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を使用して選択的に SQLGetInfo() を呼び出すように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。これらの列は、バージョン 2 からバージョン 5 までの間に変更が加えられました。バージョン 5 またはそれ以降のサーバーに対して、バージョン 2 の DB2 CLI アプリケーション (SQLColumns() を使用するもの) を実行している場合は、詳細について、825ページの『SQLColumns() 戻り値の変更』を参照してください。 **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 (詳しくは、 168 ページの『db2cli.ini の構成』を参照)
- SQL_ATTR_OPTIMIZESQLCOLUMNS 接続属性 (詳しくは、663ページの『SQLSetConnectAttr - 接続属性を設定する』を参照)

上記のどちらかの値を設定すると、以下の列は情報を戻さなくなります。

- 12 REMARKS
- 13 COLUMN_DEF

SQLColumns で戻される列

列 1 TABLE_CAT (VARCHAR(128))
これは常に NULL です。

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

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

列 4 COLUMN_NAME (VARCHAR(128) 非 NULL)
列の識別子。指定された表、視点、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、33ページの表2 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 2 進ストリングを示す場合、この列には列の最大文字数の長さが入ります。

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、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。

875ページの表194 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、ヌル終止符は含まれていません。正確な数値データ・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。

878ページの表196 も参照してください。

列 9 DECIMAL_DIGITS (SMALLINT)

列のスケール。位取りが適用できないデータ・タイプの場合は NULL が戻されます。

877ページの表195 も参照してください。

列 10 NUM_PREC_RADIX (SMALLINT)

10、2、または NULL のどれか。 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 列にはその列に許可されているビット数が入ります。

DATA_TYPE が正確な数値データ・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、350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))

列の省略時値。省略時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。省略時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。省略時値が 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 を省略時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省略時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。省略時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、350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、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))
列がヌル可でないことがわかっている場合は文字列「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 39. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。

表 39. SQLColumns SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。
HY014	もはやハンドルはありません。	DB2 CLI は、内部資源が原因でハンドルを割り当てることができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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	ドライバーが機能していません。	DB2 CLI は、表名の修飾子としてカタログ 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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

なし。

例

```
/* From the CLI sample TBINFO.C */
/* ... */
    /* call SQLColumns */
    printf("¥n    Call SQLColumns for:¥n");
    printf("        tbSchemaPattern = %s¥n", tbSchemaPattern);
    printf("        tbNamePattern = %s¥n", tbNamePattern);
    printf("        colNamePattern = %s¥n", colNamePattern);
    sqlrc = SQLColumns( hstmt, NULL, 0,
                        tbSchemaPattern, SQL_NTS,
                        tbNamePattern, SQL_NTS,
                        colNamePattern, SQL_NTS);
```

参照

- 800ページの『SQLTables - 表情報の入手』
- 343ページの『SQLColumnPrivileges - 表の列に関連した特権を入手する』
- 782ページの『SQLSpecialColumns - 特殊な (行識別子) 列の入手』

SQLConnect - データ・ソースに接続する

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、およびオプションとして許可名と認証ストリングを与える必要があります。

この関数を呼び出す前に、SQLAllocHandle() を使用して接続ハンドルを割り振る必要があります。

この関数は、SQLAllocHandle() を使用してステートメント・ハンドルを割り振る前に、呼び出さなければなりません。

構文

```
SQLRETURN   SQLConnect       (
                SQLHDBC           ConnectionHandle,  /* hdbc */
                SQLCHAR       *FAR ServerName,        /* szDSN */
                SQLSMALLINT       NameLength1,        /* cbDSN */
                SQLCHAR       *FAR UserName,          /* szUID */
                SQLSMALLINT       NameLength2,        /* cbUID */
                SQLCHAR       *FAR Authentication,    /* szAuthStr */
                SQLSMALLINT       NameLength3);       /* cbAuthStr */
```

関数引き数

表 40. 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	*NameLength1*	入力	*ServerName* 引き数の内容の長さ。
SQLCHAR *	*UserName*	入力	許可名 (ユーザー識別子)
SQLSMALLINT	*NameLength2*	入力	*UserName* 引き数の内容の長さ。
SQLCHAR *	*Authentication*	入力	認証ストリング (パスワード)
SQLSMALLINT	*NameLength3*	入力	*Authentication* 引き数の内容の長さ。

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。

SQLDataSources() がこの情報を返す前に、データベースをカタログ作成する必要があります。 Windows では、ODBC ドライバー・マネージャーを使用して、ユーザーがデータベースのカタログ作成を 2 回行わなければなりません。

1. IBM RDBMS に対して 1 回
2. ODBC に対して 1 回

これは、DB2 クライアント・アプリケーション・イネーブラー製品に組み込まれている DB2 クライアント・セットアップを使用することによって、 1 回の操作で終えることができます。カタログ作成の方法は ODBC ドライバー・マネージャーと IBM DBMS とで違いますが、 DB2 CLI アプリケーションはこの必要がありません。 (コール・レベル・インターフェースの利点の 1 つは、実行時に SQLConnect() を呼び出すまでアプリケーションがターゲット・データベースについて認識する必要がないという点です。) 実際の DBMS へのデータ・ソース名のマッピングは、 CLI アプリケーションの有効範囲と責任の範囲外です。

ODBC ドライバー・マネージャーを使用しない環境で DB2 CLI を使用する場合には、 IBM DBMS は 1 回カタログ化するだけで済みます。カタログ化の詳細については、155ページの『第4章 CLI の構成およびサンプル・アプリケーションの実行』を参照してください。

SQLConnect() の入力長さ引き数 (*NameLength1*、 *NameLength2*、*NameLength3*) は、その関連データの実際の長さ (ヌル終了文字を含まない) 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になることを指定することができます。

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。

アプリケーションが以下のことをする必要がある場合には、より拡張性の高い SQLDriverConnect() 関数を接続の際に使用します。

- ユーザーに、接続時のデータ・ソース名、ユーザー ID、およびパスワードの引き数以外についても指定するように要求する場合。
- 図形ダイアログ・ボックスを表示して、接続情報を入力要求する。

種々の接続特性 (オプション) を、 ServerName データ・ソース引き数に関連付けて db2cli.ini (および odbc.ini) 初期化ファイルのセクション内でエンド・

ユーザーが指定するか、または SQLSetConnectAttr() を使用するアプリケーションを用いて設定することができます。拡張された接続関数 SQLDriverConnect() も、追加の接続オプションを指定して呼び出すことができます。

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 の SQLConnect() とは、*ServerName*、 *UserName*、および *Authentication* 引き数ポインターがすべて 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() では最初に SQLAllocEnv() と SQLAllocConnect() を呼び出さなければなりませんが、必ずしも SQLDisconnect() の前に SQLTransact() を呼び出す必要はありません。詳細については、139ページの『ストアード・プロシージャーの例』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 41. 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。
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。 MAXCONN キーワードによって指定された接続数に達しました。
28000	許可指定が無効です。	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。

表 41. SQLConnect SQLSTATE (続き)

SQLSTATE	説明	解説
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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY090	ストリングまたはバッファー長が無効です。	引き数 *NameLength1* 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 *NameLength2* 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 *NameLength3* 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。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

IBM RDBMS の暗黙接続 (または省略時データベース) 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

```
/* From the CLI sample utilcli.c */
/* ... */
    /* connect to the database */
    printf( "¥nConnecting to %s ...¥n", dbAlias ) ;
    sqlrc = SQLConnect( *pHdbc,
                        (SQLCHAR *)dbAlias,   SQL_NTS,
                        (SQLCHAR *)user, SQL_NTS,
```

```
                    (SQLCHAR *)pswd, SQL_NTS
                  ) ;
    HANDLE_CHECK( SQL_HANDLE_DBC, *pHdbc, sqlrc, pHenv, pHdbc ) ;
    printf( "Connected to %s.¥n", dbAlias ) ;

    return( 0 ) ;
}
```

参照

- 246ページの『SQLAllocHandle - ハンドルを割り振る』
- 382ページの『SQLDriverConnect - データ・ソースに (拡張) 接続する』
- 663ページの『SQLSetConnectAttr - 接続属性を設定する』
- 480ページの『SQLGetConnectAttr - 現行属性設定を入手する』
- 246ページの『SQLAllocHandle - ハンドルを割り振る』
- 365ページの『SQLDataSources - データ・ソースのリストを入手する』
- 379ページの『SQLDisconnect - データ・サーバーからの切断』

SQLCopyDesc - ハンドル間で記述子情報をコピーする

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,
                              SQLHDESC          TargetDescHandle);
```

関数引き数

表 42. 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
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 (実装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実装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。コピーされたフィールドは、既存のフィールドを上書きします。

すべての記述子フィールドは、 *SourceDescHandle* と *TargetDescHandle* が 2 つの異なる接続または環境にある場合でも、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、整合性検査が行われます。整合性検査に失敗した場合、SQLSTATE HY021 (記述子情報が不整合です。) が返されて、 SQLCopyDesc() の呼び出しは直ちに打ち切られます。詳細については、『SQLSetDescField()』の中の『整合性検査』を参照してください。

注: 記述子ハンドルは、接続または環境を越えてコピーできます。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することにより、同じ *ConnectionHandle* 上の別の *StatementHandle* に関連付けることができます。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 *ConnectionHandles* 上の *StatementHandles* に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。

記述子ヘッダーまたはレコードのフィールドの説明については、 696ページの『SQLSetDescField - 記述子レコードの単一フィールドを設定する』を参照してください。記述子の詳細については、105ページの『記述子の使用』を参照してください。

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 を返すときは、 SQL_HANDLE_DESC の HandleType および *TargetDescHandle* の Handle を用いて SQLGetDiagRec() を呼び出すことにより、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 43. SQLCopyDesc SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
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	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY007	関連したステートメントが準備されていません。	*SourceDescHandle* が IRD と関連付けられましたが、関連したステートメント・ハンドルは、準備または実行状態にありませんでした。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。

表 43. SQLCopyDesc SQLSTATE (続き)

SQLSTATE	説明	解説
HY016	インプリメンテーションの行記述子を変更することはできません。	*TargetDescHandle* が IRD に関連付けられました。
HY021	記述子情報が不整合です。	整合性検査時に検査された記述子情報は、整合性がとれていませんでした。詳細については、SQLSetDescField() の中の 724ページの『整合性検査』を参照してください。
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。

制約

なし。

例

該当するサンプルの一覧については、 sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内の README ファイルを参照してください。

参照

なし。

SQLDataSources - データ・ソースのリストを入手する

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。カタログ化の詳細については、155ページの『第4章 CLI の構成およびサンプル・アプリケーションの実行』を参照してください。

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。

構文

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

関数引き数

表 44. 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	取り出したデータ・ソース名を入れておくバッファーを指すポインター。

表 44. 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* で指示されるバッファーの最大長。これは、SQL_MAX_DSN_LENGTH + 1 以下でなければなりません。
SQLSMALLINT *	*NameLength1Ptr*	出力	*ServerName* に返すのに使用できる最大バイト数を保管する場所を指すポインター。
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* バッファーの最大長。
SQLSMALLINT *	*NameLength2Ptr*	出力	関数が、データ・ソースの記述を返すのに使用できる実際のバイト数を返す場所を指すポインター。

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 ドライバー・マネージャーがこの関数を実行します。詳細については、835ページの『付録C. DB2 CLI と ODBC』を参照してください。

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 45. 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 を戻します。) 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 を戻します。)
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	一般的なエラーです。	特定の SQLSTATE が存在しなかったり、定義されなかった場合のエラーが発生しました。引き数 *ErrorMsg* 内の `SQLError()` から返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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。

表 45. SQLDataSources SQLSTATE (続き)

SQLSTATE	説明	解説
HY103	方向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

```
/* From the CLI sample ININFO.C */
/* ... */
    sqlrc = SQLDataSources( henv,
                            SQL_FETCH_NEXT,
                            dbAliasBuf,
                            SQL_MAX_DSN_LENGTH + 1,
                            &aliasLen,
                            dbCommentBuf,
                            255,
                            &commentLen );
```

参照

なし。

SQLDescribeCol - 列の属性のセットを返す

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の指定された列に関する共通に使用される記述子情報 (列名、タイプ、精度、位取り、ヌル可) 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。詳細については、328ページの『SQLColAttribute - 列属性を返す』を参照してください。

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。

構文

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

関数引き数

表 46. 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

表 46. SQLDescribeCol 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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、これをヌル (null) に設定してください。 列名値は環境属性 SQL_ATTR_USE_LIGHT_OUTPUT_SQLDA の影響を受ける場合があります。詳細については、 734ページの『SQLSetEnvAttr - 環境属性を設定する』を参照してください。
SQLSMALLINT	*BufferLength*	入力	*ColumnName* バッファーのサイズ。
SQLSMALLINT *	*NameLengthPtr*	出力	*ColumnName* 引き数について返すために使用できるバイト。 *NameLengthPtr* が *BufferLength* 以上である場合、列名 (*ColumnName*) が *BufferLength - 1* バイトに切り捨てられます。

表 46. SQLDescribeCol 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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、 33ページの表2 の「記号 SQL データ・タイプ」列を参照してください。
SQLUINTEGER *	*ColumnSizePtr*	出力	データベースで定義されている列の精度。 *fSqlType* が図形または 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位取り (SQL_DECIMAL、SQL_NUMERIC、SQL_TIMESTAMP だけに適用される)。各 SQL データ・タイプの位取りについては、877ページの表195 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は、列番号は 1 から始まります (SQL_ATTR_USE_BOOKMARKS ステートメント属性を SQL_UB_OFF に設定します)。
- ブックマークを使用している場合は、列番号は 0 から始まります (ステートメント属性を SQL_UB_ON に設定します)。

ポインター引き数についてヌル・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、何も返されません。

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 L_SUCCESS_WITH_INFO を返した場合、 SQLError() 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 47. 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* 内に返された列名が、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、完全列名の長さが含まれています。 (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。)

表 47. SQLDescribeCol SQLSTATE (続き)

SQLSTATE	説明	解説
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、0 より小さい値でした。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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY002	列の番号が無効です。	引き数 *ColumnNumber* に指定された値は、1 より小さい値でした。 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。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。
HY090	ストリングまたはバッファー長が無効です。	1 より小さい、引き数 *BufferLength* で指定された長さ。
HY010	関数の順序エラーです。	*StatementHandle* の 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 SQL の操作中に、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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HYC00	ドライバーが機能していません。	列 *ColumnNumber* の SQL データ・タイプは、DB2 CLI では認識されません。

表 47. SQLDescribeCol 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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

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

```
/* From the CLI sample utilcli.c */
/* ... */
        sqlrc = SQLDescribeCol( hstmt,
                        ( SQLSMALLINT ) ( i + 1 ),
                        colName,
                        sizeof(colName),
                        &colNameLen,
                        &colType,
                        &colSize,
                        &colScale,
                        NULL ) ;
```

参照

- 662ページの『SQLSetColAttributes - 列属性を設定する』
- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 619ページの『SQLNumResultCols - 結果列の数の入手』
- 630ページの『SQLPrepare - ステートメントを準備する』

SQLDescribeParam - パラメーター・マーカーの記述を返す

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。

構文

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

関数引き数

表 48. 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 が変数長ブックマークの **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ように、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ため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ように、対応するパラメーターの列または式の 10 進数での数を返すためのバッファーを指すポインターです。

表 48. SQLDescribeParam 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	NullablePtr	出力	パラメーターがヌル (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。 以下のどれかになります。 • SQL_NO_NULLS: ヌル (NULL) は許可しません (これが省略時値です)。 • SQL_NULLABLE: ヌル (NULL) を許可します。 • SQL_NULLABLE_UNKNOWN: ヌル (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・マーカーは、SQL ステートメントに表示される順番に、小さい順にパラメーター番号が付けられます。この番号は 1 から始まっています。

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。プロシージャー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。プロシージャーの呼び出しでのパラメーターのタイプをそれぞれ判別するために、アプリケーションは 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 49. SQLDescribeParam SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)

表 49. SQLDescribeParam SQLSTATE (続き)

SQLSTATE	説明	解説
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と一致しません。	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	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY008	操作が取り消されました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、 SQLCancel() が複数スレッドの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。
HY010	関数の順序エラーです。	*StatementHandle* の SQLPrepare() または SQLExecDirect() を呼び出す前に、この関数を呼び出しました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。 SQLExecute() SQLExecDirect()、SQLBulkOperations()、または SQLSetPos() が *StatementHandle* で呼び出され、SQL_NEED_DATA を戻しました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、この関数が呼び出されました。

表 49. SQLDescribeParam SQLSTATE (続き)

SQLSTATE	説明	解説
HY013	予期しないメモリーのハンドル・エラーが起きました。	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。

制約

なし。

例

該当するサンプルの一覧については、 sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内の README ファイルを参照してください。

参照

- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』
- 321ページの『SQLCancel - ステートメントを取り消す』
- 409ページの『SQLExecute - ステートメントの実行』
- 630ページの『SQLPrepare - ステートメントを準備する』

SQLDisconnect - データ・サーバーからの切断

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、 SQLFreeHandle() を呼び出します。

構文

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

関数引き数

表 50. 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が正常に行われた場合でも、追加のエラーや処理系特定の情報を使用できることを暗黙指定します。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事象 (通信障害など) のために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、 *HandleType* を SQL_HANDLE_STMT に指定して SQLFreeHandle() を実行することにより、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 51. 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活動状態であり、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活動状態であり、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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数の順序エラーです。	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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。

制約

なし。

例

```
/* From the CLI sample utilcli.c */
/* ... */
        printf( "¥nDisconnecting from the database nb. %d ...¥n", db_nb + 1) ;

        sqlrc = SQLDisconnect( a_hdbc[db_nb] ) ;
        rc    = HandleInfoPrint( SQL_HANDLE_DBC, a_hdbc[db_nb],
                                 sqlrc, __LINE__, __FILE__);
        if( rc == 0)
        {   printf( "Disconnected from the database nb. %d.¥n", db_nb + 1 ) ;
        }
```

参照

- 246ページの『SQLAllocHandle - ハンドルを割り振る』
- 356ページの『SQLConnect - データ・ソースに接続する』
- 382ページの『SQLDriverConnect - データ・ソースに (拡張) 接続する』
- 388ページの『SQLEndTran - 接続のトランザクションの終了』
- 470ページの『SQLFreeHandle - ハンドル資源を解放する』

SQLDriverConnect - データ・ソースに (拡張) 接続する

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。

構文

総称

```
SQLRETURN   SQLDriverConnect (
                SQLHDBC           ConnectionHandle,  /* hdbc */
                SQLHWND           WindowHandle,      /* hwnd */
                SQLCHAR      *FAR InConnectionString, /* szConnStrIn */
                SQLSMALLINT       StringLength1,     /* cbConnStrIn */
                SQLCHAR      *FAR OutConnectionString,/* szConnStrOut */
                SQLSMALLINT       BufferLength,      /* cbConnStrOutMax */
                SQLSMALLINT  *FAR StringLength2Ptr,   /* pcbConnStrOut */
                SQLUSMALLINT      DriverCompletion); /* fDriverCompletion */
```

関数引き数

表 52. 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

表 52. *SQLDriverConnect 引き数 (続き)*

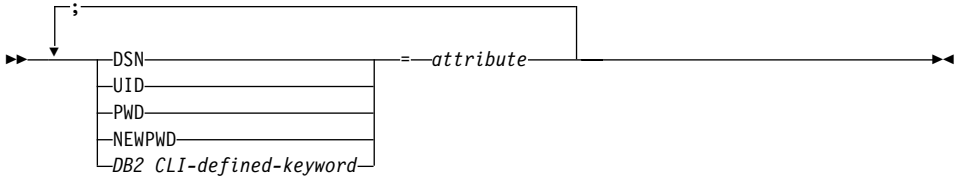
データ・タイプ	引き数	使用法	説明
SQLHWND	*hwindow*	入力	ウィンドウ・ハンドル (プラットフォーム従属): Windows では、これは親 Windows ハンドルです。 OS/2 では、これは親 PM ウィンドウ・ハンドルです。 AIX では、これは親 MOTIF ウィジェット・ウィンドウ・ハンドルです。 NULL が渡されると、ダイアログは表示されません。
SQLCHAR *	*InConnectionString*	入力	完全、一部、または空 (ヌル・ポインター) の接続ストリング (次の構文と説明を参照)。
SQLSMALLINT	*StringLength1*	入力	*InConnectionString* の長さ。
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	*BufferLength*	入力	*OutConnectionString* で指定されたバッファーの最大サイズ。
SQLCHAR *	*StringLength2Ptr*	出力	*OutConnectionString* バッファーに返すために使用できるバイト数を指すポインター。 *StringLength2Ptr* の値が *BufferLength* 以上である場合、*OutConnectionString* 内の完全接続ストリングは *BufferLength* - 1 バイトに切り捨てられます。

表 52. SQLDriverConnect 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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。 有効値は次のとおりです。 • SQL_DRIVER_PROMPT • SQL_DRIVER_COMPLETE • SQL_DRIVER_COMPLETE _REQUIRED • SQL_DRIVER_NOPROMPT

使用法

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。



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 (ユーザー識別子)。

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、使用する新規のストリングを指定することもできますし (NEWPWD=anewpass など)、 NEWPWD=; と指定することにより、DB2 CLI ドライバーにダイアログ・ボックスを生成させて、新規パスワードを求めるプロンプトを表示することもできます (*DriverCompletion* 引き数には SQL_DRIVER_NOPROMPT 以外の値を指定)。

DB2 CLI 定義のキーワードおよびそれに関連する属性値については、 174ページの『構成キーワード』で論じられます。そのセクションで扱われるキーワードはいずれも、接続ストリングに指定されるものです。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、ダイアログがオープンされる時点を判別します。

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・ボックスでデータ入力して補足できます。

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・ボックスでデータ入力して補足できます。

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設定する必要がない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ようにします。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

356ページの『SQLConnect - データ・ソースに接続する』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。次の表は、返すことのできる追加の診断を示したものです。

表 53. 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 *StringLength2Ptr* には、戻りに使用できる接続ストリングの実際の長さが含まれています。 (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、次のうちの 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、その代わりに省略時値が使用されました。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
HY000	一般的なエラーです。 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長が無効です。	*StringLength1* 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。 引き数 *BufferLength* 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。

制約

なし。

例

```
/* From the CLI sample DBCONN.C */
/* ... */
    sqlrc = SQLDriverConnect(hdbc,
                             (SQLHWND) NULL,
                             connStr,
                             SQL_NTS,
                             NULL, 0, NULL,
                             SQL_DRIVER_NOPROMPT);
```

参照

- 246ページの『SQLAllocHandle - ハンドルを割り振る』
- 356ページの『SQLConnect - データ・ソースに接続する』

SQLEndTran - 接続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、接続に関連するステートメントすべてにおける活動中の操作すべてについて、コミットとロールバック操作を要求します。 SQLEndTran() は、環境に関連している接続すべてに対してコミットおよびロールバック操作を実行するよう要求することもできます。

構文

```
SQLRETURN   SQLEndTran      (SQLSMALLINT      HandleType,
                             SQLHANDLE        Handle,
                             SQLSMALLINT      CompletionType);
```

関数引き数

表 54. SQLEndTran 引き数

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*HandleType*	入力	*Handle* タイプ識別子。 *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効力範囲を示します。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ステートメントすべてにおいて活動中の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ステートメントすべてにおいて活動中の操作すべてに対してロールバック要求を出します。活動状態に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して SQL_SUCCESS を返します。

DB2 CLI が手動コミット・モードにある場合 (SQLSetConnectAttr() を呼び出し、 SQL_ATTR_AUTOCOMMIT 属性を SQL_AUTOCOMMIT_OFF に設定するとこのモードになる)、トランザクションに入れることのできる SQL ステートメントを現行データ・ソースに対して実行するときに、新しいトランザクションが暗黙的に開始されます。

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、保留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。これらをアプリケーションは次の 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、準備済み

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、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・モードがオンになっているときに、トランザクションが 1 つも活動中でない状態で SQLEndTran() に SQL_COMMIT または SQL_ROLLBACK を指定して呼び出すと、 SQL_SUCCESS が返され (コミット可能な、またはロールバック可能な作業がないことを示す)、データ・ソースには何も影響を与えません。

自動コミット・モードがオフになっているときに、 SQLEndTran() に SQL_COMMIT か SQL_ROLLBACK のどちらかの *CompletionType* を指定して呼び出すと、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すことはしません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 55. SQLEndTran SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
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。

表 55. SQLEndTran SQLSTATE (続き)

SQLSTATE	説明	解説
40001	トランザクション・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資源デッドロックのためにロールバックされました。
HY000	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY010	関数の順序エラーです。	非同期的に実行する関数が *ConnectionHandle* に関連している *StatementHandle* のために呼び出され、 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* の SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が戻されました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、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。

制約

なし。

例

```
/* From the CLI sample utilcli.c */
/* ... */
        printf( "¥nRolling back the transaction nb. %d ...¥n", db_nb + 1) ;

        sqlrc = SQLEndTran( SQL_HANDLE_DBC, a_hdbc[db_nb], SQL_ROLLBACK );
        rc    = HandleInfoPrint( SQL_HANDLE_DBC, a_hdbc[db_nb],
                                 sqlrc, __LINE__, __FILE__);
        if( rc == 0)
        {   printf( "The transaction nb. %d rolled back.¥n", db_nb + 1 ) ;
        }
```

参照

- 533ページの『SQLGetInfo - 一般情報の入手』
- 470ページの『SQLFreeHandle - ハンドル資源を解放する』
- 475ページの『SQLFreeStmt - ステートメント・ハンドルの解放 (またはリセット)』

SQLError - エラー情報を取り出す

使用すべきでない関数

注:

ODBC バージョン 3 では、SQLError() 使用すべきではなく、代わりに SQLGetDiagRec() および SQLGetDiagField() に置き換えられています。詳細については、522ページの『SQLGetDiagRec - 診断レコードの複数のフィールド設定を取得する』と 512ページの『SQLGetDiagField - 診断データのフィールドの入手』を参照してください。

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、 SQLGetDiagRec() 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。 上記の関数と、その他の使用すべきでない関数の詳細については、822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

目的

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLError() は、特定のステートメント、接続、または環境ハンドルについて呼び出された最新の DB2 CLI 関数に関連した診断情報 (エラーおよび警告) を返します。

この情報は、標準化された SQLSTATE、ネイティブのエラー・コード、テキスト・メッセージから構成されています。詳細については、29ページの『診断』を参照してください。

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

注: データベース・サーバーの中には、ステートメントを実行した結果として SQL_NO_DATA_FOUND を返した後に製品特定の診断情報を表示するものがあります。

構文

```
SQLRETURN   SQLError         (SQLHENV           henv,
                              SQLHDBC           hdbc,
                              SQLHSTMT          hstmt,
                              SQLCHAR      FAR  *szSqlState,
```

```
SQLINTEGER   FAR  *pfNativeError,
SQLCHAR      FAR  *szErrorMsg,
SQLSMALLINT       cbErrorMsgMax,
SQLSMALLINT  FAR  *pcbErrorMsg);
```

関数引き数

表 56. SQLError 引き数

データ・タイプ	引き数	使用法	説明
SQLHENV	*henv*	入力	環境ハンドル。環境に関連した診断情報を取得するには、有効な環境ハンドルを渡します。 *hdbc* を SQL_NULL_HDBC に設定し、*hstmt* を SQL_NULL_HSTMT に設定してください。
SQLHDBC	*hdbc*	入力	データベース接続ハンドル。接続に関連した診断情報を取得するには、有効なデータベース接続ハンドルを渡し、 *hstmt* を SQL_NULL_HSTMT に設定します。*henv* 引き数は無視されます。
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル。ステートメントに関連した診断情報を取得するには、有効なステートメント・ハンドルを渡します。 *henv* 引き数と *hdbc* 引き数は無視されます。
SQLCHAR *	*szSqlState*	出力	ヌル文字で終了している 5 文字のストリングの SQLSTATE。最初の 2 文字はエラー・クラスを指示し、次の 3 文字はサブクラスを指示します。これらの値は、IBM 特定と製品特定の SQLSTATE 値が追加された、 X/Open SQL CAE 仕様と ODBC 仕様で定義されている SQLSTATE 値にそのまま対応しています。
SQLINTEGER *	*pfNativeError*	出力	ネイティブのエラー・コード。 DB2 CLI では、 *pfNativeError* 引き数には DBMS から返された SQLCODE 値が含まれています。エラーが DBMS でなく DB2 CLI で生成された場合、このフィールドは -99999 に設定されます。

表 56. SQLError 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLCHAR *	*szErrorMsg*	出力	処理系定義メッセージ・テキストを入れるバッファーを指すポインター。エラーが DB2 CLI で検出されると、エラー・メッセージの前に次のものが付けられます。 [IBM][CLI Driver] これは、エラーを検出したのが DB2 CLI であり、まだデータベース接続がないことを示します。 エラーが検出されると、DBMS から返されたエラー・メッセージの前に次のものが付けられます。 [IBM][CLI Driver][*DBMS-name*] DBMS-name は、SQLGetInfo() に SQL_DBMS_NAME 情報タイプを指定して返される名前です。 たとえば、以下のようになります。 • DB2 • DB2/6000 DBMS 名が認識されない場合、DB2 CLI はこれを DB2 ユニバーサル・データベース バージョン 5 データ・ソースとして扱います。 DBMS でエラーが生成されると、IBM 定義の SQLSTATE がテキスト・ストリングに付加されます。
SQLSMALLINT	*cbErrorMsgMax*	入力	バッファー *szErrorMsg* の最大 (つまり、割り振られた) 長さ。SQL_MAX_MESSAGE_LENGTH + 1 の長さを割り振ることをお勧めします。

表 56. SQLError 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT *	*pcbErrorMsg*	出力	*szErrorMsg* バッファーに返すために使用できる総バイト数を指すポインター。これには、ヌル終了文字が含まれていません。

使用法

SQLSTATE は、IBM 特定と製品特定の SQLSTATE 値が追加された、 X/Open SQL CAE と X/Open SQL CLI CAE で定義されたものです。

以下のものに関連した診断情報を取得する場合は、以下のようにします。

- 環境の場合は、有効な環境ハンドルを渡します。 *hdbc* を SQL_NULL_HDBC に設定し、 *hstmt* を SQL_NULL_HSTMT に設定してください。
- 接続の場合は、有効なデータベース接続ハンドルを渡し、 *hstmt* を SQL_NULL_HSTMT に設定します。 *henv* 引き数は無視されます。
- ステートメントの場合は、有効なステートメント・ハンドルを渡してください。 *henv* 引き数と *hdbc* 引き数は無視されます。

ある DB2 CLI 関数で生成された診断情報が、同じハンドルを指定して SQLError() 以外の関数を呼び出す前に取り出されない場合、直前の関数呼び出しの情報が失われます。このことは、 2 番目の DB2 CLI 関数呼び出しに関する診断情報が生成されるかどうかにかかわらず当てはまります。

指定した DB2 CLI 関数呼び出しの後に、複数の診断メッセージが使用できる場合があります。 SQLError() を繰り返し呼び出すと、これらのメッセージを一度に 1 つずつ取り出すことができます。取り出されたメッセージごとに、SQLError() は SQL_SUCCESS を返し、利用可能なメッセージのリストからそれを除去します。取り出すメッセージがそれ以上ないと、SQL_NO_DATA_FOUND が返され、 SQLSTATE は「00000」に設定され、 *pfNativeError* は 0 に設定され、 *pcbErrorMsg* と *szErrorMsg* は定義されません。

指定ハンドルに保管されている診断情報は、そのハンドルを指定して SQLError() 呼び出しが行われる場合、またはそのハンドルを指定して別の DB2 CLI 関数の呼び出しが行われる場合に、クリアされます。しかし、指定ハンドル・タイプに関連した情報は、関連しているが同じではないハンドル・タイプを指定した SQLError() 呼び出しではクリアされません。たとえば、接続

ハンドルを指定して SQLError() を呼び出しても、その接続のステートメント・ハンドルに関連したエラーはクリアされません。

アプリケーションが SQLError() を再度呼び出して同じエラー・メッセージを取り出すことはないので、エラー・メッセージのバッファー (*szErrorMsg*) が短すぎる場合でも、 SQL_SUCCESS は返されます。メッセージ・テキストの実際の長さは、*pcbErrorMsg* 内に返されます。

エラー・メッセージが切り捨てられないようにするには、バッファー長として SQL_MAX_MESSAGE_LENGTH + 1 を宣言します。メッセージ・テキストがこれ以上長くなることは決してありません。

注: SQL_MAX_MESSAGE_LENGTH に定義された値は、DB2 CLI バージョン 1 以降増えています。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

SQL_NO_DATA_FOUND は、入力ハンドルに関する診断情報を使用できない場合、または SQLError() 呼び出しによってすべてのメッセージが取り出されている場合に返されます。

診断

SQLError() は独自の診断情報を生成しないので、SQLSTATE は定義されません。

制約

ODBC は X/Open SQL CAE SQLSTATE も返しますが、 DB2 CLI (および DB2 ODBC ドライバー) は IBM 定義の追加 SQLSTATE しか返しません。ODBC ドライバー・マネージャーは、 **IM** という接頭部の付いた SQLSTATE 値も返します。これらの SQLSTATE は X/Open では定義されておらず、DB2 CLI からは返されません。 ODBC 特定の SQLSTATE については、 *ODBC 3.0 Software Development Kit and Programmer's Reference* を参照してください。

このため、標準 SQLSTATE に依存性を構築するだけで済みます。このことは、アプリケーション内の分岐論理が標準 SQLSTATE だけに基づいている必要があることを意味します。増補された SQLSTATE は、デバッグの場合に最も有効です。

注: SQLSTATE のクラス (先頭 2 文字) に関する依存性を作成すると効果的な場合があります。

例

522ページの『SQLGetDiagRec - 診断レコードの複数のフィールド設定を取得する』を参照してください。

参照

- 587ページの『SQLGetSQLCA - SQLCA データ構造を入手する』

SQLExecDirect - 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

SQLExecDirect() は、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 回だけ実行されます。また、接続しているデータベース・サーバーが、ステートメントを動的に作成できなければなりません。 (サポートされる SQL ステートメントの詳細については、 901ページの表215 を参照してください。)

構文

```
SQLRETURN   SQLExecDirect    (SQLHSTMT          StatementHandle,
                              SQLCHAR      *FAR StatementText,
                              SQLINTEGER         TextLength);
```

関数引き数

表 57. 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。詳細については、475ページの『SQLFreeStmt - ステートメント・ハンドルの解放 (またはリセット)』を参照してください。
SQLCHAR *	*StatementText*	入力	SQL ステートメント・ストリング。接続しているデータベース・サーバーが、ステートメントを作成できなければなりません。詳細については、901ページの表215 を参照してください。
SQLINTEGER	*TextLength*	入力	*StatementText* 引き数の内容の長さ。この長さは、ステートメントの正確な長さに設定するか、ステートメントがヌル終了である場合は SQL_NTS に設定する必要があります。

使用法

SQL ステートメント・テキストにベンダー・エスケープ文節順序列が含まれている場合、 DB2 CLI はまず SQL ステートメント・テキストを適切な DB2 固有の形式に修正してから、それを用いて作成および実行を依頼します。アプリケーションがベンダー・エスケープ文節順序列 (151ページの『ベンダー・エスケープ文節の使用』) を含む 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 を設定することによって、 DB2 CLI がベンダー・エスケープ文節を探索して走査することがないようにすることができます。

SQL ステートメントを COMMIT または ROLLBACK とすることはできません。その代わり、 SQLTransact() を呼び出して COMMIT または ROLLBACK を出してください。サポートされる SQL ステートメントの詳細については、901ページの表215を参照してください。

SQL ステートメント・ストリングの中に、パラメーター・マーカーが含まれている場合があります。パラメーター・マーカーは "?" 文字で表され、また、SQLExecDirect() を呼び出すときにアプリケーションに提供された値が置き換えられる、ステートメント内の位置を指示するのに使用されます。この値は、以下のものから得られます。

- アプリケーション変数。

  SQLSetParam() または SQLBindParameter() は、アプリケーション記憶域をパラメーター・マーカーにバインドするのに使用されます。

- LOB ロケーターが参照するサーバーにある LOB 値。

  SQLBindParameter() または SQLSetParam() は、 LOB ロケーターをパラメーター・マーカーにバインドするのに使用されます。 LOB の実際の値はサーバーに保持されるので、別の SQL ステートメントの入力パラメーター値として使用されるまで、アプリケーションに転送する必要はありません。

- LOB 値を含むファイル (アプリケーション環境内)。

  LOB パラメーター・マーカーにファイルをバインドするには、SQLBindFileToParam() を使用します。 SQLExecDirect() を実行すると、DB2 CLI はファイルの内容をデータベース・サーバーに直接転送します。

SQLExecDirect() を呼び出す前に、すべてのパラメーターをバインドして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については、 *SQL 解説書* の PREPARE のセクションを参照してください。

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。

結果セットを生成すると、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に、その次のデータ行 (複数行の場合もある) を取り出します (SQLBindCol() または SQLBindFileToCol() を使用します)。また、バインドされなかった列についても、 SQLGetData() を呼び出すとデータを取り出すことができます。

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

ステートメント・ハンドル上にオープン・カーソルがあってはなりません。

SQLParamOptions() 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るように指定した場合、アプリケーションは SQLExecDirect() を一度だけ呼び出して、入力パラメーター値の配列全体を処理します。

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 SQLParamData() と SQLPutData() を呼び出して、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 58. 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	ステートメントはブロック化できませんでした。	このステートメントは、記憶域以外の理由でブロック化できませんでした。
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	変換が無効です。	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と一致しません。	*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程度と一致していませんでした。
21S02	派生表の程度が列リストと一致しません。	*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程度になっていません。
22001	ストリング・データの右側が切り捨てられました。	文字タイプ列に割り当てられた文字ストリングが、列の最大長を超えました。
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 ユニバーサル・データベース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)。

表 58. SQLExecDirect SQLSTATE (続き)

SQLSTATE	説明	解説
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 SQL 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 図形データ・タイプを指示しましたが、据え置き長さ引き数 (*pcbValue*) には奇数の長さの値が含まれています。図形データ・タイプの場合、長さの値は偶数でなければなりません。
22007	日時形式が無効です。	*StatementText* には日時形式が無効な SQL ステートメントが含まれていました。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。	日時フィールドがオーバーフローしました。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にないか、または日時の値が小さ過ぎて、結合変数に割り当てることができません。
22012	ゼロによる割算は無効です。	*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* で保留状態になったか、 *hsmt* 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、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 58. 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 つ以上が含まれていました。 • COMMIT • ROLLBACK • 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 • 構文エラーを含むステートメント
40001	トランザクション・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。

表 58. SQLExecDirect SQLSTATE (続き)

SQLSTATE	説明	解説
428A1	ホスト・ファイル変数で参照される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いずれか、または両方の形式が無効です。 • 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、予期しない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、媒体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値は、そのデータ・タイプのゆえに使用できません。	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 58. SQLExecDirect SQLSTATE (続き)

SQLSTATE	説明	解説
44000	保全性制約違反。	*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、固有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。	DRDA サーバーに接続 (DB2 コネクトを使用して行う) しているときには、 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存在しています。	*StatementText* 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視点名はすでに存在しています。
S0002	データベース・オブジェクトは存在していません。	*StatementText* に、存在しない表名または視点名を参照する SQL ステートメントが含まれていました。
S0011	索引はすでに存在しています。	*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	索引がありません。	*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	列はすでに存在しています。	*StatementText* に ALTER TABLE ステートメントがあり、ADD 文節に指定されている列は固有になっていなかったか、基本表の既存の列を識別できませんでした。
S0022	列がありません。	*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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効です。	*StatementText* は、ヌル・ポインターでした。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY014	もはやハンドルはありません。	DB2 CLI は、内部資源が原因でハンドルを割り当てることができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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。

表 58. SQLExecDirect SQLSTATE (続き)

SQLSTATE	説明	解説
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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、 SQLSetConnectAttr() 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。

制約

なし。

例

```
/* From the CLI sample TBREAD.C */
/* ... */
    /* execute directly the statement */
    printf("¥n    Execute directly the statement.¥n");
    printf("        %s¥n", stmt);
    sqlrc = SQLExecDirect( hstmt, stmt, SQL_NTS ) ;
    STMT_HANDLE_CHECK( hstmt, sqlrc);

/* ... */
    /* execute directly the statement */
    printf("¥n    Execute directly the statement.¥n");
    printf("        %s¥n", stmt);
    sqlrc = SQLExecDirect( hstmt, stmt, SQL_NTS ) ;
    STMT_HANDLE_CHECK( hstmt, sqlrc);

/* ... */
    /* execute directly the statement */
    printf("¥n    Execute directly the statement¥n");
    printf("        %s.¥n", stmt);
    sqlrc = SQLExecDirect( hstmt, stmt, SQL_NTS ) ;
    STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参照

- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 267ページの『SQLBindFileToCol - LOB 列に LOB ファイル参照をバインドする』
- 273ページの『SQLBindFileToParam - LOB パラメーターに LOB ファイル参照をバインドする』
- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』
- 409ページの『SQLExecute - ステートメントの実行』
- 446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』
- 434ページの『SQLFetch - 次の行の取り出し』
- 622ページの『SQLParamData - データ値が必要な次のパラメーターを入手する』
- 655ページの『SQLPutData - パラメーターのデータ値を渡す』
- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』

SQLExecute - 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、 SQLPrepare() を使用して正常に作成されたステートメントを 1 回または数回実行します。ステートメントは、SQLBindParameter()、SQLSetParam() 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アプリケーション変数の現行値を使用して実行されます。

構文

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

関数引き数

表 59. 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。詳細については、475ページの『SQLFreeStmt - ステートメント・ハンドルの解放 (またはリセット)』を参照してください。

使用法

SQL ステートメント・ストリングの中に、パラメーター・マーカーが含まれている場合があります。パラメーター・マーカーは "?" 文字で表され、また、SQLExecute() を呼び出すときにアプリケーションに提供された値が置き換えられる、ステートメント内の位置を指示するのに使用されます。この値は、以下のものから得られます。

- アプリケーション変数。

  パラメーター・マーカーにアプリケーション記憶域をバインドするには、SQLSetParam() または SQLBindParameter() を使用します。

- LOB ロケーターが参照するサーバーにある LOB 値。 SQLBindParameter() または SQLSetParam() は、 LOB ロケーターをパラメーター・マーカーにバインドするのに使用されます。 LOB の実際の値はサーバーに保持されるので、別の SQL ステートメントの入力パラメーター値として使用されるまで、アプリケーションに転送する必要はありません。

- LOB 値を含むファイル (アプリケーション環境内)。
  LOB パラメーター・マーカーにファイルをバインドするには、SQLBindFileToParam() を使用します。 SQLExecDirect() を実行すると、DB2 CLI はファイルの内容をデータベース・サーバーに直接転送します。

SQLExecute() を呼び出す前に、すべてのパラメーターをバインドして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。したがって、SQLPrepare() を最初に呼び出す必要があります。

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。

照会を複数回実行するには、アプリケーションは SQL_CLOSE オプションを指定して SQLFreeStmt() を呼び出し、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。

結果セットを生成すると、 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にその次のデータ行 (複数行の場合もある) を取り出します。 (SQLBindCol() または SQLBindFileToCol を使用します。) また、バインドされなかった列についても、 SQLGetData() を呼び出すとデータを取り出すことができます。

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。

SQLParamOptions() 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るように指定した場合、アプリケーションは SQLExecDirect() を一度だけ呼び出して、入力パラメーター値の配列全体を処理します。実行されたステートメントが複数の結果セットを返す (入力パラメ

ーターの各集まりごとに 1 つ) 場合は、一度現在の結果セットでの処理を完了してから、 SQLMoreResults() を使用して次の結果セットに進みます。詳細については、610ページの『SQLMoreResults - さらに結果セットがあるかどうかを判別する』を参照してください。

戻りコード

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

アプリケーションが SQLParamData() と SQLPutData() を呼び出して、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 には、 SQLExecDirect() (402ページの表58 を参照) が含まれています。ただし、**HY**009 と **HY**090 は除き、以下の表に示された SQLSTATE を追加します。

表 60. SQLExecute SQLSTATE

SQLSTATE	説明	解説
HY010	関数の順序エラーです。	指定された *StatementHandle* が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

```
/* From the CLI sample TBREAD.C */
/* ... */
    /* execute the statement for divisionParam = Eastern */
    printf("    Execute the prepared statement for¥n");
    printf("        divisionParam = 'Eastern'¥n");
```

```
strcpy( divisionParam, "Eastern");
sqlrc = SQLExecute( hstmt ) ;
STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参照

- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 409ページの『SQLExecute - ステートメントの実行』
- 446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』
- 630ページの『SQLPrepare - ステートメントを準備する』
- 434ページの『SQLFetch - 次の行の取り出し』
- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』
- 626ページの『SQLParamOptions - パラメーターに入力配列を指定する』
- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』
- 273ページの『SQLBindFileToParam - LOB パラメーターに LOB ファイル参照をバインドする』
- 267ページの『SQLBindFileToCol - LOB 列に LOB ファイル参照をバインドする』
- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』

SQLExtendedBind - 列の配列のバインド

目的

仕様:	DB2 CLI 6		

SQLExtendedBind() は、 SQLBindCol() または SQLBindParameter() を繰り返し呼び出さずに、列の配列をバインドする場合に使用します。

構文

```
SQLRETURN    SQLExtendedBind (
                              SQLHSTMT          StatementHandle,
                              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 61. 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、この出力情報は 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。

表 61. SQLExtendedBind() 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLUINTEGER	puiPrecisionCType	入力	レコードの 10 進数の精度。使用されるのは、アプリケーション・データ・タイプが SQL_C_DECIMAL_IBM の場合だけです。
SQLSMALLINT	psScaleCType	入力	レコードの 10 進数の位取り。使用されるのは、アプリケーション・データ・タイプが SQL_C_DECIMAL_IBM の場合だけです。
SQLINTEGER	pcbValue	入力	長さの値を指すポインターの配列。
SQLINTEGER	piIndicator	入力	標識の値を指すポインターの配列。
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()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() 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() 引き数 *ColumnSize* と同じ目的を果たします。

表 61. SQLExtendedBind() 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	pibScale	入力	SQL 位取り値の配列。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() 引き数 *DecimalDigits* と同じ目的を果たします。

使用法

この関数は、 SQLBindCol() または SQLBindParameter() の複数回の呼び出しを置き換えるために使用します。

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

詳しくは (許可されている引き数値など)、 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』および 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』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 62. SQLExtendedBind() SQLSTATE

SQLSTATE	説明	解説
07006	変換が無効です。	*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。)

表 62. SQLExtendedBind() SQLSTATE (続き)

SQLSTATE	説明	解説
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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
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 のデータ・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。
HY009	引き数値が無効です。	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	関数の順序エラーです。	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 SQL の操作中に、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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY021	不整合な記述子情報	整合性検査時に検査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長が無効です。	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	パラメーターの数値が無効です。	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。

表 62. SQLExtendedBind() SQLSTATE (続き)

SQLSTATE	説明	解説
HY094	位取り値が無効です。	*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	ドライバーが機能していません。	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。

制約

なし。

例

該当するサンプルの一覧については、 sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内の README ファイルを参照してください。

参照

- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』

SQLExtendedFetch - 拡張取り出し (行の配列の取り出し)

使用すべきでない関数

注:

ODBC バージョン 3 では、SQLExtendedFetch() は使用すべきではありません。代わりに、SQLFetchScroll() に置き換えられています。詳細については、446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』を参照してください。

このバージョンの DB2 CLI でも引き続き SQLExtendedFetch() をサポートはしていますが、 SQLFetchScroll() 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。上記の関数と、その他の使用すべきでない関数の詳細については、822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

目的

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLExtendedFetch() は、バインドされた列ごとの配列の形式で、複数行を含むデータのブロック (*行セット* という) を返すことにより、 SQLFetch() の機能を拡張します。行セットのサイズは、SQLSetStmtAttr() 呼び出しの SQL_ROWSET_SIZE 属性で判別されます。

データ行を一度に 1 行ずつ取り出すには、アプリケーションは SQLFetch() を呼び出す必要があります。

ブロックまたは配列検索の詳細については、99ページの『配列への結果セットの取り出し』を参照してください。

構文

```
SQLRETURN   SQLExtendedFetch (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLUSMALLINT      FetchOrientation,  /* fFetchType */
                SQLINTEGER        FetchOffset,       /* irow */
                SQLUINTEGER  *FAR RowCountPtr,       /* pcrow */
                SQLUSMALLINT *FAR RowStatusArray);   /* rgfRowStatus */
```

関数引き数

表 63. SQLExtendedFetch 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	取り出しの方向とタイプ。 DB2 CLI は、取り出し方向 SQL_FETCH_NEXT だけ、つまり順方向のみのカーソル方向をサポートします。次のデータ配列 (行セット) が取り出されます。
SQLINTEGER	FetchOffset	入力	将来の利用のために予約済み。
SQLUINTEGER *	RowCountPtr	出力	実際に取り出された行数。処理中にエラーが発生した場合、 *RowCountPtr* はエラーが発生した行の前の行 (行セット内) の順位を指定します。取り出し中にエラーが発生した場合、最初の行 *RowCountPtr* は値 0 を指します。
SQLUSMALLINT *	RowStatusArray	出力	状況値の配列。要素数は、行セット内の行数 (SQL_ROWSET_SIZE 属性で定義される) と等しくなければなりません。行取り出しのたびに状況値が返されます。 • SQL_ROW_SUCCESS 取り出された行数が状況配列内の要素数より小さい (つまり、行セットのサイズより小さい) 場合、残りの状況要素は SQL_ROW_NOROW に設定されます。 DB2 CLI は、取り出しの開始以降に行が更新または削除されたかどうかを検出できません。そのため、以下の ODBC 定義の状況値は報告されません。 • SQL_ROW_DELETED • SQL_ROW_UPDATED

使用法

行の集まりの配列取り出しを実行するには、SQLExtendedFetch() を使用します。アプリケーションは、SQL_ROWSET_SIZE オプションを指定して SQLSetStmtAttr() を呼び出し、配列のサイズを指定します。

最初に SQLExtendedFetch() を呼び出す前に、カーソルを最初の行に置いてください。 SQLExtendedFetch() を呼び出した後、ちょうど検索された行セットの最後の行要素に対応する、結果セット内の行に、カーソルが置かれます。

SQLBindCol() または SQLBindFileToCol() 関数によってバインドされた結果セット内の列について、 DB2 CLI は必要に応じてバインドされた列のデータを変換し、これらの列にバインドされた場所にそのデータを保管します。 99ページの『配列への結果セットの取り出し』の節に記述されているように、列方向または行方向に結果セットをバインドすることができます。

- アプリケーション変数を列方向にバインドする場合:

  結果セットを列方向にバインドする場合、アプリケーションは SQL_ATTR_BIND_TYPE ステートメント属性に SQL_BIND_BY_COLUMN を指定します。 (これは省略時値です。) 次いで、アプリケーションは SQLBindCol() 関数を呼び出します。

  アプリケーションが SQLExtendedFetch() を呼び出すと、バッファーの初めに最初の行のデータが保管されます。それ以降のデータの各行は、*cbValueMax* バイト (SQLBindCol() 呼び出しの引き数) のオフセットに、または関連付けられている C バッファー・タイプが固定幅 (たとえば、SQL_C_LONG) である場合は、直前の行のデータからのその固定長に対応するオフセットに保管されます。

  バインドされた列ごとに、戻りに使用できるバイト数は、列にバインドされた *pcbValue* (SQLBindCol() の据え置き出力引き数) バッファーに保管されます。列の最初の行の戻りに使用できるバイト数はこのバッファーの先頭に保管され、それ以降の各行の返りに使用できるバイト数は直前の行の値から *sizeof(SQLINTEGER)* バイトのオフセットに保管されます。列のデータが特定の行に対して NULL であれば、pcbValue 配列内の関連する要素は SQL_NULL_DATA に設定されます。

- ファイル参照を列方向にバインドする場合:

  SQLBindFileToCol() の *StringLength* と *IndicatorValue* ポインターは出力配列を指すポインターです。ファイルの実際の長さは StringLength 配列の先頭に、またそれに関連づけられた最初の行の標識値は IndicatorValue 配列の先頭にそれぞれ保管されます。それより後のファイルの長さと標識値は、直前の行から *sizeof(SQLINTEGER)* バイトのオフセットにある配列に書き込まれます。

- アプリケーション変数の行方向のバインドの場合:

  最初にアプリケーションは、取り出されるデータの単一行とそれに関連した個々の列データ値のデータ長を保存できる構造のサイズに *vParam* 引き数を設定し、 SQL_ATTR_BIND_TYPE 属性を指定して、 SQLSetStmtAttr() を呼び出す必要があります。

  バインドされた列ごとに、データの最初の行は、 SQLBindCol() 呼び出しの *rgbValue* に指定されたアドレスに保管され、それ以降の各行は直前の行のデータから *vParam* バイト (SQLSetStmtAttr() 呼び出しに使用されている) のオフセットに保管されます。

  バインドされた列ごとに、最初の行の戻りに使用できるバイト数は SQLBindCol() 呼び出しの *pcbValue* 引き数で指定されたアドレスに保管され、それ以降の各行の返りに使用できるバイト数は、直前の行の値があるアドレスから *vParam* バイトのオフセットに保管されます。

ファイル参照の行方向バインドはサポートされていません。

SQLExtendedFetch() が行セット全体に適用されるエラーを戻すと、SQL_ERROR 関数戻りコードが該当する SQLSTATE とともに報告されます。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。

単一行に適用するエラーが発生する場合:

- 行の *RowStatusArray* 配列にある対応する要素は、SQL_ROW_ERROR に設定されます。
- **01**S01 の SQLSTATE が、 *SQLError()* を使用して取得できるエラーのリストに追加されます。
- 現在行のエラーを説明しているゼロまたは複数の追加 SQLSTATE が、*SQLError()* を使用して取得できるエラーのリストに追加されます。

RowStatusArray 配列に SQL_ROW_ERROR が返された場合にのみ、対応する要素のあるエラーが発生したことを示しています。これは、生成された SQLSTATE の数を示してはいません。それで、SQLSTATE **01**S01 は、各行の SQLSTATE の結果における区切り記号として使用されます。 DB2 CLI は行セットから残りの行の取り出しを継続し、関数戻りコードとして SQL_SUCCESS_WITH_INFO を戻します。 SQLExtendedFetch() が返された後、エラーが発生している各行について、 **01**S01 の SQLSTATE および現在行のエラーを示すゼロまたは複数の追加 SQLSTATE が存在します。これは SQLError() を介して取り出すことができます。特定の行にあてはまっている個々のエラーは、カーソルに影響を及ぼして先に進めなくなることはありません。

RowStatusArray 配列出力バッファー内の要素数は、行セット内の行数 (SQL_ROWSET_SIZE ステートメント属性で定義される) と等しくなければなりません。取り出された行数が状況配列内の要素数より小さい場合、残りの状況要素は SQL_ROW_NOROW に設定されます。

アプリケーションは、 SQLExtendedFetch() 呼び出しと SQLFetch() 呼び出しを混合できません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

診断

表 64. SQLExtendedFetch 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。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
01S01	行の中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 を戻します。)
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	変換が無効です。	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。
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ヌル・ポインターでしたが、対応する列の値はヌルです。 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ヌル・ポインターでしたが、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。

表 64. SQLExtendedFetch SQLSTATE (続き)

SQLSTATE	説明	解説
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 ユニバーサル・データベースによって検出された場合、関連するカーソルは未定義です。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、バインドされた列のデータ・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	日時形式が無効です。	文字ストリングから日時形式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・スタンプに関する算術演算で有効な日付の範囲内にない結果になるか、あるいは、バインドされた変数が小さすぎて、その変数に日時値を割り当てることができません。
22012	ゼロによる割算は無効です。	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 64. SQLExtendedFetch SQLSTATE (続き)

SQLSTATE	説明	解説
428A1	ホスト・ファイル変数で参照されるファイルにアクセスできません。	これは、次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いずれか、または両方の形式が無効です。 • 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、予期しない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、媒体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。
56084	DRDA では、LOB データはサポートされていません。	LOBが DRDA でサポートされていません。 DRDA サーバーに接続 (DB2 コネクトを使用して行う) しているときには、 LOB 列の選択や更新を行うことができません。

表 64. SQLExtendedFetch SQLSTATE (続き)

SQLSTATE	説明	解説
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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数の順序エラーです。	SQLFetch() 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 のために SQLExtendedFetch() を呼び出しました。 *StatementHandle* の 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 SQL の操作中に、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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
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。
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、認識されませんでした。
HYC00	ドライバーが機能していません。	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。 指定された取り出しタイプは認識されますが、サポートされません。
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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、 SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約
なし。

例
446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』 を参照してください。

参照
- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 267ページの『SQLBindFileToCol - LOB 列に LOB ファイル参照をバインドする』
- 409ページの『SQLExecute - ステートメントの実行』
- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 434ページの『SQLFetch - 次の行の取り出し』

SQLExtendedPrepare - 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() を繰り返し呼び出す代わりに、この関数を使用できます。

構文

```
SQLRETURN SQLExtendedPrepare( SQLHSTMT      StatementHandle,
                      SQLCHAR *     StatementText,
                      SQLINTEGER    TextLength,
                      SQLINTEGER    cPars,
                      SQLSMALLINT   sStmtType,
                      SQLINTEGER    cStmtAttrs,
                      SQLINTEGER *  piStmtAttr,
                      SQLINTEGER *  pvParams );
```

関数引き数

表 65. 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* 引き数の内容の長さ。 これは、*StatementText* 内の SQL ステートメントの正確な長さに設定するか、ステートメント・テキストがヌル終了の場合は SQL_NTS に設定する必要があります。
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	sStmtType	入力	ステートメント・タイプ。有効な値については、428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。

表 65. SQLExtendedPrepare() 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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。

SQLExtendedBind() 使用時の要件は、以下の 2 つです。

1. SQL ステートメントを走査して、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 に設定されているか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、SQLExtendedBind() は使用できません。
2. SQL ステートメントに含めるパラメーター・マーカーの数を事前に (*cRecords* で) 指示しておく。

StatementHandle、*StatementText*、および *TextLength* については、 630ページの『SQLPrepare - ステートメントを準備する』を参照してください。

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。有効なステートメント属性については、756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』を参照してください。

cStmtType 値のリスト

引き数 *cStmtType* は、以下のど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_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 66. SQLExtendedPrepare SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
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。

表 66. SQLExtendedPrepare SQLSTATE (続き)

SQLSTATE	説明	解説
01508	ステートメントはブロック化できませんでした。	このステートメントは、記憶域以外の理由でブロック化できませんでした。
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 を戻します。)
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と一致しません。	*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程度と一致していませんでした。
21S02	派生表の程度が列リストと一致しません。	*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程度になっていません。
22018	キャスト仕様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、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 つ以上が含まれていました。 • COMMIT • ROLLBACK • 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 • 構文エラーを含むステートメント
40001	トランザクション・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。

表 66. SQLExtendedPrepare SQLSTATE (続き)

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。
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存在しています。	*StatementText* 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視点名はすでに存在しています。
S0002	データベース・オブジェクトは存在していません。	*StatementText* に、存在していない表名または視点名を参照する SQL ステートメントがあります。
S0011	索引はすでに存在しています。	*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	索引がありません。	*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	列はすでに存在しています。	*StatementText* に ALTER TABLE ステートメントがあり、ADD 文節に指定されている列は固有になっていなかったか、基本表の既存の列を識別できませんでした。
S0022	列がありません。	*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	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 `SQLGetDiagRec()` により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、`SQLCancel()` が複数スレッドの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。

表 66. SQLExtendedPrepare SQLSTATE (続き)

SQLSTATE	説明	解説
HY009	引き数値が無効です。	*StatementText* は、ヌル・ポインターでした。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。
HY011	この段階で操作は無効です。	*Attribute* が、SQL_ATTR_CONCURRENCY、SQL_ATTR_CURSOR_TYPE、SQL_ATTR_SIMULATE_CURSOR、または SQL_ATTR_USE_BOOKMARKS であり、ステートメントが準備済みでした。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY014	もはやハンドルはありません。	DB2 CLI は、内部資源が原因でハンドルを割り当てることができませんでした。
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。	*Attribute* 引き数が SQL_ATTR_IMP_ROW_DESC または SQL_ATTR_IMP_PARAM_DESC でした。 *Attribute* 引き数が SQL_ATTR_APP_ROW_DESC または SQL_ATTR_APP_PARAM_DESC であり、**ValuePtr* の値が、暗黙的に割り当てられた記述子ハンドルでした。
HY024	属性値が無効です。	指定済みの *Attribute* 値が与えられているので、 **ValuePtr* に指定されたのは無効な値でした。 (DB2 CLI がこの SQLSTATE を戻すのは、 SQL_ATTR_ACCESS_MODE や SQL_ATTR_ASYNC_ENABLE 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に対しては、ドライバーで **ValuePtr* の指定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長が無効です。	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。	DB2 CLI のこのバージョンでは、引き数 *Attribute* の指定値が無効です。
HYC00	ドライバーが機能していません。	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・ソースによりサポートされていませんでした。

表 66. SQLExtendedPrepare 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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() 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。したがって、 SQLExecute() を呼び出すときにもアプリケーションがこれらの条件を処理できなければなりません。

制約

なし。

例

該当するサンプルの一覧については、 sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内の README ファイルを参照してください。

参照

- 630ページの『SQLPrepare - ステートメントを準備する』
- 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』

SQLFetch - 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。

列を次の位置にバインドすることができます。
- アプリケーション記憶域
- 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() は、照会を実行するか、 SQLGetTypeInfo() 関数を呼び出すか、またはカタログ関数を呼び出すかのいずれかにより、結果セットを生成した後にのみ (同一ステートメント・ハンドルを使用して) 呼び出すことができます。

一度に複数行を取り出すには、SQLFetchScroll() を使用します。

構文

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

関数引き数

表 67. 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、カーソルを進ませることだけ行います。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。 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() を使用して列をバインドすることにより、より良いパフォーマンスが得られます。

列を以下の位置にバインドすることができます。

- アプリケーション記憶域

  SQLBindCol() は、アプリケーション記憶域を列にバインドするときに使用します。データは、取り出し時にサーバーからアプリケーションへ転送されます。返すために利用できるデータの長さも設定できます。

- LOB ロケーター

  SQLBindCol() は、LOB ロケーターを列にバインドするときに使用します。取り出し時には、サーバーからアプリケーションへ LOB ロケーター (4 バイト) だけが転送されます。

  一度アプリケーションがロケーターを受け取ると、それを SQLGetSubString()、 SQLGetPosition()、SQLGetLength() に使用したり、別の SQL ステートメントのパラメーター・マーカーの値として使用することができます。 SQLGetSubString() は、別のロケーターか、またはデータ自体を返すことができます。すべてのロケーターは、そのロケーターを作成したトランザクションの終了まで (カーソルが別の行へ移動した場合も含む)、または FREE LOCATOR ステートメントでロケーターが解放されるまで有効です。

- LOB ファイル参照

  SQLBindFileToCol() は、ファイルを LOB 列にバインドするときに使用します。 DB2 CLI はデータを直接ファイルに書き込み、 SQLBindFileToCol() に指定された StringLength および IndicatorValue バッファーを更新します。

  列のデータ値が NULL で、SQLBindCol() が使用されている場合、SQL_NULL_DATA は SQLBindCol() に指定されている *pcbValue* バッファーに保管されます。

列のデータ値が NULL で、SQLBindFileToCol() が使用されている場合、*IndicatorValue* は SQL_NULL_DATA に設定され、 *StringLength* は 0 に設定されます。

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。

バインドされた記憶域バッファーが、 SQLFetch() から返されたデータを保持するほど大きくないと、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る列データの実際の長さが含まれます。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、どの文字列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)。

図形データ・タイプの切り捨ては文字データ・タイプと同じ方法で処理されます。ただし、*rgbValue* バッファーが、 SQLBindCol() 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図形 (DBCS) データは、 C バッファー・タイプが SQL_C_CHAR のときにはヌル終了しません。 (ただし、PATCH1 初期設定キーワードによる指定がない場合です。詳細については、174ページの『構成キーワード』を参照してください。) バッファー・タイプが SQL_C_DBCHAR の場合は、図形データはヌル終了します。

切り捨ては、SQL_ATTR_MAX_LENGTH ステートメント属性にも影響されます。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 (にヌル終止符を加えたもの) 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か、残りの行が不要である場合、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資源を廃棄する必要があります。

一度に複数行を取り出すには、SQLFetchScroll() を使用します。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。 **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。 *FetchOrientation* セットを使用する SQLFetchScroll() の呼び出しは、 SQL_FETCH_NEXT と同等です。詳細については、75ページの『スクロール可能カーソル』を参照してください。

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。

詳細については、 SQL_FETCH_NEXT の SQLFetchScroll()、448ページの『カーソル位置の規則』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() と同じ方法で行状況配列の値を設定します。詳しくは、SQLFetchScroll()、 451ページの『行状況』を参照してください。

行フェッチ・バッファー

SQLFetch() は、 SQLFetchScroll() と同じ方法で行フェッチ・バッファーに取り出されている行の数を返します。詳しくは、SQLFetchScroll()、452ページの『行フェッチ・バッファー』を参照してください。

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提供されます。診断レコードの詳細については、512ページの『SQLGetDiagField - 診断データのフィールドの入手』を参照してください。

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 (カーソル状況が無効) 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)) は、単一行、SQLFetch() に適用されます。

- エラーの場合は、行状況配列の対応する要素を 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、行セットにある行をすべて取り出すまで、行の取り出しを継続します。行セットの各行 (状況が SQL_ROW_NOROW である行はカウントしない) でエラーが生じない限り、 SQL_SUCCESS_WITH_INFO が返されます。エラーが生じた場合は、SQL_ERROR が返されます。特に、行セットのサイズが 1 であり、その行でエラーが生じると、 SQLFetch() は SQL_ERROR を返します。

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。詳細については、SQLGetDiagField()、519ページの『状況レコードの順序』を参照してください。

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。

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 68. 記述子フィールド

記述子フィールド	説明	位置	設定時に使用するもの
SQL_DESC_ARRAY_SIZE	ARD	ヘッダー	SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性
SQL_DESC_ARRAY_STATUS_PTR	IRD	ヘッダー	SQL_ATTR_ROW_STATUS_PTR ステートメント属性
SQL_DESC_BIND_OFFSET_PTR	ARD	ヘッダー	SQL_ATTR_ROW_BIND_OFFSET_PTR ステートメント属性
SQL_DESC_BIND_TYPE	ARD	ヘッダー	SQL_ATTR_ROW_BIND_TYPE ステートメント属性
SQL_DESC_COUNT	ARD	ヘッダー	SQLBindCol() の *ColumnNumber* 引き数
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

表 68. 記述子フィールド (続き)

記述子フィールド	説明	位置	設定時に使用するもの
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 つのフィールドを異なる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 である場合、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 69. SQLFetch SQLSTATE

SQLSTATE	説明	解説
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)。
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	変換が無効です。	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。
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ヌル・ポインターでしたが、対応する列の値はヌルです。 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ヌル・ポインターでしたが、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 ユニバーサル・データベース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。

表 69. SQLFetch SQLSTATE (続き)

SQLSTATE	説明	解説
22007	日時形式が無効です。	文字ストリングから日時形式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。	日時フィールドがオーバーフローしました。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にないか、または日時の値が小さ過ぎて、結合変数に割り当てることができません。
22012	ゼロによる割算は無効です。	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 69. SQLFetch SQLSTATE (続き)

SQLSTATE	説明	解説
428A1	ホスト・ファイル変数で参照される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いずれか、または両方の形式が無効です。 • 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、予期しない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、媒体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。
56084	DRDA では、LOB データはサポートされていません。	LOB が DRDA でサポートされていません。 DRDA サーバーに接続 (DB2 コネクトを使用して行う) しているときには、 LOB 列の選択や更新を行うことができません。

表 69. SQLFetch SQLSTATE (続き)

SQLSTATE	説明	解説
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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。
HY010	関数の順序エラーです。	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* の 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 SQL の操作中に、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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
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	ドライバーが機能していません。	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。

表 69. SQLFetch 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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

なし。

例

```
/* From the CLI sample utilcli.c */
/* ... */
    /* fetch each row and display */
    sqlrc = SQLFetch( hstmt ) ;
    if (sqlrc == SQL_NO_DATA_FOUND)
    {   printf("¥nData not found.¥n");
    }
```

参照

- 446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』
- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 267ページの『SQLBindFileToCol - LOB 列に LOB ファイル参照をバインドする』
- 409ページの『SQLExecute - ステートメントの実行』
- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 488ページの『SQLGetData - 列からのデータの入手』

SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す

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 70. 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、447ページの『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、447ページの『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た後で、その結果セット上にあるカーソルをクローズする前です。列のど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。

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。

カーソル位置の規則

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 71. 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 72. 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、かつ LastResultRow < RowsetSize	1 [a]
終了後、かつ LastResultRow >= RowsetSize	LastResultRow - RowsetSize + 1

a `SQLFetchScroll()` は SQLSTATE 01S06 (結果セットが最初の行設定を戻す前に取り出そうとしています。) 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 73. SQL_FETCH_RELATIVE 規則

条件	新しい行セットの最初の行
(開始前、かつ *FetchOffset* > 0) または (終了後、かつ *FetchOffset* < 0)	-- [a]
開始前、かつ *FetchOffset* <= 0	開始前
CurrRowsetStart = 1 かつ FetchOffset < 0	開始前
CurrRowsetStart > 1 かつ CurrRowsetStart + *FetchOffset* < 1 AND |*FetchOffset*| > RowsetSize	開始前
CurrRowsetStart > 1 かつ CurrRowsetStart + *FetchOffset* < 1 AND |*FetchOffset*| <= RowsetSize	1 [b]
1 <= CurrRowsetStart + *FetchOffset* <= LastResultRow	CurrRowsetStart + *FetchOffset*
CurrRowsetStart + *FetchOffset* > LastResultRow	終了後

表 73. SQL_FETCH_RELATIVE 規則 (続き)

条件	新しい行セットの最初の行
終了後、かつ *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設定を戻す前に取り出そうとしています。) 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 74. SQL_FETCH_ABSOLUTE 規則

条件	新しい行セットの最初の行
FetchOffset < 0 AND |*FetchOffset*| <= LastResultRow	LastResultRow + *FetchOffset* + 1
FetchOffset < 0 AND |*FetchOffset*| > LastResultRow、かつ |FetchOffset| > RowsetSize	開始前
FetchOffset < 0 AND |*FetchOffset*| > LastResultRow、かつ |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設定を戻す前に取り出そうとしています。) 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 75. 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 76. SQL_FETCH_LAST 規則

条件	新しい行セットの最初の行
RowsetSize <= LastResultRow	LastResultRow - RowsetSize + 1

表 76. SQL_FETCH_LAST 規則 (続き)

条件	新しい行セットの最初の行
RowsetSize > LastResultRow	1

SQL_FETCH_BOOKMARK 規則は以下のとおりです。

表 77. 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

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。詳細については、434ページの『SQLFetch - 次の行の取り出し』を参照してください。

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。

バッファー・アドレス

SQLFetchScroll() は、データおよび長さ / 標識バッファーのアドレスを決定するために SQLFetch() と同じ公式を使用します。詳細については、SQLBindCol() の『バッファー・アドレス』を参照してください。

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要素があるべきです。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ヌル・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。行状況配列のこの要素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ヌル・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() と同じ仕方でエラーと警告を返します。詳細については、SQLFetch()、437ページの『エラー処理』を参照してください。SQLExtendedFetch() は、SQLFetch() と同じ仕方でエラーを返します。ただし、以下の場合は例外です。

- 行セットの特定の行に当てはまる警告が生じる場合、 SQLExtendedFetch() は行状況配列にある対応する項目を SQL_ROW_SUCCESS に設定します。SQL_ROW_SUCCESS_WITH_INFO ではありません。
- 行セットのすべての行でエラーが生じると、 SQLExtendedFetch() は SQL_SUCCESS_WITH_INFO を返します。 SQL_ERROR ではありません。
- 個々の行に適用される状況レコードの各グループにおいて、SQLExtendedFetch() によって返される最初の状況レコードには SQLSTATE 01S01 (行にエラーがある) が入っていなければなりません。SQLFetchScroll() はこの SQLSTATE は返しません。 SQLExtendedFetch() が付加的な SQLSTATE を返すことができなかった場合、引き続きこの SQLSTATE を返さなければならないので注意してください。

記述子および SQLFetchScroll()

SQLFetchScroll() は、SQLFetch() と同じ仕方で記述子と対話します。詳細については、SQLFetch()、439ページの『記述子と SQLFetch』を参照してください。

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 78. SQLFetchScroll SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
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 2 進データを切り捨てた結果として、ストリングまたは 2 進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 を戻します。)
01S01	行の中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 を戻します。) (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設定を戻す前に取り出そうとしています。	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 を戻します。)
01S07	小数が切り捨てられました。	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、タイム・スタンプ、およびインターバル・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。
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	変換が無効です。	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。

表 78. SQLFetchScroll SQLSTATE (続き)

SQLSTATE	説明	解説
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。
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) がヌル・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	日時形式が無効です。	結果セットにある文字列が日付、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	ゼロによる割算は無効です。	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	キャスト仕様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正確な数値 C バッファーにバインドされ、その列にある値は有効で正確な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日時または間隔 C バッファーにバインドされ、その列にある値は有効な日時またはインターバル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・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	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。

表 78. SQLFetchScroll SQLSTATE (続き)

SQLSTATE	説明	解説
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY008	操作が取り消されました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。
HY010	関数の順序エラーです。	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。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。 *StatementHandle* のために SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が戻されました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、この関数が呼び出されました。 SQLFetch() が呼び出され、DB2 v2 またはそれ以前のサーバーに接続された後、 SQL_CLOSE オプションを使用して SQLFreeStmt() を呼び出す前、または SQLMoreResults() を呼び出す前に、SQLFetchScroll() が *StatementHandle* のために呼び出されました。 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* のために SQLFetchScroll() を呼び出しました。

表 78. SQLFetchScroll SQLSTATE (続き)

SQLSTATE	説明	解説
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ヌル・ポインターでした。
HYC00	ドライバーが機能していません。	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。

制約

なし。

例

```
/* From the CLI sample TBREAD.C */
/* ... */
    /* select using SQLFetchScroll with column-wise binding */
    rc = TbSelectUsingFetchScrollColWise( hdbc) ;

/* ... */
    sqlrc = SQLFetchScroll( hstmt, SQL_FETCH_ABSOLUTE, 15 );
    STMT_HANDLE_CHECK( hstmt, sqlrc);
    if (sqlrc == SQL_NO_DATA_FOUND)
    {   printf("¥n    Data not found.¥n");
    }
    for ( i = 0; i < rowsFetchedNb; i++)
```

```
    {   printf("        %-14s%-14s¥n", col1.val[i], col2.val[i]);
    }
    /* output the Row Status Array if the complete rowset was not returned. */
    if ( rowsFetchedNb != ROWSET_SIZE)
    {   printf("        Previous rowset was not full:¥n");
        for (i = 0; i < ROWSET_SIZE; i++)
{   printf("           Row Status Array[%i] = %s¥n",
                  i, ROWSTATVALUE[rowStatus[i]]);
        }
    }
```

参照

- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 321ページの『SQLCancel - ステートメントを取り消す』
- 369ページの『SQLDescribeCol - 列の属性のセットを返す』
- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 409ページの『SQLExecute - ステートメントの実行』
- 434ページの『SQLFetch - 次の行の取り出し』
- 619ページの『SQLNumResultCols - 結果列の数の入手』
- 744ページの『SQLSetPos - 行セット (Rowset) でカーソル位置を設定する』
- 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』

SQLForeignKeys - 外部キー列のリストを入手する

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。

構文

```
SQLRETURN   SQLForeignKeys   (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLCHAR      *FAR PKCatalogName,     /* szPkCatalogName */
                SQLSMALLINT       NameLength1,       /* cbPkCatalogName */
                SQLCHAR      *FAR PKSchemaName,      /* szPkSchemaName */
                SQLSMALLINT       NameLength2,       /* cbPkSchemaName */
                SQLCHAR      *FAR PKTableName,       /* szPkTableName */
                SQLSMALLINT       NameLength3,       /* cbPkTableName */
                SQLCHAR      *FAR FKCatalogName,     /* szFkCatalogName */
                SQLSMALLINT       NameLength4,       /* cbFkCatalogName */
                SQLCHAR      *FAR FKSchemaName,      /* szFkSchemaName */
                SQLSMALLINT       NameLength5,       /* cbFkSchemaName */
                SQLCHAR      *FAR FKTableName,       /* szFkTableName */
                SQLSMALLINT       NameLength6);      /* cbFkTableName */
```

関数引き数

表 79. 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	基本キー表のカタログ修飾子。これは、NULL ポインターまたは長さゼロのストリングでなければなりません。
SQLSMALLINT	NameLength1	入力	*PKCatalogName* の長さ。これは、0 に設定する必要があります。
SQLCHAR *	PKSchemaName	入力	基本キー表のスキーマ修飾子。
SQLSMALLINT	NameLength2	入力	*PKSchemaName* の長さ。
SQLCHAR *	PKTableName	入力	基本キーを含む表名の名前。
SQLSMALLINT	NameLength3	入力	*PKTableName* の長さ。

表 79. SQLForeignKeys 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLCHAR *	FKCatalogName	入力	外部キーを含む表のカタログ修飾子。これは、NULL ポインターまたは長さゼロのストリングでなければなりません。
SQLSMALLINT	NameLength4	入力	*FKCatalogName* の長さ。これは、0 に設定する必要があります。
SQLCHAR *	FKSchemaName	入力	外部キーを含む表のスキーマ修飾子。
SQLSMALLINT	NameLength5	入力	*FKSchemaName* の長さ。
SQLCHAR *	FKTableName	入力	外部キーを含む表の名前。
SQLSMALLINT	NameLength6	入力	*FKTableName* の長さ。

使用法

PKTableName に表名が含まれており、 *FKTableName* が空ストリングである場合、 SQLForeignKeys() は指定された表の基本キーとそ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 に表名が含まれており、 *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基本キー (他の表の中にある) を含む結果セットを返します。

PKTableName と *FKTableName* の両方に表名が含まれている場合、 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* で指定された表の基本キーを参照します。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省略時設定されます。

461ページの表80 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。基本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基本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 より小さいため、アプリケーションは出力バッファー用に常に 128 文字 (にヌル終止符を加えたも

の) を確保するようにしたり、あるいは、SQL_MAX_CATALOG_NAME_LEN、 SQL_MAX_SCHEMA_NAME_LEN、 SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を指定して SQLGetInfo() を呼び出し、接続している DBMS でサポートされる関連付けられた TABLE_CAT、TABLE_SCHEM、 TABLE_NAME、および COLUMN_NAME 列の実際の長さをそれぞれ判別することができます。

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。

表 80. SQLForeignKeys で返される列

列番号 / 名	データ・タイプ	説明
1 PKTABLE_CAT	VARCHAR(128)	これは常に NULL です。
2 PKTABLE_SCHEM	VARCHAR(128)	PKTABLE_NAME を含むスキーマの名前。
3 PKTABLE_NAME	非 NULL の VARCHAR(128)	基本キーを含む表の名前。
4 PKCOLUMN_NAME	非 NULL の VARCHAR(128)	基本キー列名。
5 FKTABLE_CAT	VARCHAR(128)	これは常に NULL です。
6 FKTABLE_SCHEM	VARCHAR(128)	FKTABLE_NAME を含むスキーマの名前。
7 FKTABLE_NAME	非 NULL の VARCHAR(128)	外部キーを含む表の名前。
8 FKCOLUMN_NAME	非 NULL の VARCHAR(128)	外部キー列名。
9 ORDINAL_POSITION	非 NULL の SMALLINT	1 から始まるキー内の列の順序を示す位置。

表 80. SQLForeignKeys で返される列 (続き)

列番号 / 名	データ・タイプ	説明
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 DBMS 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
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
12 FK_NAME	VARCHAR(128)	外部キー識別子。データに適用できない場合は、NULL。
13 PK_NAME	VARCHAR(128)	基本キー識別子。データに適用できない場合は、NULL。
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 81. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効です。	引き数 *PKTableName* と *FKTableName* はともにヌル・ポインターでした。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。
HY014	もはやハンドルはありません。	DB2 CLI は、内部資源が原因でハンドルを割り当てることができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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。 533ページの『SQLGetInfo - 一般情報の入手』を参照してください。
HYC00	ドライバーが機能していません。	DB2 CLI は、表名の修飾子としてカタログ 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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

なし。

例

```
/* From the CLI sample tbconstr.c */
/* ... */
    /* fetch each row, and display */
    printf("    Fetch each row and display.¥n");
    sqlrc = SQLFetch( hstmt );
```

参照

- 636ページの『SQLPrimaryKeys - 表の基本キー列を入手する』
- 788ページの『SQLStatistics - 基本表の索引および統計情報の入手』

SQLFreeConnect - 接続ハンドルの解放

使用すべきでない関数

注:

ODBC バージョン 3 では、SQLFreeConnect() は使用すべきではありません。代わりに、SQLFreeHandle() に置き換えられています。詳細については、470ページの『SQLFreeHandle - ハンドル資源を解放する』を参照してください。

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、 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。 上記の関数と、その他の使用すべきでない関数の詳細については、822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

新しい関数への移行

たとえば、次の旧ステートメント

```
SQLFreeConnect(hdbc);
```

は、新しい関数を使用して、以下のように書き換えます。

```
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
```

目的

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLFreeConnect() は、接続ハンドルを無効にし、解放します。接続ハンドルに関連したすべての DB2 CLI 資源が解放されます。

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

構文

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

関数引き数

表 82. SQLFreeConnect 引き数

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

使用法

接続がまだ存在しているときにこの関数を呼び出すと、 SQL_ERROR が返され、接続ハンドルはそのまま有効です。

終了を継続するには、SQLFreeEnv() を呼び出すか、または新しい接続ハンドルが必要であれば SQLAllocConnect() を呼び出してください。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 83. SQLFreeConnect SQLSTATE

SQLSTATE	説明	解説
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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数の順序エラーです。	*hdbc* の SQLDisconnect() 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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。

制約

なし。

例

470ページの『SQLFreeHandle - ハンドル資源を解放する』を参照してください。

参照

- 379ページの『SQLDisconnect - データ・サーバーからの切断』
- 470ページの『SQLFreeHandle - ハンドル資源を解放する』

SQLFreeEnv - 環境ハンドルの解放

使用すべきでない関数

注:

ODBC バージョン 3 では、SQLFreeEnv() は使用すべきではありません。代わりに、SQLFreeHandle() に置き換えられています。詳細については、470ページの『SQLFreeHandle - ハンドル資源を解放する』を参照してください。

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、 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。 上記の関数と、その他の使用すべきでない関数の詳細については、822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

新しい関数への移行

たとえば、次の旧ステートメント

```
SQLFreeEnv(henv);
```

は、新しい関数を使用して、以下のように書き換えます。

```
SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

目的

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLFreeEnv() は、環境ハンドルを無効にし、解放します。環境ハンドルに関連したすべての DB2 CLI 資源が解放されます。

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

この関数は、アプリケーションが終了前に行われなければならない最後の DB2 CLI ステップです。

構文

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

関数引き数

表 84. SQLFreeEnv 引き数

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

使用法

有効な接続ハンドルがまだ存在しているときにこの関数を呼び出すと、SQL_ERROR が返され、環境ハンドルはそのまま有効です。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 85. SQLFreeEnv SQLSTATE

SQLSTATE	説明	解説
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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数の順序エラーです。	割り当てられた状態または接続された状態の *hdbc* があります。 `SQLFreeEnv()` を呼び出す前に、*hdbc* の `SQLDisconnect()` と `SQLFreeConnect()` を呼び出してください。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。

許可

なし。

例

470ページの『SQLFreeHandle - ハンドル資源を解放する』を参照してください。

参照

- 470ページの『SQLFreeHandle - ハンドル資源を解放する』

SQLFreeHandle - ハンドル資源を解放する

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資源を解放します。

注: この関数は、資源を解放するための一般的な関数です。これは、バージョン 2 の関数 SQLFreeConnect (接続ハンドルを解放する) および SQLFreeEnv() (環境ハンドルを解放する) から置き換わったものです。SQLFreeHandle() はさらに、ステートメント・ハンドルを解放するためのバージョン 2 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,
                              SQLHANDLE         Handle);
```

関数引き数

表 86. 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は、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、以下の要領で解放するために使用します。

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検査は行いません。

環境ハンドルの解放

SQL_HANDLE_ENV の *HandleType* を使って SQLFreeHandle() を呼び出す前に、アプリケーションは、その環境のもとで割り当てられている接続すべてに対して SQL_HANDLE_DBC の *HandleType* を使って SQLFreeHandle() を呼び出さなければなりません。これを行わないと、SQLFreeHandle() の呼び出しは、 SQL_ERROR と環境を返し、活動状態にある接続はすべて有効のままになります。

接続ハンドルの解放

SQL_HANDLE_DBC の *HandleType* を使用して SQLFreeHandle() を呼び出す前に、アプリケーションは接続のために SQLDisconnect() を呼び出さなければなりません。これを行わないと、SQLFreeHandle() の呼び出しは、 SQL_ERROR を返し、接続はすべて有効のままになります。

ステートメント・ハンドルの解放

SQL_HANDLE_STMT の *HandleType* を使用して SQLFreeHandle() を呼び出すと、 SQL_HANDLE_STMT の *HandleType* を使用して行う SQLAllocHandle() の呼び出しによって割り当てられた資源をすべて解放します。アプリケーションが SQLFreeHandle() を呼び出して結果を保留にしているステートメントを解放するときに、保留になっている結果は解放されます。アプリケーションがステートメント・ハンドルを解放するときに、 DB2 CLI はそのハンドルに関連して自動的に生成された記述子をすべて解放します。 SQLFreeHandle() が呼び出されたときに結果が保留になっている場合、結果セットは破棄されます。

接続上でオープンしているステートメントと記述子を SQLDisconnect() はすべて自動的に除去するので注意してください。

記述子ハンドルの解放

SQL_HANDLE_DESC の *HandleType* を使用して SQLFreeHandle() を呼び出すと、 *Handle* の記述子ハンドルが解放されます。 SQLFreeHandle() の呼び出しは、 *Handle* の記述子レコードの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) によって参照される可能性のあるアプリケーションが割り当てるメモリーを解放することはありません。明示的に割り当

てられている記述子ハンドルが解放されると、解放されたハンドルが関連していたすべてのステートメントは、自動的に割り当てられた記述子ハンドルに返ります。

接続上でオープンしているステートメントと記述子を SQLDisconnect() はすべて自動的に除去するので注意してください。アプリケーションがステートメント・ハンドルを解放するときに、 DB2 CLI はそのハンドルに関連して自動的に生成された記述子をすべて解放します。

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 87. SQLFreeHandle SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
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	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。

表 87. SQLFreeHandle SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数の順序エラーです。	*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。データがすべての実行時データ・パラメーターまたは列用に送られる前に、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資源は、 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・エラーが起きました。	*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実装記述子のハンドルに設定されました。

制約

なし。

例

```
/* From the CLI sample utilcli.c */
/* ... */
    /* free the environment handle */
    printf("¥nFreeing the environment handle ...¥n");
    sqlrc = SQLFreeHandle( SQL_HANDLE_ENV,  *pHenv ) ;
    rc    = HandleInfoPrint( SQL_HANDLE_ENV, *pHenv,
```

```
                                sqlrc, __LINE__, __FILE__);
if( rc == 0)
{   printf("The environment handle is free.¥n");
}
```

参照

- 246ページの『SQLAllocHandle - ハンドルを割り振る』
- 321ページの『SQLCancel - ステートメントを取り消す』
- 693ページの『SQLSetCursorName - カーソル名を設定する』

SQLFreeStmt - 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・ハンドルを除去してそのステートメント・ハンドルに関連した DB2 CLI 資源を解放します。

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 88. 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、保留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・ハンドルが除去されるか、次の SQLSetCursorName() 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 (警告またはエラーが生成されません)。

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 資源が解放され、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、保留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 の SQL_DESC_COUNT フィールドを 0 に設定し、指定されている *StatementHandle* の 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用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用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 の SQL_DESC_COUNT フィールドを 0 に設定し、指定されている *StatementHandle* の 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用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用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。LOB の使用法の詳細については、123ページの『ラージ・オブジェクトの使用』を参照してください。

照会、カタログ関数、または SQLGetTypeInfo() に関連付けられているステートメント・ハンドルが次のようになっているときに、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方法を示します。

- 照会、カタログ関数、または SQLGetTypeInfo() に関連付けられている場合は、カーソルをクローズします。
- 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る場合は、パラメーターをリセットします。
- 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る場合は、列をアンバインドします。

別の方法として、ステートメント・ハンドルを除去し、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 に設定したときに SQL_SUCCESS_WITH_INFO が返されない理由は、 SQLError() を呼び出すときに使用するステートメント・ハンドルがないことが考えられます。

診断

表 89. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。

表 89. SQLFreeStmt SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数の順序エラーです。	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・エラーが起きました。	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

```
/* From the CLI sample utilcli.c */
/* ... */
    sqlrc = SQLFreeStmt( hstmt, SQL_UNBIND ) ;
    rc = HandleInfoPrint( SQL_HANDLE_STMT, hstmt,
                          sqlrc, __LINE__, __FILE__);
    if( rc != 0) return(1) ;

/* ... */
    sqlrc = SQLFreeStmt( hstmt, SQL_CLOSE ) ;
    rc = HandleInfoPrint( SQL_HANDLE_STMT, hstmt,
                          sqlrc, __LINE__, __FILE__);
    if( rc != 0) return(1) ;
```

参照

- 246ページの『SQLAllocHandle - ハンドルを割り振る』
- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』
- 267ページの『SQLBindFileToCol - LOB 列に LOB ファイル参照をバインドする』
- 273ページの『SQLBindFileToParam - LOB パラメーターに LOB ファイル参照をバインドする』
- 446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』
- 434ページの『SQLFetch - 次の行の取り出し』

- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』

SQLGetConnectAttr - 現行属性設定を入手する

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。

構文

```
SQLRETURN    SQLGetConnectAttr(SQLHDBC           ConnectionHandle,
                               SQLINTEGER        Attribute,
                               SQLPOINTER        ValuePtr,
                               SQLINTEGER        BufferLength,
                               SQLINTEGER        *StringLengthPtr);
```

関数引き数

表 90. 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	属性	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* がポインターであるが、ストリングではない場合、*BufferLength* の値は SQL_IS_POINTER になります。 • *ValuePtr* がポインターではない場合、 *BufferLength* の値は SQL_IS_NOT_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。

表 90. SQLGetConnectAttr 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER	*StringLengthPtr	出力	**ValuePtr* 内に戻すために使用できる総バイト数 (ヌル終了文字を除く) を戻すバッファーを指すポインター。*ValuePtr* がヌル・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 からヌル終了文字の長さを引いたものより大きい場合、 **ValuePtr* の値は、*BufferLength* からヌル終了文字の長さを引いたものに切り捨てられ、これは DB2 CLI によってヌル終了になります。

使用法

設定可能な属性のリストについては、663ページの『SQLSetConnectAttr - 接続属性を設定する』を参照してください。*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。ヌル終了文字を含む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、指定した属性に省略時値がなく、 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 91. SQLGetConnectAttr SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
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* からヌル終了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 を戻します。)
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	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY010	関数の順序エラーです。	*ConnectionHandle* の 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長が無効です。	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。

表 91. SQLGetConnectAttr SQLSTATE (続き)

SQLSTATE	説明	解説
HYC00	ドライバーが機能していません。	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・ソースによりサポートされていませんでした。

制約

なし。

例

```
/* From the CLI sample DBINFO.C */
/* ... */
    sqlrc = SQLGetConnectAttr( hdbc, SQL_AUTOCOMMIT, &autocommit, 0, NULL ) ;
    HANDLE_CHECK( SQL_HANDLE_DBC, hdbc, sqlrc, &henv, &hdbc ) ;
    printf("¥n    A connection attribute...¥n");
    printf( "        Autocommit is: " ) ;
    if ( autocommit == SQL_AUTOCOMMIT_ON ) printf( "ON¥n" ) ;
    else printf( "OFF¥n" ) ;
```

参照

- 592ページの『SQLGetStmtAttr - ステートメント属性の現行設定値を入手する』
- 663ページの『SQLSetConnectAttr - 接続属性を設定する』
- 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』

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

使用すべきでない関数

注:

ODBC バージョン 3 では、SQLGetConnectOption() を使用すべきではなく、代わりに SQlGetConnectAttr() を使用します。詳細については、480ページの『SQLGetConnectAttr - 現行属性設定を入手する』を参照してください。

このバージョンの DB2 CLI でも引き続き SQLGetConnectOption() をサポートしていますが、SQLGetConnectAttr() 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。 上記の関数と、その他の使用すべきでない関数の詳細については、822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

SQLGetCursorName - カーソル名の入手

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。

構文

```
SQLRETURN   SQLGetCursorName (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLCHAR      *FAR CursorName,        /* szCursor */
                SQLSMALLINT       BufferLength,      /* cbCursorMax */
                SQLSMALLINT  *FAR NameLengthPtr);    /* pcbCursor */
```

関数引き数

表 92. 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	バッファー *CursorName* の長さ。
SQLSMALLINT *	*NameLengthPtr*	出力	*CursorName* に返すために利用可能なバイト数。

使用法

SQLGetCursorName() は、 SQLSetCursorName() に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。

SQLSetCursorName() を使用して名前を明示設定すると、ステートメントを除去するか、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。カーソル名は常に 18 文字以下であり、接続で常に固有です。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 93. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY090	ストリングまたはバッファー長が無効です。	引き数 *BufferLength* に指定された値は 0 より小さい値です。

制約

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

```
/* From the CLI sample TBMOD.C */
/* ... */
    /* get the cursor name of the SELECT statement */
    sqlrc = SQLGetCursorName( hstmtSelect,
                              cursorName, 20, &cursorLen ) ;
```

参照

- 409ページの『SQLExecute - ステートメントの実行』
- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 630ページの『SQLPrepare - ステートメントを準備する』
- 693ページの『SQLSetCursorName - カーソル名を設定する』

SQLGetData - 列からのデータの入手

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。 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() 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() を呼び出して次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   *FAR StrLen_or_IndPtr); /* pcbValue */
```

関数引き数

表 94. 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

表 94. SQLGetData 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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、順番通りに番号が付けられます。 • ブックマークを使用していない場合は、列番号は 1 から始まります (SQL_ATTR_USE_BOOKMARKS ステートメント属性を SQL_UB_OFF に設定します)。 • ブックマークを使用している場合は、列番号は 0 から始まります (ステートメント属性を SQL_UB_ON または SQL_UB_VARIABLE に設定します)。
SQLSMALLINT	*TargetType*	入力	*ColumnNumber* による列識別子の 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_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_DEFAULT を指定するとデータが省略時の C データ・タイプに変換されます。詳細については、33ページの表2 を参照してください。

表 94. 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* で指示されたバッファーの最大サイズ。
SQLINTEGER *	*StrLen_or_IndPtr*	出力	DB2 CLI が使用できるバイト数を示す値を指すポインターが、 *TargetValuePtr* バッファー内に返されます。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。 列のデータ値がヌルの場合、値は SQL_NULL_DATA です。このポインターが NULL で、 SQLFetch() でヌル・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 2 進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ヌル (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: DB2 CLI は、*TargetValuePtr* が *StrLen_or_IndPtr* の後のメモリー内に連続的に設定されると、ある程度パフォーマンスが強化されます。

使用法

SQLGetData() は同じ結果セットに対しては、 SQLFetch() が使用されており SQLFetchScroll() が使用されていない場合に限り、 SQLBindCol() とともに使用できます。一般的なステップは以下のとおりです。

1. SQLFetch() - カーソルを最初の行へ進め、最初の行を検索し、結合列のデータを転送します。
2. SQLGetData() - 指定された列のデータを転送します。
3. 必要な各列ごとに、ステップ 2 を繰り返します。

4. SQLFetch() - 次の行へカーソルを進め、次の行を検索し、結合列のデータを転送します。
5. 結果セット内の行ごとに、または結果セットが不要になるまで、ステップ 2、3、および 4 を繰り返します。

C データ・タイプ (*TargetType*) が SQL_C_CHAR、SQL_C_BINARY、SQL_C_DBCHAR 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である場合、 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入手することができます。列全体を取得するために、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 (にヌル終止符を加えたもの) 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。行全体で検索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るか、または結果セットからのデータがこれ以上必要ない場合は、 SQLFreeStmt() を呼び出してカーソルをクローズします。

TargetType 入力引き数は、 *TargetValuePtr* で指示された記憶域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 図形列データの場合、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。

TargetValuePtr に返される内容は、検索する列データが 2 進ではないか、または列の SQL データ・タイプが図形 (DBCS) であり、かつ C バッファー・タイプが SQL_C_CHAR であれば、常にヌル終了です。アプリケーションが複数の部分に分けてデータを検索している場合は、適切な調整を行う必要があります (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ヌル終了文字 5 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)。

スクロール可能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。

スクロール可能カーソルでの SQLGetData() の使用

SQLGetData() はスクロール可能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 (ブックマーク) を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。

詳細については、75ページの『スクロール可能カーソル』を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE
- 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 にヌル終止符が入れられます。

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 95. 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。

表 95. SQLGetData SQLSTATE (続き)

SQLSTATE	説明	解説
07006	変換が無効です。	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。
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ヌル・ポインターでしたが、列の値はヌルです。 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	日時形式が無効です。	文字ストリングから日時形式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・スタンプに関する算術演算で有効な日付の範囲内にない結果になるか、あるいは、バインドされた変数が小さすぎて、その変数に日時値を割り当てることができません。
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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY002	列の番号が無効です。	指定された列は、0 より小さいか、結果の列数より大きい値でした。 指定した列は 0 でしたが、アプリケーションはブックマークの使用を許可しませんでした (SQL_ATTR_USE_BOOKMARKS ステートメント属性を設定)。 `SQLExtendedFetch()` がこの結果セットに呼び出されました。

表 95. 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	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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、省略時タイプ SQL_C_CHAR、SQL_C_BINARY、または SQL_C_DBCHAR のいずれか) です。
HYC00	ドライバーが機能していません。	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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

なし。

例

バインドされた列を使用する方法と SQLGetData() を使用する方法の比較については、 445ページの『例』を参照してください。

```
/* From the CLI sample TBREAD.C */
/* ... */
        sqlrc = SQLGetData( hstmt, 2, SQL_C_CHAR, location.val, 15,
                            &location.ind ) ;
        STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参照

- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』
- 434ページの『SQLFetch - 次の行の取り出し』
- 598ページの『SQLGetSubString - ストリング値の部分を取り出す』

SQLGetDataLinkAttr - データ・リンク属性値を入手する

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,
                               SQLSMALLINT       Attribute,
                               SQLCHAR     FAR   *DataLink,
                               SQLINTEGER        DataLinkLength,
                               SQLPOINTER        *ValuePtr,
                               SQLINTEGER        BufferLength,
                               SQLINTEGER  FAR   *StringLengthPtr);
```

関数引き数

表 96. 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。

表 96. SQLGetDataLinkAttr 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	属性	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	属性が抽出される元の DATALINK 値。
SQLINTEGER	DataLinkLength	入力	DATALINK 値の長さ。
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	属性バッファーの長さ。

表 96. SQLGetDataLinkAttr 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER	*StringLength	出力	**Attribute* 内に戻すために使用できる総バイト数 (ヌル終了文字を除く) を戻すバッファーを指すポインター。 *Attribute* がヌル・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、ヌル終了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。ヌル終了文字を含むストリングの最大長は、*BufferLength* バイトになります。

データ・リンクの詳細については、*管理の手引き: 計画* を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_NO_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 97. SQLExtendedBind() 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 がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。

表 97. SQLExtendedBind() SQLSTATE (続き)

SQLSTATE	説明	解説
01004	切り捨てられたデータ。	**ValuePtr* に戻されるデータは、 *BufferLength* からヌル終了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失敗です。	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を割り当てられませんでした。
HY009	引き数値が無効です。	引き数 **DataLink* に指定された値は、ヌル・ポインター、または無効でした。
HY090	無効なストリングまたはバッファー長。	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。

制約

なし。

例

304ページの『例』を参照してください。

参照

- 302ページの『SQLBuildDataLink - DATALINK 値の作成』

SQLGetDescField - 記述子レコードの単一フィールド設定を入手する

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。

構文

```
SQLRETURN   SQLGetDescField  (SQLHDESC          DescriptorHandle,
                              SQLSMALLINT       RecNumber,
                              SQLSMALLINT       FieldIdentifier,
                              SQLPOINTER        ValuePtr,
                              SQLINTEGER        BufferLength,
                              SQLINTEGER        *StringLengthPtr);
```

関数引き数

表 98. 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情報を求める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省略時値を返します。詳しくは、699ページの『記述子フィールドの初期設定』 の SQLSetDescField() を参照してください。

表 98. SQLGetDescField 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*FieldIdentifier*	入力	値を返す記述子のフィールドを指定します。詳しくは、707ページの『FieldIdentifier 引き数』 の SQLSetDescField() を参照してください。
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* がポインターであるが、ストリングではない場合、*BufferLength* の値は SQL_IS_POINTER になります。 • *ValuePtr* がポインターではない場合、 *BufferLength* の値は SQL_IS_NOT_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 (ヌル終了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() を呼び出すことはできません。

名前、データ・タイプ、および列またはパラメーター・データの記憶域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、記述子ヘッダー (ステートメント属性でもある) にある単一のフィールドの設定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() を呼び出すとき、この関数は SQLSTATE HY091 (無効な記述子フィールド識別子) を返します。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 SQLGetDescField() を呼び出したものの、省略時値がなく、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返される値は未定義です。詳しくは、699ページの『記述子フィールドの初期設定』の SQLSetDescField() 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。

以下のレコード・フィールドは、読み取り専用として使用可能です。これらのフィールドそれぞれは、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説明と、記述子ヘッダーまたはレコードに設定可能なフィールドの説明については、 SQLSetDescField() のセクションを参照してください。記述子の詳細については、105ページの『記述子の使用』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 99. SQLGetDescField SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
HY013	予期しないメモリーのハンドル・エラーが起きました。	*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。
HY021	記述子情報が不整合です。	整合性検査時に検査された記述子情報は、整合性がとれていませんでした。詳細については、724ページの『整合性検査』 の `SQLSetDescField()` を参照してください。
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 を戻します。)
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_BOOKMARK 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。

表 99. SQLGetDescField SQLSTATE (続き)

SQLSTATE	説明	解説
HY000	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY007	関連した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	関数の順序エラーです。	*DescriptorHandle* が関連付けられていた StatementHandle に対して、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* に対して SQLExecute() または SQLExecDirect() が呼び出され、SQL_NEED_DATA が返されました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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	記述子タイプが範囲外です。	*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。

制約

なし。

例

```
/* From the CLI sample DBUSE.C */
/* ... */
    /* bind the file-parameter
    /* see how the header field SQL_DESC_ALLOC_TYPE is set. */
    sqlrc = SQLGetDescField( hIPD,
                             0, /* ignored for header fields */
                             SQL_DESC_ALLOC_TYPE,
```

```
                            &descFieldAllocType, /* The result */
                            SQL_IS_SMALLINT,
                            NULL ); /* ignored */
    STMT_HANDLE_CHECK( hstmt, sqlrc);

/* ... */
    /* see how the field SQL_DESC_PARAMETER_TYPE is set. */
    sqlrc = SQLGetDescField( hIPD,
                            1, /* Look at the parameter */
                            SQL_DESC_PARAMETER_TYPE,
                            &descFieldParameterType, /* The result */
                            SQL_IS_SMALLINT,
                            NULL ); /* ignored */
    STMT_HANDLE_CHECK( hstmt, sqlrc);

/* ... */
        /* see how the descriptor record field SQL_DESC_TYPE_NAME is set */
        rc = SQLGetDescField( hIRD,
                             (SQLSMALLINT)colCount,
                             SQL_DESC_TYPE_NAME, /* record field */
                             descFieldTypeName, /* The result */
                             25,
                             NULL ); /* ignored */
        STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参照

- 507ページの『SQLGetDescRec - 記述子レコードの複数フィールド設定を入手する』
- 696ページの『SQLSetDescField - 記述子レコードの単一フィールドを設定する』
- 728ページの『SQLSetDescRec - 列またはパラメーター・データに複数の記述子フィールドを設定する』

SQLGetDescRec - 記述子レコードの複数フィールド設定を入手する

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・データの記憶域を記述します。

構文

```
SQLRETURN   SQLGetDescRec    (SQLHDESC          DescriptorHandle,
                              SQLSMALLINT       RecNumber,
                              SQLCHAR           *Name,
                              SQLSMALLINT       BufferLength,
                              SQLSMALLINT       *StringLengthPtr,
                              SQLSMALLINT       *TypePtr,
                              SQLSMALLINT       *SubTypePtr,
                              SQLINTEGER        *LengthPtr,
                              SQLSMALLINT       *PrecisionPtr,
                              SQLSMALLINT       *ScalePtr,
                              SQLSMALLINT       *NullablePtr);
```

関数引き数

表 100. 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。
SQLSMALLINT	*RecNumber*	入力	アプリケーションが情報を求める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、行に列またはパラメーターのデータが含まれない場合、SQLGetDescRec() の呼び出しはフィールドの省略時値を返します (詳細については、 SQLSetDescField() の『記述子フィールドの初期化』を参照してください)。

表 100. SQLGetDescRec 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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* バッファーの長さ (バイト数)。
SQLSMALLINT	*StringLengthPtr	出力	**Name* バッファーに返すときに利用可能なデータのバイト数 (ヌル終了文字は除く) を返す先のバッファーを指すポインターです。バイト数が *BufferLength* と同じか大きい場合、 **Name* のデータは、*BufferLength* からヌル終了文字の長さを引いたものに切り捨てられ、 DB2 CLI によってヌル終了になります。
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	レコード (タイプが SQL_DATETIME または SQL_INTERVAL) 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。
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、単一の列またはパラメーターについて、以下のフィールドの値を検索します。

- 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、ヌル・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、省略時値がなく、まだ設定されていないときには、この関数は SQL HY091 (無効な記述子フィールド識別子) を返します。

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。記述子ヘッダーまたはレコードのフィールドの説明については、696ページの『SQLSetDescField - 記述子レコードの単一フィールドを設定する』を参照してください。記述子の詳細については、105ページの『記述子の使用』を参照してください。

戻りコード

- SQL_SUCCESS
- 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 101. SQLGetDescRec SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01004	データが切り捨てられました。	バッファー **Name* には記述子フィールド全体を返すのに十分な大きさがなかったため、フィールドが切り捨てられました。切り捨てられていない記述子フィールドの長さが、*StringLengthPtr* に返されます。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
07009	記述子索引が無効です。	*RecNumber* 引き数は 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	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 `SQLGetDiagRec()` により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY007	関連した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。

表 101. SQLGetDescRec SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数の順序エラーです。	*DescriptorHandle* が関連付けられていた *StatementHandle* に対して、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* に対して SQLExecute() または SQLExecDirect() が呼び出され、SQL_NEED_DATA が返されました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。

制約

なし。

例

該当するサンプルの一覧については、 sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内の README ファイルを参照してください。

参照

- 728ページの『SQLSetDescRec - 列またはパラメーター・データに複数の記述子フィールドを設定する』
- 501ページの『SQLGetDescField - 記述子レコードの単一フィールド設定を入手する』
- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』

SQLGetDiagField - 診断データのフィールドの入手

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。

構文

```
SQLRETURN   SQLGetDiagField  (SQLSMALLINT       HandleType,
                              SQLHANDLE         Handle,
                              SQLSMALLINT       RecNumber,
                              SQLSMALLINT       DiagIdentifier,
                              SQLPOINTER        DiagInfoPtr,
                              SQLSMALLINT       BufferLength,
                              SQLSMALLINT       *StringLengthPtr);
```

関数引き数

表 102. 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・タイプ識別子。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情報を求める状況レコード。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・レコードのフィールドを示す場合、*RecNumber* は 0 となります。そうでない場合、0 より大きい数になります。
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、514ページの『DiagIdentifier 引き数』を参照してください。

表 102. SQLGetDiagField 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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* の値によります。
SQLINTEGER	BufferLength	入力	*DiagInfoPtr* が文字ストリングを指す場合、この引き数の長さは *ValuePtr です。 ValuePtr がポインターである (しかしストリングに対するものではない) 場合、 *BufferLength* の値は SQL_IS_POINTER となります。 ValuePtr がポインターでない場合には、 *BufferLength* の値は SQL_IS_NOT_POINTER となります。 **DiagInfoPtr* の値が 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	**DiagInfoPtr* に戻すために使用できる総バイト数 (ヌル終了文字に必要なバイト数を除く) を戻すバッファーを指すポインター (文字データ用)。戻りに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* からヌル終了文字の長さを引いた長さまで切り捨てられます。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()` 関数の SQL_NEED_DATA) が戻されたとき、特定のエラーまたは警告情報を得る。
2. `SQLExecute()` または `SQLExecDirect()` への呼び出しで、挿入、削除 (DEL)、または更新操作が行われた時点で影響を受けるデータ・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スクロール可能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 関数のどれかが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 SQLGetDiagField() を呼び出すことができます。 SQLGetDiagField() は、 *Handle* 引き数で指定される診断データ構造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走査することができます。状況記録の数値は、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() 以外の他の関数が途中で呼び出されない限り、アプリケーションは後で再び SQLGetDiagField() を呼び出し、レコードからフィールドを取り出すことができます。上記以外の他の関数が途中で呼び出されると、同一ハンドルにレコードを追加してしまいます。

アプリケーションは SQLGetDiagField() を呼び出し、いつでも何らかの診断フィールドを返すことができます。例外は SQL_DIAG_ROW_COUNT です。 *Handle* が SQL ステートメントが実行されているところのステートメント・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 (その他のエラーに遭遇しない場合)。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 特定のヘッダー診断フィールドは、環境ハンドルと関連することはありません。

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識別子を示します。 *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ります。そして、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。

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 103. 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は、ステートメント・ハンドルに対してのみ定義され、SQLExecute()、SQLExecDirect()、または SQLMoreResults() が呼び出されてから定義されます。ステートメント・ハンドル以外で、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文字列です (DB2 CLI がサポートしている値については、 518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は、ステートメント・ハンドルに対してのみ定義され、なおかつ SQLExecute() または SQLExecDirect() への呼び出しがなされた後に定義されます。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_DYNAMIC_FUNCTION_CODE (戻りタイプ SQLINTEGER)

表 103. DiagIdentifier 引き数のヘッダー・フィールド (続き)

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、 518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は、ステートメント・ハンドルに対してのみ定義され、なおかつ SQLExecute() または SQLExecDirect() への呼び出しがなされた後に定義されます。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_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。戻りコードの一覧については、29ページの『関数戻りコード』を参照してください。 *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 104. 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)
: 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 *)

表 104. DiagIdentifier 引き数のレコード・フィールド (続き)

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 と同じ形式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 105. 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

表 105. 動的関数フィールドの値 (続き)

実行される SQL ステートメント	SQL_DIAG_DYNAMIC_FUNCTION の値	SQL_DIAG_DYNAMIC_ FUNCTION_CODE の値
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
不明	空ストリング	SQL_DIAG_UNKNOWN_STATEMENT

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、他のレコードすべて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走査して、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 ～ HZ) 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。
- **処理系定義の 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、 *RecNumber* が正の値の SQL_NO_DATA も戻されます。

制約

なし。

例

該当するサンプルの一覧については、 sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内の README ファイルを参照してください。

参照

- 522ページの『SQLGetDiagRec - 診断レコードの複数のフィールド設定を取得する』

SQLGetDiagRec - 診断レコードの複数のフィールド設定を取得する

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。

構文

```
SQLRETURN   SQLGetDiagRec    (SQLSMALLINT       HandleType,
                              SQLHANDLE         Handle,
                              SQLSMALLINT       RecNumber,
                              SQLCHAR           *SQLState,
                              SQLINTEGER        *NativeErrorPtr,
                              SQLCHAR           *MessageText,
                              SQLSMALLINT       BufferLength,
                              SQLSMALLINT       *TextLengthPtr);
```

関数引き数

表 106. 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・タイプ識別子。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情報を求める状況レコード。状況レコードは、1 から数えます。

表 106. 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* バッファーの長さ (バイト単位)。
SQLSMALLINT	*TextLengthPtr*	出力	**MessageText* に戻すために使用できる総バイト数 (ヌル終了文字に必要なバイト数を除く) を戻すバッファーを指すポインター。戻りに使用できるバイト数が *BufferLength* より大きい場合、**MessageText* のエラー・メッセージ・テキストは *BufferLength* からヌル終了文字の長さを引いたもの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が含まれている文字ストリングを戻します。戻りデータの詳細については、512ページの『SQLGetDiagField - 診断データのフィールドの入手』に記載されています。

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走査できます。 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。
SQLGetDiagRec() または SQLGetDiagField() 以外の関数を一時的に呼び出している場合、アプリケーションは、あとでもう一度 SQLGetDiagRec() を呼び出し、レコードからフィールドを検索できます。また、SQLGetDiagField() を呼び出して SQL_DIAG_NUMBER フィールドの値を検索し、その回数分だけ SQLGetDiagRec() を呼び出せば、使用可能な診断レコードの合計数を取り出せます。

診断データ構造のフィールドの説明については、512ページの『SQLGetDiagField - 診断データのフィールドの入手』を参照してください。

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これらのハンドルは、515ページの『ヘッダー・フィールド』、 516ページの『レコード・フィールド』、および SQLGetDescField() の説明に記載されています。

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、 *RecNumber* が正の値の SQL_NO_DATA も戻されます。

例

```
/* From the CLI sample utilcli.c */
/* ... */
    while ( SQLGetDiagRec( htype,
                           hndl,
                           i,
                           sqlstate,
                           &sqlcode,
                           message,
                           SQL_MAX_MESSAGE_LENGTH + 1,
                           &length
                         ) == SQL_SUCCESS ) {
       printf( "¥n  SQLSTATE          = %s¥n", sqlstate ) ;
       printf( "  Native Error Code = %ld¥n", sqlcode ) ;
       printf( "%s¥n", message ) ;
       i++ ;
    }
                              &fileOption, 14, &fileInd);
```

参照

- 512ページの『SQLGetDiagField - 診断データのフィールドの入手』

SQLGetEnvAttr - 現行の環境属性値を検索する

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   *FAR StringLengthPtr);  /* StringLength */
```

関数引き数

表 107. 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、734ページの『環境属性』を参照してください。
SQLPOINTER	ValuePtr	出力	*Attribute* に関連した現行値。戻される値のタイプは、*Attribute* により異なります。
SQLINTEGER	BufferLength	入力	*ValuePtr* が指すバッファーの最大サイズ。属性値は文字ストリングで、それ以外は無視されます。

表 107. SQLGetEnvAttr 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER *	StringLengthPtr	出力	*ValuePtr* に戻すために使用できる総バイト数 (ヌル終了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* がヌル・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* は *BufferLength* からヌル終了文字の長さを引いた長さまで切り捨てられ、 DB2 CLI によりヌル終了させられます。

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。

有効な環境属性のリストは、734ページの『環境属性』を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 108. SQLGetEnvAttr SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振り失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
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。

制約

なし。

例

```
/* From the CLI sample APINFO.C */
/* ... */
    sqlrc = SQLGetEnvAttr( henv, SQL_ATTR_OUTPUT_NTS, &output_nts, 0, NULL ) ;
    ENV_HANDLE_CHECK( henv, sqlrc) ;
```

参照

- 734ページの『SQLSetEnvAttr - 環境属性を設定する』

SQLGetFunctions - 関数の入手

目的

仕様:	DB2 CLI 2.1	ODBC 1.0	ISO CLI

SQLGetFunctions() は、特定の関数がサポートされるかどうかを照会します。これにより、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 *FAR SupportedPtr);     /* pfExists */
```

関数引き数

表 109. 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。有効な *FunctionId* 値は 531ページの図17 に示されています。
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 TRUE または FALSE を戻す場所を指すポインター。

使用法

531ページの図17 は、*FunctionId* 引き数の有効な値と、対応する関数がサポートされているかどうかを示しています。 (このリストは、サンプル・アプリケーション getfuncs.c を使用して生成されました。)

FunctionId が SQL_API_ALL_FUNCTIONS に設定されている場合、*SupportedPtr* は 100 個の要素の SQLSMALLINT 配列を指していなければなりません。この配列は *FunctionId* 値によって指標として使われ、多くの関数を識別するために使用されます。この配列の一部の要素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、関数のリストを入手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 *FunctionId* 値には、 `SQLGetFunction()` 呼び出しを明示的に出すことが必要です。 *FunctionId* 値の完全セットは `sqlcli1.h` に定義されています。

```
------------------------------------------
Connected to Server: SAMPLE
 Database Name: SAMPLE
 Instance Name: db2inst1
     DBMS Name: DB2/6000
  DBMS Version: 05.00.0000
------------------------------------------
SQLALLOCCONNECT      is supported   SQLALLOCENV          is supported
SQLALLOCHANDLE       is supported   SQLALLOCSTMT         is supported
SQLBINDCOL           is supported   SQLBINDFILETOCOL     is supported
SQLBINDFILETOPARAM   is supported   SQLBINDPARAM         is supported
SQLBINDPARAMETER     is supported   SQLBROWSECONNECT     is supported
SQLCANCEL            is supported   SQLCLOSECURSOR       is supported
SQLCOLATTRIBUTE      is supported   SQLCOLATTRIBUTES     is supported
SQLCOLUMNPRIVILEGES  is supported   SQLCOLUMNS           is supported
SQLCONNECT           is supported   SQLCOPYDESC          is supported
SQLDATASOURCES       is supported   SQLDESCRIBECOL       is supported
SQLDESCRIBEPARAM     is supported   SQLDISCONNECT        is supported
SQLDRIVERCONNECT     is supported   SQLENDTRAN           is supported
SQLERROR             is supported   SQLEXECDIRECT        is supported
SQLEXECUTE           is supported   SQLEXTENDEDFETCH     is supported
SQLFETCH             is supported   SQLFETCHSCROLL       is supported
SQLFOREIGNKEYS       is supported   SQLFREECONNECT       is supported
SQLFREEENV           is supported   SQLFREEHANDLE        is supported
SQLFREESTMT          is supported   SQLGETCONNECTATTR    is supported
SQLGETCONNECTOPTION  is supported   SQLGETCURSORNAME     is supported
SQLGETDATA           is supported   SQLGETDESCFIELD      is supported
SQLGETDESCREC        is supported   SQLGETDIAGFIELD      is supported
SQLGETDIAGREC        is supported   SQLGETENVATTR        is supported
SQLGETFUNCTIONS      is supported   SQLGETINFO           is supported
SQLGETLENGTH         is supported   SQLGETPOSITION       is supported
SQLGETSQLCA          is supported   SQLGETSTMTATTR       is supported
SQLGETSTMTOPTION     is supported   SQLGETSUBSTRING      is supported
SQLGETTYPEINFO       is supported   SQLMORERESULTS       is supported
SQLNATIVESQL         is supported   SQLNUMPARAMS         is supported
SQLNUMRESULTCOLS     is supported   SQLPARAMDATA         is supported
SQLPARAMOPTIONS      is supported   SQLPREPARE           is supported
SQLPRIMARYKEYS       is supported   SQLPROCEDURECOLUMNS  is supported
SQLPROCEDURES        is supported   SQLPUTDATA           is supported
SQLROWCOUNT          is supported   SQLSETCOLATTRIBUTES  is supported
SQLSETCONNECTATTR    is supported   SQLSETCONNECTION     is supported
SQLSETCONNECTOPTION  is supported   SQLSETCURSORNAME     is supported
SQLSETDESCFIELD      is supported   SQLSETDESCREC        is supported
SQLSETENVATTR        is supported   SQLSETPARAM          is supported
SQLSETPOS            is supported   SQLSETSCROLLOPTIONS  is supported
SQLSETSTMTATTR       is supported   SQLSETSTMTOPTION     is supported
SQLSPECIALCOLUMNS    is supported   SQLSTATISTICS        is supported
SQLTABLEPRIVILEGES   is supported   SQLTABLES            is supported
SQLTRANSACT          is supported
```

図 17. サポートされる関数のリスト (getfuncs.c からの出力)

注: バージョン 2.1 以前または LOB データ・タイプをサポートしない他の IBM RDBMS では、 LOB サポート関数 (SQLGetLength()、

SQLGetPosition()、 SQLGetSubString()、SQLBindFileToCol()、SQLBindFileToCol()) は、DB2 共通サーバー用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 110. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数の順序エラーです。	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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。

許可

なし。

例

以下の例では、531ページの図17 に記述されている、任意のデータ・ソースに関するリストを生成します。

```
/* From the CLI sample ILINFO.C */
/* ... */
    /* check to see if SQLGetInfo() is supported */
    sqlrc = SQLGetFunctions(hdbc, SQL_API_SQLGETINFO, &supported);
    HANDLE_CHECK( SQL_HANDLE_DBC, hdbc, sqlrc, &henv, &hdbc ) ;
```

参照

なし。

SQLGetInfo - 一般情報の入手

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されている DBMS に関する一般情報 (サポートされるデータ変換も含む) を戻します。

構文

```
SQLRETURN   SQLGetInfo       (
                SQLHDBC           ConnectionHandle,  /* hdbc */
                SQLUSMALLINT      InfoType,          /* fInfoType */
                SQLPOINTER        InfoValuePtr,      /* rgbInfoValue */
                SQLSMALLINT       BufferLength,      /* cbInfoValueMax */
                SQLSMALLINT  *FAR StringLengthPtr);  /* pcbInfoValue */
```

関数引き数

表 111. 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
SQLUSMALLINT	*InfoType*	入力	ご希望の情報タイプ。引き数は、32ページの『データ・タイプとデータ変換』にある表の最初の列の値のうちの 1 つでなければなりません。
SQLPOINTER	*InfoValuePtr*	出力 (または入力)	この関数がご希望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 2 進数値 • 32 ビット・マスク • NULL 終了文字ストリング
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。

表 111. SQLGetInfo 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT *	*StringLengthPtr*	出力	希望の情報を戻すことができるバイトの総数をこの関数が戻す場所を指すポインター。ストリング出力の場合、このサイズには NULL 終了文字が含まれません。 *StringLengthPtr* が指す場所の値が *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、 SQLGetInfo() がその値に戻す情報に関する説明については、 『SQLGetInfo によって戻される情報』を参照してください。

DB2 CLI バージョン 5 の情報タイプの数は、名前変更されました。リストについては、826ページの『SQLGetInfo() の InfoTypes の変更』を参照してください。 『SQLGetInfo によって戻される情報』のリストには、古い値と新しい値が両方記載されています。

SQLGetInfo によって戻される情報

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。

- 'Y' または 'N' を含む文字ストリングの場合は、″N″ が返されます。
- 'Y' または 'N' 以外の値だけを含む文字ストリングの場合は、空ストリングが返されます。
- 16 ビット整数の場合は、0 (ゼロ) が返されます。
- 32 ビット整数の場合は、0 (ゼロ) が返されます。
- 32 ビット・マスクの場合は、0 (ゼロ) が返されます。

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、ユーザーが、関数 SQLTables() で返されるすべての表に対する SELECT 特権を保証されることを示します。 "N" は、アクセスできない表が返されている可能性があることを示します。

SQL_ACTIVE_ENVIRONMENTS (16 ビット整数)
この *InfoType* は、SQL_MAX_CONCURRENT_ACTIVITIES に置き換えられています。

DB2 CLI ドライバーがサポートできる活動環境の最大数。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。

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_ACTIVE_CONNECTIONS (16 ビット整数)
この *InfoType* は、SQL_MAX_DRIVER_CONNECTIONS に置き換えられています。

アプリケーションごとにサポートされている活動接続の最大数。

ゼロが返された場合は、限度はシステム資源に従属していることを示しています。

db2cli.ini 初期設定ファイルの MAXCONN キーワードまたは SQL_ATTR_MAX_CONNECTIONS 環境 / 接続オプションを使用して、接続数の限界を定めることができます。限度がゼロ以外の値に設定されていると、その限界が返されます。

SQL_ACTIVE_STATEMENTS (16 ビット整数)

この *InfoType* は、SQL_MAX_CONCURRENT_ACTIVITIES に置き換えられています。

接続ごとの活動ステートメントの最大数。

ゼロが返された場合は、限度はデータベース・システムおよび DB2 CLI の資源と限度に従属していることを示しています。

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_ASYNC_MODE (32 ビット無符号整数)

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。詳しくは 145ページの『CLI の非同期実行』を参照してください。

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、ブックマークを有効にしておくべき時を示します。

- SQL_BP_CLOSE。ブックマークが有効なのは、アプリケーションが SQL_CLOSE オプション付きの `SQLFreeStmt()`、 `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_BP_DROP が指定されていれば、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に続く修飾名要素の間の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)
データベース・ベンダーの修飾子用の用語

ベンダーが、3 部分から成る名前の高位の部分に使用する名前。

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 SQL_OWNER_USAGE と同様ですが、カタログ用に使われます。

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、このサーバーの省略時文字セットにおける省略時の照合名 (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 RDBM 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

(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_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 COLATION 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。列の省略時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_CLOSE_BEHAVIOR (32 ビット無符号整数)

カーソルがクローズされる時にロックが解放されるかどうかを示します。有効値は以下のとおりです。

- SQL_CC_NO_RELEASE: このステートメント・ハンドルのカーソルがクローズされても、ロックは解放されません。これが省略時です。

- SQL_CC_RELEASE: このステートメント・ハンドルのカーソルがクローズされると、ロックは解放されます。

一般に、 SQL_CLOSE オプションを指定して関数 SQLFreeStmt() を呼び出したり、 *HandleType* を SQL_HANDLE_STMT に設定して SQLFreeHandle() を呼び出したりすると、カーソルは明示的にクローズされます。また、(コミットまたはロールバックを出して) トランザクションが終了しても、カーソルがクローズされる場合があります (現在使用中の WITH HOLD 属性に基づく)。

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・プランが除去されます。
- 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・プランが除去されます。
- 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。ステートメント・ハンドル上のカーソルが、同一のトランザクション内の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、 tt では指定しません。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)

SQLConnect() への入力に関するデータ・ソースとして使用される名前。または、SQLDriverConnect() 接続ストリング中の DSN キーワード値。

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。

SQL_DATABASE_NAME (ストリング)

現在使用中のデータベースの名前。

注: IBM DBMS の SELECT CURRENT SERVER によっても返されます。

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、以下のとおりです。
- ″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省略時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、および単独読み取りが可能です)。

  これは IBM の UR レベルと等価です。
- 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単独読み取りが可能です)。

  これは IBM の CS レベルと等価です。

- 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保留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 (反復可能読み取りと単独読み取りが可能です)。

  これは IBM の RS レベルと等価です。
- SQL_TXN_SERIALIZABLE = 保留中のトランザクションによる影響を受けたデータは他のトランザクションで使用できません (反復可能読み取りはできますが、単独読み取りはできません)。

  これは IBM の RR レベルと等価です。
- 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 ユニバーサル・データベース (AS/400 版) 分離レベルです。

IBM の用語では、
- SQL_TXN_READ_UNCOMMITTED は、未確約読み取りです。
- 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 ドライバー・マネージャーのある 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 ドライバー・マネージャーは、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_DROP_SCHEMA
- 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_DROP_VIEW
- 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。

次のビット・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の前にあるアンバインド列について 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合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合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 = 識別子名は大文字でシステム・カタログに保管されます。
- SQL_IC_LOWER = 識別子名は小文字でシステム・カタログに保管されます。
- SQL_IC_SENSITIVE = 識別子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 = 識別子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 の識別子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)

区切り識別子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)

サポートされる、INFORMATIONAL_SCHEMA 中の視点を示します。DB2 CLI は常にゼロを返し、INFORMATIONAL_SCHEMA 中の視点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)
これは DBMS で、予約語に関する ODBC のリストにないすべてのキーワード のストリングです。

SQL_LIKE_ESCAPE_CLAUSE (ストリング)
LIKE 述部のメタキャラクター・パーセントと下線について、エスケープ文字がサポートされているかどうかを示す文字ストリング。

SQL_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活動状態の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 ステートメントの 16 進数リテラルの最大長を指定する、 32 ビット無符号整数値。

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)。

SQL_MAX_COLUMN_NAME_LEN (16 ビット整数)
列名の最大長 (バイト単位)。

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活動環境の最大数。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活動接続の最大数。

ゼロが返された場合は、限度はシステム資源に従属していることを示しています。

`db2cli.ini` 初期設定ファイルの MAXCONN キーワードまたは SQL_ATTR_MAX_CONNECTIONS 環境 / 接続オプションを使用して、接続数の限界を定めることができます。限度がゼロ以外の値に設定されていると、その限界が返されます。

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_OWNER_NAME_LEN (16 ビット整数)
この *fInfoType* は、SQL_MAX_SCHEMA_NAME_LEN に置き換えられています。

スキーマ修飾子名の最大長 (バイト単位)。

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_QUALIFIER_NAME_LEN (16 ビット整数)
この *fInfoType* は、SQL_MAX_CATALOG_NAME_LEN に置き換えられています。

カタログ修飾子名の最大長。 3 つの部分から成る表名の最初の部分 (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特定の長ストリング・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活動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 つの接続だけが活動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、調整済み分散作業単位 (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)
ヌル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 として定義できません。
- SQL_NNC_NULL。列を NOT NULL として定義できます。

SQL_NULL_COLLATION (16 ビット整数)
リスト中で NULL がソートされるかどうかを示します。
- SQL_NC_HIGH。ヌル値は高位にソートされます。
- SQL_NC_LOW。ヌル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 スカラー数値関数を示します。この関数は、 151ページの『ベンダー・エスケープ文節の使用』に記述されている ODBC ベンダー・エスケープ・シーケンスとともに使用するよう意図されています。

次のビット・マスクを用いて、どの数値関数がサポートされているかを示します。
- 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要素、 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、スクロール可能カーソル、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、 1 次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_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、付与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_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。

上記の 3 タイプの ODBC SQL 文法の定義については、 *ODBC 3.0 Software Development Kit and Programmer's Reference* を参照してください。

SQL_ODBC_SQL_OPT_IEF (ストリング)

この *InfoType* は、SQL_INTEGRITY に置き換えられています。

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。

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 : 外部結合 ON 文節に述部を使用できます。このビットを設定しないと、ON 文節中で有効な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。

(151ページの『ベンダー・エスケープ文節の使用』を参照してください。)

SQL_OWNER_TERM (ストリング)

この *InfoType* は、SQL_SCHEMA_TERM に置き換えられています。

データベース・ベンダーのスキーマ (所有者) 用の用語。

SQL_OWNER_USAGE (32 ビット・マスク)

この *InfoType* は、SQL_SCHEMA_USAGE に置き換えられています。

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_OU_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_OU_PROCEDURE_INVOCATION - プロシージャー呼び出しステートメントでサポートされています。
- SQL_OU_TABLE_DEFINITION - すべての表定義ステートメントでサポートされています。
- SQL_OU_INDEX_DEFINITION - すべての索引定義ステートメントでサポートされています。
- SQL_OU_PRIVILEGE_DEFINITION - すべての特権定義ステートメント (つまり、付与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。

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使用可能度を示します。

- 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使用可能度を示します。

- 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、 DB2 CLI が 131ページの『ストアード・プロシージャーの使用』で指定されている ODBC プロシージャー呼び出し構文を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_QUALIFIER_LOCATION (16 ビット整数)
この *InfoType* は、 SQL_CATALOG_LOCATION に置き換えられています。

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_QL_START が返されます。

SQL_QUALIFIER_NAME_SEPARATOR (ストリング)
カタログ名とその後に続く修飾名要素の間の区切り記号として使用される文字。

この *InfoType* は、SQL_CATALOG_NAME_SEPARATOR に置き換えられています。

SQL_QUALIFIER_TERM (ストリング)
データベース・ベンダーの修飾子用の用語

ベンダーが、3 部分から成る名前の高位の部分に使用する名前。

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。

この *InfoType* は、SQL_CATALOG_TERM に置き換えられています。

SQL_QUALIFIER_USAGE (32 ビット・マスク)
この *fInfoType* は、SQL_CATALOG_USAGE に置き換えられています。

これは SQL_OWNER_USAGE と同様ですが、カタログ用に使われます。

SQL_QUOTED_IDENTIFIER_CASE (16 ビット整数)
以下の値を返します。

- SQL_IC_UPPER - SQL 中の引用符付き識別子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 中の引用符付き識別子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 中の引用符付き識別子 (区切り識別子) 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 中の引用符付き識別子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) 識別子がシステム・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″ は、同一列を複数回取り出した際、その間に変更が検出されたことを示し、″N″ は、変更が検出されなかったことを示します。

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)

スクロール可能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
- SQL_SO_KEYSET_DRIVEN
- SQL_SO_STATIC
- SQL_SO_DYNAMIC
- SQL_SO_MIXED

スクロール可能カーソルについて詳しくは、75ページの『スクロール可能カーソル』を参照してください。

SQL_SEARCH_PATTERN_ESCAPE (ストリング)

カタログ関数 (SQLTables() や SQLColumns() など) 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)

a〜z、A〜Z、0〜9、および _ に加えて、区切り識別子以外に使えるすべての文字が含まれています。

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。エントリー・レベルの SQL-92 に準拠。
- SQL_SC_FIPS127_2_TRANSITIONAL。 FIPS 127-2 変換レベルに準拠。

- 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_SQL92_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 - すべての述部
- 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間隔を示します。

以下のビット・マスクを用いて、どの間隔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間隔を示します。

以下のビット・マスクを用いて、どの間隔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識別子です。

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。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 112. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY090	ストリングまたはバッファー長が無効です。	*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	ドライバーが機能していません。	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。

制約

なし。

例

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

```
sqlrc = SQLGetInfo(hdbc, SQL_DBMS_VER, imageInfoBuf, 255, &outlen);
HANDLE_CHECK( SQL_HANDLE_DBC, hdbc, sqlrc, &henv, &hdbc ) ;
printf("            Remote DBMS Version: %s¥n", imageInfoBuf);
```

参照

- 603ページの『SQLGetTypeInfo - データ・タイプ情報の入手』

SQLGetLength - ストリング値の長さを取り出す

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   FAR  *StringLength,
                              SQLINTEGER   FAR  *IndicatorValue);
```

関数引き数

表 113. 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られているが、現時点で割り当てられたステートメントがまだ準備されていない任意のステートメント・ハンドルとすることができます。
SQLSMALLINT	LocatorCType	入力	C タイプのソース LOB ロケーター。以下のいずれかになります。 • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	Locator	入力	LOB ロケーター値に設定する必要があります。
SQLINTEGER *	StringLength	出力	ターゲットの C バッファー・タイプが 2 進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のバイト数[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。

表 113. SQLGetLength 引き数 (続き)

データ・タイプ	引き数	使用法	説明

注:

a これは、DBCLOB データの場合であってもバイト数です。

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。

ロケーター引き数には、FREE LOCATOR ステートメントを用いて明示的に解放されてはいない、またはロケーターが作成されたトランザクションが終了したために暗黙的に解放されていない有効な LOB ロケーターを入れることができます。

このステートメント・ハンドルは、作成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 114. SQLGetLength SQLSTATE

SQLSTATE	説明	解説
07006	変換が無効です。	*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。	
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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。

表 114. SQLGetLength SQLSTATE (続き)

SQLSTATE	説明	解説
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 のいずれかではありません。
HY009	引き数値が無効です。	*StringLength* を指すポインターが NULL でした。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HYC00	ドライバーが機能していません。	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。

制約

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

```
/* From the CLI sample dtlob.c */
/* ... */
    /* get the length of the whole CLOB data */
    sqlrc = SQLGetLength( hstmtLocUse, SQL_C_CLOB_LOCATOR,
                          clobLoc, &clobLen, &ind ) ;
    STMT_HANDLE_CHECK( hstmtLocUse, sqlrc);
```

参照

- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』

- 446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』
- 434ページの『SQLFetch - 次の行の取り出し』
- 582ページの『SQLGetPosition - ストリングの開始位置を戻す』
- 598ページの『SQLGetSubString - ストリング値の部分を取り出す』

SQLGetPosition - 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      FAR  *SearchLiteral,
                              SQLINTEGER        SearchLiteralLength,
                              SQLUINTEGER       FromPosition,
                              SQLUINTEGER  FAR  *LocatedAt,
                              SQLINTEGER   FAR  *IndicatorValue);
```

関数引き数

表 115. 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られているが、現時点で割り当てられたステートメントがまだ準備されていない任意のステートメント・ハンドルとすることができます。
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。

表 115. SQLGetPosition 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER	SearchLocator	入力	*SearchLiteral* ポインターが NULL であるか、*SearchLiteralLength* を 0 に設定する場合、*SearchLocator* は探索ストリングに関連付けられた LOB ロケーターに設定しなければなりません。そうしないと、この引き数は無視されます。
SQLCHAR *	SearchLiteral	入力	この引き数は、探索ストリング・リテラルを含む記憶域を指します。 *SearchLiteralLength* が 0 であれば、このポインターは NULL でなければなりません。
SQLINTEGER	SearchLiteralLength	入力	*SearchLiteral* 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、この値が関数で戻されます。 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 *	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、ストリングの任意の部分をランダムな方法で取得するために、 `SQLGetSubString()` と、一緒に使用します。 `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場所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 と *SearchLocator* (使用されている場合) 引き数には、 FREE LOCATOR ステートメントを使用して明示的に解放されていない、または LOB ロケーターが作成されたトランザクションが終了したために暗黙的に解放されていない有効な LOB ロケーターが入ります。

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、作成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 116. SQLGetPosition SQLSTATE

SQLSTATE	説明	解説
07006	変換が無効です。	*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。	パターンの長さが 4000 バイトを超えています。
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。
HY001	メモリーの割り振り失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。

表 116. SQLGetPosition SQLSTATE (続き)

SQLSTATE	説明	解説
HY009	引き数値が無効です。	*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 のいずれかではありません。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY090	ストリングまたはバッファー長が無効です。	*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	ドライバーが機能していません。	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。

制約

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

```
/* From the CLI sample dtlob.c */
/* ... */
    /* get the starting postion of the CLOB piece of data */
    sqlrc = SQLGetPosition( hstmtLocUse,
                            SQL_C_CLOB_LOCATOR,
```

```
                                clobLoc,
                                0,
                                ( SQLCHAR * ) "Interests",
                                strlen( "Interests"),
                                1,
                                &clobPiecePos,
                                &ind ) ;
```

参照

- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 418ページの『SQLExtendedFetch - 拡張取り出し (行の配列の取り出し)』
- 434ページの『SQLFetch - 次の行の取り出し』
- 578ページの『SQLGetLength - ストリング値の長さを取り出す』
- 598ページの『SQLGetSubString - ストリング値の部分を取り出す』

SQLGetSQLCA - SQLCA データ構造を入手する

目的

仕様	DB2 CLI 2.1		

SQLGetSQLCA() は、SQL ステートメントの作成と実行に関連する SQLCA や、データの取り出し、カーソルのクローズに関連する SQLCA を戻すときに使用します。 SQLCA は、情報のほかに、 SQLError() によって使用可能になるものを戻すこともあります。

注: SQLGetSQLCA() を、 SQLGetDiagField() と SQLGetDiagRec() の代わりに使用しないでください。

SQLCA 構造の詳細な説明については、 *SQL 解説書* の SQLCA の付録を参照してください。

ステートメント・ハンドルの割り振りなど、アプリケーション側で関数を厳密に処理する場合は、SQLCA は使用できません。この場合、すべての値をゼロに設定した状態で空 SQLCA が戻されます。

構文

```
SQLRETURN   SQLGetSQLCA      (SQLHENV           EnvironmentHandle, /* henv */
                              SQLHDBC           ConnectionHandle,  /* hdbc */
                              SQLHSTMT          StatementHandle,   /* hstmt */
                              struct sqlca FAR  *SqlcaPtr);        /* pSqlca */
```

関数引き数

表 117. SQLGetSQLCA 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。環境に関連した SQLCA を取得するには、有効な環境ハンドルを渡します。 *ConnectionHandle* を SQL_NULL_HDBC に設定し、 *StatementHandle* を SQL_NULL_HSTMT に設定してください。
SQLHDBC	ConnectionHandle	入力	接続ハンドル。接続に関連した SQLCA を取得するには、有効なデータベース接続ハンドルを渡し、 *StatementHandle* を SQL_NULL_HSTMT に設定します。 *EnvironmentHandle* 引き数は無視されます。

表 117. SQLGetSQLCA 引き数 (続き)

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。ステートメントに関連した SQLCA を取得するには、有効なステートメント・ハンドルを渡します。 *EnvironmentHandle* 引き数と *ConnectionHandle* 引き数は、無視されます。
SQLCA *	sqlCA	出力	SQL 連絡域。

使用法

ハンドルは、SQLError() 関数の場合と同じ方法で使用します。以下のものに関連付けられている SQLCA を取得するには、それぞれ次のようにします。

- 環境の場合は、有効な環境ハンドルを渡します。 *ConnectionHandle* を SQL_NULL_HDBC に設定し、 *StatementHandle* を SQL_NULL_HSTMT に設定してください。
- 接続の場合は、有効なデータベース接続ハンドルを渡し、 *StatementHandle* を SQL_NULL_HSTMT に設定します。 *EnvironmentHandle* 引き数は無視されます。
- ステートメントの場合は、有効なステートメント・ハンドルを渡してください。 *EnvironmentHandle* 引き数と *ConnectionHandle* 引き数は、無視されます。

ある DB2 CLI 関数で生成された診断情報が、同じハンドルを指定して SQLError() 以外の関数を呼び出す前に取り出されない場合、直前の関数呼び出しの情報が失われます。このことは、 2 番目の DB2 CLI 関数呼び出しに関する診断情報が生成されるかどうかにかかわらず当てはまります。

DBMS との対話にならない DB2 CLI 関数が呼び出される場合、 SQLCA にはすべてゼロが含まれます。通常、以下の関数には、有用な情報が戻されます。

- SQLBrowseConnect()
- SQLCancel(),
- SQLCloseCursor()
- SQLColAttribute()
- SQLColumnPrivileges()
- SQLColumns()
- SQLConnect()、SQLDisconnect()
- SQLCopyDesc()
- SQLDataSources()

- SQLDescribeCol()
- SQLDescribeParam()
- SQLEndTran()
- SQLExecDirect()、SQLExecute()
- SQLFetch()
- SQLFetchScroll()
- SQLForeignKeys()
- SQLFreeHandle()
- SQLGetData() (LOB 列が含まれている場合)
- SQLMoreResults()
- SQLPrepare() (** 下記の注を参照)
- SQLPrimaryKeys()
- SQLProcedureColumns()
- SQLProcedures()
- SQLRowCount()
- SQLSetConnectAttr() (SQL_ATTR_AUTOCOMMIT、および SQL_ATTR_DB2EXPLAIN の場合)
- SQLStatistics()
- SQLTables()
- SQLTablePrivileges()
- SQLTransact()

DB2 ユニバーサル・データベース バージョン 2 以降のサーバーに接続される場合、 SQLCA には、特に関心を向けるべき次の 2 つのフィールドがあります。

- SQLERRD(3) フィールド (たとえば、sqlca.errd[2]):
  - PREPARE (** 下記の注を参照) の後で、ステートメントを実行すると、ユーザーに戻される行数の見積もりが入ります。アプリケーションは、適切な照会が出されたかどうかを評価するのに役立つよう、この情報をユーザーに知らせることができます。
  - INSERT、DELETE、UPDATE の後では、影響を受けた実際の行数が入れられます。
  - 複合 SQL 処理の後では、INSERT、UPDATE、または DELETE ステートメントによる影響を受けたすべてのサブステートメントの累計が入ります。
- SQLERRD(4) フィールド (たとえば、sqlca.errd[3]):
  - PREPARE (** 下記の注を参照) の後では、ステートメントを処理するのに必要な資源の相対コスト見積もりが入ります。

この値は、174ページの『構成キーワード』で説明した DB2ESTIMATE 構成キーワード、および 663ページの『SQLSetConnectAttr - 接続属性を設定する』の関数説明で説明する SQL_ATTR_DB2ESTIMATE 接続属性と比較される数値です。

– 複合 SQL の処理の後では、成功したサブステートメント数のカウントが入れられます。

注: SQLERRD(3) および SQLERRD(4) フィールドに戻される情報の正確度は、パラメーター・マーカーの使用のようなさまざまな要素、およびステートメント内のさまざまな式によって異なります。その主な要素のうちで制御可能なものは、データベース統計の正確度です。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 ユニバーサル・データベースでは、RUNSTATS コマンドの最終実行時にします。)

DB2 CLI バージョン 5 では、デフォルトで据え置き準備がオンになります。対応する実行要求が発行されるまで、PREPARE 要求はサーバーに送られません。つまり、PREPARE に関連する SQLERRD(3) フィールドおよび SQLERRD(4) フィールドのデータは、 SQLPrepare() が呼び出された後、意味を持たなくなります。詳細は 827ページの『デフォルトでの据え置き準備』を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

なし。

制約

なし。

例

```
/* From the CLI sample APSQLCA.C */
/* ... */
    /* call SQLGetSQLCA */
    printf("    Call SQLGetSQLCA and print some SQLCA fields.¥n");
    sqlrc = SQLGetSQLCA( henv, hdbc, hstmt, &sqlca ) ;
    HANDLE_CHECK( SQL_HANDLE_STMT, hstmt, sqlrc, &henv, &hdbc ) ;
```

参照

- 522ページの『SQLGetDiagRec - 診断レコードの複数のフィールド設定を取得する』

SQLGetStmtAttr - ステートメント属性の現行設定値を入手する

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。

構文

```
SQLRETURN   SQLGetStmtAttr  (SQLHSTMT         StatementHandle,
                             SQLINTEGER       Attribute,
                             SQLPOINTER       ValuePtr,
                             SQLINTEGER       BufferLength,
                             SQLINTEGER       *StringLengthPtr);
```

関数引き数

表 118. 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。

表 118. SQLGetStmtAttr 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER	BufferLength	入力	*Attribute* が ODBC 定義の属性で、 *ValuePtr* が文字ストリングか 2 進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* には次の値が有効です。 • **ValuePtr* が文字ストリングを指すポインターの場合、 *BufferLength* はストリングまたは SQL_NTS の長さです。 • **ValuePtr* が 2 進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。 *BufferLength* には負の値が入れられます。 • **ValuePtr* が文字ストリングまたは 2 進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。
SQLSMALLINT	*StringLengthPtr	出力	**ValuePtr* に戻すために使用できる総バイト数 (ヌル終了文字を除く) を戻すバッファーを指すポインター。このポインターがヌル・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* は *BufferLength* からヌル終了文字の長さを引いた長さまで切り捨てられ、 DB2 CLI がヌル文字で終了させます。

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 ビット値またはヌル文字で終了する文字ストリングのどちらかです。この値がヌル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 **StringLengthPtrPtr* 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。

SQLGetStmtAttr() を呼び出す DB2 CLI バージョン 5.2 アプリケーションを DB2 CLI バージョン 2 と一緒に使用して作業できるようにするため、 SQLGetStmtAttr() への呼び出しが SQLGetStmtOption() にマップされます。

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属性のリストについては、 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』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 119. SQLGetStmtAttr SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージです。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
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* からヌル終了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 を戻します。)
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たか、カーソルが結果セットの開始点の前または終了点の後ろに配置されていました。

表 119. SQLGetStmtAttr SQLSTATE (続き)

SQLSTATE	説明	解説
HY000	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY010	関数の順序エラーです。	*StatementHandle* の非同期実行関数が呼び出され、この関数が呼び出された時点でまだ実行中でした。 *StatementHandle* のために SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が戻されました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY090	ストリングまたはバッファー長が無効です。	*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、このバージョンの DB2 CLI では無効でした。
HY109	カーソル位置が無効です。	*Attribute* 引き数は SQL_ATTR_ROW_NUMBER で、行は削除されたか、フェッチできませんでした。
HYC00	ドライバーが機能していません。	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。

制約

なし。

例

```
/* From the CLI sample DBINFO.C */
/* ... */
    sqlrc = SQLGetStmtAttr( hstmt, SQL_CURSOR_HOLD, &cursor_hold, 0, NULL ) ;
    HANDLE_CHECK( SQL_HANDLE_STMT, hstmt, sqlrc, &henv, &hdbc ) ;
    printf("    A statement attribute...¥n");
    printf( "        Cursor With Hold is: " ) ;
    if ( cursor_hold == SQL_CURSOR_HOLD_ON ) printf( "ON¥n" ) ;
    else printf( "OFF¥n" ) ;
```

参照

- 480ページの『SQLGetConnectAttr - 現行属性設定を入手する』
- 663ページの『SQLSetConnectAttr - 接続属性を設定する』
- 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』

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

使用すべきでない関数

注:

ODBC バージョン 3 では、 SQLGetStmtOption() は使用すべきでない関数であり、 SQLGetStmtAttr() に置き換えられました。詳細については、592ページの『SQLGetStmtAttr - ステートメント属性の現行設定値を入手する』を参照してください。

このバージョンの DB2 CLI では引き続き SQLGetStmtOption() をサポートしていますが、 DB2 CLI プログラムでは SQLGetStmtAttr() を使い始め、最新の規格に従うようお勧めします。 上記の関数と、その他の使用すべきでない関数の詳細については、
822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

SQLGetSubString - ストリング値の部分を取り出す

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   FAR  *StringLength,
                SQLINTEGER   FAR  *IndicatorValue);
```

関数引き数

表 120. 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られているが、現時点で割り当てられたステートメントがまだ準備されていない任意のステートメント・ハンドルとすることができます。
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は番号 1 が付けられます。

表 120. SQLGetSubString 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLUINTEGER	ForLength	入力	これは、関数から戻されるストリングの長さです。 BLOB と CLOB の場合、これはバイト単位の長さです。 DBCLOB の場合、これは文字単位の長さです。 *FromPosition* がソース・ストリングの長さより小さい値で、 *FromPosition* + *ForLength* - 1 がソース・ストリングの終わりを超えている場合、その結果は必要な文字数で右側を埋め込まれます (BLOB の場合は X'00'、 CLOB の場合は単一バイト・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、LOB ロケーター C バッファー・タイプ (SQL_C_CLOB_LOCATOR、SQL_C_BLOB_LOCATOR、SQL_C_DBCLOB_LOCATOR) または C ストリング変数 (CLOB の場合は SQL_C_CHAR、BLOB の場合は SQL_C_BINARY、 DBCLOB の場合は SQL_C_DBCHAR) のどちらかでなければなりません。
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。
SQLINTEGER *	StringLength	出力	ターゲットの C バッファー・タイプが 2 進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 (バイト単位)[a]。 ポインターを 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 の代わりとして使用し、データを分割して入手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。

ロケーター引き数には、FREE LOCATOR ステートメントを用いて明示的に解放されてはいない、またはロケーターが作成されたトランザクションが終了したために暗黙的に解放されていない有効な LOB ロケーターを入れることができます。

このステートメント・ハンドルは、作成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 121. 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。戻りに使用できる実際の長さは、*StringLength* に保管されます。
07006	変換が無効です。	*TargetCType* に指定された値は、SQL_C_CHAR、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)。

表 121. SQLGetSubString SQLSTATE (続き)

SQLSTATE	説明	解説
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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
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	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY090	ストリングまたはバッファー長が無効です。	*BufferLength* の値は、0 より小さい値でした。
HYC00	ドライバーが機能していません。	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。

制約

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

```
/* From the CLI sample dtlob.c */
/* ... */
    /* read the piece of CLOB data in buffer */
    sqlrc = SQLGetSubString( hstmtLocUse,
                             SQL_C_CLOB_LOCATOR,
                             clobLoc,
                             clobPiecePos,
                             clobLen - clobPiecePos,
                             SQL_C_CHAR,
                             buffer,
                             clobLen - clobPiecePos + 1,
                             &clobPieceLen,
                             &ind ) ;
    STMT_HANDLE_CHECK( hstmtLocUse, sqlrc);
```

参照

- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』
- 434ページの『SQLFetch - 次の行の取り出し』
- 488ページの『SQLGetData - 列からのデータの入手』
- 578ページの『SQLGetLength - ストリング値の長さを取り出す』
- 582ページの『SQLGetPosition - ストリングの開始位置を戻す』

SQLGetTypeInfo - データ・タイプ情報の入手

目的

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,
                              SQLSMALLINT       DataType);
```

関数引き数

表 122. 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。

表 122. SQLGetTypeInfo 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*DataType*	入力	照会される SQL データ・タイプ。サポートされるタイプは次のとおりです。 • SQL_ALL_TYPES • SQL_BIGINT • SQL_BINARY • SQL_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_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 ごとに降順で戻されます。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_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)
SQL データ・タイプ名 (たとえば、VARCHAR、BLOB、DATE、INTEGER) の文字表示。

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 2 進ストリングの場合、この列には最大長がバイト単位で入ります。グラフィック (DBCS) ストリングの場合は、これはその列の 2 バイト文字の数になります。

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不能である場合には、データ・タイプのこの列はヌルです。

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不能である場合には、データ・タイプのこの列はヌルです。

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 構文での使用が要求されてい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、ヌル標識が戻されます。

注: CREATE_PARAM 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ルに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・タイプがヌル値を受け入れるかどうかを指示します。
- ヌル値を使用できない場合は SQL_NO_NULLS に設定します。
- ヌル値を使用できる場合は SQL_NULLABLE に設定します。

列 8 CASE_SENSITIVE (SMALLINT NOT NULL データ・タイプ)
データ・タイプを照合するときに大文字小文字を区別するかどうかを示します。有効な値は TRUE と 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・タイプが無符号である場所を示します。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、ヌル標識が戻されます。

列 11 FIXED_PREC_SCALE (SMALLINT NOT NULL データ・タイプ)
データ・タイプが正確な数値であり、かつ常に同じ精度と位取りである場合には、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固有の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省略時文字セットです。

列 14 MINIMUM_SCALE (INTEGER データ・タイプ)
SQL データ・タイプの最小位取り。データ・タイプが固定位取りの場合、MINIMUM_SCALE 列と MAXIMUM_SCALE 列には同じ値が入れられます。位取りが適用できないと、NULL が戻されます。

列 15 MAXIMUM_SCALE (INTEGER データ・タイプ)
SQL データ・タイプの最大位取り。位取りが適用できないと、NULL が戻されます。最大位取りが 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 10 進数値を 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 123. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY004	SQL のデータ・タイプが範囲外です。	指定された *DataType* は無効です。
HY010	関数の順序エラーです。	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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

次の ODBC 指定の SQL データ・タイプ (および対応する *DataType* 定義値) は、 IBM RDBMS ではサポートされません。

データ・タイプ	DataType
TINY INT	SQL_TINYINT
BIG INT	SQL_BIGINT
BIT	SQL_BIT

例

```
/* From the CLI sample DTINFO.C */
/* ... */
    /* call SQLTables */
    printf("¥n    Call SQLGetTypeInfo.¥n");
    sqlrc = SQLGetTypeInfo( hstmt, SQL_ALL_TYPES);
    STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参照

- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 662ページの『SQLSetColAttributes - 列属性を設定する』
- 446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』
- 533ページの『SQLGetInfo - 一般情報の入手』

SQLMoreResults - さらに結果セットがあるかどうかを判別する

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、または
- 結果セットを戻しているストアード・プロシージャー。

構文

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

関数引き数

表 124. 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。

- SQLParamOptions() または SQLBindParameter() に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 照会を含むストアード・プロシージャー。この照会のカーソルはオープンされているので、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することができます。

詳細については、91ページの『配列の使用によるパラメーター値の入力』と136ページの『ストアード・プロシージャーから結果セットを返す』を参照してください。

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。

SQL_CLOSE オプションを指定して SQLFreeStmt() が呼び出された場合、または *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保留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 125. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数の順序エラーです。	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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() 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。

制約

ODBC 仕様の SQLMoreResults() も、入力パラメーター値の配列が指定されたパラメーター化 INSERT、UPDATE、および DELETE ステートメントの実行に関連付けられたカウントを戻すことができます。しかし、DB2 CLI はこのようにカウント情報を戻すことをサポートしていません。

例

```
/* From the CLI sample PCALL.C */
/* ... */
    /* print result sets, if any */
    do
    {   rc = StmtResultPrint( hstmt1);
    }
    while( SQLMoreResults( hstmt1) == SQL_SUCCESS);
```

参照

- 626ページの『SQLParamOptions - パラメーターに入力配列を指定する』

SQLNativeSql - ネイティブの SQL テキストを入手する

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・エスケープ文節順序列が含まれていた場合、 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。)

構文

```
SQLRETURN   SQLNativeSql     (
                SQLHDBC           ConnectionHandle,  /* hdbc */
                SQLCHAR      FAR  *InStatementText,  /* szSqlStrIn */
                SQLINTEGER        TextLength1,       /* cbSqlStrIn */
                SQLCHAR      FAR  *OutStatementText, /* szSqlStr */
                SQLINTEGER        BufferLength,      /* cbSqlStrMax */
                SQLINTEGER   FAR  *TextLength2Ptr);  /* pcbSqlStr */
```

関数引き数

表 126. 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* の長さ。
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* が指すバッファーのサイズ。
SQLINTEGER *	TextLength2Ptr	出力	*OutStatementText* 内に戻すために使用できる総バイト数 (ヌル終止符を除く)。戻りに使用できるバイト数が *BufferLength* 以上の場合は、*OutStatementText* 内の出力 SQL ストリングが *BufferLength - 1* バイトに切り捨てられます。

使用法

この関数は、アプリケーションが DB2 CLI によってデータ・ソースに渡される変換済み SQL ストリングを検査または表示したいときに呼び出します。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・エスケープ文節順序列が含まれているときしか行われません。ベンダー・エスケープ文節順序列の詳細については、 151ページの『ベンダー・エスケープ文節の使用』を参照してください。

DB2 CLI が行えるのは、ベンダー・エスケープ文節の構文エラーを検出することだけです。 DB2 CLI は変換済み SQL ストリングを準備のためにデータ・ソースに渡すことはないため、 DBMS によって検出された構文エラーはこのときには生成されません。 (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 127. 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 を戻します。)
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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効です。	引き数 *InStatementText* はヌル・ポインターです。 引き数 *OutStatementText* はヌル・ポインターです。
HY090	ストリングまたはバッファー長が無効です。	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。

制約

なし。

例

```
/* From the CLI sample DBNATIVE.C */
/* ... */
    SQLNativeSql(hdbc, inODBCStmt, SQL_NTS, outDbStmt, 1024, &dbStmtLen);
    HANDLE_CHECK( SQL_HANDLE_DBC, hdbc, sqlrc, &henv, &hdbc ) ;
    if ( dbStmtLen == SQL_NULL_DATA ) printf( "Invalid ODBC statement¥n" ) ;
    else {
       printf( "¥n   the data source specific format is:¥n   %s¥n", outDbStmt);
    }
```

参照

- 151ページの『ベンダー・エスケープ文節の使用』

SQLNumParams - SQL ステートメント内のパラメーター数を入手する

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,
                              SQLSMALLINT  FAR  *ParameterCountPtr);
```

関数引き数

表 128. 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

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 129. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。
HY010	関数の順序エラーです。	指定された *StatementHandle* について 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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

なし。

例

該当するサンプルの一覧については、 sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内の README ファイルを参照してください。

参照

- 273ページの『SQLBindFileToParam - LOB パラメーターに LOB ファイル参照をバインドする』
- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』
- 630ページの『SQLPrepare - ステートメントを準備する』

SQLNumResultCols - 結果列の数の入手

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  FAR  *ColumnCountPtr);  /* pccol */
```

関数引き数

表 130. 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 131. SQLNumResultCols 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。
HY010	関数の順序エラーです。	*StatementHandle* の 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 SQL の操作中に、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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() 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

```
/* From the CLI sample utilcli.c */
/* ... */
    /* identify the output columns */
    sqlrc = SQLNumResultCols( hstmt, &nResultCols ) ;
    STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参照

- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 267ページの『SQLBindFileToCol - LOB 列に LOB ファイル参照をバインドする』
- 662ページの『SQLSetColAttributes - 列属性を設定する』
- 369ページの『SQLDescribeCol - 列の属性のセットを返す』
- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 488ページの『SQLGetData - 列からのデータの入手』
- 630ページの『SQLPrepare - ステートメントを準備する』

SQLParamData - データ値が必要な次のパラメーターを入手する

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、固定長データを送るときにも使用することができます。この入力方法の正確な順序に関する説明は、87ページの『長形式データの分割送信 / 取り出し』を参照してください。

構文

```
SQLRETURN   SQLParamData     (SQLHSTMT          StatementHandle,
                              SQLPOINTER   FAR  *ValuePtrPtr );
```

関数引き数

表 132. 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 *hdbc* を参照するすべての関数呼び出しに、属性や接続状況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 *hdbc* に対する以下の関数も許可されません。

- SQLAllocConnect()
- SQLAllocStmt()
- SQLSetConnectAttr()
- SQLNativeSql()
- SQLTransact()

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、SQLExecDirect() 関数と 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 133. SQLParamData SQLSTATE

SQLSTATE	説明	解説
07006	変換が無効です。	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 133. 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 情報タイプが 'Y' で、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) に送られたデータが、SQLBindParameter() で *StrLen_or_IndPtr* 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 情報タイプが 'Y' で、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) に送られたデータが、 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・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	一般的なエラーです。	特定の SQLSTATE が存在しなかったり、定義されなかった場合のエラーが発生しました。引き数 *szErrorMsg* 内の SQLError() から戻されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。
HY010	関数の順序エラーです。	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。

表 133. SQLParamData SQLSTATE (続き)

SQLSTATE	説明	解説
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
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・エラーが起きました。	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が起きました。	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。
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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

なし。

例

```
/* From the CLI sample dtlob.c */
```

659ページの『例』を参照してください。

参照

- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』
- 321ページの『SQLCancel - ステートメントを取り消す』
- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 655ページの『SQLPutData - パラメーターのデータ値を渡す』
- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』

SQLParamOptions - パラメーターに入力配列を指定する

使用法

注:

ODBC バージョン 3 では、SQLParamOptions() は使用すべきでない関数であり、 SQLSetStmtAttr() に置き換えられました。 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』を参照してください。

このバージョンの DB2 CLI では引き続き SQLParamOptions() をサポートしていますが、 DB2 CLI プログラムでは SQLSetStmtAttr() を使い始め、最新の規格に従うようお勧めします。 上記の関数と、その他の使用すべきでない関数の詳細については、
822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

目的

仕様:	**DB2 CLI** 2.1	**ODBC** 1.0	

SQLParamOptions() には、 SQLBindParameter() がパラメーターの集合ごとに複数値を設定できるようにする機能があります。このため、アプリケーションは準備、実行、および SQLBindParameter() 呼び出しという 1 組の操作で同一 SQL ステートメントのバッチ処理を行うことができます。

構文

```
SQLRETURN   SQLParamOptions  (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLUINTEGER       Crow,              /* crow */
                              SQLUINTEGER  FAR  *FetchOffsetPtr);  /* pirow */
```

関数引き数

表 134. SQLParamOptions 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。
SQLUINTEGER	Crow	入力	各パラメーターの値の個数。これが 1 より大きい場合、 SQLBindParameter() の *rgbValue* 引き数はパラメーター値の配列を指し、 *pcbValue* は長さの配列を指します。

表 134. SQLParamOptions 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLUINTEGER *	FetchOffsetPtr	出力 (据え置き)	現行パラメーター配列指数の保管用のバッファーを指すポインター。パラメーター値の個々の集合を処理するたびに、 *FetchOffsetPtr* をその集合の配列指数に設定します。ステートメントが失敗すると、*FetchOffsetPtr* を使用して、正常に処理されたステートメントの個数を判別することができます。 *FetchOffsetPtr* ポインターが NULL の場合は、何も戻されません。

使用法

ステートメントを実行するときに、 *FetchOffsetPtr* をパラメーター値の現行配列の索引に設定します。配列内の特定の要素の実行時にエラーが発生すると、実行が停止し、 SQLExecute()、SQLExecDirect()、または SQLParamData() は SQL_ERROR を戻します。

FetchOffsetPtr の内容には、次の使用法があります。

- SQLParamData() が SQL_NEED_DATA を戻した場合に、アプリケーションは *FetchOffsetPtr* 内の値にアクセスして、どのパラメーターの集合に値を割り当てるかを判別することができます。
- SQLExecute() または SQLExecDirect() がエラーを戻した場合に、アプリケーションは *FetchOffsetPtr* 内の値にアクセスして、パラメーター値配列内のどの要素が失敗したかを判別することができます。
- SQLExecute()、SQLExecDirect()、SQLParamData()、または SQLPutData() が成功すると、 *FetchOffsetPtr* 内の値が *Crow* 内の入力値に設定され、配列のすべての要素が正常に処理されたことを示します。

出力引き数 *FetchOffsetPtr* は、正常に処理されたパラメーターの集合の個数を示します。処理されたステートメントが照会である場合、 *FetchOffsetPtr* は、SQLMoreResults() が戻した現行結果セットと関連付けられた配列指数を示しており、 SQLMoreResults() が呼び出されるたびに増加します。

基礎サポートにより複合 SQL (DB2 ユニバーサル・データベース、または DB2 コネクト V2.3 での DRDA 環境) を行える環境であれば、実行要求とともに配列されているすべてのデータは、 1 つのネットワーク・フローとしてともにパッケージされます。

DB2 ユニバーサル・データベース V2.1 以降に接続した場合、アプリケーションは ATOMIC または NOT ATOMIC 複合 SQL を選択できます。 ATOMIC

複合 SQL (これは省略時値) がある場合、配列のすべての要素の処理が成功するか、すべて失敗するかのどちらかです。 NOT ATOMIC 複合 SQL があれば、中間配列要素の 1 つにエラーが検出される場合にも、実行は継続します。アプリケーションは、 SQLSetStmtAttr() 呼び出しの SQL_ATTR_PARAMOPT_ATOMIC 属性を設定して、複合 SQL のタイプを選択できます。

DRDA 環境の場合、基礎複合 SQL サポートは常に NOT ATOMIC COMPOUND SQL (ゆえに DRDA シナリオにおける省略時値) です。

アプリケーションで SQL_ATTR_PARAMOPT_ATOMIC 属性の現行値が分からない場合は、 SQLGetStmtOption() を呼び出してください。

複合 SQL をサポートしていないサーバーに接続したときには、 DB2 CLI はステートメントを準備し、パラメーター・マーカーの配列用に繰り返し実行します。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 135. SQLParamOptions 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数の順序エラーです。	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。
HY107	行の値が範囲外です。	引き数 *Crow* の値は、1 より小さい値でした。

制約

なし。

例

98ページの『配列の入力例』を参照してください。

参照

- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』
- 610ページの『SQLMoreResults - さらに結果セットがあるかどうかを判別する』
- 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』

SQLPrepare - ステートメントを準備する

目的

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLPrepare() は、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、そのステートメントを DBMS に送って準備します。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 SQLFreeStmt() を呼び出してカーソルをクローズする必要があります。

構文

```
SQLRETURN   SQLPrepare      (SQLHSTMT          StatementHandle,
                             SQLCHAR      FAR  *StatementText,
                             SQLINTEGER        TextLength);
```

関数引き数

表 136. 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。
SQLINTEGER	*TextLength*	入力	*StatementText* 引き数の内容の長さ。 これは、*szSqlstr* 内の SQL ステートメントの正確な長さに設定するか、ステートメント・テキストがヌル文字で終了している場合は SQL_NTS に設定する必要があります。

使用法

DB2 CLI バージョン 5 では、デフォルトで据え置き準備がオンになります。対応する実行要求が発行されるまで、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。詳細は 827ページの『デフォルトでの据え置き準備』を参照してください。

SQL ステートメント・テキストにベンダー・エスケープ文節順序列を含む場合、 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 特定形式に修正します。アプリケーションがベンダー・エスケープ文節順序列 (151ページの『ベンダー・エスケープ文節の使用』を参照) を含む 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 がベンダー・エスケープ文節に走査を実行しないようにすることができます。

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) の形式に関する情報を要求することができます。

- SQLNumResultCols()
- SQLDescribeCol()
- SQLColAttribute()

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 “?” 文字によって表されており、アプリケーションに提供された値が SQLExecute() の呼び出し時に置き換えられる、ステートメント内の位置を指示するのに使用されます。バインド・パラメーター関数である SQLBindParameter()、 SQLSetParam() および SQLBindFileToParam() は、アプリケーションの値を各パラメーター・マーカーにバインドする (関連付ける) ため、またデータの転送時にデータの変換が実行される場合にそのことを示すために使用されます。

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。詳細は 409ページの『SQLExecute - ステートメントの実行』を参照してください。

パラメーター・マーカーに関する規則については、 *SQL 解説書* の PREPARE のセクション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。

SQL ステートメントを COMMIT または ROLLBACK とすることはできません。 SQLTransact() を呼び出して COMMIT または ROLLBACK を出してく

ださい。 DB2 ユニバーサル・データベースでサポートされる SQL ステートメントの詳細については、 901ページの表215 を参照してください。

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 137. 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	ステートメントはブロック化できませんでした。	このステートメントは、記憶域以外の理由でブロック化できませんでした。
21S01	挿入値リストが列リストと一致しません。	*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程度と一致していませんでした。
21S02	派生表の程度が列リストと一致しません。	*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程度になっていません。
22018	キャスト仕様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。

表 137. SQLPrepare SQLSTATE (続き)

SQLSTATE	説明	解説
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、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、次のうちの 1 つ以上が含まれていました。 • COMMIT • ROLLBACK • 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 • 構文エラーを含むステートメント
40001	トランザクション・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存在しています。	*StatementText* 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視点名はすでに存在しています。
S0002	データベース・オブジェクトは存在していません。	*StatementText* に、存在していない表名または視点名を参照する SQL ステートメントがあります。
S0011	索引はすでに存在しています。	*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	索引がありません。	*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	列はすでに存在しています。	*StatementText* に ALTER TABLE ステートメントがあり、ADD 文節に指定されている列は固有になっていなかったか、基礎表の既存の列を識別できませんでした。
S0022	列がありません。	*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。

表 137. SQLPrepare SQLSTATE (続き)

SQLSTATE	説明	解説
HY001	メモリーの割り振り失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。
HY009	引き数値が無効です。	*StatementText* は、ヌル・ポインターでした。
HY010	関数の順序エラーです。	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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY014	もはやハンドルはありません。	DB2 CLI は、内部資源が原因でハンドルを割り当てることができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() 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。したがって、 SQLExecute() を呼び出すときにもアプリケーションがこれらの条件を処理できなければなりません。

許可
なし。

例

```
/* From the CLI sample TBREAD.C */
/* ... */
    /* prepare the statement */
    printf("¥n    Prepare the statement¥n");
    printf("        %s¥n", stmt);
    sqlrc = SQLPrepare( hstmt, stmt, SQL_NTS ) ;
    STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参照
- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』
- 273ページの『SQLBindFileToParam - LOB パラメーターに LOB ファイル参照をバインドする』
- 662ページの『SQLSetColAttributes - 列属性を設定する』
- 369ページの『SQLDescribeCol - 列の属性のセットを返す』
- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 409ページの『SQLExecute - ステートメントの実行』
- 616ページの『SQLNumParams - SQL ステートメント内のパラメーター数を入手する』
- 619ページの『SQLNumResultCols - 結果列の数の入手』
- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』

SQLPrimaryKeys - 表の基本キー列を入手する

目的

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLPrimaryKeys() は、表の基本キーを構成する列名のリストを戻します。情報は SQL 結果セット内に戻されますが、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。

構文

```
SQLRETURN   SQLPrimaryKeys   (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLCHAR      FAR  *CatalogName,      /* szCatalogName */
                SQLSMALLINT       NameLength1,       /* cbCatalogName */
                SQLCHAR      FAR  *SchemaName,       /* szSchemaName */
                SQLSMALLINT       NameLength2,       /* cbSchemaName */
                SQLCHAR      FAR  *TableName,        /* szTableName */
                SQLSMALLINT       NameLength3);      /* cbTableName */
```

関数引き数

表 138. 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。 これは、ヌル・ポインターまたは長さゼロのストリングでなければなりません。
SQLSMALLINT	NameLength1	入力	*CatalogName* の長さ。
SQLCHAR *	SchemaName	入力	表名のスキーマ修飾子。
SQLSMALLINT	NameLength2	入力	*SchemaName* の長さ。
SQLCHAR *	TableName	入力	表名。
SQLSMALLINT	NameLength3	入力	*TableName* の長さ。

使用法

SQLPrimaryKeys() は、シングル表から基本キー列を戻します。検索パターンを使用してスキーマ修飾子または表名を指定することはできません。

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。

ほとんどの場合に SQLPrimaryKeys() に対する呼び出しは、システム・カタログに対し複雑な、またそれゆえに費用のかかる照会へマップされることになるので、その使用を制限しなければなりません。それで何度も呼び出しを行うよりも結果を保管してしまう方が良いでしょう。

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 文字よりも少ないので、アプリケーションは、出力バッファー用に 128 文字 (およびヌル終止符) を常にとっておくか、あるいは、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 列の実際の長さをそれぞれ判別するために、SQL_MAX_CATALOG_NAME_LEN、 SQL_MAX_SCHEMA_NAME_LEN、 SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を使用して、選択的に SQLGetInfo() を呼び出すように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。

SQLPrimaryKeys で戻される列

列 1 TABLE_CAT (VARCHAR(128))
これは常にヌルです。

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)
基本キー列名。

列 5 ORDINAL_POSITION (SMALLINT 非 NULL)
1 を最初の番号とする基本キー内の列順序番号。

列 6 PK_NAME (VARCHAR(128))
基本キー識別子。データ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。

指定した表に基本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 139. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、`SQLCancel()` が複数スレッドの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。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。
HY014	もはやハンドルはありません。	DB2 CLI は、内部資源が原因でハンドルを割り当てることができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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	ドライバーが機能していません。	DB2 CLI は、表名の修飾子としてカタログ をサポートしません。

表 139. SQLPrimaryKeys 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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

なし。

例

```
/* From the CLI sample tbconstr.c */
/* ... */
    /* call SQLPrimaryKeys */
    printf("¥n    Call SQLPrimaryKeys for the table %s.%s¥n",
            tbSchema, tbName);
    sqlrc = SQLPrimaryKeys(hstmt, NULL, 0,
                           tbSchema, SQL_NTS, tbName, SQL_NTS);
    STMT_HANDLE_CHECK( hstmt, sqlrc ) ;
```

参照

- 459ページの『SQLForeignKeys - 外部キー列のリストを入手する』
- 788ページの『SQLStatistics - 基本表の索引および統計情報の入手』

SQLProcedureColumns - プロシージャーの入力 / 出力情報を入手する

目的

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLProcedureColumns() は、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。

構文

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

関数引き数

表 140. 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 つの部分から成るプロシージャー名のカタログ修飾子。 これは、ヌル・ポインターまたは長さゼロのストリングでなければなりません。
SQLSMALLINT	NameLength1	入力	*CatalogName* の長さ。これは、0 に設定する必要があります。

表 140. SQLProcedureColumns 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLCHAR *	SchemaName	入力	スキーマ名で結果セットを修飾するためのパターン*値* を入れられるバッファー。 DB2 (MVS/ESA 版) V 4.1 の場合、すべてのストアード・プロシージャーは 1 つのスキーマにあります。 *SchemaName* 引き数の唯一の受け入れ可能な値はヌル・ポインターです。 DB2 ユニバーサル・データベースの場合、*SchemaName* には有効なパターン値が含まれます。有効な検索パターンの詳細については、72ページの『システム・カタログ情報の照会』を参照してください。
SQLSMALLINT	NameLength2	入力	*SchemaName* の長さ。
SQLCHAR *	ProcName	入力	プロシージャー名で結果セットを修飾するためのパターン*値* を入れられるバッファー。
SQLSMALLINT	NameLength3	入力	*ProcName* の長さ。
SQLCHAR *	ColumnName	入力	パラメーター名で結果セットを修飾するためのパターン*値* を入れられるバッファー。この引き数は、 ProcName または SchemaName (あるいはその両方) に空ではない値を指定することにより、すでに制限されている結果セットをさらに修飾するために使用します。
SQLSMALLINT	NameLength4	入力	*ColumnName* の長さ。

使用法

DB2 ユニバーサル・データベース バージョン 5 では、すべてのストアード・プロシージャー (SYSCAT.PROCEDURES と SYSCAT.PROCPARMS) に関する情報をサーバーに保管するのに使う 2 つのシステム・カタログ視点が導入されました。これらの視点については、895ページの『付録G. ストアード・プロシージャーのカタログ表示』を参照してください。

バージョン 5 以前では、 DB2 CLI はストアード・プロシージャー登録の疑似カタログ表を使用していました。デフォルトで、DB2 CLI は新しいシステム・カタログ表示を使用することになります。アプリケーションで疑似カタログ表の使用が見込まれる場合、 CLI/ODBC 構成キーワード PATCH1 を 262144 に設定してください。詳細については、823ページの『ストアード・プロシージャーの疑似カタログ表の置換』を参照してください。

ストアード・プロシージャーが DB2 (MVS/ESA 版) V 4.1 サーバーまたはそれ以降のものにある場合、ストアード・プロシージャーの名前はサーバーの SYSIBM.SYSPROCEDURES カタログ表に登録される必要があります。

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・プロシージャーと関連のある入力、入出力および出力パラメーターに情報を戻しますが、結果セットが戻された記述子情報に情報を戻すことはできません。

SQLProcedureColumns() は、PROCEDURE_CAT、PROCEDURE_SCHEM、PROCEDURE_NAME、および COLUMN_TYPE の順序で結果セット内の情報を戻します。 643ページの『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 文字よりも少ないので、アプリケーションは、出力バッファー用に 128 文字 (およびヌル終止符) を常にとっておくか、あるいは、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 列の実際の長さをそれぞれ判別するために、SQL_MAX_CATALOG_NAME_LEN、 SQL_MAX_SCHEMA_NAME_LEN、SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を使用して選択的に SQLGetInfo() を呼び出すように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。これらの列は、バージョン 2 からバージョン 5 までの間に変更が加えられました。バージョン 5 以降のサーバーに対して、 SQLProcedureColumns() を使

用するバージョン 2 DB2 CLI アプリケーションを実行する場合の詳細については、 826ページの『SQLProcedureColumns() 戻り値の変更』を参照してください。

SQLProcedureColumns で戻される列

列 1 PROCEDURE_CAT (VARCHAR(128))
これは常にヌルです。

列 2 PROCEDURE_SCHEM (VARCHAR(128))
PROCEDURE_NAME を含むスキーマの名前。 (これも DB2 (MVS/ESA 版) V 4.1 SQLProcedureColumns() 結果セットに対し 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 列の値が文字または 2 進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。それがグラフィック (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、時間、タイム・スタンプ・データ・タイプであれば、これは文字に変換されるときの値を表示するのに必要なバイトの総数です。

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。

875ページの表194 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、ヌル終止符は含まれません。正確な数値データ・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。

878ページの表196 を参照してください。

列 10 DECIMAL_DIGITS (SMALLINT)
パラメーターの位取り。位取りが適用できないデータ・タイプの場合は NULL が戻されます。

877ページの表195 を参照してください。

列 11 NUM_PREC_RADIX (SMALLINT)
10、2、または NULL のどれか。 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 が正確な数値データ・タイプの場合、この列には値 10 が含まれており、 COLUMN_SIZE 列と DECIMAL_DIGITS 列にはそのパラメーターに入れられる小数桁数が含まれています。

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

基数が適用できないデータ・タイプの場合は NULL が戻されます。

列 12 NULLABLE (SMALLINT 非 NULL)
パラメーターが NULL を受け入れない場合は SQL_NO_NULLS。

パラメーターがヌル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)
列の省略時値。

NULL を省略時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省略時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。省略時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順序数が入れられます。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”。
- ヌル可能かどうか不明の場合は、ゼロ長ストリング。

ヌル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 141. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。

表 141. SQLProcedureColumns SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。
HY014	もはやハンドルはありません。	DB2 CLI は、内部資源が原因でハンドルを割り当てることができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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	ドライバーが機能していません。	DB2 CLI は、プロシージャー名の修飾子として *カタログ* をサポートしません。 接続されているサーバーは、プロシージャー名の修飾子として *スキーマ* をサポートしません。
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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

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・カタログへのサポートを提供していない、またはストアード・プロシージャーへのサポートを提供していない DB2 へアプリケーションを接続する場合、 SQLProcedureColumns() は空の結果セットを戻します。

例

```
/* From the CLI sample STPCLI.C */
/* ... */
    rc = SQLProcedureColumns(hstmt, NULL, 0,
```

```
                                    stpSchemaPattern, SQL_NTS,
                                    stpNamePattern, SQL_NTS,
                                    colNamePattern, SQL_NTS);
```

参照

- 649ページの『SQLProcedures - プロシージャー名のリストを入手する』

SQLProcedures - プロシージャー名のリストを入手する

目的

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLProcedures() は、サーバーに登録されており、指定した検索パターンと一致するプロシージャー名のリストを戻します。

情報は SQL 結果セット内に戻されますが、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。

構文

```
SQLRETURN   SQLProcedures    (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLCHAR      FAR  *CatalogName,      /* szProcCatalog */
                SQLSMALLINT       NameLength1,       /* cbProcCatalog */
                SQLCHAR      FAR  *SchemaName,       /* szProcSchema */
                SQLSMALLINT       NameLength2,       /* cbProcSchema */
                SQLCHAR      FAR  *ProcName,         /* szProcName */
                SQLSMALLINT       NameLength3);      /* cbProcName */
```

関数引き数

表 142. 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 つの部分から成るプロシージャー名のカタログ修飾子。 これは、ヌル・ポインターまたは長さゼロのストリングでなければなりません。
SQLSMALLINT	NameLength1	入力	*CatalogName* の長さ。これは、0 に設定する必要があります。

表 142. SQLTables 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLCHAR *	SchemaName	入力	スキーマ名で結果セットを修飾するためのパターン*値* を入れられるバッファー。 DB2 (MVS/ESA 版) V 4.1 の場合、すべてのストアード・プロシージャーは 1 つのスキーマにあります。 *SchemaName* 引き数の唯一の受け入れ可能な値はヌル・ポインターです。 DB2 ユニバーサル・データベースの場合、*SchemaName* には有効なパターン値が含まれます。有効な検索パターンの詳細については、72ページの『システム・カタログ情報の照会』を参照してください。
SQLSMALLINT	NameLength2	入力	*SchemaName* の長さ。
SQLCHAR *	ProcName	入力	表名で結果セットを修飾するためのパターン*値*を入れられるバッファー。
SQLSMALLINT	NameLength3	入力	*ProcName* の長さ。

使用法

DB2 ユニバーサル・データベース バージョン 5 では、すべてのストアード・プロシージャー (SYSCAT.PROCEDURES と SYSCAT.PROCPARMS) に関する情報をサーバーに保管するのに使う 2 つのシステム・カタログ視点が導入されました。これらの視点については、895ページの『付録G. ストアード・プロシージャーのカタログ表示』を参照してください。 `SQLProcedures()` は、これらの視点からストアード・プロシージャーのリストを戻します。

バージョン 5 以前では、DB2 CLI はストアード・プロシージャー登録の疑似カタログ表を使用していました。デフォルトで、DB2 CLI は新しいシステム・カタログ表示を使用することになります。アプリケーションで疑似カタログ表の使用が見込まれる場合、 CLI/ODBC 構成キーワード PATCH1 を 262144 に設定してください。詳細については、823ページの『ストアード・プロシージャーの疑似カタログ表の置換』を参照してください。

ストアード・プロシージャーが DB2 (MVS/ESA 版) V 4.1 サーバーまたはそれ以降のものにある場合、ストアード・プロシージャーの名前はサーバーの SYSIBM.SYSPROCEDURES カタログ表に登録される必要があります。

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() 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 文字よりも少ないので、アプリケーションは、出力バッファー用に 128 文字 (およびヌル終止符) を常にとっておくか、あるいは、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 列の実際の長さをそれぞれ判別するために、SQL_MAX_CATALOG_NAME_LEN、 SQL_MAX_SCHEMA_NAME_LEN、SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を使用して選択的に SQLGetInfo() を呼び出すように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。

SQLProcedures で戻される列

列 1 PROCEDURE_CAT (VARCHAR(128))
これは常にヌルです。

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。

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。

これは、バージョン 5 より前の DB2 CLI で使用されていたものです。後方互換性のために、旧 DB2CLI.PROCEDURES 疑似カタログ表と一緒に使用できます (PATCH1 CLI/ODBC 構成キーワードを設定)。詳細については、823ページの『ストアード・プロシージャーの疑似カタログ表の置換』を参照してください。

列 5 NUM_OUTPUT_PARAMS (INTEGER 非 NULL)
出力パラメーター数。

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。

これは、バージョン 5 より前の DB2 CLI で使用されていたものです。後方互換性のために、旧 DB2CLI.PROCEDURES 疑似カタログ表と一緒に使用できます (PATCH1 CLI/ODBC 構成キーワードを設定)。詳細については、823ページの『ストアード・プロシージャーの疑似カタログ表の置換』を参照してください。

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。

これは、バージョン 5 より前の DB2 CLI で使用されていたものです。後方互換性のために、旧 DB2CLI.PROCEDURES 疑似カタログ表と一緒に使用できます (PATCH1 CLI/ODBC 構成キーワードを設定)。詳細については、823ページの『ストアード・プロシージャーの疑似カタログ表の置換』を参照してください。

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 143. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。
HY014	もはやハンドルはありません。	DB2 CLI は、内部資源が原因でハンドルを割り当てることができませんでした。

表 143. SQLProcedures SQLSTATE (続き)

SQLSTATE	説明	解説
HY090	ストリングまたはバッファー長が無効です。	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	ドライバーが機能していません。	DB2 CLI は、プロシージャー名の修飾子としてカタログ をサポートしません。 接続されているサーバーは、プロシージャー名の修飾子としてスキーマをサポートしません。
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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

ストアード・プロシージャー・カタログへのサポートを提供していない、またはストアード・プロシージャーへのサポートを提供していない DB2 へアプリケーションを接続する場合、 SQLProcedureColumns() は空の結果セットを戻します。

例

```
/* From the CLI sample STPCLI.C */
/* ... */
    /* call SQLProcedures */
    printf("¥n    Call SQLProcedures for:¥n");
    printf("        schemaPattern = %s¥n", stpSchemaPattern);
    printf("        namePattern = %s¥n", stpNamePattern);
    sqlrc = SQLProcedures(hstmt, NULL, 0,
                          stpSchemaPattern, SQL_NTS,
                          stpNamePattern, SQL_NTS);
```

参照

- 640ページの『SQLProcedureColumns - プロシージャーの入力 / 出力情報を入手する』

SQLPutData - パラメーターのデータ値を渡す

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。

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。

構文

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

関数引き数

表 144. 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。

表 144. SQLPutData 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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・タイプの省略時値が 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

SQLParamData() と SQLPutData() の順序については、 87ページの『長形式データの分割送信 / 取り出し』を参照してください。

アプリケーションは、SQL_NEED_DATA 状態のステートメントについて SQLParamData() を呼び出した後に 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。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 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 2 進データの場合、 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。

StatementHandle の親 *hdbc* を参照するすべての関数呼び出しに、属性や接続状況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 *hdbc* に対する以下の関数も許可されません。

- SQLAllocConnect()
- SQLAllocStmt()
- SQLSetConnectAttr()
- SQLNativeSql()
- SQLTransact()

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 145. SQLPutData SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	数値パラメーターに送られるデータは、有効数字が失われないようにして切り捨てられます。 送信された日付と時刻の列に関するタイム・スタンプ・データが切り捨てられました。 関数は、 SQL_SUCCESS_WITH_INFO で戻ります。

表 145. SQLPutData SQLSTATE (続き)

SQLSTATE	説明	解説
22001	ストリング・データの右側が切り捨てられました。	2 進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。	数値パラメーターに送られたデータが原因で、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	日時形式が無効です。	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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。
HY009	引き数値が無効です。	引き数 *DataPtr* がヌル・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	関数の順序エラーです。	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長が無効です。	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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

StrLen_or_Ind の新しい値 SQL_DEFAULT_PARAM が ODBC 2.0 に導入され、プロシージャーで使用する値をアプリケーションから送信された値ではなく、パラメーターの省略時値にするように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には省略時値の概念がないので、*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 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、 SQLGetInfo() に SQL_NEED_LONG_DATA_LEN オプションを指定して、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、この情報が SQLPutData() に必要ないことを示す場合、'N' を戻します。

例

```
/* From the CLI sample dtlob.c */
/* ... */
    /*
     * This paramter will use SQLPutData
     */
    blobInd = SQL_DATA_AT_EXEC;
    sqlrc = SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_BLOB,
                             BUFSIZ, 0, (SQLPOINTER)inputParam, BUFSIZ, &blobInd);
```

参照

- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』
- 409ページの『SQLExecute - ステートメントの実行』
- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 622ページの『SQLParamData - データ値が必要な次のパラメーターを入手する』
- 321ページの『SQLCancel - ステートメントを取り消す』

SQLRowCount - 行カウントを入手する

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視点に対して実行された UPDATE、INSERT、または DELETE 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   FAR  *RowCountPtr);     /* pcrow */
```

関数引き数

表 146. 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場所を指すポインター。

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、または DELETE 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、連鎖削除) 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 147. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数の順序エラーです。	*StatementHandle* の 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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。

許可

なし。

例

該当するサンプルの一覧については、 sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内の README ファイルを参照してください。

参照

- 399ページの『SQLExecDirect - ステートメントの直接実行』
- 409ページの『SQLExecute - ステートメントの実行』
- 619ページの『SQLNumResultCols - 結果列の数の入手』

SQLSetColAttributes - 列属性を設定する

使用すべきでない関数

注:

SQLSetColAttributes() は、ODBC バージョン 3 で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 SQLSetColAttributes() 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。 SQLSetColAttributes() に渡すすべての引き数は無視され、関数呼び出しは、常に SQL_SUCCESS を戻します。

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() の機能性を必要としません。詳しくは、827ページの『デフォルトでの据え置き準備』を参照してください。

例

該当するサンプルの一覧については、 sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内の README ファイルを参照してください。

SQLSetConnectAttr - 接続属性を設定する

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。

構文

```
SQLRETURN   SQLSetConnectAttr(SQLHDBC           ConnectionHandle,
                              SQLINTEGER        Attribute,
                              SQLPOINTER        ValuePtr,
                              SQLINTEGER        StringLength);
```

関数引き数

表 148. 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	設定する属性 (667ページの『属性値』のリストに記載)。
SQLPOINTER	ValuePtr	入力	*Attribute* と関連付けられる値を指すポインター。 *Attribute* の値に応じて、**ValuePtr* は、 32 ビットの無符号整数値になるか、ヌル終了文字ストリングを指します。 *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になりますので、注意してください。

表 148. 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 2 進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、*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* 引き数の設定により、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* はストリングまたは SQL_NTS の長さです。 • *ValuePtr* が 2 進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。これは負の値を *StringLength* に入れます。 • *ValuePtr* が文字ストリングまたは 2 進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() を使用できる

ように更新する必要があります。詳細については、824ページの『SQLSetConnectAttr() を使用してステートメント属性のサブセットを設定する』を参照してください。

引き続き SQLSetConnectAttr() を使用してステートメント属性を設定しているバージョン 2 のアプリケーションでは、ステートメント属性が複数の活動状態のステートメントのうちどれかに設定された場合にエラーが戻されると、このステートメント属性は、後で接続に割り当てられるステートメントの省略時値として設定されます。しかし、SQLSetConnectAttr() への同じ呼び出しで直前に設定されたステートメント属性が、関数が異常終了してエラーが戻された後も設定されたままになるかどうかは、定義されていません。SQLSetConnectAttr() を使ってステートメント属性を設定しないようお勧めする理由の 1 つはこの点にあります。

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、記述子フィールドは、接続にある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

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 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。

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任意の時点で 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。次の表に、それぞれの接続属性を設定できるのはいつかを示します。

表 149. 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	はい	はい	はい [a]
SQL_ATTR_ASYNC_ENABLE	はい	はい	いいえ [b]

表 149. 接続属性をいつ設定するか (続き)

属性	接続前	接続後	ステートメントを割り当てた後
SQL_ATTR_AUTO_IPD (読み取り専用)	いいえ	はい	はい
SQL_ATTR_AUTOCOMMIT	はい	はい	はい [c]
SQL_ATTR_CLISCHEMA	はい	はい	はい
SQL_ATTR_CLOSE_BEHAVIOR	いいえ	はい	はい [c]
SQL_ATTR_CONN_CONTEXT	はい	いいえ	いいえ
SQL_ATTR_CONNECT_NODE	はい	いいえ	いいえ
SQL_ATTR_CONNECTTYPE	はい	いいえ	いいえ
SQL_ATTR_CURRENT_SCHEMA	はい	はい	はい
SQL_ATTR_DB2_SQLERRP (読み取り専用)	いいえ	はい	はい
SQL_ATTR_DB2ESTIMATE	いいえ	はい	はい
SQL_ATTR_DB2EXPLAIN	いいえ	はい	はい
SQL_ATTR_ENLIST_IN_DTC	いいえ	はい	はい
SQL_ATTR_INFO_ACCTSTR	いいえ	はい	はい
SQL_ATTR_INFO_APPLNAME	いいえ	はい	はい
SQL_ATTR_INFO_USERID	いいえ	はい	はい
SQL_ATTR_INFO_WRKSTNNAME	いいえ	はい	はい
SQL_ATTR_LOGIN_TIMEOUT	はい	いいえ	いいえ
SQL_ATTR_LONGDATA_COMPAT	はい	はい	はい
SQL_ATTR_MAXCONN	はい	いいえ	いいえ
SQL_ATTR_OPTIMIZE_SQLCOLUMNS	はい	はい	はい
SQL_ATTR_QUIET_MODE	はい	はい	はい
SQL_ATTR_SYNC_POINT	はい	いいえ	いいえ
SQL_ATTR_TRANSLATE_OPTION	はい	はい	はい
SQL_ATTR_TXN_ISOLATION	いいえ	はい [c]	はい [a]
SQL_ATTR_WCHARTYPE	はい	はい [c]	はい [c]

[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] 属性は、活動状態のステートメントが出現する前に設定してください。

[c] 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。

データ・ソースが **ValuePtr* に指定し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。たとえば、アプリケーションが SQL_ATTR_ASYNC_ENABLE を SQL_ASYNC_ENABLE_ON に設定しようとしたのに、サーバーがこの設定をサポートしていない場合、 DB2 CLI は代わ

りに値 SQL_ASYNC_ENABLE_OFF を代用します。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* によって設定した情報の形式は、*Attribute* の指定により異なります。 SQLSetConnectAttr() は、2 つの異なる形式、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形式で、属性情報を受け入れます。それぞれの形式の注記は、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。

DB2 CLI バージョン 5.2 以前に導入されたすべての属性と同じく、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。

属性値

以下のバージョン 2 接続属性 (SQLSetConnectOption() を使用して設定された) の名前は、バージョン 5 で変更されました。

表 150. 名前が変更された接続属性

バージョン 2 での名前	バージョン 5 以降の名前
SQL_ACCESS_MODE	SQL_ATTR_ACCESS_MODE
SQL_AUTOCOMMIT	SQL_ATTR_AUTOCOMMIT
SQL_CONNECTTYPE	SQL_ATTR_CONNECTTYPE
SQL_CURRENT_SCHEMA	SQL_ATTR_CURRENT_SCHEMA
SQL_DB2ESTIMATE	SQL_ATTR_DB2ESTIMATE
SQL_DB2EXPLAIN	SQL_ATTR_DB2EXPLAIN
SQL_LOGIN_TIMEOUT	SQL_ATTR_LOGIN_TIMEOUT
SQL_LONGDATA_COMPAT	SQL_ATTR_LONGDATA_COMPAT
SQL_MAXCONN	SQL_ATTR_MAXCONN
SQL_QUIET_MODE	SQL_ATTR_QUIET_MODE
SQL_SCHEMA	SQL_ATTR_SCHEMA
SQL_SYNC_POINT	SQL_ATTR_SYNC_POINT
SQL_TXN_ISOLATION	SQL_ATTR_TXN_ISOLATION
SQL_WCHARTYPE	SQL_ATTR_WCHARTYPE

属性　*ValuePtr の内容

SQL_ATTR_ACCESS_MODE (DB2 CLI v2)

次のどれかである 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。それゆえ、制限の少ない分離レベルおよびロックが、トランザクションで使用可能です。つまり非コミット読み取り (SQL_TXN_READ_UNCOMMITTED) が可能です。

DB2 CLI は、データベースに対する要求が*読取専用* であることを確認しません。更新要求を出すと、 DB2 CLI は SQL_MODE_READ_ONLY 設定値の結果として選択されたトランザクション分離レベルを使用して、その要求を処理します。

- **SQL_MODE_READ_WRITE**: 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、この接続に関する省略時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。

  SQL_MODE_READ_WRITE が省略時値です。

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_ASYNC_ENABLE (DB2 CLI v5)

指定された接続上でステートメントを使用して呼び出された関数を非同期で実行するかどうかを指定する 32 ビット整数値。

- **SQL_ASYNC_ENABLE_OFF** = Off (省略時値)
- SQL_ASYNC_ENABLE_ON = On

SQL_ASYNC_ENABLE_ON を設定すると、今後この接続上に割り当てられるすべてのステートメント・ハンドルを非同期実行できるようになります。また、この設定により、この接続に関連する既存のステートメント・ハンドルの非同期実行も可能になります。接続上に活動状態のステートメントがあるのに非同期実行がオンになっていると、エラーが戻されます。

この属性により、 *InfoType* SQL_ASYNC_MODE を使用して呼び出される `SQLGetInfo()` が、 SQL_AM_CONNECTION と SQL_AM_STATEMENT のどちらを戻すかを設定できます。

関数が非同期で呼び出されると、元の関数が SQL_STILL_EXECUTING 以外のコードを戻すまでは、 *StatementHandle* に関連するステートメントまたは接続時に、元の関数、 `SQLAllocHandle()`、 `SQLCancel()`、`SQLGetDiagField()`、または `SQLGetDiagRec()` しか呼び出せません。 *StatementHandle* または *StatementHandle* に関連する接続時に呼び出される他の関数は、 SQLSTATE HY010 の SQL_ERROR (関数順序エラー) を戻します。関数は、他のステートメント時に呼び出せます。

一般に、アプリケーションは、シングル・スレッドのオペレーティング・システム上で、非同期で関数を実行する必要があります。マルチ・スレッドのオペレーティング・システムの場合は、同じスレッドで非同期に実行するのではなく、別々のスレッドでアプリケーションを実行すべきです。マルチ・スレッドのオペレーティング・システムでしか実行できないアプリケーションは、非同期実行をサポートする必要はありま

せん。詳細については、50ページの『マルチスレッドのアプリケーション作成』および 145ページの『CLI の非同期実行』を参照してください。

以下の関数は、非同期で実行できます。

SQLColAttribute()	SQLGetTypeInfo()
SQLColumnPrivileges()	SQLMoreResults()
SQLColumns()	SQLNumParams()
SQLCopyDesc()	SQLNumResultCols()
SQLDescribeCol()	SQLParamData()
SQLDescribeParam()	SQLPrepare()
SQLExecDirect()	SQLPrimaryKeys()
SQLExecute()	SQLProcedureColumns()
SQLFetch()	SQLProcedures()
SQLFetchScroll()	SQLPutData()
SQLForeignKeys()	SQLSetPos()
SQLGetData()	SQLSpecialColumns()
SQLGetDescField() 1*	SQLStatistics()
SQLGetDescRec() 1*	SQLTablePrivileges()
SQLGetDiagField()	SQLTables()
SQLGetDiagRec()	

1* これらの関数は、記述子がインプリメンテーション記述子であり、アプリケーション記述子でない場合にのみ、非同期で呼び出せます。

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、読み取り専用の 32 ビット整数値。

- 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 の省略時値は、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 ビット整数値。

- SQL_AUTOCOMMIT_OFF: アプリケーションは、 `SQLTransact()` 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**: 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 は省略時値です。

  注: この値が調整された分散作業接続単位の場合、省略時値は **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。

DB2 CLI バージョン 1 アプリケーションは、省略時値が手動コミット・モードであると想定しています。 822ページの『非互換性』を参照してください。

SQL_ATTR_CLISCHEMA (DB2 CLI v6)

DB2 ODBC カタログ視点 (これは、使用するホスト DBMS 上に保管されている) の名前が入っているヌル終了文字ストリング。

DB2 ODBC カタログは、 DB2 コネクトを使用してホスト DBMS に接続する ODBC アプリケーションによる、表リストのスキーマ呼び出しのパフォーマンスを向上させるよう設計されています。

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、事

前に結合されており、 ODBC アプリケーションによる高速カタログ・アクセスをサポートするための索引付けがなされています。

システム管理者は、複数の DB2 ODBC カタログ視点を作成し、それぞれに特定のユーザー・グループが必要とする行だけを含めることができます。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。

この属性には SYSSCHEMA キーワードと似た効果がありますが、可能な場合には SQL_ATTR_CLISCHEMA の方を使用するようにしてください。 SQL_ATTR_CLISCHEMA を使用すると、データ・アクセスの効率が向上します。 SYSSCHEMA で使用するユーザー定義の表は DB2 カタログ表のミラー・イメージであり、 ODBC ユーザーに必要な情報を作成するためには、 ODBC ドライバーが複数の表の行を結合する必要がありました。また、SQL_ATTR_CLISCHEMA を使用すると、カタログ表の競合も減少します。

これに対応する DB2 CLI/ODBC ドライバー構成キーワード (180ページの『CLISCHEMA』) もあります。

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。これは省略時値です。
- 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単独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。

詳しくは、325ページの『SQLCloseCursor - カーソルをクローズして保留の結果を廃棄する』を参照してください。

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 は次のどちらかになります。

- 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ヌル・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。省略時設定では、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。

いつアプリケーションがコンテキストを管理すべきかについては、 50 ページの『マルチスレッドのアプリケーション作成』を参照してください。

コンテキストの詳細については、 *管理 API 解説書* の sqleBeginCtx() API を参照してください。

SQL_ATTR_CONNECT_NODE (DB2 CLI v6)

この属性は、接続先の DB2 エンタープライズ拡張エディション・データベース区画サーバーのターゲット論理ノードを指定するのに使用します。この設定は、環境変数 DB2NODE の値を上書きします。次のどちらかに設定できます。

- 0 ～ 999 の整数
- SQL_CONN_CATALOG_NODE

この変数を設定しないと、ターゲット論理ノードは、デフォルトにより、マシンでポート 0 として定義されている論理ノードに設定されます。

これに対応する DB2 CLI/ODBC ドライバー構成キーワード (182ページの『CONNECTTYPE』) 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 - エラーが起こり、サーバーへの接続が切断されています。ユーザーは切断を実行して、DB2CLI 資源をクリーンアップする必要があります。

この属性は、主に Microsoft ODBC ドライバー・マネージャー 3.5x によって、接続をプールする前に使用されます。

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入手しようとすると、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 (省略時値) 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、または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 `SQLTransact()` 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回復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。

  これは省略時値です。

- 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 接続オプションの省略時値である 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、SQLSetConnectOption() 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。整合 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、 SQLSetEnvAttr() が ODBC でサポートされていないので、各接続の接続レベルでこれらの属性を設定する必要があります。

この省略時接続タイプは、 CONNECTTYPE DB2 CLI/ODBC 構成キーワードを使用して設定することもできます。詳しくは 168ページの『db2cli.ini の構成』を参照してください。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_CURRENT_CATALOG (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入手しようとすると、SQLSTATE が HYC00 (ドライバーが機能しない) になります。

データ・ソースで使用するカタログの名前が入っているヌル終了文字ストリング。

SQL_ATTR_CURRENT_SCHEMA (DB2 CLI v2)

szSchemaName ポインターをヌルに設定する場合に、 SQLColumns() 呼び出しに DB2 CLI が使用するスキーマの名前を含む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* 引き数を長さゼロまたはヌル・ポインターとして、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、 *szSchemaName* ポインターがヌルである次の SQLColumns() 呼び出しで有効になります。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_DB2_SQLERRP (DB2 CLI v6)

sqlca の *sqlerrp* フィールドを含む読み取り専用のヌル文字で終了するストリング。

製品を表す 3 文字の識別子で始まり、その後にバージョン、リリース、および修正レベルを表す 5 つの数字が続きます。たとえば、SQL05000 は DB2 ユニバーサル・データベースのバージョン 5 リリース 0 修正レベル 0 を表します。

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。

sqlca およびこの特定のフィールドの詳細については、*SQL 解説書* を参照してください。

SQL_ATTR_DB2ESTIMATE (DB2 CLI v2)

DB2 CLI が、最適化プログラムから戻された見積もりを報告するため、 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した 32 ビット整数。

- **0**: 見積もりは戻されません。

  これは省略時値です。

- 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限界値。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。 DB2ESTIMATE 値の方が大きければ、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、最適化プログラムの評価と、この照会を続行するか取り消すかを選択するための押し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せずに、 `SQLPrepare()` の次に `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、この機能を直接実装し、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。

新規の 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。

- 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 表を作成しなければなりません。補足情報については、*SQL 解説書* を参照してください。

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。

現行の許可 ID は、Explain 表に対して INSERT 特権を持っていなければなりません。

この値は、 DB2EXPLAIN DB2 CLI/ODBC 構成キーワードを使用して設定することもできます。詳しくは 168ページの『db2cli.ini の構成』を参照してください。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_ENLIST_IN_DTC (DB2 CLI v5.2)

次のいずれかの SQLPOINTER。

- 非ヌル・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調整プログラム (DTC) によって行われます。

- ヌル:

  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。

この属性に非ヌル・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。非ヌル・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列に実行することはできません。

SQL_ATTR_INFO_ACCTSTR (DB2 CLI v6)

DB2 コネクトを使用しているときに、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ストリングを識別するのに使用される、ヌル文字で終了する文字ストリング。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 (OS/390 版) 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。
- DRDA サーバー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 ～ Z、 0 ～ 9、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 コネクトを使用しているときに、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、ヌル文字で終了する文字ストリング。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 (OS/390 版) 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。
- DRDA サーバー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 ～ Z、 0 ～ 9、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_USERID (DB2 CLI v6)

DB2 コネクトを使用しているときに、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、ヌル文字で終了する文字ストリング。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 (OS/390 版) 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。
- DRDA サーバー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 ～ Z、 0 ～ 9、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 コネクトを使用しているときに、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、ヌル文字で終了する文字ストリング。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 (OS/390 版) 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。
- DRDA サーバー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 ～ Z、 0 ～ 9、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_LOGIN_TIMEOUT (DB2 CLI v2)

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入手しようとすると、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、ログイン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。 *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 2 進ラージ・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、そのまま (SQL_BLOB、SQL_CLOB、SQL_DBCLOB) 報告されます。これは省略時値です。
- SQL_LD_COMPAT_YES: ラージ・オブジェクト・データ・タイプ (BLOB、CLOB、および DBCLOB) は、SQL_LONGVARBINARY、SQL_LONGVARCHAR、および

SQL_LONVARGRAPHIC にマップされます。 SQLGetTypeInfo() は、SQL_LONGVARBINARY、 SQL_LONGVARCHAR 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_MAXCONN (DB2 CLI v2)

アプリケーションが設定する同時接続の最大数に対応する 32 ビット整数値。省略時値は **0** で、これは最大値がないことを意味します。アプリケーションはシステム資源で許可される個数の接続を設定することができます。整数値は 0 または正の数でなければなりません。

これは、アプリケーション・ベースでの接続の最大数のための管理プログラムとして使用できます。

OS/2、Windows 95、および Windows NT で NetBIOS プロトコルが使用中の場合に、この値は、アプリケーションで並行してセットアップされる接続 (NetBIOS セッション) の数に対応します。 OS/2 NetBIOS の値の範囲は、1 から 254 までです。 0 (省略時値) を指定すると、**5 つの** *予約済み* 接続が行われます。他のアプリケーションは、*予約済み NetBIOS セッション*を使用できません。このパラメーターで指定された接続数は、 DB2 NetBIOS プロトコルがリモート・サーバーに接続するのに使用するアダプターに適用されます (アダプター番号は NetBIOS ノードのノード・ディレクトリーで指定します)。

最初の接続が確立される時点で有効な値は、使用しようとしている値です。最初の接続がすでに確立されていると、この値を変更しようとしても拒否されます。アプリケーションが SQL_ATTR_MAXCONN 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てきました。 ODBC アプリケーションは、SQLSetEnvAttr() がサポートされていないので、接続レベルでこの属性を設定する必要があります。

同時接続の最大数は、 MAXCONN DB2 CLI/ODBC 構成キーワードを使用して設定することもできます。詳しくは 168ページの『db2cli.ini の構成』を参照してください。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_METADATA_ID (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入手しようとすると、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 (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入手しようとすると、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 (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入手しようとすると、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_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 `SQLSetConnectOption()` を呼び出したことがない場合、 DB2 CLI はこのオプションに、`SQLGetConnectOption()` に対するヌルの親ウィンドウ・ハンドルを戻し、ヌルの親ウィンドウ・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) 最適化プログラム情報を表示するよう求めると、DB2 CLI はヌルのウィンドウ・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 がヌル・ポインターである場合、 DB2 CLI はダイアログ・ボックスを表示しません。エンド・ユーザーが最適化プログラム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 がヌル・ポインターではない場合、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間で選択できるようにする 32 ビット整数値。以下の値を指定することができます。

- **SQL_ONEPHASE**: 各データベースが複数のデータベース・トランザクションで行う作業をコミットするために、 1 フェーズ・コミットが使用されます。データ保全性を確かにするには、 1 つのトランザクションで 2 つ以上のデータベースを更新することがないようにします。 1 つのトランザクションで実行された更新のうち最初のデータベースだけが、そのトランザクションでのただ 1 つの更新側になり、それ以外のすべてのデータベースはアクセスされても読み取り専用になります。このトランザクション内でこれらの読み取り専用のデータベースを更新しようとしても拒否されます。これは省略時値です。
- 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。

分散作業単位 (トランザクション) の詳細については、*SQL 解説書* を参照してください。

アプリケーション内のすべての接続の SQL_ATTR_CONNECTTYPE 値と SQL_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。整合 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、 SQLSetEnvAttr() が ODBC でサポートされていないので、接続レベルでこれらの属性を設定する必要があります。

調整トランザクションのタイプを SYNCPOINT DB2 CLI/ODBC 構成キーワードを使用して設定することもできます。詳しくは 168ページの『db2cli.ini の構成』を参照してください。

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 (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入手しようとすると、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。

DB2 CLI トレース機能は、属性ではなく TRACE DB2 CLI/ODBC 構成キーワードを使用して設定できます。詳しくは 168ページの『db2cli.ini の構成』を参照してください。

SQL_ATTR_TRACEFILE (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入手しようとすると、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ヌルで終了する文字ストリング。

DB2 CLI トレース・ファイル名は、この属性ではなく、TRACEFILENAME DB2 CLI/ODBC 構成キーワードを使用して設定されます。詳しくは 168ページの『db2cli.ini の構成』を参照してください。

SQL_ATTR_TRANSLATE_LIB (DB2 CLI v5)

この接続属性は ODBC により定義されますが、Windows 3.1 の DB2 CLI でのみサポートされます。その結果、その他のプラットフォームでこの属性を設定または入手しようとすると、 SQLSTATE が HYC00 (ドライバーが機能しない) になります。

DB2 クライアント・アプリケーション・イネーブラー (Windows 版) またはソフトウェア開発者キット (Windows 版) 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。

このオプションは、DB2 (OS/2 版) バージョン 1 に接続されているとき、またはバージョン 2.3 以前のバージョンの DDCS (OS/2 版) を使用しているときに、 16 ビット・バージョンの Windows 上で使用できます。 TRANSLATEOPTION と組み合わせて使用され、NLS SBCS 文字 (ドイツ語のウムラウト文字など) と、 Windows コード・ページ 1004 の対応する文字の正確なマッピングを提供します。

注: このオプションは、 Windows アプリケーションが接続するサーバーが下位バージョンのサーバー (たとえば DB2 バージョン 1) で、そのサーバーでは非同等コード・ページ変換がサポートされていない場合には便利です。

SQL_ATTR_TRANSLATE_OPTION (DB2 CLI v5)

この接続属性は ODBC により定義されますが、Windows 3.1 の DB2 CLI でのみサポートされます。その結果、その他のプラットフォームでこの属性を設定または入手しようとすると、 SQLSTATE が HYC00 (ドライバーが機能しない) になります。

DB2 バージョン 1 でのデータベースのコード・ページ番号 (データベース構成パラメーターを照会すると取得できます) を定義します。 TRANSLATEDLL および TRANSLATEOPTION を指定すると、コード・ページ番号 *database codepage number* から Windows 1004 コード・ページに文字を変換できるようになります。

database codepage number でサポートされている値には、437 と 850 の 2 つがあります。それ以外の値を指定すると、変換が不可能であるということを示す警告が、接続要求時に戻されます。

注: このオプションは、 Windows アプリケーションが接続するサーバーが下位バージョンのサーバー (たとえば DB2 バージョン 1) で、そのサーバーでは非同等コード・ページ変換がサポートされていない場合には便利です。

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()` を呼び出すことにより、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、繰り返し不能な読み取り、および単独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。繰り返し不能な読み取り、および単独読み取りが可能です。

  これは省略時値です。
- SQL_TXN_REPEATABLE_READ - ダーティー読み取りと繰り返し不能な読み取りが不可能です。単独読み取りが可能です。
- SQL_TXN_SERIALIZABLE - トランザクションが直列化可能です。ダーティー読み取り、繰り返し不能な読み取り、単独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 ユニバーサル・データベース (AS/400 版) 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、未確約読み取りです。
- 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。

分離レベルの詳細については、*SQL 解説書* を参照してください。

このオプションは、hstmt にオープン・カーソルがあったり、この接続に未解決のトランザクションがあったりすると指定できません。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、省略時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。 `SQLSetStmtOption()` の関数記述にある SQL_STMTTXN_ISOLATION オプション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) 文字形式を、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形式にするか、またはワイド文字形式にするか選択する融通性があります。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グラフィック 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 (C 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形式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。この送受信により、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形式に変換したり、この形式から変換したりする、という不利な点があります。

  これは省略時値です。

マルチバイト・アプリケーション変数の使用法については、*アプリケーション開発の手引き* を参照してください。

注: これは、IBM 定義の拡張機能です。

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 151. SQLSetConnectAttr SQLSTATE

SQLSTATE	説明	解説
01000	一般的なエラーです。	通知メッセージです。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
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 を戻します。)
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、結果セットは保留になっていました。
HY000	一般的なエラーです。	処理系固有の SQLSTATE が存在しなかったり、定義されなかった場合のエラーが発生しました。 *MessageText バッファー内の SQLGetDiagRec() が戻したエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY009	引き数値が無効です。	*ValuePtr* にヌル・ポインターが渡され、**ValuePtr* の値はストリング値でした。

表 151. SQLSetConnectAttr SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数の順序エラーです。	*ConnectionHandle* と関連する *StatementHandle* の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* の SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が戻されました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、この関数が呼び出されました。 *ConnectionHandle* の 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。
HY011	この段階で操作は無効です。	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 や SQL_ATTR_ASYNC_ENABLE 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長が無効です。	*StringLength* 引き数は 0 未満でしたが、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。	DB2 CLI のこのバージョンでは、引き数 *Attribute* の指定値が無効です。
HYC00	ドライバーが機能していません。	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・ソースによりサポートされていませんでした。

制約

なし。

例

```
/* From the CLI sample utilcli.c */
    /* set AUTOCOMMIT off or on */
    sqlrc = SQLSetConnectAttr( *pHdbc,
                               SQL_ATTR_AUTOCOMMIT,
                               autocommitValue, SQL_NTS) ;
    HANDLE_CHECK( SQL_HANDLE_DBC, *pHdbc, sqlrc, pHenv, pHdbc ) ;
```

参照

- 480ページの『SQLGetConnectAttr - 現行属性設定を入手する』
- 592ページの『SQLGetStmtAttr - ステートメント属性の現行設定値を入手する』
- 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』
- 246ページの『SQLAllocHandle - ハンドルを割り振る』

SQLSetConnection - 接続ハンドルを設定する

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。この関数は、アプリケーションが DB2 CLI 関数呼び出しを組み込み SQL 関数呼び出しと混合し、複数の接続が関係するときにだけ使用してください。

構文

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

関数引き数

表 152. 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たい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 接続関数によって接続要求を出した場合に、組み込み SQL を含む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 では複数の並行接続が行えます。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視点から言えば、これは必ずしも決定的なものでなく、この情報を追跡することは困難です。 SQLSetConnection() は、どの接続が活動状態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。

SQLSetConnection() は、アプリケーションが単に DB2 CLI 呼び出しだけを行う場合、必要ありません。これは、各ステートメント・ハンドルが暗黙に接続ハンドルに関連付けられ、特定の DB2 CLI 関数がどの接続を適用するかについてまったく混乱がないからです。

DB2 CLI アプリケーション内に組み込まれた SQL の使用の詳細については、143ページの『組み込み SQL と DB2 CLI の混合』を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 153. 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	一般的なエラーです。	特定の SQLSTATE でなく、処理系定義 SQLSTATE で定義されなかった種類のエラーが発生しました。引き数 *szErrorMsg* 内の SQLError から戻されたエラー・メッセージに、そのエラーと原因が記述されています。

制約

なし。

例

```
/* From the CLI sample dbmusemx.sqc */
/* ... */
    sqlrc = SQLSetConnection( a_hdbc[0]);
    MC_HANDLE_CHECK( SQL_HANDLE_DBC, a_hdbc[0], sqlrc,
                     &henv, a_hdbc, MAX_CONNECTIONS ) ;
```

参照

- 356ページの『SQLConnect - データ・ソースに接続する』
- 382ページの『SQLDriverConnect - データ・ソースに (拡張) 接続する』

SQLSetConnectOption - 接続オプションを設定する

使用すべきでない関数

注:

ODBC バージョン 3 では、SQLSetConnectOption() は使用すべきでない関数であり、 SQLSetConnectAttr() に置き換えられました。詳細については、663ページの『SQLSetConnectAttr - 接続属性を設定する』を参照してください。

このバージョンの DB2 CLI では引き続き SQLSetConnectOption() をサポートしていますが、 DB2 CLI プログラムでは SQLSetConnectAttr() を使い始め、最新の規格に従うようお勧めします。 上記の関数と、その他の使用すべきでない関数の詳細については、
822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

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。詳細については、822ページの『64 ビット環境でサポートされない、使用すべきでない関数』を参照してください。

新しい関数への移行

たとえば、次の旧ステートメント

```
SQLSetConnectOption(
           *hdbc,
           SQL_AUTOCOMMIT,
           SQL_AUTOCOMMIT_OFF);
```

は、新しい関数を使用して、以下のように書き換えます。

```
SQLSetConnectAttr(
           *hdbc,
           SQL_ATTR_AUTOCOMMIT,
           SQL_AUTOCOMMIT_OFF,
           0);
```

DB2 ユニバーサル・データベース バージョン 5 以前の DB2 バージョンでは、 SQLSetConnectOption() を使用して、特定のステートメント属性と接続属性を設定できました。この動作は除去されたので、 SQLSetConnectAttr() を使ってステートメント属性を設定することはできなくなりました。この機能を利用していたバージョン 2 アプリケーションを移行する場合の詳細については、824ページの『SQLSetConnectAttr() を使用してステートメント属性のサブセットを設定する』を参照してください。

SQLSetCursorName - カーソル名を設定する

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、ステートメント・ハンドルが割り振られるときに、カーソル名を暗黙に生成するので、この関数は任意選択です。

構文

```
SQLRETURN   SQLSetCursorName (SQLHSTMT          StatementHandle,
                              SQLCHAR      FAR  *CursorName,
                              SQLSMALLINT       NameLength);
```

関数引き数

表 154. 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* 引き数の内容の長さ。

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。ハンドルが除去されるまで、名前はステートメント・ハンドルと関連付けられたままであるか、このステートメント・ハンドルに他の SQLSetCursorName() が呼び出されます。

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。このため、SQLSetCursorName() を使用せず、代わりに SQLGetCursorName() を呼び出して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固有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対立しないように、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 でカーソル名は識別子と見なされるため、先頭は英字 (a〜z、A〜Z) で、その後は数字 (0〜9)、英字、または下線文字 (_) の任意の組み合わせ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 155. 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〜z または A〜Z 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。

表 155. SQLSetCursorName SQLSTATE (続き)

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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効です。	*CursorName* は、ヌル・ポインターでした。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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

```
/* From the CLI sample TBMOD.C */
/* ... */
    /* set the name of the cursor */
    rc = SQLSetCursorName(hstmtSelect, (SQLCHAR *)"CURSNAME", SQL_NTS);
    STMT_HANDLE_CHECK( hstmtSelect, sqlrc);
```

参照

- 485ページの『SQLGetCursorName - カーソル名の入手』

SQLSetDescField - 記述子レコードの単一フィールドを設定する

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。

構文

```
SQLRETURN   SQLSetDescField  (SQLHDESC          DescriptorHandle,
                              SQLSMALLINT       RecNumber,
                              SQLSMALLINT       FieldIdentifier,
                              SQLPOINTER        ValuePtr,
                              SQLINTEGER        BufferLength);
```

関数引き数

表 156. 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。詳細については、707ページの『FieldIdentifier 引き数』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 156. 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 2 進数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。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* には次の値が有効です。 • *ValuePtr* が文字ストリングへのポインターであれば、 *BufferLength* はそのストリングまたは SQL_NTS の長さとなります。 • *ValuePtr* が 2 進数バッファーへのポインターであれば、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。 • *BufferLength* には負の値が入れられます。 • *ValuePtr* が文字ストリングや 2 進ストリング以外の値へのポインターであれば、*BufferLength* には必ず値 SQL_IS_POINTER が入ります。 • *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。この関数を

呼び出すことにより、対象となるフィールドが設定可能なものであれば、任意の記述子タイプにおける任意のフィールドを設定できます (このセクションで後ほど示される表を参照)。

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。

他の関数を呼び出して設定を行うと、関数の 1 回の呼び出しで複数の記述子フィールドを設定することが可能です。 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 (TYPE、 DATETIME_INTERVAL_CODE、OCTET_LENGTH、PRECISION、SCALE、 DATA_PTR、OCTET_LENGTH_PTR、および INDICATOR_PTR) 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 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() を呼び出せば、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) へのオフセットを追加することにより、バインド用バッファーを変更できます。この関数は、SQLBindCol() や SQLBindParameter() を呼び出すことなくバインド用バッファーを変更します。これにより、アプリケーションは他のフィールド、たとえば SQL_DESC_DATA_TYPE を変更することなく SQL_DESC_DATA_PTR を変更することが可能です。

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。

1. 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。
2. 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・タイプの適切な省略時値に設定します。タイプ属性フィールドの省略時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、省略時の属性は指定変更されます。
3. 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検査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 をヌル・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省略時値に初期設定したり、省略時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。以下の表は、記述子のタイプごとの初期設定が示されています。 “D” はそのフィールドが省略時値で初期設定されることを、また “ND” はそのフィールドが省略時値なし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省略時値です。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。

IRD のフィールドに省略時値があるのは、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、データの論理視点であるため、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 (詳しくは、 105ページの『記述子の使用』 を参照)。

SQLGetDescField() がアクセスできるフィールドは必然的に、SQLSetDescField() では設定できません。 SQLSetDescField() が設定できるフィールドは、以下の表に示されています。

ヘッダー・フィールドの初期設定は次のとおりです。

表 157. 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	省略時値:	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	省略時値:	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	省略時値:	ARD: ヌル・ポインター APD: ヌル・ポインター IRD: ヌル・ポインター IPD: ヌル・ポインター
SQL_DESC_BIND_OFFSET_PTR (SQLINTEGER *)			

表 157. ヘッダー・フィールドの初期設定 (続き)

	R/W: ARD: R/W	省略時値:	ARD: ヌル・ポインター
	APD: R/W		APD: ヌル・ポインター
	IRD: 未使用		IRD: 未使用
	IPD: 未使用		IPD: 未使用
SQL_DESC_BIND_TYPE (SQLINTEGER)			
	R/W: ARD: R/W	省略時値:	ARD: SQL_BIND_BY_COLUMN
	APD: R/W		APD: SQL_BIND_BY_COLUMN
	IRD: 未使用		IRD: 未使用
	IPD: 未使用		IPD: 未使用
SQL_DESC_COUNT (SQLSMALLINT)			
	R/W: ARD: R/W	省略時値:	ARD: 0
	APD: R/W		APD: 0
	IRD: R		IRD: D
	IPD: R/W		IPD: 0
SQL_DESC_ROWS_PROCESSED_PTR (SQLUINTEGER *)			
	R/W: ARD: 未使用	省略時値:	ARD: 未使用
	APD: 未使用		APD: 未使用
	IRD: R/W		IRD: ヌル・ポインター
	IPD: R/W		IPD: ヌル・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 (記述子タイプが範囲外です。) が返されます。

レコード・フィールドの初期設定は次のとおりです。

表 158. 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	省略時値:	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_BASE_COLUMN_NAME (SQLCHAR *)			

表 158. レコード・フィールドの初期設定 (続き)

R/W:	ARD: 未使用 APD: 未使用 IRD: R IPD: 未使用	省略時値:	ARD: 未使用 APD: 未使用 IRD: D IPD: 未使用
SQL_DESC_BASE_TABLE_NAME (SQLCHAR *)			
R/W:	ARD: 未使用 APD: 未使用 IRD: R IPD: 未使用	省略時値:	ARD: 未使用 APD: 未使用 IRD: D IPD: 未使用
SQL_DESC_CASE_SENSITIVE (SQLINTEGER)			
R/W:	ARD: 未使用 APD: 未使用 IRD: R IPD: R	省略時値:	ARD: 未使用 APD: 未使用 IRD: D IPD: D [a]
SQL_DESC_CATALOG_NAME (SQLCHAR *)			
R/W:	ARD: 未使用 APD: 未使用 IRD: R IPD: 未使用	省略時値:	ARD: 未使用 APD: 未使用 IRD: D IPD: 未使用
SQL_DESC_CONCISE_TYPE (SQLSMALLINT)			
R/W:	ARD: R/W APD: R/W IRD: R IPD: R/W	省略時値:	ARD: SQL_C_DEFAULT APD: SQL_C_DEFAULT IRD: D IPD: ND
SQL_DESC_DATA_PTR (SQLPOINTER)			
R/W:	ARD: R/W APD: R/W IRD: 未使用 IPD: 未使用	省略時値:	ARD: ヌル・ポインター APD: ヌル・ポインター IRD: 未使用 IPD: 未使用 [b]
SQL_DESC_DATETIME_INTERVAL_CODE (SQLSMALLINT)			

表 158. レコード・フィールドの初期設定 (続き)

R/W:	ARD: R/W	省略時値:	ARD: ND
	APD: R/W		APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_DATETIME_INTERVAL_PRECISION (SQLINTEGER)

R/W:	ARD: R/W	省略時値:	ARD: ND
	APD: R/W		APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_DISPLAY_SIZE (SQLINTEGER)

R/W:	ARD: 未使用	省略時値:	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_FIXED_PREC_SCALE (SQLSMALLINT)

R/W:	ARD: 未使用	省略時値:	ARD: 未使用
	APD: 未使用		APD: 未使用
	IRD: R		IRD: D
	IPD: R		IPD: D [a]

SQL_DESC_INDICATOR_PTR (SQLINTEGER *)

R/W:	ARD: R/W	省略時値:	ARD: ヌル・ポインター
	APD: R/W		APD: ヌル・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 *)

R/W:	ARD: 未使用	省略時値:	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_LENGTH (SQLUINTEGER)

表 158. レコード・フィールドの初期設定 (続き)

R/W:	ARD: R/W	省略時値:	ARD: ND
	APD: R/W		APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_LITERAL_PREFIX (SQLCHAR *)

R/W:	ARD: 未使用	省略時値:	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_LITERAL_SUFFIX (SQLCHAR *)

R/W:	ARD: 未使用	省略時値:	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_LOCAL_TYPE_NAME (SQLCHAR *)

R/W:	ARD: 未使用	省略時値:	ARD: 未使用
	APD: 未使用		APD: 未使用
	IRD: R		IRD: D
	IPD: R		IPD: D [a]

SQL_DESC_NAME (SQLCHAR *)

R/W:	ARD: 未使用	省略時値:	ARD: ND
	APD: 未使用		APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_NULLABLE (SQLSMALLINT)

R/W:	ARD: 未使用	省略時値:	ARD: ND
	APD: 未使用		APD: ND
	IRD: R		IRD: N
	IPD: R		IPD: ND

SQL_DESC_NUM_PREC_RADIX (SQLINTEGER)

表 158. レコード・フィールドの初期設定 (続き)

R/W:	ARD: R/W	省略時値:	ARD: ND
	APD: R/W		APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_OCTET_LENGTH (SQLINTEGER)

R/W:	ARD: R/W	省略時値:	ARD: ND
	APD: R/W		APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_OCTET_LENGTH_PTR (SQLINTEGER *)

R/W:	ARD: R/W	省略時値:	ARD: ヌル・ポインター
	APD: R/W		APD: ヌル・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	省略時値:	ARD: 未使用
	APD: 未使用		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	省略時値:	ARD: ND
	APD: R/W		APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_SCALE (SQLSMALLINT)

R/W:	ARD: R/W	省略時値:	ARD: ND
	APD: R/W		APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_SCHEMA_NAME (SQLCHAR *)

表 158. レコード・フィールドの初期設定 (続き)

R/W:	ARD: 未使用 APD: 未使用 IRD: R IPD: 未使用	省略時値:	ARD: 未使用 APD: 未使用 IRD: D IPD: 未使用
SQL_DESC_SEARCHABLE (SQLSMALLINT)			
R/W:	ARD: 未使用 APD: 未使用 IRD: R IPD: 未使用	省略時値:	ARD: 未使用 APD: 未使用 IRD: D IPD: 未使用
SQL_DESC_TABLE_NAME (SQLCHAR *)			
R/W:	ARD: 未使用 APD: 未使用 IRD: R IPD: 未使用	省略時値:	ARD: 未使用 APD: 未使用 IRD: D IPD: 未使用
SQL_DESC_TYPE (SQLSMALLINT)			
R/W:	ARD: R/W APD: R/W IRD: R IPD: R/W	省略時値:	ARD: SQL_C_DEFAULT APD: SQL_C_DEFAULT IRD: D IPD: ND
SQL_DESC_TYPE_NAME (SQLCHAR *)			
R/W:	ARD: 未使用 APD: 未使用 IRD: R IPD: R	省略時値:	ARD: 未使用 APD: 未使用 IRD: D IPD: D [a]
SQL_DESC_UNNAMED (SQLSMALLINT)			
R/W:	ARD: 未使用 APD: 未使用 IRD: R IPD: R/W	省略時値:	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_UNSIGNED (SQLSMALLINT)			

表 158. レコード・フィールドの初期設定 (続き)

R/W:	ARD: 未使用 APD: 未使用 IRD: R IPD: R	省略時値:	ARD: 未使用 APD: 未使用 IRD: D IPD: D [a]
SQL_DESC_UPDATABLE (SQLSMALLINT)			
R/W:	ARD: 未使用 APD: 未使用 IRD: R IPD: 未使用	省略時値:	ARD: 未使用 APD: 未使用 IRD: D 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 (記述子タイプが範囲外です。) が返されます。

b IPD 内の SQL_DESC_DATA_PTR フィールドは、整合性検査を強制実行するよう設定できます。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。

FieldIdentifier 引き数

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。省略時値は 1 です。このフィールドは、SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性によっても設定されます。

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。

このフィールドの省略時値は 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_ATTR_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。これらの配列は次のような名前になっています。

- 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。この配列の要素の数は、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。省略時設定では、このフィールドはヌル・ポインターに設定されます。SQL_DESC_ARRAY_STATUS_PTR フィールドがヌル・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要素を、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。

配列内の要素には、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、行状況配列のこの要素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 フィールドがヌル・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。

配列内の要素には、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・セットのど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。

配列内の要素には、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。

この配列の要素が 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ヌル・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、配列のすべての要素が SQL_ROW_PROCEED である場合と同じです。配列中のある要素が 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。配列内の要素には、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。

この配列の要素が 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ヌル・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、配列のすべての要素が 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。省略時設定では、これはヌル・ポインターに設定されます。このフィールドがヌル・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。

行方向のバインドについては、 『SQLFetchScroll』および『SQLBindParameter』の項にある説明を参照してください。

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 (省略時値) 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 (省略時値) 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 を基数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 (ヌル・ポインターに設定された *Target ValuePtr* 引き数のある SQLBindCol()、またはヌル・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_DSEC_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列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 [実装記述子] 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。これがヌル・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。省略時設定では、これはヌル・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不能である場合、この変数には空ストリングが入ります。

SQL_DESC_CASE_SENSITIVE [実装記述子] 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視点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 (日時およびインターバルといったデータ・タイプを含む) 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() への呼び出しにより設定できます。 SQL_DESC_TYPE は、 SQLSetDescField() または SQLSetDescRec() への呼び出しにより設定できます。

SQL_DESC_CONCISE_TYPE をインターバルまたは日時以外のデータ・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_CONCISE_TYPE をインターバルまたは日時コンサイス・データ・タイプに設定すると、 SQL_DESC_TYPE フィールドはそれに対応する冗長タイプ (SQL_DATETIME または SQL_INTERVAL) に設定され、 SQL_DESC_DATETIME_INTERVAL_CODE フィールドは適切なサブコードに設定されます。

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* がヌル・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 フィールドがヌル・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 フィールドがヌル・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、さらにそのデータ・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検査します。 724ページの『整合性検査』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・タイプにおいて当てはまります。

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 159. 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

このフィールドは、インターバル・データ・タイプに対しては DB2 CLI がサポートしていない他の値 (ここには示されていません) に設定することもできます。

SQL_DESC_DATETIME_INTERVAL_PRECISION [すべて] この SQLINTEGER レコード・フィールドには、 TYPE フィールドが SQL_INTERVAL (DB2 CLI はサポートしていない) になっている場合には、インターバル先行精度が入っています。

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 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 [実装記述子] この読み取り専用の SQLSMALLINT レコード・フィールドは、列が正確な数値列であり、精度が固定されていて位取りがゼロ以外である (MONEY データ・タイプなど) 場合に

は SQL_TRUE に、また列が正確な数値列ではなく、精度と位取り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 フィールドがヌル・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 がヌル・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 フィールドがヌル・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 2 進データ・タイプです。これは、固定長データ・タイプの最大文字長であるか、可変長データ・タイプの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ヌル終了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。

このフィールド内の値は、DB2 CLI バージョン 2 で定義された長さの値とは異なっている場合があります。

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不能であるデータ・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不能であるデータ・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 [実装記述子] 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 [実装記述子] 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識別子が無効 (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。 列名値は環境属性 SQL_ATTR_USE_LIGHT
_OUTPUT_SQLDA の影響を受ける場合があります。詳細については、 734ページの『SQLSetEnvAttr - 環境属性を設定する』を参照してください。

SQL_DESC_NULLABLE [実装記述子] IRD において、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 値が入っていない場合には 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ヌル可能であるため、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・タイプが正確な数値データ・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 10 進数であるためです。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 2 進データ・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 2 進タイプの場合、これは最大長 (バイト単位) になります。この値は常に、実装記述子の場合にはヌル終了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ヌル終了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 2 進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 フィールドがヌル・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 フィールドがヌル・ポインターである場合は、 DB2 CLI は文字ストリングと 2 進値がヌル終了であるとみなします。(2 進値は 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。

省略時設定では、これは 4 バイト値へのポインターです。SQL_ATTR_USE_2BYTES_OCTET_LENGTH 環境変数を設定して、これを変更することができます。詳しくは、734ページの『SQLSetEnvAttr - 環境属性を設定する』を参照してください。

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。省略時設定では、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、このフィールドは省略時で 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、正確な数値タイプの場合には桁数が、近似値タイプの場合には仮数 (2 進精度) におけるビット数が、また SQL_TYPE_TIME、SQL_TYPE_TIMESTAMP、SQL_INTERVAL_SECOND の各タイプの場合には小数表現による秒の部分の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。

このフィールド内の値は、 DB2 CLI バージョン 2 で定義された精度の値とは異なっている場合があります。

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位取りが入れられます。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。

このフィールド内の値は、 DB2 CLI バージョン 2 で定義された位取りの値とは異なっている場合があります。詳しくは、付録 D の『データ・タイプ』を参照してください。

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、DB2 CLI バージョン 2 における SQL_UNSEARCEABLE 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、DB2 バージョン 2 における 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、DB2 CLI バージョン 2 における 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、日時およびインターバルを除くすべてのデータ・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。日時データ・タイプおよびインターバル・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 や SQL_INTERVAL を指定します。

このフィールドに SQL_DATETIME または SQL_INTERVAL が入っているときは常に、 SQL_DESC_DATETIME_INTERVAL_CODE フィールドにはそのコンサイス・タイプに対応したサブコードが入っていなければなりません。日時データ・タイプの場合、SQL_DESC_TYPE には SQL_DATETIME が入れられ、 SQL_DESC_DATETIME_INTERVAL_CODE フィールドにはその特定の日時データ・タイプに対するサブコードが入れられます。インターバル・データ・タイプの場合、SQL_DESC_TYPE には SQL_INTERVAL が入れられ、SQL_DESC_DATETIME_INTERVAL_CODE フィールドにはその特定のインターバル・データ・タイプに対するサブコードが入れられます。

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()` への呼び出しにより設定できます。SQL_DESC_CONCISE_TYPE は、 `SQLBindCol()` または `SQLBindParameter()`、あるいは `SQLSetDescField()` への呼び出しにより設定できます。

SQL_DESC_TYPE をインターバルまたは日時以外のデータ・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。

SQL_DESC_TYPE が日時またはインターバルのいずれかの冗長データ・タイプ (SQL_DATETIME または SQL_INTERVAL) に設定され、SQL_DESC_DATETIME_INTERVAL_CODE フィールドが適切なサブコードに

設定されると、 SQL_DESC_CONCISE_TYPE フィールドはそれに対応するコンサイス・タイプに設定されます。 SQL_DESC_TYPE を日時またはインターバルのいずれかのコンサイス・タイプに設定しようとすると、 SQLSTATE HY021 (記述子情報に不整合 (Inconsistent descriptor information)) が返されます。

SQL_DESC_TYPE フィールドを SQLSetDescField() への呼び出しにより設定すると、以下のフィールドが次のような省略時値に設定されます。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 160. 省略時値

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 の省略時の精度に設定されます。
SQL_DATETIME	DB2 CLI はこのデータ・タイプをサポートしません。
SQL_INTERVAL	DB2 CLI はこのデータ・タイプをサポートしません。

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 [実装記述子] 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 [実装記述子] 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識別子が無効 (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識別子が無効 (Invalid descriptor field identifier)) が返されます。

SQL_DESC_UNSIGNED [実装記述子] 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。

整合性検査

整合性検査は、アプリケーションが ARD、APD、 IPD のいずれかの SQL_DESC_DATA_PTR フィールドに値を渡した時点で、 DB2 CLI が自動的に実行します。いずれかのフィールドに、他のフィールドとの不整合があると、 `SQLSetDescField()` は SQLSTATE HY021、「記述子情報に不整合 (Inconsistent descriptor information)」を返します。詳しくは、730ページの『整合性検査』にある `SQLSetDescRec()` の項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 161. 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 を戻します。)
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 を戻します。)
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	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY010	関数の順序エラーです。	*DescriptorHandle* が関連付けられていた StatementHandle に対して、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* の SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が戻されました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、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の行記述子を変更することはできません。	*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。

表 161. SQLSetDescField SQLSTATE (続き)

SQLSTATE	説明	解説
HY021	記述子情報が不整合です。	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。 整合性検査時に検査された記述子情報は、整合性がとれていませんでした。 (724ページの『整合性検査』を参照。)
HY091	記述子タイプが範囲外です。	*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。)

制約

なし。

例

該当するサンプルの一覧については、 sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内の README ファイルを参照してください。

参照

- 728ページの『SQLSetDescRec - 列またはパラメーター・データに複数の記述子フィールドを設定する』
- 501ページの『SQLGetDescField - 記述子レコードの単一フィールド設定を入手する』
- 507ページの『SQLGetDescRec - 記述子レコードの複数フィールド設定を入手する』
- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』

- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』

SQLSetDescRec - 列またはパラメーター・データに複数の記述子フィールドを設定する

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 162. 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 以上になっている必要があります。 *RecNumber* が SQL_DESC_COUNT 値より大きい場合、 *RecNumber* は SQL_DESC_COUNT 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 または SQL_INTERVAL 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。

表 162. SQLSetDescRec 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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 PRECISION フィールドを設定する値。
SQLSMALLINT	*Scale*	入力	記述子レコードに 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* をヌル・ポインターに設定して、 SQL_DESC_DATA_PTR フィールドをヌル・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* をヌル・ポインターに設定して、 SQL_DESC_OCTET_LENGTH_PTR フィールドをヌル・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* をヌル・ポインターに設定して、 SQL_DESC_INDICATOR_PTR フィールドをヌル・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 も ODBC で定義されていますが、 DB2 CLI ではサポートされていません。)

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。

整合性検査

アプリケーションが APD、ARD、または IPD の SQL_DESC_DATA_PTR フィールドを設定すると、 DB2 CLI により整合性検査が自動的に実行されます。 SQLSetDescRec() を呼び出すと、必ず整合性検査が実行されます。他のフィールドと整合性がとれていないフィールドが見つかると、 SQLSetDescRec() が SQLSTATE HY021「整合性がとれていない記述子に関する情報です。(Inconsistent descriptor information.)」を戻します。

アプリケーション記述子

アプリケーションが APD、ARD、または IPD の SQL_DESC_DATA_PTR フィールドを設定すると、 DB2 CLI は SQL_DESC_TYPE の値とその SQL_DESC_TYPE フィールドに適用可能な値が有効で整合性がとれているかどうか検査します。この検査は、SQLBindParameter() または SQLBindCol() が呼び出されたり、 APD、ARD、または IPD の SQLSetDescRec() が呼び出されたりすると、必ず実行されます。この整合性検査には、アプリケーション記述子フィールドに関する以下の検査も含まれます。

- SQL_DESC_TYPE フィールドは、有効な C タイプか SQL タイプのどちらかになっていなければならない。 SQL_DESC_CONCISE_TYPE フィールドは、有効な C タイプか SQL タイプのどちらかになっていなければならない。

- SQL_DESC_TYPE フィールドに数値タイプが示されている場合は、SQL_DESC_PRECISION フィールドと SQL_DESC_SCALE フィールドが有効かどうか確認する。
- SQL_DESC_CONCISE_TYPE フィールドが時刻データ・タイプの場合は、SQL_DESC_PRECISION フィールドの秒精度が有効かどうか検査する。

IPD の SQL_DESC_DATA_PTR フィールドは通常設定されませんが、アプリケーションはこのフィールドを設定して、 IPD フィールドの整合性検査を強行できます。整合性検査は、IRD では実行できません。 IPD の SQL_DESC_DATA_PTR フィールドに設定される値は実際には保管されず、`SQLGetDescField()` または `SQLGetDescRec()` では取り出せません。この設定は、整合性検査を強行する目的で行われます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 163. SQLSetDescRec SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージです。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
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。

表 163. SQLSetDescRec SQLSTATE (続き)

SQLSTATE	説明	解説
HY000	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY010	関数の順序エラーです。	*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* の SQLExecute() または SQLExecDirect() が呼び出され、SQL_NEED_DATA が戻されました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、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・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。
HY016	インプリメンテーションの行記述子を変更することはできません。	*DescriptorHandle* 引き数は、IRD と関連していました。
HY021	記述子情報が不整合です。	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。 整合性検査時に検査された記述子情報は、整合性がとれていませんでした。 (724ページの『整合性検査』にある SQLSetDescField() を参照してください。)

制約

なし。

例

該当するサンプルの一覧については、 sqllib¥samples¥cli (または sqllib/samples/cli) サブディレクトリー内の README ファイルを参照してください。

参照

- 696ページの『SQLSetDescField - 記述子レコードの単一フィールドを設定する』

- 501ページの『SQLGetDescField - 記述子レコードの単一フィールド設定を入手する』
- 507ページの『SQLGetDescRec - 記述子レコードの複数フィールド設定を入手する』
- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』

SQLSetEnvAttr - 環境属性を設定する

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 164. SQLSetEnvAttr 引き数

データ・タイプ	引き数	使用法	説明
SQLHENV	EnviornmentHandle	入力	環境ハンドル
SQLINTEGER	Attribute	入力	設定する環境属性。属性とその説明のリストについては、『環境属性』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。

環境属性

SQL_ATTR_CONNECTION_POOLING

環境レベルでの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。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。これが省略時値です。

- 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。この呼び出しは、接続のプール処理が使用可能になっている共用環境をアプリケーションが割り振る前に呼び出してください。 SQLSetEnvAttr() への呼び出しの環境ハンドルがヌルに設定されているので、SQL_ATTR_CONNECTION_POOLING が処理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、暗黙的に共用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ヌル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用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用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用環境だけです。

SQL_ATTR_CONNECTTYPE

このアプリケーションを整合分散環境で実行するか、または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。有効値は以下のとおりです。

- **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 SQLTransact() 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回復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。

これが省略時値です。

- 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。この属性は、SQLSetConnectAttr 関数を使用して設定することもできます。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。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、 SQLSetEnvAttr() が ODBC でサポートされていないので、 SQLSetConnectAttr() を使用して各接続の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ドライバー・マネージャー) はどの接続をプールから再利用するか決定します。 DB2 CLI (またはドライバー・マネージャー) は、呼び出しの接続オプション、アプリケーションがキーワードに設定した接続属性、およびプールにある接続の接続属性の突き合わせを試行します。この属性値で、突き合わせ基準の精度レベルが判別されます。

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。これが省略時値です。

- SQL_CP_RELAXED_MATCH = 接続ストリング・キーワードが一致する接続が使用されます。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

アプリケーションがセットアップしたい最大並行接続数に対応する 32 ビット整数値。省略時値は **0** で、これは最大値がないことを意味します。アプリケーションはシステム資源で許可される個数の接続を設定することができます。整数値は 0 または正の数でなければなりません。

これは、アプリケーション・ベースでの接続の最大数のための管理プログラムとして使用できます。

OS/2 で NetBIOS プロトコルが使用中の場合に、この値は、アプリケーションで並行してセットアップされる接続 (NetBIOS セッション) の数に対応します。 OS/2 NetBIOS の値の範囲は、1 から 254 です。 0 (省略時値) を指定すると、**5 つの** *予約済み* 接続が行われます。他のアプリケーションは、*予約済み NetBIOS セッション*を使用できません。このパラメーターで指定された接続数は、 DB2 NetBIOS プロトコルがリモート・サーバーに接続するのに使用するアダプターに適用されます (アダプター番号は NetBIOS ノードのノード・ディレクトリーで指定します)。

最初の接続が確立される時点で有効な値は、使用しようとしている値です。最初の接続がすでに確立されていると、この値を変更しようとしても拒否されます。アプリケーションが SQL_ATTR_MAXCONN 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てきました。 ODBC アプリケーションは、`SQLSetEnvAttr()` がサポートされていないので、接続レベルでこ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_ODBC_VERSION

特定の機能性に 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 (関数の順序エラーです。) 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。
  - DB2 CLI は、SQLError()、SQLGetDiagField()、SQLGetDiagRec() が呼び出されると、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。
  - DB2 CLI は、SQLError()、SQLGetDiagField()、SQLGetDiagRec() が呼び出されると、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ヌル終了の使用を制御する 32 ビット整数値。有効値は以下のとおりです。

- **SQL_TRUE**: DB2 CLI は、ヌル終了を使用して出力文字ストリングの長さを指示します。

  これが省略時値です。
- SQL_FALSE: DB2 CLI は、ヌル終了を出力文字ストリングに使用しません。

この属性の影響を受ける CLI 関数は、文字ストリング・パラメーターのある環境 (およびその環境で割り振られている接続とステートメント) について呼び出されたすべての関数です。

この属性は、この環境に接続ハンドルが割り振られていないときにのみ、設定することができます。

SQL_ATTR_PROCESSCTRL

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。

SQLSetEntAttr() 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 呼び出しが逐次化されます。これを設定すると、DB2 CLI は、CLI への呼び出しを逐次化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 を検査する必要がなくなります。

SQL_ATTR_SYNC_POINT

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間で選択できるようにする 32 ビット整数値。有効値は以下のとおりです。

- SQL_ONEPHASE: 複数のデータベース・トランザクションの各データベースが行う作業をコミットするときは、 1 フェーズ・コミットが使用されます。データ保全性を確かにするには、 1 つのトランザクションで 2 つ以上のデータベースを更新することがないようにします。 1 つのトランザクションで実行された更新のうち最初のデータベースだけが、そのトランザクションでのただ 1 つの更新側になり、それ以外のすべてのデータベースはアクセスされても読み取り専用になります。このトランザクション内でこれらの読み取り専用のデータベースを更新しようとしても拒否されます。
- 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。

分散作業単位 (トランザクション) の詳細については、*SQL 解説書* を参照してください。

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。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、 SQLSetEnvAttr() が ODBC でサポートされていないので、SQLSetConnectAttr() を使用して各接続の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 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

SQL_DESC_OCTET_LENGTH_PTR 記述子フィールド (APD または ARD のどちらかとして使用される) が、 2 バイト値と 4 バイト値 (省略時値) のどちらを指すポインターであるかを指定する、4 バイト整数値。有効値は以下のとおりです。

- SQL_TRUE: SQL_DESC_OCTET_LENGTH_PTR は、2 バイト値へのポインターです。
- **SQL_FALSE**: SQL_DESC_OCTET_LENGTH_PTR は、 4 バイト値へのポインターです (省略時値)。

記述子フィールドの詳細については、696ページの『SQLSetDescField - 記述子レコードの単一フィールドを設定する』を参照してください。

この属性はいつでも設定することができ、記述子関数が次に呼び出されるときに有効となります。この属性は、SQLGetData() 呼び出しの *StrLen_or_IndPtr* 値には影響を与えません。

注: これは、IBM 定義の拡張機能です。

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。有効値は以下のとおりです。

- 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 (省略時値)。

注: これは、IBM 定義の拡張機能です。

SQL_CONNECTTYPE

この *Attribute* は、 SQL_ATTR_CONNECTTYPE に置き換えられました。

SQL_MAXCONN

この *Attribute* は、 SQL_ATTR_MAXCONN に置き換えられました。

SQL_SYNC_POINT

この *Attribute* は、 SQL_ATTR_SYNC_POINT に置き換えられました。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 165. SQLSetEnvAttr SQLSTATE

SQLSTATE	説明	解説
HY009	引き数値が無効です。	*fOption* 値が与えられているので、引き数 *vParam* に指定されたのは無効な値でした。
HY011	この段階で操作は無効です。	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。	指定済みの *Attribute* 値が与えられているので、 **ValuePtr* に指定されたのは無効な値でした。
HY090	無効なストリングまたはバッファー長。	*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 165. SQLSetEnvAttr SQLSTATE (続き)

SQLSTATE	説明	解説
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	ドライバーが機能していません。	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。

制約

なし。

例

61ページの『DB2 複数サイト更新の例』も参照してください。

```
/* From the CLI sample PCALL.C */
/* ... */
    /* enable sending column names over the network */
    sqlrc = SQLSetEnvAttr( henv,
                        SQL_ATTR_USE_LIGHT_OUTPUT_SQLDA,
                        ( SQLPOINTER ) SQL_FALSE,
                        SQL_FALSE
                      ) ;
```

参照

- 526ページの『SQLGetEnvAttr - 現行の環境属性値を検索する』

SQLSetParam - バッファーまたは LOB ロケーターに 1 つのパラメーター・マーカーをバインドする

使用すべきでない関数

注:

ODBC バージョン 3 では、SQLSetParam() は使用すべきでない関数であり、 SQLBindParameter() に置き換えられました。詳細については、278ページの『SQLBindParameter - バッファーまたは LOB ロケーターにパラメーター・マーカーをバインドする』を参照してください。

このバージョンの DB2 CLI では引き続き SQLSetParam() をサポートしていますが、 DB2 CLI プログラムでは SQLBindParameter() を使い始め、最新の規格に従うようお勧めします。 上記の関数と、その他の使用すべきでない関数の詳細については、
822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

等価の関数: SQLBindParam()

CLI 関数 SQLBindParam() は、関数 SQLSetParam() と厳密に同じです。どちらも同じ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。

SQLSetPos - 行セット (Rowset) でカーソル位置を設定する

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,
                             SQLUSMALLINT      RowNumber,
                             SQLUSMALLINT      Operation,
                             SQLUSMALLINT      LockType);
```

関数引き数

表 166. 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 だと、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、745ページの『RowNumber 引き数』を参照してください。

表 166. SQLSetPos 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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 追加情報については、746ページの『Operation 引き数』を参照してください。 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。詳しくは、306ページの『SQLBulkOperations - 行のセットの追加、更新、削除、または取り出し』を参照してください。
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、748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 だと、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_POISTION *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、SQLFetchScroll() にある行ごとのバインドおよび列ごとのバインドの説明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 操作 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要素が、

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、749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要素を 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) 実装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、749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。

詳しくは、306ページの『SQLBulkOperations - 行のセットの追加、更新、削除、または取り出し』を参照してください。

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、あるステートメントが接続で得たロックは、同じ接続の別のステートメントによってロック解除可能です。

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 167. 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ロック解除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。
SQL_LOCK_UNLOCK	DB2 CLI ではサポートされない。行をロック解除する。

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。配列の各要素は、SQL_ROW_PROCEED (省略時値) 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要素数は、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。)

行状況配列については、434ページの『SQLFetch - 次の行の取り出し』を参照してください。

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 168. SQLSetPos SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)
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 2 進データは、非ブランク文字または非ヌル文字の 2 進データの短縮形となります。
01S01	行の中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。)
01S07	小数が切り捨てられました。	*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	変換が無効です。	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。

表 168. SQLSetPos SQLSTATE (続き)

SQLSTATE	説明	解説
07009	記述子索引が無効です。	引き数 *Operation* は、SQL_REFRESH または SQL_UPDATE で、列は、結果セットの列数より大きい列番号でバインドされました。
21S02	派生表の程度が列リストと一致しません。	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、読取専用であるか、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 2 進値の割り当てが、非ブランク文字 (文字の場合) または非ヌル文字 (2 進数の場合) 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	日時形式が無効です。	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。

表 168. SQLSetPos SQLSTATE (続き)

SQLSTATE	説明	解説
HY000	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY008	操作が取り消されました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。
HY010	関数の順序エラーです。	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。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。 SQLExecute()、SQLExecDirect()、または SQLSetPos() が *StatementHandle* で呼び出され、 SQL_NEED_DATA を戻しました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、この関数が呼び出されました。 SQLFetchScroll() が呼び出される前に、または SQLFetch() が呼び出された後で、しかも SQLFreeStmt() が SQL_CLOSE オプションで呼び出された後に、バージョン 2 DB2 CLI アプリケーションが *StatementHandle* の SQLSetPos() を呼び出しました。
HY011	この段階で操作は無効です。	バージョン 2 DB2 CLI 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、 SQLSetPos() が、SQLFetch()、SQLFetchScroll()、または SQLExtendedFetch() の呼び出し前に呼び出されました。

表 168. SQLSetPos SQLSTATE (続き)

SQLSTATE	説明	解説
HY090	ストリングまたはバッファー長が無効です。	*Operation* 引き数は SQL_ADD、SQL_UPDATE、または SQL_UPDATE_BY_BOOKMARK であり、データ値はヌル・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、データ値はヌル・ポインターでなく、列長値は 0 よりも小さいが、SQL_DATA_AT_EXEC、 SQL_COLUMN_IGNORE、SQL_NTS、または SQL_NULL_DATA ではないか、または SQL_LEN_DATA_AT_EXEC_OFFSET 以下でした。 長さ / 標識バッファーの値は SQL_DATA_AT_EXEC でした。 SQL タイプは、SQL_LONGVARCHAR、SQL_LONGVARBINARY、または他のデータ・ソース固有のデータ・タイプでした。また、SQLGetInfo() の情報タイプ SQL_NEED_LONG_DATA_LEN は “Y” でした。
HY092	オプション・タイプが範囲外です。	*Operation* 引き数は 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	ドライバーが機能していません。	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。

表 168. SQLSetPos SQLSTATE (続き)

SQLSTATE	説明	解説
HYT00	タイムアウト期間が満了しました。	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() を通じて設定します。

制約

なし。

例

```
/* From the CLI sample TBREAD.C */
/* ... */
    sqlrc = SQLSetPos( hstmt, 3, SQL_POSITION, SQL_LOCK_NO_CHANGE);
    STMT_HANDLE_CHECK( hstmt, sqlrc);
    sqlrc = SQLGetData(hstmt, 0, SQL_C_LONG, bookmark.val, 4, &bookmark.ind);
    STMT_HANDLE_CHECK( hstmt, sqlrc);
    sqlrc = SQLSetStmtAttr( hstmt,
                        SQL_ATTR_FETCH_BOOKMARK_PTR,
                        (SQLPOINTER) bookmark.val,
                        0);
    STMT_HANDLE_CHECK( hstmt, sqlrc);

/* ... */
    sqlrc = SQLSetPos( hstmt, 3, SQL_POSITION, SQL_LOCK_NO_CHANGE);
    STMT_HANDLE_CHECK( hstmt, sqlrc);
    sqlrc = SQLGetData(hstmt, 0, SQL_C_LONG, bookmark.val, 4, &bookmark.ind);
    STMT_HANDLE_CHECK( hstmt, sqlrc);
    sqlrc = SQLSetStmtAttr( hstmt,
                        SQL_ATTR_FETCH_BOOKMARK_PTR,
                        (SQLPOINTER) bookmark.val,
                        0);
    STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参照

- 254ページの『SQLBindCol - アプリケーション変数または LOB ロケーターに列をバインドする』
- 321ページの『SQLCancel - ステートメントを取り消す』
- 446ページの『SQLFetchScroll - バインド列すべての行セットを取り出し、データを返す』
- 501ページの『SQLGetDescField - 記述子レコードの単一フィールド設定を入手する』
- 507ページの『SQLGetDescRec - 記述子レコードの複数フィールド設定を入手する』

- 696ページの『SQLSetDescField - 記述子レコードの単一フィールドを設定する』
- 728ページの『SQLSetDescRec - 列またはパラメーター・データに複数の記述子フィールドを設定する』
- 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』

SQLSetStmtAttr - 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。

構文

```
SQLRETURN   SQLSetStmtAttr   (SQLHSTMT          StatementHandle,
                              SQLINTEGER        Attribute,
                              SQLPOINTER        ValuePtr,
                              SQLINTEGER        StringLength);
```

関数引き数

表 169. 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	設定オプション。 759ページの『ステートメント属性』にリストされる。

表 169. SQLSetStmtAttr 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*ValuePtr	入力	*Attribute* が ODBC 定義の属性で、 *ValuePtr* が文字ストリングか 2 進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、*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* 引き数の設定により、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* はストリングまたは SQL_NTS の長さです。 • *ValuePtr* が 2 進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。これは負の値を *StringLength* に入れます。 • *ValuePtr* が文字ストリングまたは 2 進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。
SQLINTEGER	*StringLength*	入力	*ValuePtr* が文字ストリングまたは 2 進数バッファーへのポインターの場合、この引き数の長さは **ValuePtr* になります。 *ValuePtr* が文字ストリングまたは 2 進数バッファー以外へのポインターの場合、 *StringLength* の値は SQL_IS_POINTER になります。 *ValuePtr* がポインターでない場合、 *StringLength* の値は SQL_IS_NOT_POINTER になります。

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

テートメントが除去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。

データ・ソースが、*ValuePtr* に指定されている値をサポートしない場合、ステートメント属性の中には、類似の値の代替となるものもあります。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、*Attribute* が SQL_ATTR_CONCURRENCY、 **ValuePtr* が SQL_CONCUR_ROWVER で、データ・ソースがこれをサポートしない場合、 DB2 CLI は SQL_CONCUR_VALUES を代用し、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、代用された値を判別します。

ValuePtr によって設定した情報の形式は、*Attribute* の指定により異なります。SQLSetStmtAttr() が受け入れる属性情報の形式は、ヌル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。それぞれの形式の注記は、その属性の説明にあります。この形式は、SQLGetStmtAttr() のそれぞれの属性ごとに戻される情報に適用されます。 SQLSetStmtAttr() の *ValuePtr* 引き数が指し示す文字ストリングの長さは、 *StringLength* です。

記述子の設定によってステートメント属性を設定する

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に 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 APD または 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。 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() 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 fHandleType 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 170. 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	説明
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

ステートメント属性

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 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。

注: DB2 CLI バージョン 2 のステートメント属性は、すべて名前変更されています。バージョン 2 では SQL_ で始まっていましたが、現在は SQL_ATTR_ になっています。

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。この属性の値が 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。この属性の値が 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_ASYNC_ENABLE (DB2 CLI v2)

指定のステートメントでの関数コールを非同期で実行するかどうかを指定する 32 ビット整数値。

- **SQL_ASYNC_ENABLE_OFF** = Off (省略時値)
- SQL_ASYNC_ENABLE_ON = On

関数が非同期で呼び出されると、元の関数が SQL_STILL_EXECUTING 以外のコードを戻すまでは、元の関数 SQLAllocHandle()、SQLCancel()、SQLSetStmtAttr()、 SQLGetDiagField()、SQLGetDiagRec()、または SQLGetFunctions() だけがそのステートメントで、またはそのステートメントに関連した接続で呼び出すことができます。そのステートメントで、またはそのステートメントに関連した接続で呼び出されるその他の関数は、 HY010 の SQLSTATE を伴う SQL_ERROR を戻します (関数順序エラー)。関数は、他のステートメントで呼び出せます。

DB2 CLI はステートメント・レベル非同期実行をサポートしているため、ステートメント属性 SQL_ATTR_ASYNC_ENABLE を設定できます。その初期値は、ステートメント・ハンドル割り当て時と同じ名前を持つ接続レベル属性値と同じです。

一般に、アプリケーションは、シングル・スレッドのオペレーティング・システム上で、非同期で関数を実行する必要があります。マルチ・スレッドのオペレーティング・システムの場合は、同じスレッドで非同期に実行するのではなく、別々のスレッドでアプリケーションを実行すべきです。マルチスレッドのオペレーティング・システムだけで動作する DB2 CLI アプリケーションは、非同期実行のサポートが必要ありません。詳しくは、50ページの『マルチスレッドのアプリケーション作成』、および 145ページの『CLI の非同期実行』を参照してください。

以下の関数は、非同期で実行できます。

SQLColAttribute()	SQLGetTypeInfo()
SQLColumnPrivileges()	SQLMoreResults()
SQLColumns()	SQLNumParams()
SQLCopyDesc()	SQLNumResultCols()
SQLDescribeCol()	SQLParamData()
SQLDescribeParam()	SQLPrepare()
SQLExecDirect()	SQLPrimaryKeys()
SQLExecute()	SQLProcedureColumns()
SQLFetch()	SQLProcedures()
SQLFetchScroll()	SQLPutData()
SQLForeignKeys()	SQLSetPos()
SQLGetData()	SQLSpecialColumns()
SQLGetDescField() 1*	SQLStatistics()
SQLGetDescRec() 1*	SQLTablePrivileges()
SQLGetDiagField()	SQLTables()
SQLGetDiagRec()	

1* これらの関数は、記述子がインプリメンテーション記述子であり、アプリケーション記述子でない場合にのみ、非同期で呼び出せます。

非同期実行は、 ASYNCENABLE DB2 CLI/ODBC 構成キーワードを使用しても設定可能です。詳しくは、168ページの『db2cli.ini の構成』を参照してください。

SQL_ATTR_BIND_TYPE (DB2 CLI v2)

このステートメント・ハンドルを指定して SQLExtendedFetch() 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 *vParam* に値 **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 *vParam* に値を指定すると、*行ごとのバインド* が選択されます。

行ごとのバインドで、*vParam* 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が指定の長さで増分された際に、必ず結果が次の行の同じ列の先頭を指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、このオプションの省略時値です。

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、カーソルがオープンするとエラーが戻されます。これが省略時値です。
- 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、この省略時値は役立ちません。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、厳密でない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 の省略時値は、SQL_CONCUR_READ_ONLY です。キー・セット・カーソルの省略時値は、SQL_CONCUR_VALUES です。

この属性は SQLSetScrollOptions() の Concurrency 引き数を使用して設定することもできます。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() や 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、厳密でない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** (これは省略時値です)
- SQL_CURSOR_HOLD_OFF

StatementHandle が最初に割り当てられるときの省略時値は、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することによっても、カーソル保留を設定することができます。詳しくは、168ページの『db2cli.ini の構成』を参照してください。

注: このオプションは、IBM 拡張機能です。

SQL_ATTR_CURSOR_SCROLLABLE (DB2 CLI v6)

アプリケーションが要求す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** = スクロール可能カーソルは、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。これが省略時値です。
- SQL_SCROLLABLE = スクロール可能カーソルが、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。スクロール可能カーソルの詳細については、75ページの『スクロール可能カーソル』を参照してください。

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。これが省略時値です。
- 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。
- SQL_CURSOR_STATIC = 結果セット内のデータは、静的です。これが省略時値です。
- 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。

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。

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_CURSOR_DYNAMIC

  この値が使用されると、DB2 CLI は、ステートメント属性を SQL_CURSOR_STATIC SQL_CURSOR_FORWARD_ONLY に設定し、 SQLSTATE 01S02 (オプション値が変更された) を戻します。この場合、アプリケーションは `SQLGetStmtAttr()` を呼び出して、実際の値を照会する必要があります。

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、その発行時に実行されます。
- **SQL_DEFERRED_PREPARE_ON** (省略時値) = 据え置き準備を使用可能にする。付随する実行要求が発行されるまで、PREPARE 要求の実行は遅れます。それから、2 つの要求は、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に、 (2 つではなく) 1 つのコマンド / 応答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。

省略時設定の動作は DB2 バージョン 2 から変更されました。現在は据え置き準備が省略時値になっており、必要なら明示的にオフにする必要があります。

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。このことは、これらの値を使用して SQL ステートメントを継続するかどうかを決めているユーザーにとって重要となります。

このオプションは、 CLI/ODBC オプション DB2ESTIMATE がゼロ以外の値に設定されていれば、オフになります。

DEFERREDPREPARE DB2 CLI/ODBC 構成キーワードを使用することによっても、据え置き準備を設定することができます。詳しくは、168 ページの『db2cli.ini の構成』を参照してください。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_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 (省略時値)。

  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。
  - ステートメントのブロッキングが適格でない場合
  - カーソル・タイプが SQL_CURSOR_FORWARD_ONLY 以外である場合

EARLYCLOSE DB2 CLI/ODBC 構成キーワードを使用することによっても、優先クローズ機能を設定することができます。詳しくは、168ページの『db2cli.ini の構成』を参照してください。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_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 の省略時値は、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)

2 進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、省略時で 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_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_MAX_LENGTH (DB2 CLI v2)

単一文字または 2 進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。 SQL_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。 *vParam* の省略時値は **0** です。 0 の場合は、DB2 CLI が文字または 2 進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。 *vParam* の省略時値は **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、識別子として扱われます。大文字小文字の区別はありません。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、識別子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。

  これが省略時値です。

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_NODESCRIBE (DB2 CLI v2)

このステートメント属性は、DB2 CLI バージョン 5 以降では不要です。現在では、DB2 CLI は省略時で据え置き準備を使用するため、SQLSetColAttributes() の機能性を必要としません。詳しくは、827ページの『デフォルトでの据え置き準備』を参照してください。

DB2 CLI が、結果セットの列属性を自動的に記述するか、SQLSetColAttributes() によってアプリケーションから通知されるまで待たなければならないかを指定する 32 ビット整数。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_NOSCAN (DB2 CLI v2)

DB2 CLI が SQL を走査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 ストリングを走査して、エスケープ文節シーケンスを探します。これが省略時値です。
- SQL_NOSCAN_ON - SQL ストリングを走査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走査をオフにすることを選択できます。この選択を行うと、走査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。 n を 0 (省略時値) に設定すると、この文節は追加されません。

OPTIMIZE FOR n ROWS ステートメントの影響に関する詳細については、 *管理の手引き* を参照してください。

この値は、 OPTIMIZEFORNROWS DB2 CLI/ODBC 構成キーワードを使用して設定することもできます。詳しくは、168ページの『db2cli.ini の構成』を参照してください。

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、システム表が走査されないよう、この呼び出しを最適化します。

  呼び出しが最適化されると、 `SQLColumns()` からの COLUMN_DEF 情報 (列の省略時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 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。

この値は、 OPTIMIZESQLCOLUMNS DB2 CLI/ODBC 構成キーワードを使用して設定することもできます。詳しくは、168ページの『db2cli.ini の構成』を参照してください。

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ヌルの場合、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。これは省略時でヌルに設定されます。

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ても、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** (省略時値) 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時にパラメーターを無視するために使用される 16 ビット無符号整数値の配列を示す 16 ビット無符号整数 * 値。それぞれの値は、SQL_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、配列のどの要素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、PARAMSET_SIZE が 1 より大きい場合にのみ必要です。

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・セットのど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 複合 SQL のどちらを介して行うかを判別する 32 ビット整数値。有効値は以下のとおりです。

- **SQL_ATOMIC_YES** - 基礎処理で ATOMIC 複合 SQL を使用します。これが省略時値です。
- SQL_ATOMIC_NO - 基礎処理で NON-ATOMIC 複合 SQL を使用します。

ATOMIC 複合 SQL は、以下のものでは使用することができません。バージョン 2.1 以前の共通サーバー用の DB2 または DRDA サーバー。上記のサーバーの 1 つに接続されるときに 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。これがヌル・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。

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事前取り出しするかどうか (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入手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。

有効値は以下のとおりです。

- SQL_PREFETCH_ON - サーバーがサポートしていれば、事前取り出しが行われます。
- SQL_PREFETCH_OFF - 事前取り出しは行われません。これが省略時値です。

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。

使用されている現行値は、0、1、2、3、5、7、および 9 です。照会最適化レベルの詳細については、 *SQL 解説書* の SET CURRENT QUERY OPTIMIZATION コマンドを参照してください。

SQL_ATTR_QUERY_TIMEOUT (DB2 CLI v2)

アプリケーションに戻る前に、 SQL ステートメントが実行されるまで待機する秒数を示す 32 ビット整数値。 Windows 3.1 以外では、DB2 CLI は **0** しかサポートしません。 0 はタイムアウトがないという意味です。

注: Windows 3.1 上では、このオプションを設定して、長時間の照会を終了させることができます。このオプションを指定すると、基礎となる Windows 3.1 接続コードはダイアログ・ボックスを表示して、指定秒数が経過したことを知らせ、照会の続行または中断のいずれかを指定するようプロンプトが出されます。

このオプションは Windows 3.1 以外のプラットフォームでは無効であり、 **S1**C00 が戻されます。

SQL_ATTR_RETRIEVE_DATA (DB2 CLI v2)

32 ビット整数値。

- **SQL_RD_ON** = SQLFetchScroll() および DB2 CLI バージョン 5.2 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す。これが省略時値です。
- SQL_RD_OFF = SQLFetchScroll() および DB2 CLI バージョン 5.2 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。

SQL_RETRIEVE_DATA を SQL_RD_OFF に設定することにより、アプリケーションは行が存在するかどうかを検査したり、あるいは検索行のオーバーヘッドを損なわずにブックマークを検索して行を探すことができます。

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。省略時値は 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・カーソルに指定でき、 SQLSetScrollOptions() の RowsetSize 引き数を使用して設定することもできます。

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ヌルの場合、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。これは省略時でヌルに設定されます。

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() の省略時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。この配列の要素の数は、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_ATTR_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。詳しくは 78ページの『結果セットから返される行セットの指定』を参照してください。

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。省略時値は **1** であり、これは単一の SQLFetch() 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更新および削除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。

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 ドライバー・マネージャーは、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。

トランザクション分離レベルは、 TXNISOLATION DB2 CLI/ODBC 構成キーワードを使用して設定することもできます。詳しくは、168ページの『db2cli.ini の構成』を参照してください。

この属性 (または対応するキーワード) を使用できるのは、省略時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: IBM 拡張機能により、このオプションをステートメント・レベルに設定することができます。

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 (省略時値)
- 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与えます。

カーソルで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このオプションで SQL_UB_VARIABLE 値を指定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

表 171. SQLSetStmtAttr SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を戻します。)

表 171. SQLSetStmtAttr SQLSTATE (続き)

SQLSTATE	説明	解説
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 を戻します。)
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	一般的なエラーです。	特定の SQLSTATE がなかった場合のエラーが発生しました。 SQLGetDiagRec() により **MessageText* バッファーに返されたエラー・メッセージに、そのエラーと原因が記述されています。
HY001	メモリーの割り振り失敗です。	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を割り当てられませんでした。
HY009	引き数値が無効です。	*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	関数の順序エラーです。	*StatementHandle* の非同期実行関数が呼び出され、この関数が呼び出された時点でまだ実行中でした。 *StatementHandle* のために SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が戻されました。データがすべての実行時データ・パラメーターまたは列用に送られる前に、この関数が呼び出されました。
HY011	この段階で操作は無効です。	*Attribute* が、SQL_ATTR_CONCURRENCY、SQL_ATTR_CURSOR_TYPE、SQL_ATTR_SIMULATE_CURSOR、または SQL_ATTR_USE_BOOKMARKS であり、ステートメントが準備済みでした。
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* 引き数が SQL_ATTR_APP_ROW_DESC または SQL_ATTR_APP_PARAM_DESC であり、 **ValuePtr* の値が、暗黙的に割り当てられた記述子ハンドルでした。

表 171. SQLSetStmtAttr SQLSTATE (続き)

SQLSTATE	説明	解説
HY024	属性値が無効です。	指定済みの *Attribute* 値が与えられているので、 **ValuePtr* に指定されたのは無効な値でした。 (DB2 CLI がこの SQLSTATE を戻すのは、 SQL_ATTR_ACCESS_MODE や SQL_ATTR_ASYNC_ENABLE 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に対しては、ドライバーで **ValuePtr* の指定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長が無効です。	*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。	DB2 CLI のこのバージョンでは、引き数 *Attribute* の指定値が無効です。
HYC00	ドライバーが機能していません。	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・ソースによりサポートされていませんでした。

制約

なし。

例

`SQLFetchScroll()` を参照してください。

参照

- 321ページの『SQLCancel - ステートメントを取り消す』
- 480ページの『SQLGetConnectAttr - 現行属性設定を入手する』
- 592ページの『SQLGetStmtAttr - ステートメント属性の現行設定値を入手する』
- 663ページの『SQLSetConnectAttr - 接続属性を設定する』
- 696ページの『SQLSetDescField - 記述子レコードの単一フィールドを設定する』

SQLSetStmtOption - ステートメント・オプションの設定

使用すべきでない関数

注:

ODBC バージョン 3 では、SQLSetStmtOption() は使用すべきでない関数であり、代わりに SQLSetStmtAttr() を使用します。詳しくは 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』を参照してください。

DB2 CLI のこのバージョンでは、引き続き SQLSetStmtOption() がサポートされていますが、最新の標準に合わせて、SQLSetStmtAttr() を DB2 CLI プログラムでご使用になることをお勧めします。 上記の関数と、その他の使用すべきでない関数の詳細については、
822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

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。詳細については、822ページの『64 ビット環境でサポートされない、使用すべきでない関数』を参照してください。

新しい機能への移行

たとえば、次の旧ステートメント

```
SQLSetStmtOption(
           hstmt,
           SQL_ROWSET_SIZE,
           RowSetSize);
```

は、新しい関数を使用して、以下のように書き換えます。

```
SQLSetStmtAttr(
           hstmt,
           SQL_ATTR_ROW_ARRAY_SIZE,
           (SQLPOINTER) RowSetSize,
           0);
```

SQLSpecialColumns - 特殊な (行識別子) 列の入手

目的

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLSpecialColumns() は、表の固有行の識別子情報 (基本キーまたは固有索引) 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。

構文

```
SQLRETURN   SQLSpecialColumns(
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLUSMALLINT      IdentifierType,    /* fColType */
                SQLCHAR      FAR  *CatalogName,      /* szCatalogName */
                SQLSMALLINT       NameLength1,       /* cbCatalogName */
                SQLCHAR      FAR  *SchemaName,       /* szSchemaName */
                SQLSMALLINT       NameLength2,       /* cbSchemaName */
                SQLCHAR      FAR  *TableName,        /* szTableName */
                SQLSMALLINT       NameLength3,       /* cbTableName */
                SQLUSMALLINT      Scope,             /* fScope */
                SQLUSMALLINT      Nullable);         /* fNullable */
```

関数引き数

表 172. 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	戻される固有行識別子のタイプ。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。これは、ヌル・ポインターまたは長さゼロのストリングでなければなりません。

表 172. SQLSpecialColumns 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	NameLength1	入力	*CatalogName* の長さ。これは、0 に設定する必要があります。
SQLCHAR *	SchemaName	入力	指定した表のスキーマ修飾子。
SQLSMALLINT	NameLength2	入力	*SchemaName* の長さ。
SQLCHAR *	TableName	入力	表名。
SQLSMALLINT	NameLength3	入力	*NameLength3* の長さ。
SQLUSMALLINT	効力範囲	入力	固有行識別子が有効になる最大所要時間。 *Scope* は、次のうちの 1 つでなければなりません。 • SQL_SCOPE_CURROW: 行識別子は、その行にある間だけ有効であることが保証されます。同じ行識別子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: 行識別子は、現行トランザクションの持続期間だけ有効であることが保証されます。 • SQL_SCOPE_SESSION: 行識別子は、接続の持続期間だけ有効であることが保証されます。 行識別子値が有効であることが保証される期間は、現行のトランザクション分離レベルにより異なります。分離レベルに関連する情報とシナリオについては、IBM DB2 SQL 解説書を参照してください。
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。 次のうちの 1 つでなければなりません。 • SQL_NO_NULLS - 戻される行識別子の列の集まりに、NULL 値を入れることができません。 • SQL_NULLABLE - 戻される行識別子の列の集まりに、NULL 値を入れられる列を含めることができます。

使用法

表の任意の行を固有に識別する方法が複数ある場合 (つまり、指定した表に複数の固有な索引がある場合)、 DB2 CLI は内部の基準に基づいて最良 の行識別子列の集まりを戻します。

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。

固有な行識別子情報が戻される際の結果セットの形式は、行識別子の各列が結果セット内の 1 行で表されるという形式です。 『SQLSpecialColumns で戻される列』 は、SQLSpecialColumns() で戻され、 SCOPE がソートする結果セット内の列の順序を示します。

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 文字未満なので、アプリケーションは出力バッファー用に 128 文字 (およびヌル終止符) を常にとっておくか、あるいは、接続している DBMS がサポートしている COLUMN_NAME 列の実際の長さを判別するために、SQL_MAX_COLUMN_NAME_LEN を使用して選択的に SQLGetInfo() を呼び出すように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 (行識別子の実際のスコープ)。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、782ページの表172 にある *Scope* を参照してください。

列 2 COLUMN_NAME (VARCHAR(128) 非 NULL)
表の基本キーである (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。 33ページの表2 にある記号 SQL データ・タイプ列の値の 1 つ。

列 4 TYPE_NAME (VARCHAR(128) 非 NULL)
DATA_TYPE 列の値に関連した名前で表される DBMS 文字ストリング。

列 5 COLUMN_SIZE (INTEGER)
DATA_TYPE 列の値が文字または 2 進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。それがグラフィック (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。

データ、時間、タイム・スタンプ・データ・タイプであれば、これは文字に変換されるときの値を表示するのに必要なバイトの総数です。

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。

875ページの表194 も参照してください。

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、ヌル終止符は含まれていません。正確な数値データ・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。

878ページの表196 も参照してください。

列 7 DECIMAL_DIGITS (SMALLINT)
列のスケール。位取りが適用できないデータ・タイプの場合は NULL が戻されます。 877ページの表195 も参照してください。

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 173. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、`SQLCancel()` が複数スレッドの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。
HY009	引き数値が無効です。	*TableName* はヌルです。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。

表 173. SQLSpecialColumns SQLSTATE (続き)

SQLSTATE	説明	解説
HY014	もはやハンドルはありません。	DB2 CLI は、内部資源が原因でハンドルを割り当てることができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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	スコープ・タイプが範囲外です。	無効な *Scope* 値を指定しました。
HY099	Nullable タイプが範囲外です。	無効な *Nullable* 値を指定しました。
HYC00	ドライバーが機能していません。	DB2 CLI は、表名の修飾子としてカタログ 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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

なし。

例

```
/* From the CLI sample tbconstr.c */
/* ... */
    /* call SQLSpecialColumns */
    printf("¥n    Call SQLSpecialColumns for the table %s.%s¥n",
           tbSchema, tbName);
    sqlrc = SQLSpecialColumns(hstmt, SQL_BEST_ROWID, NULL, 0,
                              tbSchema, SQL_NTS, tbName, SQL_NTS,
                              SQL_SCOPE_CURROW, SQL_NULLABLE);
```

参照

- 348ページの『SQLColumns - 表の列の情報を入手する』
- 788ページの『SQLStatistics - 基本表の索引および統計情報の入手』
- 800ページの『SQLTables - 表情報の入手』

SQLStatistics - 基本表の索引および統計情報の入手

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。

構文

```
SQLRETURN   SQLStatistics    (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLCHAR      FAR *CatalogName,       /* szCatalogName */
                SQLSMALLINT       NameLength1,       /* cbCatalogName */
                SQLCHAR      FAR *SchemaName,        /* szSchemaName */
                SQLSMALLINT       NameLength2,       /* cbSchemaName */
                SQLCHAR      FAR *TableName,         /* szTableName */
                SQLSMALLINT       NameLength3,       /* cbTableName */
                SQLUSMALLINT      Unique,            /* fUnique */
                SQLUSMALLINT      Reserved);         /* fAccuracy */
```

関数引き数

表 174. 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。これは、ヌル・ポインターまたは長さゼロのストリングでなければなりません。
SQLSMALLINT	NameLength1	入力	*NameLength1* の長さ。これは、0 に設定する必要があります。
SQLCHAR *	SchemaName	入力	指定した表のスキーマ修飾子。
SQLSMALLINT	NameLength2	入力	*SchemaName* の長さ。
SQLCHAR *	TableName	入力	表名。
SQLSMALLINT	NameLength3	入力	*NameLength3* の長さ。

表 174. SQLStatistics 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLUSMALLINT	固有	入力	戻される索引情報のタイプ。 • SQL_INDEX_UNIQUE 固有索引だけを戻します。 • SQL_INDEX_ALL すべての索引を戻します。
SQLUSMALLINT	予約済み。	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の利用の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。値は現行値でない場合があり、それらの値が最新の値であるかどうかを確認することは試行されません。

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)。
  - 下記の表の TYPE 列を SQL_TABLE_STAT に設定する場合、表の中の行数と表を保管するためのページ数。
  - TYPE 列が索引、索引内の固有の値の個数、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 790ページの『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、そして ORDINAL_POSITION 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。

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 文字未満なので、アプリケーションは、出力バッファー用に 128 文字 (およびヌル終止符) を常にとっておくか、あるいは、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_OWNER_SCHEMA_LEN、SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を使用して選択的に `SQLGetInfo()` を呼び出すように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))
これは常にヌルです。

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

列 3 TABLE_NAME (VARCHAR(128) 非 NULL)
表の名前。

列 4 NON_UNIQUE (SMALLINT)
索引で重複値が禁止されるかどうかを示します。
- 索引値で重複値が使用できる場合、SQL_TRUE。
- 索引値が固有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))

列のソート順序です (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固有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: SQLERRD(3) および SQLERRD(4) フィールドに戻される情報の正確度は、パラメーター・マーカーの使用のようなさまざまな要素、およびステートメント内のさまざまな式によって異なります。その主な要素のうちで制御可能なものは、データベース統計の正確度です。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 ユニバーサル・データベースでは、RUNSTATS コマンドの最終実行時にします。)

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 175. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。
HY009	引き数値が無効です。	*TableName* は NULL です。

表 175. SQLStatistics SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。
HY014	もはやハンドルはありません。	DB2 CLI は、内部資源が原因でハンドルを割り当てることができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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	固有オプション・タイプが範囲外です。	無効な *Unique* 値を指定しました。
HY101	正確度オプション・タイプが範囲外です。	無効な *Reserved* 値を指定しました。
HYC00	ドライバーが機能していません。	DB2 CLI は、表名の修飾子としてカタログ 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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

なし。

例

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

```
printf("¥n    Call SQLStatistics for:¥n");
printf("        tbSchema = %s¥n", tbSchema);
printf("        tbName = %s¥n", tbName);
sqlrc = SQLStatistics( hstmt, NULL, 0,
                       tbSchema, SQL_NTS,
                       tbName, SQL_NTS,
                       SQL_INDEX_UNIQUE, SQL_QUICK);
```

参照

- 348ページの『SQLColumns - 表の列の情報を入手する』
- 782ページの『SQLSpecialColumns - 特殊な (行識別子) 列の入手』

SQLTablePrivileges - 表に関連した特権の入手

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。

構文

```
SQLRETURN SQLTablePrivileges (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLCHAR      FAR  *CatalogName,      /*  szCatalogName */
                SQLSMALLINT       NameLength1,       /* cbCatalogName */
                SQLCHAR      FAR  *SchemaName,       /* szSchemaName */
                SQLSMALLINT       NameLength2,       /* cbSchemaName */
                SQLCHAR      FAR  *TableName,        /* szTableName */
                SQLSMALLINT       NameLength3);      /* cbTableName */
```

関数引き数

表 176. 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。
SQLCHAR *	szTableQualifier	入力	3 つの部分から成る表名のカタログ修飾子。これは、ヌル・ポインターまたは長さゼロのストリングでなければなりません。
SQLSMALLINT	cbTableQualifier	入力	*CatalogName* の長さ。これは、0 に設定する必要があります。
SQLCHAR *	SchemaName	入力	スキーマ名で結果セットを修飾するためのパターン*値* を入れられるバッファー。
SQLSMALLINT	NameLength2	入力	*SchemaName* の長さ。
SQLCHAR *	TableName	入力	表名で結果セットを修飾するためのパターン*値*を入れられるバッファー。
SQLSMALLINT	NameLength3	入力	*TableName* の長さ。

SchemaName 引き数と *TableName* 引き数は検索パターンを受け入れることに注意してください。有効な検索パターンの詳細については、73ページの『カタログ関数での入力引き数』を参照してください。

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 文字未満なので、アプリケーションは、出力バッファー用に 128 文字 (およびヌル終止符) を常にとっておくか、あるいは、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_OWNER_SCHEMA_LEN、SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を使用して、選択的に SQLGetInfo() を呼び出すように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))
これは常にヌルです。

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

列 3 TABLE_NAME (VARCHAR(128) 非 NULL)
表の名前。

列 4 GRANTOR (VARCHAR(128))
特権を付与したユーザーの許可 ID。

列 5 GRANTEE (VARCHAR(128))
特権が付与されたユーザーの許可 ID。

列 6 PRIVILEGE (VARCHAR(128))
表特権。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 のどれかです。

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 177. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。

表 177. SQLTablePrivileges SQLSTATE (続き)

SQLSTATE	説明	解説
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。
HY009	引き数値が無効です。	*TableName* はヌルです。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。
HY014	もはやハンドルはありません。	DB2 CLI は、内部資源が原因でハンドルを割り当てることができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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。
HYC00	ドライバーが機能していません。	DB2 CLI は、表名の修飾子としてカタログ 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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

なし。

例

```
/* From the CLI sample TBINFO.C */
/* ... */
    /* call SQLTablePrivileges */
    printf("¥n    Call SQLTablePrivileges for:¥n");
    printf("        tbSchemaPattern = %s¥n", tbSchemaPattern);
    printf("        tbNamePattern = %s¥n", tbNamePattern);
    sqlrc = SQLTablePrivileges( hstmt, NULL, 0,
                                tbSchemaPattern, SQL_NTS,
                                tbNamePattern, SQL_NTS);
    STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参照

• 800ページの『SQLTables - 表情報の入手』

SQLTables - 表情報の入手

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。

構文

```
SQLRETURN   SQLTables       (
                SQLHSTMT         StatementHandle,   /* hstmt */
                SQLCHAR      FAR *CatalogName,      /* szCatalogName */
                SQLSMALLINT      NameLength1,       /* cbCatalogName */
                SQLCHAR      FAR *SchemaName,       /* szSchemaName */
                SQLSMALLINT      NameLength2,       /* cbSchemaName */
                SQLCHAR      FAR *TableName,        /* szTableName */
                SQLSMALLINT      NameLength3,       /* cbTableName */
                SQLCHAR      FAR *TableType,        /* szTableType */
                SQLSMALLINT      NameLength4);      /* cbTableType */
```

関数引き数

表 178. 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	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。*カタログ* は、3 部分から成る表名の最初の部分です。 これは、NULL ポインターまたは長さゼロのストリングでなければなりません。
SQLSMALLINT	NameLength1	入力	*CatalogName* の長さ。これは、0 に設定する必要があります。
SQLCHAR *	SchemaName	入力	スキーマ名で結果セットを修飾するためのパターン*値*を入れられるバッファー。
SQLSMALLINT	NameLength2	入力	*SchemaName* の長さ。
SQLCHAR *	TableName	入力	表名で結果セットを修飾するためのパターン*値*を入れられるバッファー。
SQLSMALLINT	NameLength3	入力	*TableName* の長さ。

表 178. 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、対象のタイプに関するコンマで区切られた単一引用符値 (大文字) のリストです。 有効な表タイプ識別子には TABLE、VIEW、SYSTEM TABLE、ALIAS、SYNONYM 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、表タイプ識別子のすべての可能性を指定することに等しくなります。 SYSTEM TABLE を指定すると、システム表とシステム視点 (存在する場合) の両方が戻されます。
SQLSMALLINT	NameLength4	入力	*NameLength4* のサイズ。

CatalogName、*SchemaName*、および *TableName* 引き数は、検索パターンを受け入れることに留意してください。有効な検索パターンの詳細については、73 ページの『カタログ関数での入力引き数』を参照してください。

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。そうでない場合には、SELECT 特権が付与されていない表をユーザーが選択する状況を、アプリケーションがハンドルすることができる必要があります。

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、視点、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。 3 つのキーワード (SCHEMALIST、SYSCHEMA、TABLETYPE) を CLI 初期設定ファイルに設定することにより、アプリケーションが *SchemaName* と *TableType* の一方または両方に NULL ポインターを与えた場合に、結果セットを制限することができます。これらのキーワードと使用法については、174ページの『構成キーワード』で詳細に説明しています。アプリケーションが *SchemaName* や *TableType* に NULL ポインターを指定しなかった場合、 CLI 初期設定ファイルの関連キーワード指定は無視されます。

SQLTables() から戻された結果セットには、所定の順序で 803ページの表179 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 文字未満なので、アプリケーションは、出力バッファー用に 128 文字 (およびヌル終止符) を常にとっておくか、あるいは、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_OWNER_SCHEMA_LEN、SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を使用して選択的に SQLGetInfo() を呼び出すように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。

表 179. SQLTables で戻される列

列名	データ・タイプ	説明
TABLE_CAT	VARCHAR(128)	TABLE_SCHEM を含むカタログの名前。この列には、NULL 値が含まれています。
TABLE_SCHEM	VARCHAR(128)	TABLE_NAME を含むスキーマの名前。
TABLE_NAME	VARCHAR(128)	表、視点、別名、同義語のどれかの名前。
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。そのタイプは、ストリング値 'TABLE'、'VIEW'、'INOPERATIVE VIEW'、'SYSTEM TABLE'、'ALIAS'、または 'SYNONYM' です。
REMARKS	VARCHAR(254)	表に関する記述情報が含まれています。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 180. 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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。

表 180. SQLTables SQLSTATE (続き)

SQLSTATE	説明	解説
HY008	操作が取り消しになりました。	非同期処理が *StatementHandle* に対して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 *StatementHandle* で呼び出されました。そして、関数が再び *StatementHandle* で呼び出されました。 関数が呼び出され、その実行が完了する前に、SQLCancel() が複数スレッドの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。
HY009	引き数値が無効です。	*TableName* はヌルです。
HY010	関数の順序エラーです。	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 SQL の操作中に、関数が呼び出されました。 非同期実行関数 (この関数ではない) が *StatementHandle* で呼び出され、この関数は、呼び出し時に依然実行中でした。
HY014	もはやハンドルはありません。	DB2 CLI は、内部資源が原因でハンドルを割り当てることができませんでした。
HY090	ストリングまたはバッファー長が無効です。	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。
HYC00	ドライバーが機能していません。	DB2 CLI は、表名の修飾子としてカタログ 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返す前に、タイムアウト期間が満了しました。タイムアウトは、 Windows 3.1 や Macintosh System 7 のようなマルチタスクではないシステム上でのみサポートされています。タイムアウト期間は、SQLSetConnectAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制約

なし。

例

44ページの『環境情報の照会の例』も参照してください。

```
/* From the CLI sample TBREAD.C */
/* ... */
    /* call SQLTables */
    printf("¥n    Call SQLTables.¥n");
    sqlrc = SQLTables( hstmt, NULL, 0,
                       tbSchemaPattern, SQL_NTS,
                       tbNamePattern, SQL_NTS,
                       NULL, 0);
    STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参照

- 348ページの『SQLColumns - 表の列の情報を入手する』
- 795ページの『SQLTablePrivileges - 表に関連した特権の入手』

SQLTransact - トランザクション管理

使用すべきでない関数

注:

ODBC バージョン 3 では、SQLTransact() は使用すべきでない関数で、代わりに SQLEndTran() を使用します。詳しくは 388ページの『SQLEndTran - 接続のトランザクションの終了』を参照してください。

DB2 CLI のこのバージョンでは、引き続き SQLTransact() がサポートされていますが、最新の標準に合わせて、SQLEndTran() を DB2 CLI プログラムでご使用になることをお勧めします。 上記の関数と、その他の使用すべきでない関数の詳細については、822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。

新しい関数への移行

たとえば、次の旧ステートメント

```
SQLTransact(henv, hdbc, SQL_COMMIT);
```

は、新しい関数を使用して、以下のように書き換えます。

```
SQLEndTran(SQL_HANDLE_ENV, henv, SQL_COMMIT);
```

目的

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLTransact() は、指定の接続にある現在のトランザクションをコミットまたはロールバックします。 SQLTransact() は、環境に関連付けられている各接続に対して出されるコミットまたはロールバックを要求する際に使用することができます。

接続時刻または直前の SQLTransact() 呼び出し (どちらか現在に近い方) より後に、接続でデータベースに加えられた変更は、すべてコミットまたはロールバックされます。

接続でトランザクションが活動状態の場合、アプリケーションはデータベースから切断する前に SQLTransact() を呼び出しておく必要があります。

構文

```
SQLRETURN   SQLTransact      (SQLHENV          EnvironmentHandle, /* henv */
                              SQLHDBC          ConnectionHandle,  /* hdbc */
                              SQLUSMALLINT     Type);             /* fType */
```

関数引き数

表 181. SQLTransact 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 *ConnectionHandle* が有効な接続ハンドルである場合、 *EnvironmentHandle* は無視されます。
SQLHDBC	*ConnectionHandle*	入力	データベース接続ハンドル。 *ConnectionHandle* を SQL_NULL_HDBC に設定しない場合、 *EnvironmentHandle* には接続を関連付ける環境ハンドルを入れる必要があります。
SQLUSMALLINT	*Type*	入力	ご希望のトランザクション用のアクション。この引き数の値は、次のうちの 1 つでなければなりません。 • SQL_COMMIT • SQL_ROLLBACK

使用法

DB2 CLI では、活動トランザクションがまだないアプリケーションが SQLPrepare()、 SQLExecDirect()、SQLExecDirect()、SQLGetTypeInfo()、またはカタログ関数のどれかを出すと、トランザクションが暗黙に開始されます。アプリケーションが SQLTransact() を呼び出すと、トランザクションが終了します。

入力接続ハンドルが SQL_NULL_HDBC であり、かつ環境ハンドルが有効である場合、コミットやロールバックは環境内の各オープン接続で出されます。SQL_SUCCESS が戻されるのは、すべての接続でコミットやロールバックの成功が報告されたときだけです。 1 つまたは複数の接続でコミットやロールバックが失敗すると、 SQLTransact() が SQL_ERROR を戻します。どの接続でコミットやロールバック操作が失敗したかを判別するには、環境内の接続ハンドルでアプリケーションが SQLError() を呼び出すことが必要です。

重要な点として、接続オプション SQL_ATTR_CONNECTTYPE が SQL_COORDINATED_TRANS (整合分散トランザクションを示す) に指定され

ていないかぎり、整合グローバル・トランザクションに 1 フェーズまたは 2 フェーズのコミット・プロトコルが提供されることはありません。

トランザクションが終了すると、次のような結果になります。

- 準備済み SQL ステートメント (SQLPrepare() を経由しての) は複数のトランザクションで持続するので、最初に SQLPrepare() を呼び出さなくても再び実行することができます。
- カーソル位置は、以下の 1 つまたは複数の条件が当てはまる場合には、コミット後も維持されます。
  - サーバーが SQL/DS である。
  - このハンドルの SQL_ATTR_CURSOR_HOLD ステートメント属性が SQL_CURSOR_HOLD_OFF に設定されている。
  - DB2 CLI 初期設定ファイルに CURSORHOLD キーワードが設定されているため、保留カーソルは有効ではなく、 SQL_ATTR_CURSOR_HOLD ステートメント属性をリセットしても指定変更されなかった。
  - この接続を設定した SQLDriverConnect() 呼び出しの接続ストリングに CURSORHOLD キーワードがあり、それによると保留カーソルが有効ではなく、 SQL_ATTR_CURSOR_HOLD ステートメント属性をリセットしても指定変更されなかった。

  上記の状況のいずれかの理由で、カーソル位置が維持されない場合、カーソルはクローズされ、保留されているすべての結果が破棄されます。

  カーソル位置がコミット後に維持される場合、残りの結果セットの処理が続行される前に、アプリケーションは取り出しを発行してカーソルを (次の行に) 位置指定し直すことが必要です。

  コミット後にカーソル位置が維持されているかどうかを判別するには、SQL_CURSOR_COMMIT_BEHAVIOR 情報タイプを指定して SQLGetInfo() を呼び出してください。
- ロールバック後にカーソルはクローズされ、保留されているすべての結果は廃棄されます。
- ステートメント・ハンドルは SQLTransact() 呼び出しの後もまだ有効であり、以降の SQL ステートメントに再利用できるか、 SQLFreeStmt() を呼び出して割り振り解除することができます。
- カーソル名、結合パラメーター、および列結合は、トランザクションにまたがって有効です。

接続上で現在活動状態になっているトランザクションがない場合は、SQLTransact() を呼び出してもデータベース・サーバーには影響はなく、SQL_SUCCESS が戻されます。

接続が失われたために、 COMMIT または ROLLBACK の実行中に SQLTransact() が失敗するおそれがあります。この場合、アプリケーションは COMMIT または ROLLBACK を処理したかどうかを判別することができない場合があり、またデータベース管理者の援助が必要となる場合があります。トランザクション・ログやその他のトランザクション管理タスクの詳細については、 DBMS 製品情報を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 182. SQLTransact SQLSTATE

SQLSTATE	説明	解説
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* に関連した接続は関数の実行時に失敗し、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。
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失敗です。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY012	トランザクション・コードが無効です。	引き数 *Type* に指定された値は、 SQL_COMMIT でも SQL_ROLLBACK でもありませんでした。
HY013	予期しないメモリーのハンドル・エラーが起きました。	DB2 CLI は、関数の実行または完了をサポートするのに必要なメモリーを使用することができませんでした。

制約

なし。

例

388ページの『SQLEndTran - 接続のトランザクションの終了』を参照してください。

参照

- 756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』

- 533ページの『SQLGetInfo - 一般情報の入手』

付録A. プログラミングのヒントと提案

この節では、 DB2 CLI および ODBC アプリケーションのパフォーマンスと移植性を向上させるのに役立つヒントと提案を提供します。

共通接続属性の設定

以下の接続属性は、DB2 CLI アプリケーションによって設定 (または考慮) することが必要な場合があります。

SQL_ATTR_AUTOCOMMIT

各コミット要求が余分なネットワーク・フローを生成できるので、通常この属性は SQL_AUTOCOMMIT_OFF に設定します。特に必要な場合に限って、SQL_AUTOCOMMIT をオンにしておきます。

注: 省略時値は SQL_AUTOCOMMIT_ON です。

SQL_ATTR_TXN_ISOLATION

この接続属性は、接続またはステートメント操作時の分離レベルを判別します。分離レベルとは、可能な並行性のレベル、およびステートメントを実行するのに必要なロックのレベルを決めるものです。アプリケーションにしてみれば、並行性を最大にし、一方でデータ一貫性が保証される分離レベルを選択することが必要になります。

分離レベルとその影響の完全な説明については、*SQL 解説書* を参照してください。

共通ステートメント属性の設定

以下のステートメント属性は、 DB2 CLI アプリケーションによって設定することが必要な場合があります。

SQL_ATTR_MAX_ROWS

このオプションを設定することにより、アプリケーションに戻される行数を制限することができます。アプリケーション (とりわけ、メモリー資源が制限さ

れているクライアント上のアプリケーション) で非常に大きな結果セットが不用意に生成されて処理できなくなるといった状況を避けるために、このオプションを使用することができます。

DB2 UDB v6 では、SQL_ATTR_MAX_ROWS を設定すると、ステートメントに 『OPTIMIZE for n ROWS』 および 『Fetch n Rows ONLY』 文節が追加されます。その他のサーバーでは、完全な結果セットはサーバーで引き続き生成され、 DB2 CLI が取り出すのは SQL_ATTR_MAX_ROWS 行までです。

SQL_ATTR_CURSOR_HOLD

このステートメント属性は、このステートメントのカーソルを CURSOR WITH HOLD 文節と同等のものを用いて定義するかどうかを決めます。

CURSOR WITH HOLD を必要としないステートメントが SQL_CURSOR_HOLD_OFF に設定されていれば、ステートメント・ハンドルに関連する資源は DB2 CLI によってよりよく活用されます。

注: ODBC アプリケーションの多くは、コミット後にカーソル位置が保持されている場合に、省略時の動作を予期します。

SQL_ATTR_TXN_ISOLATION

DB2 CLI では分離レベルをステートメント・レベルで設定することが可能です (ただし、分離レベルは接続レベルで設定することをお勧めします)。分離レベルとは、可能な並行性のレベル、およびステートメントを実行するのに必要なロックのレベルを決めるものです。

すべてのステートメントが省略時の分離レベルのままにされているのではなく、要求された分離レベルに設定されていれば、ステートメント・ハンドルに関連する資源は DB2 CLI によってよりよく活用されます。以上のことは、接続している DBMS のロックおよび分離レベルについて完全に理解している場合にのみ行ってみてください。分離レベルとその影響の完全な説明については、*SQL 解説書* を参照してください。

アプリケーションは、並行性を最大にするように、最小の分離レベルを使用すべきです。

バインドと SQLGetData との比較

通常は、SQLGetData() の使用に比べて、アプリケーション変数または結果セットへのファイル参照をバインドするほうが効率的です。データ値が大きい可変長データであるために、状況が以下のようである場合には、 SQLGetData()、または LOB 関数 (この方が好まれる) を使用してください。

- データを分割して受け取らなければならない。または、
- データを検索する必要がない (別のアプリケーション・アクションに依存する)。

転送効率を向上する

メモリー内の *pcbValue* 引き数と *rgbValue* 引き数が連続している場合に、バインドされたアプリケーション変数と DB2 CLI の間での文字データの転送効率を向上させることが可能です。 (これにより、DB2 CLI は一回のコピー操作で両方の値を取り出すことができます。)

たとえば、

```
struct {  SQLINTEGER  pcbValue;
          SQLCHAR     rgbValue[MAX_BUFFER];
       } column;
```

カタログ関数の使用を制限する

一般論として、カタログ関数を呼び出せる回数の制限、および戻される行数の制限について論じます。

一度関数を呼び出してから、その情報をアプリケーションに保管することによって、カタログ関数呼び出しの回数を減らすことが可能です。

戻される行数は、以下を指定することによって制限することができます。

- すべてのカタログ関数のスキーマ名またはパターン
- SQLTables 以外のすべてのカタログ関数の表名またはパターン
- 詳細な列情報を戻すカタログ関数の列名またはパターン

ただ、アプリケーションの開発やテストを何百もの表を持つデータ・ソースに対して行ったとしても、アプリケーションの実行は何百もの表を持つデータベースに対して行われることがある、という点を覚えておいてください。将来の計画を立てることが大切です。

カタログ照会に使用されたステートメント・ハンドルのカーソルでオープンしているものをクローズし (SQL_CLOSE を指定して SQLFreeStmt() を呼び出

す)、カタログ表へのロックをすべて解除します。カタログ表に未解決のロックがあると、CREATE、 DROP または ALTER ステートメントを実行できなくなることがあります。

関数生成による結果セットの列名を使用する

カタログおよび情報関数により生成される結果セットの列名は、 ODBC および CLI 標準が変化するにつれて変更されることがあります。ただし、*列の位置* が変更されることはありません。

アプリケーション依存性は列の位置 (*icol* パラメーター) に基づいており、列の名前には基づいていません。

ODBC アプリケーションから DB2 CLI 固有の関数をロードする

DB2 CLI ステートメント・ハンドル (HSTMT) を入手するには、`SQLGetInfo()` に SQL_DRIVER_HSTMT オプションを指定して呼び出します。

DB2 CLI 関数は、オペレーティング・システム呼び出しを使用し HTSMT 引き数を渡すことによって、共用ライブラリーまたは DLL から直接呼び出すことができます。 ODBC ドライバー・マネージャーは、自分のステートメント・ハンドルのセットを保持しつつ、それを各呼び出しの*実際の* ドライバー・ハンドルにマッピングします。 DB2 CLI 関数が直接呼び出されると、`SQLGetInfo()` が呼び出されて、このマッピングが明示的に行われます。

動的 SQL ステートメント・キャッシュを使用する

注: このセクションの記述は、グローバル・ステートメント・キャッシュのないサーバーに接続する場合にのみ該当します。 DB2 ユニバーサル・データベースなどのグローバル動的ステートメント・キャッシュのあるサーバーに接続する場合、アプリケーションは動的 SQL ステートメント・キャッシュを使用すべきではありません。詳しくは 815ページの『グローバル動的ステートメント・キャッシュを利用する』を参照してください。

*動的*キャッシュ を使用するには (サーバーがシステム作成バージョンの動的 SQL ステートメントをキャッシュする場合)、アプリケーションは同じ SQL ステートメントに対して同じステートメント・ハンドルを使用しなければなりません。

たとえば、あるアプリケーションが 10 個の SQL ステートメントの集まりを決まって使用する場合には、 10 個のステートメント・ハンドルを割り振り、

それらのステートメントのおのおのに関連づけることになります。そのステートメントを実行している間は、ステートメント・ハンドルを解放しないでください。 (トランザクションは、作成されたどのステートメントにも影響を与えることなく、ロールバックまたはコミットすることができます。) アプリケーションは通常と同じくステートメントの作成と実行を継続し、 DB2 CLI は作成が実際に必要かどうかを判別します。

関数呼び出しのオーバーヘッドを削減するために、一度ステートメントを作成し、その後アプリケーション全体を繰り返し実行することができます。

注: サーバーが動的キャッシュをサポートしていない場合、 DB2 CLI は必要に応じてステートメントの作成命令を内部的に出します。

グローバル動的ステートメント・キャッシュを利用する

DB2 ユニバーサル・データベースのバージョン 5 以降では、サーバーに*グローバルな動的ステートメント・キャッシュ* が保管されています。このキャッシュは、準備状態の SQL ステートメントに対する最も一般的なアクセス・プランを保管するのに使用します。

各ステートメントが準備される前に、サーバーは自動的にこのキャッシュを検索して、 (このアプリケーションか別のアプリケーション、またはクライアントによって) この SQL ステートメント用のアクセス・プランが作成済みであるかどうかを調べます。作成済みであれば、サーバーが新たにアクセス・プランを生成する必要はなく、代わりに、キャッシュの中にあるものを使用します。現在では、グローバル動的ステートメント・キャッシュのないサーバー (DB2 共通サーバー v2 など) への接続でなければ、アプリケーションがクライアントで接続をキャッシュする必要はありません。クライアントでのキャッシュ接続については、移行のセクションの 825ページの『クライアントでのステートメント・ハンドルのキャッシュ』を参照してください。

データの挿入および検索を最適化する

91ページの『配列の使用によるパラメーター値の入力』および 99ページの『配列への結果セットの取り出し』に記載されている方法では、ネットワーク・フローを最適化するために複合 SQL を使用します。

可能な限りこれらの方法を用いるようにしてください。

ラージ・オブジェクト・データを最適化する

長形式ストリングには、可能な限り LOB データ・タイプおよびそれをサポートする関数を使用してください。 LONG VARCHAR、LONG VARBINARY、および LONG VARGRAPHIC タイプとは異なり、 LOB データ値は LOB ロケーターおよび SQLGetPostion() や SQLGetSubString() などの関数を使用して、サーバーの大きなデータ値を操作することができます。

LOB 値をファイルに直接取り出すこともできますし、 LOB パラメーター値をファイルから直接読み取ることができます。このようにすると、アプリケーション・バッファーを経由してデータを転送するアプリケーションのオーバーヘッドを節約することができます。

オブジェクト識別子の大小文字の区別

すべてのデータベース・オブジェクト (表、視点、列など) の識別子は、その識別子が区切られていなければ、カタログ表には大文字で保管されます。区切り名を用いて識別子が作成された場合には、名前の記述に用いられた文字がそのままカタログ表に保管されます。

識別子が SQL ステートメント内で参照されると、識別子が区切られていなければ、大小文字は*区別なし* として処理されます。

たとえば、次の 2 つの表が作成された場合、

```
CREATE TABLE MyTable   (id INTEGER)
CREATE TABLE "YourTable" (id INTEGER)
```

2 つの表 MYTABLE と YourTable が存在することになります。

以下の 2 つのステートメントは同等です。

```
SELECT * FROM MyTable (id INTEGER)
SELECT * FROM MYTABLE (id INTEGER)
```

次の 2 番目のステートメントは、YOURTABLE と命名された表がないため、TABLE NOT FOUND というエラーが出されて失敗します。

```
SELECT * FROM "YourTable" (id INTEGER)   // executes without error
SELECT * FROM YourTable (id INTEGER)     // error, table not found
```

すべての DB2 CLI カタログ関数の引き数は、オブジェクトの名前を大文字小文字の*区別あり* 、すなわち各名前が区切られているものとして処理します。

SQLConnect の代わりに SQLDriverConnect を使用する

SQLDriverConnect() を使用することにより、アプリケーションはユーザーへの接続情報の入力要求を DB2 CLI によって提供されるダイアログ・ボックスに任せることができます。

あるアプリケーションが接続情報の照会にアプリケーション自身のダイアログ・ボックスを使用している場合、ユーザーは接続ストリングに追加の接続オプションを指定できます。このストリングも保管され、それ以後の接続では省略時値として使用されます。

SQL 管理プログラムを使用する

SQL ステートメントが作成されるたびに、サーバーはステートメントのコストを*見積もります* 。次いで、アプリケーションはそのステートメントの実行を継続するかどうかを判断します。

この見積もりは SQLCA (SQLERRD(4)) から入手し、アプリケーションが直接使用することができますし、または SQL_DB2ESTIMATE 接続オプションをしきい値に設定することができます。見積もられたステートメントのコストがどれもしきい値を超えている場合には、 DB2 CLI が警告のダイアログ・ボックスを表示し、ステートメントの継続または取り消しを入力要求します。

提案されているしきい値は 60000 ですが、通常のアプリケーションではエンド・ユーザーがしきい値を設定できるようになっています。

注: この見積もりは、サーバーがステートメントを実行するために使用している資源の合計の見積もりだけであって、ステートメントを実行するために必要な時間を示してはいません。

結果の行数の見積もりは SQLCA (SQLERRD(3)) からも利用可能で、大きい照会を制限するためにアプリケーションが使用することも可能です。

注: SQLERRD(3) および SQLERRD(4) フィールドに戻される情報の正確度は、パラメーター・マーカーの使用のようなさまざまな要素、およびステートメント内のさまざまな式によって異なります。その主な要素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 ユニバーサル・データベースでは、RUNSTATS コマンドの最終実行時にします。)

ステートメント走査をオフにする

DB2 CLI は省略時設定で、ベンダー・エスケープ文節順序列を探索して、SQL ステートメントを一つ走査します。

アプリケーションがベンダー・エスケープ文節順序列 (151ページの『ベンダー・エスケープ文節の使用』) を含む SQL ステートメントを生成しない場合は、 SQL_NO_SCAN ステートメント・オプションを接続レベルで SQL_NOSCAN_ON に設定することによって、 DB2 CLI がベンダー・エスケープ文節を探索して文節走査を実行することがないようにすることができます。

複数のロールバックでカーソルを保留する

トランザクション管理上の複雑な問題を処理することが必要なアプリケーションでは、同一のデータベースに複数の並行接続を確立するとよい場合があります。 DB2 CLI 内の各接続にはそれぞれトランザクション有効範囲があり、 1 つの接続で実行されるアクションが他の接続のトランザクションに影響を与えることはありません。

たとえば、ある 1 つのトランザクション内でオープンされているすべてのカーソルは、問題が起こってそのトランザクションがロールバックされるとクローズしてしまいます。カーソルは一つのトランザクション内に個別にあり、あるステートメントでロールバックがなされても他のステートメントのカーソルには影響しないので、アプリケーションは同一のデータベースに複数の接続を使用して、オープンしているカーソルを持つ複数のステートメントを分離しておきます。

複数の接続を使用するということは、ある接続でクライアントにデータを渡してから、別の接続でサーバーにそのデータを戻すということを意味します。たとえば、

接続 #1 で、ラージ・オブジェクト列にアクセスしており、かつラージ・オブジェクト値の一部にマッピングする LOB ロケーターを作成していると仮定します。

接続 #2 で、LOB ロケーターにより表される LOB 値の一部を使用 (挿入) する場合、まず接続 #1 の LOB 値をアプリケーションに移動し、それから接続 #2 で作業中の表に渡す必要があります。そうする理由は、接続 #2 が接続 #1 の LOB ロケーターを認識しないためです。

接続が 1 つしかなければ、LOB ロケーターを直接使用することができます。ただし、トランザクションをロールバックするとすぐに、LOB ロケーターは失われてしまいます。

複合 SQL サブステートメントの作成

複合ステートメントの効率を最大にするには、 BEGIN COMPOUND ステートメントの前にサブステートメントを作成し、次いで複合ステートメント内でそのサブステートメントを実行します。

このようにすることによっても、作成エラーが複合ステートメントの外側で処理されるので、エラー処理が単純化されます。

ユーザー定義タイプ (UDT) のキャスト

照会ステートメントの述部にパラメーター・マーカーが使用されており、かつ、そのパラメーターがユーザー定義タイプである場合には、ステートメントに CAST 関数を使用して、パラメーター・マーカーまたは UDT のいずれかをキャストしなければなりません。

たとえば、次のようにタイプおよび表が定義されているとします。

```
CREATE DISTINCT TYPE CNUM AS INTEGER WITH COMPARISONS
CREATE TABLE CUSTOMER (
        Cust_Num     CNUM NOT NULL,
        First_Name   CHAR(30) NOT NULL,
        Last_Name    CHAR(30) NOT NULL,
        Phone_Num    CHAR(20) WITH DEFAULT,
        PRIMARY KEY  (Cust_Num) )
```

このステートメントはパラメーター・マーカーがタイプ CNUM にはならないために失敗し、したがってタイプに互換性がないことから比較が失敗して、次のようになります。

```
SELECT  first_name, last_name, phone_num FROM customer
where cust_num = ?
```

列を整数 (その基本 SQL タイプ) にキャストすると、パラメーターには整数のタイプが与えられるので、比較を実行することができます。

```
SELECT  first_name, last_name, phone_num from customer
where cast( cust_num as integer ) = ?
```

あるいは、パラメーター・マーカーを INTEGER にキャストすることによって、サーバーは INTEGER に CNUM 変換を適用することができます。

```
SELECT  first_name, last_name, phone_num FROM customer
where cust_num = cast( ? as integer )
```

完全な作業サンプルについては、custrep.c サンプル・ファイルを参照してください。

以下の詳細については、*SQL 解説書* を参照してください。
- パラメーター・マーカー。PREPARE ステートメントを参照
- キャスト。CAST 関数を参照

非同期実行を使用せずマルチ・スレッドを使用する

非同期 SQL モデルは、 Windows 3.1 のような非スレッド・オペレーティング・システム上でのみ使用してください。ご使用になるアプリケーションでマルチ・スレッドを使用できない場合、 145ページの『CLI の非同期実行』を参照してください。

非同期 SQL は、マルチ・スレッドをサポートしているプラットフォーム上で使用しないでください。

50ページの『マルチスレッドのアプリケーション作成』には、DB2 CLI でマルチ・スレッドを使用する理由とその方法が説明されています。一般的な使用方法として、以下のようなものがあります。

- 実行中のものとは別のスレッドを使用して、 `SQLCancel()` を呼び出すことができます。 (たとえば、時間のかかる照会を取り消します。)
- たいていの GUI ベースのアプリケーションではスレッドを使用して、ユーザーとの対話が優先度の高いスレッドで扱われるようにしています。それに比べると、他のアプリケーション・タスク (たとえば、データベースへのアクセスなど) は優先度が低くなっています。
- マルチ・スレッドで DB2 CLI 関数を実行すると、スループットが向上します。

据え置き準備を使用してネットワーク・フローを減らす

DB2 CLI バージョン 5 では、デフォルトで据え置き準備がオンになります。対応する実行要求が発行されるまで、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。

詳細は 827ページの『デフォルトでの据え置き準備』を参照してください。

付録B. アプリケーションの移行

この節では、前のバージョンの DB2 CLI 以降に変更された点、および非互換性の部分とその対処方法について説明します。

変更の要約

DB2 ユニバーサル・データベースのバージョン 5 には、 DB2 CLI アプリケーションの作成方法に影響を及ぼす新機能が入っています。 *SQL 解説書* には、変更の要約が完全な形で載せられています。

DB2 CLI 関数の要約、およびどのバージョンでその関数が追加されたかについては、 235ページの『DB2 CLI 関数の要約』を参照してください。

DB2 CLI のバージョン 5 では数多くの変更点があります。そのうちの目立ったものとして、以下のようなものがあります。

- DB2 CLI 関数のうち使用すべきでない関数となったものが多くあります。詳しくは 822ページの『バージョン 5 で使用すべきでない DB2 CLI 関数』を参照してください。
- 名前変更された関数があります。 (たとえば、SQLColAttributes() は SQLColAttribute() になっています。)
- ハンドルを割り当てるための 3 つの関数 (SQLAllocConnect()、SQLAllocEnv()、および SQLAllocStmt()) は、 1 つの多目的関数 (SQLAllocHandle()) に統廃合されました。
- DB2 CLI および ODBC で現在使用可能な新機能をサポートするため、新しい DB2 CLI 機能が追加されました (記述子、スクロール可能カーソルなど)。

また、DB2 CLI には引き続き、 ODBC アプリケーションによってアクセスできない DB2 機能にアクセスするための次の拡張が含まれています。たとえば、

- ラージ・オブジェクト (LOB)、 LOB ロケーターとファイル参照バッファーへのランダム・アクセスのサポート (順次アクセスも可能)
- 詳細な DB2 特定診断情報に関する SQLCA アクセス
- 出力ストリングの NULL 終了の制御

非互換性

すべてのバージョン 1 およびバージョン 2 アプリケーションは、バイナリー互換性があるため、何ら変更を加えることなく DB2 ユニバーサル・データベース バージョン 5 製品で実行できるはずです。場合によっては、アプリケーションを実行するのに、 DB2 CLI 初期設定ファイル (db2cli.ini) のカスタマイズを行うことが必要になることもあります。このファイルおよび各種キーワードの詳細は、168ページの『db2cli.ini の構成』を参照してください。

アプリケーションを再コンパイルした場合、いくらかのマイナー・チェンジ (小規模な変更) が必要になることがありますが、 DB2CLI_VER 定義を設定することによってこれらの変更を最小限にすることができます (詳細については、 832ページの『DB2CLI_VER 定義の設定』を参照してください)。また、環境属性 SQL_ATTR_ODBC_VERSION を使用して、必要な変更を最小限にすることもできます。詳しくは、734ページの『SQLSetEnvAttr - 環境属性を設定する』にある属性の説明を参照してください。

64 ビット環境でサポートされない、使用すべきでない関数

以下の関数は、64 ビット環境ではサポートされず、使用すべきではありません。

- `SQLSetConnectOption()`
- `SQLSetStmtOption()`

上記の関数は 64 ビット環境ではサポートされません。これらの関数には、ポインターにキャストできる 32 ビット整数パラメーター (64 ビット・システムでは不可能) を指定するからです。

`SQLSetConnectAttr()` および `SQLSetStmtAttr()` を 64 ビット環境で使用するには、アプリケーションを事前に変更しておく必要があります。

バージョン 2.1.1 から 5.0.0 への変更点

バージョン 5 で使用すべきでない DB2 CLI 関数

バージョン 2 にあってバージョン 5 では使用すべきでないとされた各 DB2 CLI 関数は、現在も関数参照にリストされています。それぞれの関数についての説明の冒頭に、その旨が示されています。

DB2 CLI バージョン 5 では、使用すべきでない関数も引き続きすべてサポートしていますが、最新の標準に合わせて、新しい関数を DB2 CLI プログラムでご使用になることをお勧めします。

ものによっては、使用すべきでない関数の機能および引き数と、それに代わる関数が酷似している場合があります (たとえば、SQLColAttributes() と SQLColAttribute() など)。そのような場合、使用すべきでない関数の説明は省かれています。

また、別の場合には、使用すべきでない関数の説明がそのままになっているものもあり、その場合、新しい関数に切り替えになったことを理解する助けとなります。

下記の表には、使用すべきでない関数とその説明、および代わりの関数がそれぞれリストされています。

表 183. 使用すべきでない関数とそれに代わる関数

使用すべきでない関数	説明	代わりの関数
SQLAllocConnect()	除去	SQLAllocHandle()
SQLAllocEnv()	除去	SQLAllocHandle()
SQLAllocStmt()	除去	SQLAllocHandle()
SQLColAttributes()	除去	SQLColAttribute()
SQLError()	変更なし	SQLGetDiagField() および SQLGetDiagRec()
SQLExtendedFetch()	変更なし	SQLFetchScroll()
SQLFreeConnect()	変更なし	SQLFreeHandle()
SQLFreeEnv()	変更なし	SQLFreeHandle()
SQLGetConnectOption()	除去	SQLGetConnectAttr()
SQLGetStmtOption()	除去	SQLGetStmtAttr()
SQLParamOptions()	変更なし	SQLSetStmtAttr()
SQLSetConnectOption()	除去	SQLSetConnectAttr()
SQLSetParam()	除去	SQLBindParameter()
SQLSetStmtOption()	除去	SQLSetStmtAttr()
SQLTransact()	変更なし	SQLEndTran()

ストアード・プロシージャーの疑似カタログ表の置換

DB2 ユニバーサル・データベース バージョン 5 では、サーバーの全ストアード・プロシージャーに関する情報を保管するための 2 つのシステム・カタログ表示を導入しています。バージョン 5 以前では、DB2 CLI はストアード・プロシージャー登録の疑似カタログ表を使用していました。デフォルトで、DB2 CLI は新しいシステム・カタログ表示を使用することになります。アプリケー

ションで疑似カタログ表の使用が見込まれる場合、 CLI/ODBC 構成キーワード PATCH1 を 262144 に設定してください。

バージョン 2 DB2 共通サーバーへのアプリケーションの接続時に、SQLProcedureColumns() および SQLProcedures() がストアード・プロシージャーの情報を (疑似カタログ表から) 戻せるように、ストアード・プロシージャー登録の疑似カタログ表がすでに作成され、満たされている必要があります。これは DB2CLI スキーマにおける PROCEDURES という名前のついた表です。この疑似カタログ表の詳細は、897ページの『付録H. ストアード・プロシージャー登録の疑似カタログ表』を参照してください。この表を満たす際、897ページの『付録H. ストアード・プロシージャー登録の疑似カタログ表』にある規則どおりに行うことが肝要であり、そうしないと SQLProcedureColumns() および SQLProcedures() 呼び出しはエラーになります (SQLSTATE **42**601)。

SQLSetConnectAttr() を使用してステートメント属性のサブセットを設定する

SQLSetConnectAttr() 関数は、ステートメント属性のサブセットを設定する場合にのみ使用できます。この関数はまもなく廃止になり、新しい DB2 CLI ではこの機能を使いません。バージョン 5 以前の DB2 CLI では、SQLSetConnectOption() を使用してこの機能を使っていましたが、その後除去されて、関数も使用すべきでないものとされています。

バージョン 5 以前に書かれた DB2 CLI アプリケーションをサポートするため、現在 SQLSetConnectAttr() は、バージョン 2 で定義された以下のオプション値を使用して、以前の SQLSetConnectOption() と同様の働きをしています。 (バージョン 5 では '_ATTR_' が追加になっています。)

- SQL_ATTR_ASYNC_ENABLE
- SQL_ATTR_BIND_TYPE
- SQL_ATTR_CONCURRENCY
- SQL_ATTR_CURSOR_HOLD
- SQL_ATTR_CURSOR_TYPE
- SQL_ATTR_MAX_LENGTH
- SQL_ATTR_MAX_ROWS
- SQL_ATTR_NODESCRIBE
- SQL_ATTR_NOSCAN
- SQL_ATTR_PARAMOPT_ATOMIC
- SQL_ATTR_QUERY_TIMEOUT
- SQL_ATTR_RETRIEVE_DATA
- SQL_ATTR_ROWSET_SIZE
- SQL_ATTR_STMTTXN_ISOLATION
- SQL_ATTR_TXN_ISOLATION

クライアントでのステートメント・ハンドルのキャッシュ

DB2 の前のバージョンには、グローバル・ステートメント・キャッシュがありませんでした。しかし、サーバーで*動的*ステートメント・キャッシュ を提供していました。 DB2 CLI でこのことは、あるステートメント・ハンドルについて、ステートメントが一度準備されると、そのステートメント・ハンドルが解放される場合を除いて再度準備する必要がないことを意味します (コミットやロールバックの後も同様)。複数のトランザクションで同一の SQL ステートメントを繰り返し実行するアプリケーションの場合、次のことを実行すると、処理時間とネットワーク通信量を相当量節約できます。

1. 毎回そのステートメントを独自のステートメント・ハンドルと関連付けます。
2. さらに、アプリケーションの先頭でそのステートメントを一度準備します。
3. その後で、アプリケーション全体で必要な回数だけそのステートメントを実行します。

DB2 ユニバーサル・データベースのバージョン 5 では、グローバル動的ステートメント・キャッシュがあるため、これが不要になりました。最初のステートメントを準備すると、グローバル・キャッシュにパッケージが作成されます。それ以降の準備要求は、いずれもキャッシュの中から最初のアクセス・プランを見つけて、すぐに使用します。

トランザクション境界間の動的ステートメント・キャッシュをサポートしないサーバーにアプリケーションが接続すると、 DB2 CLI は必要に応じて各ステートメントを内部的に準備します。このことは、RDBMS とは関係なくすべてのアプリケーションについて上記の方式を使用できることを意味します。

SQLColumns() 戻り値の変更

次の表は、SQLColumns() によって戻される列への変更を、バージョン 2.1.1 とバージョン 5 で対比したリストです。

現在の値については、350ページの『SQLColumns() で戻される列』を参照してください。

DB2 CLI をバージョン 2 の場合と同じように (同じ列名および順序で) 動作させるには、 DB2 CLI/ODBC 構成キーワード PATCH2 を設定します。このキーワード設定について、詳しくは 167ページの『CLI/ODBC 構成キーワードの設定方法』を参照してください。

表 184. SQLColumns で戻される列の変更

バージョン 2 の列	バージョン 5 の列	変更
14 DATETIME_CODE	バージョン 5 では除去	この情報は戻されなくなった。バージョン 2 では常に NULL だった。
バージョン 2 と同等のものはない。	14 SQL_DATA_TYPE	この情報はバージョン 2 では戻されなかった。
バージョン 2 と同等のものはない。	15 SQL_DATETIME_SUB	この情報はバージョン 2 では戻されなかった。
15 CHAR_OCTET_LENGTH	16 CHAR_OCTET_LENGTH	列番号が変更。名前と説明は変更なし。
16 ORDINAL_POSITION	17 ORDINAL_POSITION	列番号が変更。名前と説明は変更なし。
17 IS_NULLABLE	18 IS_NULLABLE	列番号が変更。名前と説明は変更なし。

SQLProcedureColumns() 戻り値の変更

次の表は、SQLProcedureColumns() によって戻される列への変更を、バージョン 2.1.1 とバージョン 5 で対比したリストです。

現在の値については、643ページの『SQLProcedureColumns で戻される列』を参照してください。

DB2 CLI をバージョン 2 の場合と同じように (同じ列名および順序で) 動作させるには、 DB2 CLI/ODBC 構成キーワード PATCH2 を設定します。このキーワード設定について、詳しくは 167ページの『CLI/ODBC 構成キーワードの設定方法』を参照してください。

表 185. SQLProcedureColumns で戻される列の変更

バージョン 2 の列	バージョン 5 の列	変更
14 ORDINAL_POSITION	18 ORDINAL_POSITION	列番号が変更。名前と説明は変更なし。

SQLGetInfo() の InfoTypes の変更

新しい *InfoTypes* を多数 SQLGetInfo() に追加しただけでなく、以下のバージョン 2 の値がバージョン 5 では名前変更されています。

表 186. SQLGetInfo() で使用する値の変更

バージョン 2 の InfoType	バージョン 5 の InfoType
SQL_ACTIVE_CONNECTIONS	SQL_MAX_DRIVER_CONNECTIONS
SQL_ACTIVE_STATEMENTS	SQL_MAX_CONCURRENT_ACTIVITIES
SQL_MAX_OWNER_NAME_LEN	SQL_MAX_SCHEMA_NAME_LEN
SQL_MAX_QUALIFIER_NAME_LEN	SQL_MAX_CATALOG_NAME_LEN
SQL_ODBC_SQL_OPT_IEF	SQL_INTEGRITY
SQL_SCHEMA_TERM	SQL_OWNER_TERM
SQL_OWNER_USAGE	SQL_SCHEMA_USAGE
SQL_QUALIFIER_LOCATION	SQL_CATALOG_LOCATION
SQL_QUALIFIER_NAME_SEPARATOR	SQL_CATALOG_NAME_SEPARATOR
SQL_QUALIFIER_TERM	SQL_CATALOG_TERM
SQL_QUALIFIER_USAGE	SQL_CATALOG_USAGE

デフォルトでの据え置き準備

DB2 CLI バージョン 5 では、デフォルトで据え置き準備がオンになります。対応する実行要求が発行されるまで、PREPARE 要求はサーバーに送られません。それから、2 つの要求は、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に、 (2 つではなく) 1 つのコマンド / 応答フローに結合されます。これが最大の益となるのは、アプリケーションが照会を生成して応答のセットが非常に少ない場合や、別々の要求と回答のオーバーヘッドが照会データの複数ブロックに広がっていない場合です。 DB2 コネクト接続や DDCS ゲートウェイを使用する環境では、要求と回答の 4 つの組み合わせが 2 つに減るため、コスト削減の機会ともなります。

要求次第 PREPARE が実行されることを期待する DB2 CLI バージョン 2 アプリケーションでは、期待どおりの操作にならないことがあります。 (アプリケーションは、たとえば、通常は準備ステートメントの SQLCA の SQLERRD(3) と SQLERRD(4) に戻される行およびコスト見積もりに依存しますが、据え置き準備を使用可能にすると、これらの値はゼロになることがあります。) このようなプログラムに、バージョン 2 におけるのと同様の働きをさせるには、 DB2 CLI/ODBC 構成キーワード DEFERREDPREPARE を設定して、据え置き準備を使用不能にします。詳しくは 195ページの『DEFERREDPREPARE』を参照してください。

また、ステートメント属性 SQL_ATTR_DEFERRED_PREPARE を使用して、それが発行されたならただちに DB2 CLI がステートメントを準備するようにす

ることもできます。詳しくは、756ページの『SQLSetStmtAttr - ステートメントに関連したオプションの設定』を参照してください。

バージョン 2 では、DB2 CLI アプリケーションは、関数 SQLSetColAttributes() を使用し、結果セットのすべての列に結果記述子情報を記述することによって、ネットワーク通信量を減らすことができました。据え置き準備では、この方法は無駄です。また、 SQLSetColAttributes() は使用すべきでない関数になっています。アプリケーションがこの関数を呼び出すと、すべての引き数が無視されることになり、必ず SQL_SUCCESS が戻されます。

バージョン 2.1.0 から 2.1.1 への変更点

複数行の結果セットを戻すストアード・プロシージャー

DB2 CLI の前のバージョンでは、複数行の結果セットをサポートしていませんでした。バージョン 2.1.1 では、ストアード・プロシージャーが存在するときに、それぞれが照会と関連付けられている 1 つまたは複数のカーソルをオープンして、ストアード・プロシージャー呼び出しから 1 つまたは複数の結果セットを取り出すことができるようになりました。

SQLGetInfo のデータ変換および値

DB2 CLI バージョン 2.1.1 では、ODBC によって定義される、ベンダー・エスケープ文節を使用した変換関数をサポートするようになりました。この関数により、chars (CHAR、VARCHAR、 LONG VARCHAR および CLOB) と DOUBLE (または FLOAT) の間で変換を行えます。

DB2 CLI バージョン 2.1.0 では、SQLGetInfo() を指定したすべての SQL_CONVERT *fInfoTypes* にゼロを戻しました。現在のバージョン 2.1.1 では変換をサポートしており、 SQLGetInfo() は、SQL_CONVERT (たとえば、SQL_CONVERT_INTEGER など) で始まる *fInfoTypes* であり、 SQL_CVT_ (たとえば、SQL_CVT_CHAR など) で始まるビット・マスクとの比較に使用できるものに、セットになったビット・マスクを戻します。

CONVERT 関数に加え、DB2 CLI バージョン 2.1.1 では、 ODBC ベンダー・エスケープ文節を使用してアクセスできる、以下の 2 つの日時の関数が提供されます。

JULIAN_DAY(day_expr)

紀元前 4712 年 1 月 1 日から数えた日数に対応する整数値を date_exp に戻します。

SECONDS_SINCE_MIDNIGHT(time_expr)
午前零時から数えた秒数に対応する整数値を time_expr に戻します。

バージョン 1.x から 2.1.0 への変更点

バージョン 2.1.0 と 1.x の相違点を以下で説明します。

AUTOCOMMIT と CURSOR WITH HOLD の省略時値

DB2 CLI の以前のバージョンでは、自動コミットはサポートされておらず (各ステートメントはトランザクションである)、そのため AUTOCOMMIT が**オフ**に設定されているのと同じことでした。 DB2 CLI バージョン 2.1 になって自動コミットがサポートされるようになりましたが、 ODBC との整合性をとるために、省略時の自動コミットの動作は**オン**になっています。

既存の DB2 CLI アプリケーションを以前のバージョンと同じ動作で実行させるには、 AUTOCOMMIT キーワードを `0` に設定します。このキーワードはすべてのアプリケーションに適用されます。それで、新規のアプリケーションではこのキーワードを明示的に指定変更し、自動コミット接続オプションを要求された値に設定します。

バージョン 2.1 では CURSOR WITH HOLD の指定がサポートされており、その省略時設定は、*保留の指定* が**オン**です。以前のバージョンではこの指定がサポートされていなかったので、保留の指定 (with hold) は事実上**オフ**に設定されていました。

この変更は自動コミットほどアプリケーションの動作に影響を与えるものではありませんが、 CURSORHOLD キーワードは `0` (コミット後にカーソルは保守されない) に設定してください。このキーワードはすべてのアプリケーションに適用されます。それで、新規のアプリケーションではこのキーワードを明示的に指定変更し、 CURSORHOLD オプションを要求された値に設定します。

図形データ・タイプの値

以下に示す `#define` 値は、

- SQL_GRAPHIC
- SQL_VARGRAPHIC
- SQL_LONGVARGRAPHIC

ODBC アプリケーションで使用できるようにバージョン 2.1 で変更されています。 DB2 CLI は以前の値も受け入れますが、これらの値を使用する既存のアプリケーションを再コンパイルすることをお勧めします。

上記の値は sqlcli.h ヘッダー・ファイルに定義されています。

SQLSTATES

以前のバージョンでは、DB2 CLI は **S1**009 SQLSTATE を戻し、 ODBC 定義のより明示的な **S1**090～**S1**110 の一連の SQLSTATE は戻しません。

X/Open もこの範囲の SQLSTATE を使用しているので、 DB2 CLI バージョン 2.1 も、このより明示的な SQLSTATE を戻します。

組み込み SQL の混合 (CONNECT RESET を用いない場合)

DB2 CLI のバージョン 2.1 で複数の接続をサポートしていることにより、組み込み SQL と DB2 CLI を混合して使用している既存のアプリケーションには影響が出る可能性があります。

ご使用のアプリケーションが、以下の条件を満たしている場合、

1. 組み込み SQL を使用して (コマンド行プロセッサーや管理 API の使用を含む) データベースに接続している。
2. (リセットを出さ「ない」)。
3. DB2 CLI を使用してデータベースに接続している。

2 番目の接続は、最初の接続と接続タイプが異なるため、失敗します。

アプリケーションは、DB2 CLI 関数を呼び出す前に、CONNECT RESET を出すことが必要になります。

注: アプリケーション側では、常に接続を明示的にリセットするようにします。

VARCHAR FOR BIT DATA の使用

FOR BIT DATA 文節を用いて定義された文字データは、 SQL_C_BINARY の省略時 C バッファー・タイプに関連付けられています。データが FOR BIT DATA として定義されている場合は、そのデータは以下の宛先に転送されます。

- 未変更の SQL_C_BINARY バッファー。
- SQL_C_CHAR バッファー (データの 16 進値の文字表示として)。各バイトは ASCII 文字 2 文字によって表されます。 (これは、SQL_C_CHAR が FOR BIT DATA ストリングの 2 倍のサイズでなければならないことを意味します。)

SQL_C_CHAR に FOR BIT DATA として定義されたデータを指定して明示的に使用する既存のアプリケーションは、入手する結果が異なり、元のデータの

半分だけを受け取ることになる可能性があります。初期設定キーワード BITDATA を 0 に設定することにより、 DB2 CLI が FOR BIT DATA を必ず以前のバージョンと同様に処理するようにしておくことができます。

述部内でのユーザー定義タイプ

既存のアプリケーションは、ユーザー定義タイプを利用できるように表が修正されているかどうかによって影響を受ける場合があります。

照会ステートメントの述部にパラメーター・マーカーが使用されており、かつ、そのパラメーターがユーザー定義タイプである場合には、ステートメントに CAST 関数を使用して、パラメーター・マーカーまたは UDT のいずれかをキャストしなければなりません。

たとえば、次のようにタイプおよび表が定義されているとします。

```
CREATE DISTINCT TYPE CNUM AS INTEGER WITH COMPARISONS
CREATE TABLE CUSTOMER (
        Cust_Num     CNUM NOT NULL,
        First_Name   CHAR(30) NOT NULL,
        Last_Name    CHAR(30) NOT NULL,
        Phone_Num    CHAR(20) WITH DEFAULT,
        PRIMARY KEY  (Cust_Num) )
```

このステートメントはパラメーター・マーカーがタイプ CNUM にはならないために失敗し、したがってタイプに互換性がないことから比較が失敗して、次のようになります。

```
SELECT  first_name, last_name, phone_num FROM customer
where cust_num = ?
```

列を整数 (その基本 SQL タイプ) にキャストすると、パラメーターには整数のタイプが与えられるので、比較を実行することができます。

```
SELECT  first_name, last_name, phone_num from customer
where cast( cust_num as integer ) = ?
```

あるいは、パラメーター・マーカーを INTEGER にキャストすることもできます。これにより、サーバーにパラメーター・マーカーのタイプが通知され、省略時設定の INTEGER - CNUM 変換を適用することができます。

```
SELECT  first_name, last_name, phone_num FROM customer
where cust_num = cast( ? as integer )
```

完全な作業サンプルについては、custrep.c サンプル・ファイルを参照してください。

以下の詳細については、*SQL 解説書* を参照してください。

- UDT。CREATE DISTINCT TYPE を参照。
- パラメーター・マーカー。PREPARE ステートメントを参照。
- キャスト。CAST 関数を参照。

SQLGetInfo のデータ変換値

バージョン 2.1 以前のバージョンでは、DB2 CLI は SQL_CONVERT_ で始まる (たとえば、 SQL_CONVERT_INTEGER) *fInfoTypes* のビット・マスクの集まりを戻します。 *fInfoTypes* は、ここでは SQL_CVT_ で始まる (たとえば、SQL_CVT_CHAR) 比較ビット・マスクに対応して用いられています。

この *fInfoType* は、サポートされている変換関数を示すために ODBC によって定義されたものであり、現在これらの関数はサポートされていないため、DB2 CLI バージョン 2.1 は今のところすべての SQL_CONVERT *fInfoTypes* に (正しく) ゼロを戻します。

関数プロトタイプの変更

X/Open と ODBC の連携を高めるため、 DB2 CLI 関数の引き数の一部が無符号データ・タイプに変更され、かつ 2 つのタイプ SQLUINTEGER および SQLUSMALLINT が新たに導入されました。

バージョン 2.1 以前の DB2 CLI アプリケーションでは、バージョン 2.1 のヘッダー・ファイルを用いてコンパイルをかけたときに DB2CLI_VER が定義されていないと、関数引き数の不一致によるコンパイラー・エラーが生成されます。 『DB2CLI_VER 定義の設定』を参照してください。

SQLUINTEGER および SQLUSMALLINT のタイプを使用することによって既存のソース・ファイルを修正し、必要な引き数を宣言しておくことをお勧めします。

DB2CLI_VER 定義の設定

DB2CLI_VER 定義により、アプリケーションは DB2 CLI ヘッダー・ファイルが以前のバージョンの DB2 CLI との互換性を保つように指定することができます。

DB2CLI_VER は、DB2 CLI を組み込む前にコンパイル・フラグまたは `#define` のいずれかとして 16 進値で設定しなければなりません。たとえば、`-D` コンパイラー・フラグを使用した場合、

```
-DDB2CLI_VER=0x0110
```

は、DB2CLI_VER をバージョン 1.1 に設定していることを表します。

バージョン 2.1 で DB2CLI_VER が定義されていない場合、省略時設定で 0x0210 になります。

付録C. DB2 CLI と ODBC

この付録では、DB2 ODBC ドライバーで用意されているサポートについて説明するとともに、 DB2 CLI との相違点も説明します。

下記の 836ページの図18 では、DB2 CLI と DB2 ODBC ドライバーを比較しています。

1. ODBC ドライバー・マネージャーの ODBC ドライバー
2. DB2 CLI、DB2 固有のアプリケーション用に設計された呼び出し可能インターフェース

DB2 クライアントとは、使用可能なすべての DB2 クライアントを指します。DB2 とある場合、すべての DB2 ユニバーサル・データベース製品を指します。

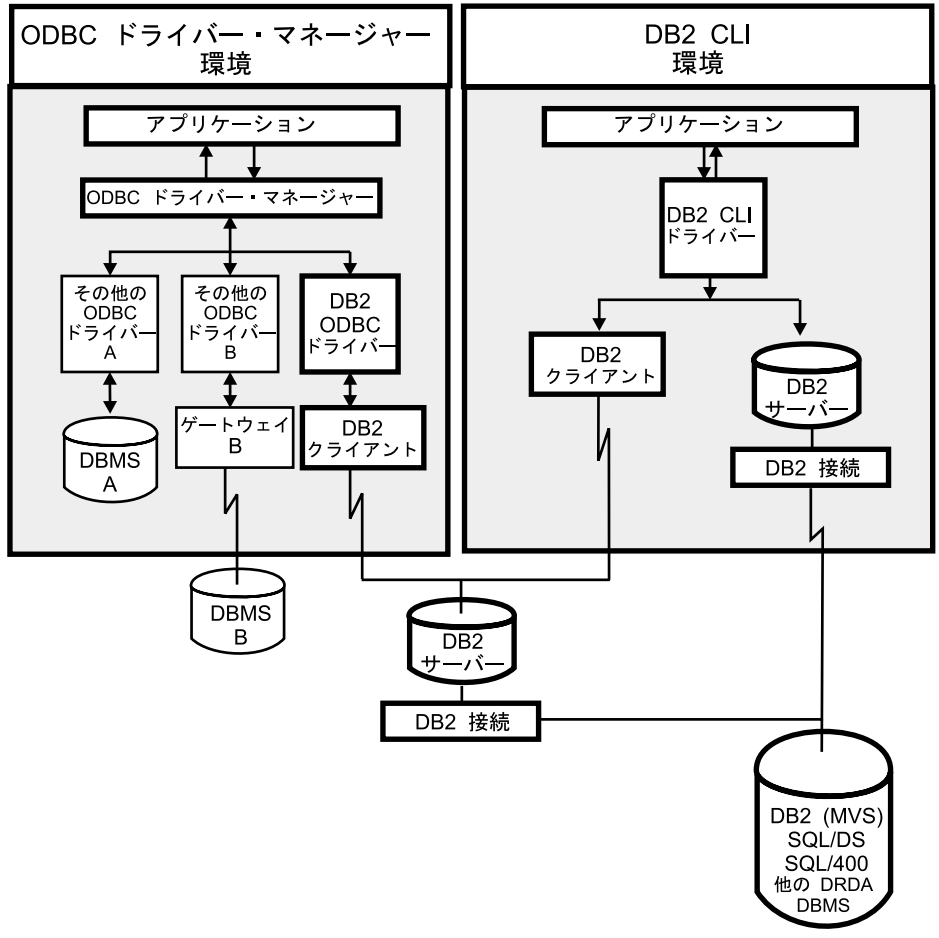


図 18. DB2 CLI と ODBC.

ODBC 環境では、ドライバー・マネージャーがアプリケーションへのインターフェースを提供します。また、アプリケーションの接続先のデータベース・サーバーに必要なドライバー を動的にロードします。 ODBC 関数の集まりを使用するのはドライバーです。ただし、いくつかの拡張機能は例外で、ドライバー・マネージャーによって使用されます。この環境では、DB2 CLI は、ODBC 2.0 のレベル 2 および ODBC 3.0 のレベル 1 に合致します。さらに、以下の ODBC 3.0 レベル 2 インターフェース合致項目とも合致します。

202 SQLDescribeParam を使用して記述した動的パラメーター

203 入力、出力、および入出力パラメーターのほか、ストアード・プロシージャーの結果値のサポート

205 SQLColumnPriviledge、SQLForeignKey、および SQLTablePrivileges からのデータ・ディクショナリーに関する拡張情報

207 該当する関数の非同期実行

209 SQL_ATTR_CONCURRENCY ステートメント属性は、SQL_CONCUR_READ_ONLY 以外の少なくとも 1 つの値に設定可能

211 省略時の分離レベルは変更可能。トランザクションは、“シリアル化” レベルの分離で実行可能

ODBC アプリケーションの開発の際には、 ODBC ソフトウェア開発キットを (Microsoft プラットフォームの場合は Microsoft から、 Microsoft 以外のプラットフォームの場合は他のベンダーから) 入手してください。 DB2 サーバーに接続する可能性のある ODBC アプリケーションを開発する場合、本書 (DB2 特定の拡張についての情報および診断情報) と *ODBC 3.0 Software Development Kit and Programmer's Reference* を併用してください。

ODBC ドライバー・マネージャーのない環境では、DB2 CLI は自己完結的なドライバーとなり、 ODBC ドライバーが提供する関数のサブセットをサポートします。 表187 には 2 つのレベルのサポートが要約されており、また、235 ページの表12 には ODBC 3.0 関数の完全なリストがあって、それらがサポートされているかどうかが示されています。

表 187. DB2 CLI ODBC サポート

ODBC 機能	DB2 ODBC ドライバー	DB2 CLI
コア・レベル関数	すべて	すべて
レベル 1 関数	すべて	すべて
レベル 2 関数	すべて	SQLDrivers() 以外すべて
付加的な DB2 CLI 関数	すべて。関数は DB2 CLI ライブラリーを動的にロードすることによってアクセス可能。詳細については 811ページの『付録A. プログラミングのヒントと提案』を参照。	• SQLSetConnection() • SQLGetEnvAttr() • SQLSetEnvAttr() • SQLSetColAttributes() • SQLGetSQLCA() • SQLBindFileToCol() • SQLBindFileToParam() • SQLExtendedBind() • SQLExtendedPrepare() • SQLGetLength() • SQLGetPosition() • SQLGetSubString()

表 187. DB2 CLI ODBC サポート (続き)

ODBC 機能	DB2 ODBC ドライバー	DB2 CLI
SQL データ・タイプ	DB2 CLI 用にリストされているすべてのタイプ、および次のもの • SQL_BINARY • SQL_VARBINARY • SQL_LONGVARBINARY	• SQL_BIGINT • SQL_BINARY • SQL_BLOB • SQL_BLOB_LOCATOR • SQL_CHAR • SQL_CLOB • SQL_CLOB_LOCATOR • SQL_DBCLOB • SQL_DBCLOB_LOCATOR • SQL_DECIMAL • SQL_DOUBLE • SQL_FLOAT • SQL_GRAPHIC • SQL_INTEGER • SQL_LONGVARBINARY • SQL_LONGVARCHAR • SQL_LONGVARGRAPHIC • SQL_NUMERIC • SQL_REAL • SQL_SMALLINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_VARBINARY • SQL_VARCHAR • SQL_VARGRAPHIC

表 187. DB2 CLI ODBC サポート (続き)

ODBC 機能	DB2 ODBC ドライバー	DB2 CLI
C データ・タイプ	DB2 CLI 用にリストされているすべてのタイプ、および次のもの。	• SQL_C_BINARY • SQL_C_BIT • SQL_C_BLOB_LOCATOR • SQL_C_CHAR • SQL_C_CLOB_LOCATOR • SQL_C_DATE • SQL_C_DBCHAR • SQL_C_DBCLOB_LOCATOR • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_SHORT • SQL_C_TIME • SQL_C_TIMESTAMP • SQL_C_TINYINT • SQL_C_SBIGINT • SQL_C_UBIGINT • SQL_C_NUMERIC ** **** 32 ビット Windows 上でのみサポートされる**
戻りコード	DB2 CLI 用にリストされているすべてのコード。	• SQL_SUCCESS • SQL_SUCCESS_WITH_INFO • SQL_STILL_EXECUTING • SQL_NEED_DATA • SQL_NO_DATA_FOUND • SQL_ERROR • SQL_INVALID_HANDLE
SQLSTATES	付加的な IBM SQLSTATES を用いて X/Open SQLSTATES にマッピングされる。例外は ODBC タイプ **08**S01。	付加的な IBM SQLSTATES を用いて X/Open SQLSTATES にマッピングされる。
アプリケーションごとに複数の接続	サポートされる	サポートされる
ドライバーの動的ロード	サポートされる	該当せず

ODBC について、詳しくは *ODBC 3.0 Software Development Kit and Programmer's Reference* を参照してください。

ODBC 関数リスト

235ページの表12 は、すべての Microsoft ODBC 3.0 関数を完全にリストしたものです。 ODBC 準拠レベルとそれが DB2 CLI によってサポートされるかどうかを、関数ごとに示します。

DB2 CLI 関数の完全なリスト、および X/Open および ISO 呼び出し可能 SQL 規格については、 235ページの『DB2 CLI 関数の要約』を参照してください。

分離レベル

次の表は、IBM RDBM 分離レベルを ODBC トランザクション分離レベルにマップしています。 SQLGetInfo() 関数は、使用可能な分離レベルを示します。

表 188. ODBC での分離レベル

IBM 分離レベル	ODBC 分離レベル
カーソル固定	SQL_TXN_READ_COMMITTED
反復可能読み取り	SQL_TXN_SERIALIZABLE_READ
読み取り固定;	SQL_TXN_REPEATABLE_READ
非コミット読み取り	SQL_TXN_READ_UNCOMMITTED
コミットなし	(ODBC には同等のものはない)
注: サポートされていない分離レベルを設定しようとすると、 SQLSetConnectOption() および SQLSetStmtOption は、 **HY**009 の SQLSTATE で SQL_ERROR を戻します。	

付録D. 拡張スカラー関数

以下の関数は、ODBC でベンダー・エスケープ文節を使用して定義されます。各関数は、エスケープ文節構文を使用するか、または同等の DB2 関数を呼び出すことによって呼び出すことができます。

これらの関数は、次のように区分されています。

- 842ページの『ストリング関数』
- 845ページの『数値関数』
- 848ページの『日時関数』
- 852ページの『システム関数』
- 853ページの『変換関数』

ベンダー・エスケープ文節の詳細については、154ページの『ODBC スカラー関数』を参照してください。

以下の節にでてくる表には、バージョン 2.1 の DB2 CLI を使用してアプリケーションから呼び出したときに、関数にアクセスできるサーバー (およびその最も古いバージョン) が示されています。

DB2 バージョン 2 サーバーへ接続したときに、以下の関数によって検出されたすべてのエラーは、SQLSTATE 38552 を戻します。メッセージのテキスト部分は、SYSFUN:*nn* という書式になります。ここで *nn* は、以下の理由コードの 1 つです。

01 範囲外の数値。
02 ゼロ除算。
03 算術桁あふれまたは下位桁あふれ。
04 無効な日付形式。
05 無効な時刻形式。
06 無効なタイム・スタンプ形式。
07 タイム・スタンプ期間の無効な文字表記。
08 無効な間隔タイプ。 (1、2、4、8、16、32、64、128、256 の 1 つでなければならない)
09 ストリングが長すぎる。
10 ストリング関数の長さまたは位置が範囲外。
11 浮動小数点数の無効な文字表記。

ストリング関数

このセクションのストリング関数は、DB2 CLI でサポートされ、 ODBC でベンダー・エスケープ文節を使用して定義されます。

- スカラー関数に対する引き数として使用される文字ストリング・リテラルは、単一引用符でバインドしなくてはなりません。
- *string_exp* として示される引き数は、列の名前、ストリング・リテラル、または別のスカラー関数の結果であり、基礎となるデータ・タイプは、SQL_CHAR、SQL_VARCHAR、SQL_LONGVARCHAR、または SQL_CLOB として表せます。
- *start*、*length*、*code* または *count* として示される引き数は、数値リテラルまたは別のスカラー関数の結果であり、基礎となるデータ・タイプは、整数ベースのものです (SQL_SMALLINT、SQL_INTEGER)。
- ストリングの先頭文字は、位置 1 にあるとみなされます。

表 189. ストリング・スカラー関数

ASCII(*string_exp* **)**

string_exp の左端の文字の ASCII コード値を整数として戻します。

DB2	(共通サーバー版 2.1)			

CHAR(*code* **)**

code で指定された ASCII コード値がある文字を戻します。 *code* の値は、0 から 255 までででなければなりません。それ以外は、戻り値は NULL です。

DB2	(共通サーバー版 2.1)			

CONCAT(*string_exp1*, *string_exp2* **)**

string_exp2 を *string_exp1* に連結した結果の文字ストリングを戻します。

DB2	(共通サーバー版 1.1)	MVS	VM/VSE	AS/400

DIFFERENCE(*string_exp1, string_exp2* **)**

整数値を戻しますが、これは、 SOUNDEX 関数によって *string_exp1* と *string_exp2* 用に戻される値の差を示します。

DB2	(共通サーバー版 1.1)			

表 189. ストリング・スカラー関数 (続き)

INSERT(*string_exp1, start, length, string_exp2* **)**

start から始まる *length* 個の文字が、 *length* 文字を含む *string_exp2* によって置換された文字ストリングを戻します。

DB2	(共通サーバー版 1.1)	MVS	VM/VSE	AS/400

LCASE(*string_exp* **)**

string_exp 内のすべての大文字を小文字に変換します。

DB2	(共通サーバー版 1.0)		VM/VSE	

LEFT(*string_exp,count* **)**

string_exp の文字の左端の *count* を戻します。

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

LENGTH(*string_exp* **)**

後書きブランクおよびストリング終了文字を除く、*string_exp* の文字数を戻します。

注: 後書きブランクはバージョン 2.1 以前には組み込まれていました。現在でも、DB2 (MVS/ESA 版) には後書きブランクが組み込まれています。

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

LOCATE(*string_exp1*, *string_exp2* **[** ,*start* **])**

string_exp2 内の *string_exp1* の先頭オカレンスの開始位置を戻します。 *string_exp1* の先頭オカレンスの検索は、任意選択の引き数 *start* を指定しなければ、*string_exp2* 内の先頭文字の位置から始まります。 *start* を指定すると、検索は *start* の値で示される文字の位置から始まります。 *string_exp2* の先頭文字位置は値 1 で示されます。 *string_exp1* が *string_exp2* で見つからないと、値 0 が戻されます。

DB2	(共通サーバー版 2.1)			

LTRIM(*string_exp* **)**

先行ブランクを除いて *string_exp* の文字を戻します。

DB2	(共通サーバー版 2.1)	VM/VSE		AS/400

表 189. ストリング・スカラー関数 (続き)

REPEAT(*string_exp*, *count* **)**
string_exp が *count* 回繰り返された文字ストリングを戻します。

DB2	(共通サーバー版 2.1)	MVS	VM/VSE	AS/400

REPLACE(*string_exp1*, *string_exp2*, *string_exp3* **)**
string_exp1 内の *string_exp2* のすべてのオカレンスを *string_exp3* で置換します。

DB2	(共通サーバー版 2.1)			

RIGHT(*string_exp*, *count* **)**
string_exp の文字の右端の count を戻します。

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

RTRIM(*string_exp* **)**
後書きブランクを除いて *string_exp* の文字を戻します。

DB2	(共通サーバー版 2.1)		VM/VSE	AS/400

SOUNDEX(*string_exp1* **)**
string_exp1 の音を表す 4 文字コードを戻します。データ・ソースが異なれば、*string_exp1* の音を表すアルゴリズムも異なるものを使用することに注意してください。

DB2	(共通サーバー版 1.1)			

SPACE(*count* **)**
count 個のスペースから成る文字ストリングを戻します。

DB2	(共通サーバー版 2.1)			

SUBSTRING(*string_exp*, *start*, *length* **)**
length 文字の、 *start* で指定された文字位置から始まる *string_exp* から導出された文字ストリングを戻します。

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

UCASE(*string_exp* **)**
string_exp 内のすべての小文字を大文字に変換します。

表 189. ストリング・スカラー関数 (続き)

DB2	(共通サーバー版 1.0)		VM/VSE	AS/400

数値関数

この節に示す数値関数は、DB2 CLI でサポートされ、 ODBC でベンダー・エスケープ文節を使用して定義されます。

- *numeric_exp* として示される引き数は、列の名前、別のスカラー関数の結果、または数値リテラルであり、基礎となるデータ・タイプは、浮動小数点ベース (SQL_NUMERIC、SQL_DECIMAL、SQL_FLOAT、SQL_REAL、SQL_DOUBLE)、または整数ベース (SQL_SMALLINT、SQL_INTEGER) のいずれかです。
- *double_exp* として示される引き数は、列の名前、別のスカラー関数の結果、または数値リテラルで、基礎となるデータ・タイプは浮動小数点ベースです。
- *integer_exp* として示される引き数は、列の名前、別のスカラー関数の結果、または数値リテラルで、基礎となるデータ・タイプは整数ベースです。

表 190. 数値スカラー関数

ABS(*numeric_exp* **)**
numeric_exp の絶対値を戻します。

DB2	(共通サーバー版 2.1)			AS/400

ACOS(*double_exp* **)**
角度としての *double_exp* の逆余弦を、ラジアンで表して戻します。

DB2	(共通サーバー版 2.1)			AS/400

ASIN(*double_exp* **)**
角度としての *double_exp* の逆正弦を、ラジアンで表して戻します。

DB2	(共通サーバー版 2.1)			AS/400

ATAN(*double_exp* **)**
角度としての *double_exp* の逆正接を、ラジアンで表して戻します。

DB2	(共通サーバー版 2.1)			AS/400

表 190. 数値スカラー関数 (続き)

ATAN2(*double_exp1*, *double_exp2* **)**

double_exp1 で指定された *x*座標、および *double_exp2* で指定された *y* 座標の逆正接を、ラジアンで表された角度として戻します。

DB2	(共通サーバー版 2.1)			

CEILING(*numeric_exp* **)**

numeric_exp 以上の最小整数を戻します。

DB2	(共通サーバー版 2.1)			

COS(*double_exp* **)**

double_exp がラジアンで表された角度のとき、 *double_exp* の余弦を戻します。

DB2	(共通サーバー版 2.1)			AS/400

COT(*double_exp* **)**

double_exp がラジアンで表された角度のとき、 *double_exp* の余接を戻します。

DB2	(共通サーバー版 2.1)			AS/400

DEGREES(*numeric_exp* **)**

numeric_exp ラジアンから変換された度数を戻します。

DB2	(共通サーバー版 2.1)			AS/400 3.6

EXP(*double_exp* **)**

double_exp の指数値を戻します。

DB2	(共通サーバー版 2.1)			AS/400

FLOOR(*numeric_exp* **)**

numeric_exp 以下の最大整数を戻します。

DB2	(共通サーバー版 2.1)			AS/400 3.6

LOG(*double_exp* **)**

double_exp の自然対数を戻します。

DB2	(共通サーバー版 2.1)			AS/400

表 190. 数値スカラー関数 (続き)

LOG10(*double_exp* **)**

double_exp の 10 を底とする対数 (常用対数) を戻します。

DB2	(共通サーバー版 2.1)			AS/400

MOD(*integer_exp1*, *integer_exp2* **)**

integer_exp2 で除算された *integer_exp1* の剰余 (モジュラス) を戻します。

DB2	(共通サーバー版 2.1)			AS/400

PI()

πの定数値を浮動小数点値として戻します。

DB2	(共通サーバー版 2.1)			AS/400

POWER(*numeric_exp*, *integer_exp* **)**

値 *numeric_exp* の *integer_exp* 乗を戻します。

DB2	(共通サーバー版 2.1)			AS/400 3.6

RADIANS(*numeric_exp* **)**

numeric_exp 度から変換されたラジアン数を戻します。

DB2	(共通サーバー版 2.1)			

RAND([*integer_exp* **])**

integer_exp を任意選択の核値として使用してランダム浮動小数点値を戻します。

DB2	(共通サーバー版 2.1)			

ROUND(*numeric_exp*, *integer_exp.* **)**

小数点の右 *integer_exp* 桁に丸められた *numeric_exp* を戻します。 *integer_exp* が負の数の場合、 *numeric_exp* は小数点の左 | *integer_exp* | 桁に丸められます。

DB2	(共通サーバー版 2.1)			

SIGN(*numeric_exp* **)**

numeric_exp の標識つまり符号を戻します。 *numeric_exp* がゼロより小さいと、-1 が戻されます。 *numeric_exp* がゼロの場合は、0 が戻されます。 *numeric_exp* がゼロより大きいと、1 が戻されます。

表 190. 数値スカラー関数 (続き)

DB2	(共通サーバー版 2.1)			

SIN(*double_exp* **)**
double_exp がラジアンで表される角度のとき、 *double_exp* の正弦を戻します。

DB2	(共通サーバー版 2.1)			AS/400

SQRT(*double_exp* **)**
double_exp の平方根を戻します。

DB2	(共通サーバー版 2.1)			AS/400

TAN(*double_exp* **)**
double_exp がラジアンで表される角度のとき、 *double_exp* の正接を戻します。

DB2	(共通サーバー版 2.1)			AS/400

TRUNCATE(*numeric_exp*, *integer_exp* **)**
小数点の右 *integer_exp* 桁に切り捨てられた *numeric_exp* を戻します。 *integer_exp* が負の数の場合、*numeric_exp* は、小数点の左 | *integer_exp* | 桁に切り捨てられます。

DB2	(共通サーバー版 2.1)			

日時関数

この節に示す日時関数は、DB2 CLI でサポートされ、 ODBC でベンダー・エスケープ文節を使用して定義されます。

- *timestamp_exp* として示される引き数は、列の名前、別のスカラー関数の結果、または時刻、日付、またはタイム・スタンプのリテラルです。
- *date_exp* として示される引き数は、列の名前、別のスカラー関数の結果、または日付かタイム・スタンプのリテラルで、基礎となるデータ・タイプは文字ベース、または日付かタイム・スタンプ・ベースです。
- *time_exp* として示される引き数は、列の名前、別のスカラー関数の結果、または時刻かタイム・スタンプのリテラルで、基礎となるデータ・タイプは文字ベース、または時刻かタイム・スタンプ・ベースです。

表 191. 日時スカラー関数

CURDATE()
現在日付を日付値として戻します。

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

CURTIME()
現在地域別時刻を時刻値として戻します。

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

DAYNAME(*date_exp* **)**
date_exp の曜日部分について、曜日の名前 (Sunday、Monday、Tuesday、Wednesday、Thursday、Friday、Saturday) を含む文字ストリングを戻します。

DB2	(共通サーバー版 2.1)			

DAYOFMONTH (*date_exp* **)**
date_exp の月間の日付を整数値として、1 から 31 までの範囲で戻します。

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

DAYOFWEEK(*date_exp* **)**
date_exp の週の曜日を整数値として、1 から 7 までの範囲で戻します。 1 は日曜日を表します。

DB2	(共通サーバー版 2.1)			AS/400 3.6

DAYOFYEAR(*date_exp* **)**
date_exp の年間の日付を整数値として、1 から 366 までの範囲で戻します。

DB2	(共通サーバー版 2.1)			AS/400 3.6

HOUR(*time_exp* **)**
time_exp の時間を整数値として、0 から 23 までの範囲で戻します。

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

JULIAN_DAY(*date_exp* **)**
紀元前 4712 年 1 月 1 日 (ユリウス日付カレンダーの開始日) から *date_exp* までの日数を返します。 (v2.1.0 から v2.1.1 データベースへ移行するときには、 移行ユーティリティーを実行して、この関数へアクセスしなければなりません。)

表 191. 日時スカラー関数 (続き)

DB2	(共通サーバー版 2.1.1)			

MINUTE(*time_exp* **)**

time_exp の分数を整数値として、0 から 59 までの範囲で戻します。

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

MONTH(*date_exp* **)**

date_exp の月数を整数値として、1 から 12 までの範囲で戻します。

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

MONTHNAME(*date_exp* **)**

date_exp の月部分について、月の名前 (January、February、March、April、 May、June、July、August、 September、October、November、December) を含む文字ストリングを戻します。

DB2	(共通サーバー版 2.1)			

NOW()

現在日付および時刻をタイム・スタンプ値として戻します。

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

QUARTER(*date_exp* **)**

date_exp の四半期を整数値として、1 から 4 までの範囲で戻します。

DB2	(共通サーバー版 2.1)			AS/400 3.6

SECOND(*time_exp* **)**

time_exp の秒数を整数値として、0 から 59 までの範囲で戻します。

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

SECONDS_SINCE_MIDNIGHT(*time_exp* **)**

午前零時から数えた *time_exp* における秒数を、 0 から 86400 の範囲の整数値で戻します。 *time_exp* に小数秒の構成要素がある場合、小数秒は廃棄されます。 (v2.1.0 から v2.1.1 データベースへ移行するときには、 移行ユーティリティーを実行して、この関数へアクセスしなければなりません。)

DB2	(共通サーバー版 2.1.1)			

表 191. 日時スカラー関数 (続き)

TIMESTAMPADD(*interval*, *integer_exp*, *timestamp_exp* **)**

タイプ *interval* の *integer_exp* 間隔を *timestamp_exp* に加算して計算されたタイム・スタンプを戻します。間隔の有効値は、次のとおりです。

- SQL_TSI_FRAC_SECOND
- SQL_TSI_SECOND
- SQL_TSI_MINUTE
- SQL_TSI_HOUR
- SQL_TSI_DAY
- SQL_TSI_WEEK
- SQL_TSI_MONTH
- SQL_TSI_QUARTER
- SQL_TSI_YEAR

小数秒は、十億分の一秒で表されます。 *timestamp_exp* が時刻値を指定し、*interval* が日、週、月、四半期、または年を指定する場合、 *timestamp_exp* の日付部分は、タイム・スタンプを計算して結果を得る前の現在日付に設定されます。*timestamp_exp* が日付値であり、 *interval* が小数秒、秒、分、または時間を指定する場合、 *timestamp_exp* の時刻部分は、タイム・スタンプを計算して結果を得る前の 00:00:00.000000 に設定されます。アプリケーションは、SQL_TIMEDATE_ADD_INTERVALS オプションを指定して ***SQLGetInfo()*** を呼び出し、どの間隔がサポートされているかを判別します。

DB2	(共通サーバー版 2.1)			

表 191. 日時スカラー関数 (続き)

TIMESTAMPDIFF(*interval*, *timestamp_exp1*, *timestamp_exp2* **)**

timestamp_exp2 が *timestamp_exp1* より大きい部分のタイプ *interval* の間隔の整数を戻します。間隔の有効値は、次のとおりです。

- SQL_TSI_FRAC_SECOND
- SQL_TSI_SECOND
- SQL_TSI_MINUTE
- SQL_TSI_HOUR
- SQL_TSI_DAY
- SQL_TSI_WEEK
- SQL_TSI_MONTH
- SQL_TSI_QUARTER
- SQL_TSI_YEAR

小数秒は、十億分の一秒で表されます。タイム・スタンプ式が時刻値であり、*interval* が日、週、月、四半期、または年を指定する場合、そのタイム・スタンプの日付部分は、タイム・スタンプどうしの差を計算する前の現在日付に設定されます。タイム・スタンプ式が日付値であり、 *interval* が小数秒、秒、分、または時間を指定する場合、そのタイム・スタンプの時刻部分は、タイム・スタンプどうしの差を計算する前の 0 に設定されます。アプリケーションは、SQL_TIMEDATE_DIFF_INTERVALS オプションを指定して ***SQLGetInfo()*** を呼び出し、どの間隔がサポートされているかを判別します。

DB2	(共通サーバー版 2.1)			

WEEK(*date_exp* **)**

date_exp の年間の週を整数値として、1 から 54 までの範囲で戻します。

DB2	(共通サーバー版 2.1)			AS/400 3.6

YEAR(*date_exp* **)**

date_exp の年数を整数値として、1 から 9999 までの範囲で戻します。

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

曜日の名前または月の名前を含む文字ストリングを戻す関数の場合、これらの文字ストリングは使用可能な各国語サポートになります。

システム関数

この節に示すシステム関数は、DB2 CLI でサポートされ、 ODBC でベンダー・エスケープ文節を使用して定義されます。

- *exp* として示される引き数は、列の名前、別のスカラー関数の結果、またはリテラルです。
- *value* として示される引き数は、リテラル定数です。

表 192. システム・スカラー関数

DATABASE()
接続ハンドルに対応するデータベースの名前を戻します (*hdbc*)。 (データベースの名前は情報タイプ SQL_DATABASE_NAME を指定した ***SQLGetInfo()*** によって得ることもできます。)

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

IFNULL(*exp*, *value*)
exp がヌルの場合、*value* が戻されます。 *exp* がヌルではない場合、*exp* が戻されます。 *value* に指定するデータ・タイプは、 *exp* のデータ・タイプと互換性がなければなりません。

DB2	(共通サーバー版 1.2)	MVS	VM/VSE	AS/400

USER()
ユーザーの許可名を戻します。 (ユーザーの許可名は、情報タイプの SQL_USER_NAME を指定した ***SQLGetInfo()*** によって得ることもできます。)

DB2	(共通サーバー版 1.0)	MVS	VM/VSE	AS/400

変換関数

変換関数は DB2 CLI によってサポートされており、ベンダー・エスケープ文節を使用して ODBC によって定義されています。

各ドライバーおよびデータ・ソースは、可能なデータ・タイプの間で、有効な変換を判別します。ドライバーは ODBC 構文をネイティブの構文に変換するので、 ODBC 構文が有効であるとしても、データ・ソースによってサポートされていない変換は拒否されます。

関数 ***SQLGetInfo()*** を適切な変換関数マスクと共に使用して、データ・ソースによってサポートされている変換を判別します。

表 193. 変換関数

CONVERT(*expr_value*, *data_type* **)**

- *data_type* は、 *expr_value* の変換後のデータ・タイプを示し、 SQL_CHAR または SQL_DOUBLE のいずれかになります。
- *expr_value* は、変換する値です。これは、ドライバーおよびデータ・ソースによりサポートされる変換の種類によって、さまざまなタイプになります。関数 ***SQLGetInfo()*** を適切な変換関数マスクと共に使用して、データ・ソースによってサポートされている変換を判別します。

(v2.1.0 から v2.1.1 データベースへ移行するときには、 移行ユーティリティーを実行して、この関数へアクセスしなければなりません。)

DB2	(共通サーバー版 2.1)			

付録E. SQLSTATE 相互参照

この表は、すべての SQLSTATE の相互参照です。それらは、『第5章 DB2 CLI 関数』にある各関数の説明の*診断* の節にリストされています。

注: DB2 CLI は、この表にはリストされていない、サーバーによって生成される SQLSTATE を戻す場合もあります。戻された SQLSTATE がこの表にリストされていない場合、追加の SQLSTATE 情報については、サーバー用の資料を参照してください。

SQLState 相互参照表

01000 (警告。)

- SQLAllocHandle()
- SQLBrowseConnect()
- SQLBuildDataLink()
- SQLBulkOperations()
- SQLCloseCursor()
- SQLColAttribute()
- SQLCopyDesc()
- SQLDescribeParam()
- SQLEndTran()
- SQLExtendedPrepare()
- SQLFetchScroll()
- SQLFreeHandle()
- SQLGetConnectAttr()
- SQLGetDataLinkAttr()
- SQLGetDescField()
- SQLGetDescRec()
- SQLGetStmtAttr()
- SQLSetConnectAttr()
- SQLSetDescField()
- SQLSetDescRec()
- SQLSetPos()
- SQLSetStmtAttr()

01002 (切断エラーです。)

- SQLDisconnect()

01004 (データが切り捨てられました。)

- SQLBrowseConnect()
- SQLBuildDataLink()
- SQLBulkOperations()
- SQLColAttribute()
- SQLDataSources()
- SQLDescribeCol()
- SQLDriverConnect()
- SQLExtendedFetch()
- SQLFetch()
- SQLFetchScroll()
- SQLGetConnectAttr()
- SQLGetCursorName()
- SQLGetData()
- SQLGetDataLinkAttr()
- SQLGetDescField()
- SQLGetDescRec()
- SQLGetInfo()
- SQLGetStmtAttr()
- SQLGetSubString()
- SQLNativeSql()
- SQLPutData()
- SQLSetPos()

01504 (UPDATE または DELETE ステートメントに WHERE 文節がありません。)

- SQLExecDirect()
- SQLPrepare()
- SQLExtendedPrepare()

01508 (ステートメントはブロック化できませんでした。)

- SQLExecDirect()
- SQLPrepare()
- SQLExtendedPrepare()

01S00 (接続ストリング属性が無効です。)

- SQLBrowseConnect()
- SQLDriverConnect()

01S01 (行の中にエラーがあります。)

- SQLExtendedFetch()
- SQLFetchScroll()
- SQLSetPos()
- SQLBulkOperations()

01S02 (オプション値が変更されました。)

- SQLBrowseConnect()
- SQLSetConnectAttr()
- SQLSetDescField()
- SQLSetStmtAttr()
- SQLExtendedPrepare()

01S06 (結果セットが最初の行設定を戻す前に取り出そうとしています。)

- SQLFetchScroll()

01S07 (小数が切り捨てられました。)

- SQLFetchScroll()
- SQLSetPos()
- SQLBulkOperations()

07001 (パラメーターの数が正しくありません。)

- SQLExecDirect()

07002 (列が多すぎます。)

- SQLExtendedFetch()
- SQLFetch()
- SQLFetchScroll()

07005 (ステートメントが結果セットを返しませんでした。)

- SQLColAttribute()
- SQLDescribeCol()

07006 (変換が無効です。)

- SQLBindParameter()
- SQLBulkOperations()
- SQLExecDirect()
- SQLExtendedBind()
- SQLExtendedFetch()
- SQLFetch()
- SQLFetchScroll()
- SQLGetData()
- SQLGetLength()
- SQLGetPosition()
- SQLGetSubString()
- SQLParamData()
- SQLSetPos()

07009 (無効な記述子索引)

- SQLBindCol()
- SQLBulkOperations()
- SQLColAttribute()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLExtendedBind()
- SQLFetch()
- SQLFetchScroll()
- SQLGetDescField()
- SQLGetDescRec()
- SQLSetDescField()
- SQLSetDescRec()
- SQLSetPos()

08001 (データ・ソースに接続できませんでした。)

- SQLBrowseConnect()
- SQLConnect()

08002 (接続が使用中です。)

- SQLBrowseConnect()
- SQLConnect()
- SQLSetConnectAttr()

08003 (接続がクローズされています。)

- SQLAllocHandle()
- SQLDisconnect()
- SQLEndTran()
- SQLGetConnectAttr()
- SQLGetInfo()
- SQLNativeSql()
- SQLSetConnectAttr()
- SQLSetConnection()
- SQLTransact()

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。)

- SQLBrowseConnect()
- SQLConnect()

08007 (トランザクション中に、接続に障害が起きました。)

- SQLEndTran()
- SQLTransact()

08S01 (通信リンクに障害が起きました。)

- SQLBrowseConnect()
- SQLCopyDesc()
- SQLDescribeParam()
- SQLExtendedPrepare()
- SQLFetchScroll()
- SQLFreeHandle()
- SQLGetDescField()
- SQLGetDescRec()
- SQLSetConnectAttr()
- SQLSetDescField()
- SQLSetDescRec()
- SQLSetStmtAttr()

0F001 (LOB トークン変数は、現在何も値を表していません。)

- SQLGetLength()
- SQLGetPosition()
- SQLGetSubString()

21S01 (挿入値リストが列リストと一致しません。)

- SQLDescribeParam()
- SQLExecDirect()
- SQLPrepare()
- SQLExtendedPrepare()

21S02 (派生表の程度が列リストと一致しません。)

- SQLExecDirect()
- SQLPrepare()
- SQLSetPos()
- SQLBulkOperations()
- SQLExtendedPrepare()

22001 (ストリング・データの右側が切り捨てられました。)

- SQLExecDirect()
- SQLFetchScroll()
- SQLPutData()
- SQLSetPos()
- SQLBulkOperations()

22002 (無効な出力または標識バッファーが指定されました。)

- SQLExtendedFetch()
- SQLFetch()
- SQLFetchScroll()
- SQLGetData()

22003 (数値が範囲外です。)

- SQLBulkOperations()
- SQLExecDirect()
- SQLExtendedFetch()
- SQLFetch()
- SQLFetchScroll()
- SQLGetData()
- SQLPutData()
- SQLSetPos()

22005 (割り当てにエラーがありました。)

- SQLExecDirect()
- SQLExtendedFetch()
- SQLFetch()
- SQLGetData()
- SQLPutData()

22007 (日時形式が無効です。)

- SQLBulkOperations()
- SQLExecDirect()
- SQLExtendedFetch()
- SQLFetch()
- SQLFetchScroll()
- SQLGetData()
- SQLPutData()
- SQLSetPos()

22008 (日時フィールドがオーバーフローしました。)

- SQLBulkOperations()
- SQLExecDirect()
- SQLExtendedFetch()
- SQLFetch()
- SQLGetData()
- SQLSetPos()

22011 (サブストリング・エラーが起きました。)

- SQLGetSubString()

22012 (ゼロによる割算は無効です。)

- SQLExecDirect()
- SQLExtendedFetch()
- SQLFetch()
- SQLFetchScroll()

22015 (インターバル・フィールドの桁あふれ)

- SQLBulkOperations()

22018 (キャスト仕様の文字値が無効です。)

- SQLFetchScroll()
- SQLPrepare()
- SQLBulkOperations()
- SQLExtendedPrepare()

22019 (無効なエスケープ文字)

- SQLPrepare()
- SQLExtendedPrepare()

22025 (無効なエスケープ・シーケンス)

- SQLPrepare()
- SQLExtendedPrepare()

22026 (ストリング・データ、長さの不一致)

- SQLParamData()

23000 (保全性制約違反です。)

- SQLExecDirect()
- SQLBulkOperations()

24000 (カーソル状態が無効です。)

- SQLBulkOperations()
- SQLCloseCursor()
- SQLColumnPrivileges()
- SQLColumns()
- SQLExecDirect()
- SQLExtendedFetch()
- SQLExtendedPrepare()
- SQLFetch()
- SQLFetchScroll()
- SQLForeignKeys()
- SQLGetData()
- SQLGetStmtAttr()
- SQLGetTypeInfo()
- SQLPrepare()
- SQLPrimaryKeys()
- SQLProcedureColumns()
- SQLProcedures()
- SQLSetConnectAttr()
- SQLSetStmtAttr()
- SQLSpecialColumns()
- SQLStatistics()
- SQLTablePrivileges()
- SQLTables()

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。)

- SQLExecDirect()

2500025501 (トランザクション状態が無効です。)

- SQLDisconnect()

25501 (トランザクション状態が無効です。)

- SQLDisconnect()

28000 (許可指定が無効です。)

- SQLBrowseConnect()
- SQLConnect()

34000 (カーソル名が無効です。)

- SQLExecDirect()
- SQLPrepare()
- SQLSetCursorName()
- SQLExtendedPrepare()

37000 (SQL 構文が無効です。)

- SQLNativeSql()

37xxx (SQL 構文が無効です。)

- SQLExecDirect()
- SQLPrepare()
- SQLExtendedPrepare()

40001 (逐次化障害)

- SQLBulkOperations()
- SQLColumnPrivileges()
- SQLEndTran()
- SQLExecDirect()
- SQLExtendedPrepare()
- SQLFetchScroll()
- SQLParamData()
- SQLPrepare()

40003 (ステートメント完了が不明)

- SQLBulkOperations()

4000308S01 (通信リンクに障害が起きました。)

- SQLBindCol()
- SQLBindFileToCol()
- SQLBindFileToParam()
- SQLBindParameter()
- SQLCancel()
- SQLColumnPrivileges()
- SQLColumns()
- SQLDescribeCol()
- SQLExecDirect()
- SQLExtendedBind()
- SQLExtendedFetch()
- SQLExtendedPrepare()
- SQLFetch()
- SQLForeignKeys()
- SQLFreeStmt()
- SQLGetCursorName()
- SQLGetData()
- SQLGetFunctions()
- SQLGetInfo()
- SQLGetLength()
- SQLGetPosition()
- SQLGetSubString()
- SQLGetTypeInfo()
- SQLMoreResults()
- SQLNumParams()
- SQLNumResultCols()
- SQLParamData()
- SQLParamOptions()
- SQLPrepare()
- SQLPrimaryKeys()
- SQLProcedureColumns()
- SQLProcedures()
- SQLPutData()
- SQLRowCount()
- SQLSetCursorName()
- SQLSpecialColumns()
- SQLStatistics()
- SQLTablePrivileges()
- SQLTables()

42000 (構文エラーまたはアクセス違反)

- SQLBulkOperations()

42601 (PARMLIST 構文エラーです。)

- SQLProcedureColumns()

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。)

- SQLGetPosition()

42895 (EXECUTE または OPEN ステートメント内のホスト変数値は、そのデータ・タイプのゆえに使用できません。)

- SQLExecDirect()

428A1 (ホスト・ファイル変数で参照されるファイルにアクセスできません。)

- SQLExecDirect()
- SQLExtendedFetch()
- SQLFetch()

42xxx (構文エラーまたはアクセス規則違反。)

- SQLExecDirect()
- SQLPrepare()
- SQLExtendedPrepare()

44000 (保全性制約違反。)

- SQLExecDirect()
- SQLBulkOperations()

54028 (並行 LOB ハンドルが最大数に達しました。)

- SQLExtendedFetch()
- SQLFetch()

56084 (DRDA では、LOB データはサポートされていません。)

- SQLExecDirect()
- SQLExtendedFetch()
- SQLFetch()

58004 (予期しないシステム障害です。)

- SQLBindCol()
- SQLBindFileToCol()
- SQLBindFileToParam()
- SQLBindParameter()
- SQLConnect()
- SQLDataSources()
- SQLDescribeCol()
- SQLDisconnect()
- SQLExecDirect()
- SQLExtendedBind()
- SQLExtendedFetch()
- SQLExtendedPrepare()
- SQLFetch()
- SQLFreeConnect()
- SQLFreeEnv()
- SQLFreeStmt()
- SQLGetCursorName()
- SQLGetData()
- SQLGetFunctions()
- SQLGetInfo()
- SQLGetLength()
- SQLGetPosition()
- SQLGetSubString()
- SQLMoreResults()
- SQLNumResultCols()
- SQLPrepare()
- SQLRowCount()
- SQLSetCursorName()
- SQLTransact()

HY000 (一般的なエラーです。)

- SQLAllocHandle()
- SQLBrowseConnect()
- SQLBuildDataLink()
- SQLBulkOperations()
- SQLCloseCursor()
- SQLColAttribute()
- SQLCopyDesc()
- SQLDataSources()
- SQLDescribeParam()
- SQLDriverConnect()
- SQLEndTran()
- SQLExtendedPrepare()
- SQLFetchScroll()
- SQLFreeHandle()
- SQLGetConnectAttr()
- SQLGetDataLinkAttr()
- SQLGetDescField()
- SQLGetDescRec()
- SQLGetStmtAttr()
- SQLParamData()
- SQLSetConnectAttr()
- SQLSetConnection()
- SQLSetDescField()
- SQLSetDescRec()
- SQLSetPos()
- SQLSetStmtAttr()

HY001 (メモリーの割り振り失敗です。)

- SQLAllocHandle()
- SQLBindCol()
- SQLBindFileToCol()
- SQLBindFileToParam()
- SQLBindParameter()
- SQLBrowseConnect()
- SQLBuildDataLink()
- SQLBulkOperations()
- SQLCancel()
- SQLCloseCursor()
- SQLColAttribute()
- SQLColumnPrivileges()
- SQLColumns()
- SQLConnect()
- SQLCopyDesc()
- SQLDataSources()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLDisconnect()
- SQLEndTran()
- SQLExecDirect()
- SQLExtendedBind()
- SQLExtendedFetch()
- SQLExtendedPrepare()
- SQLFetch()
- SQLFetchScroll()
- SQLForeignKeys()
- SQLFreeConnect()
- SQLFreeEnv()
- SQLFreeHandle()
- SQLFreeStmt()
- SQLGetConnectAttr()
- SQLGetCursorName()
- SQLGetData()
- SQLGetDataLinkAttr()
- SQLGetDescField()
- SQLGetDescRec()
- SQLGetEnvAttr()
- SQLGetFunctions()
- SQLGetInfo()
- SQLGetLength()
- SQLGetPosition()
- SQLGetStmtAttr()
- SQLGetSubString()
- SQLGetTypeInfo()
- SQLMoreResults()
- SQLNativeSql()
- SQLNumParams()
- SQLNumResultCols()
- SQLParamData()
- SQLParamOptions()
- SQLPrepare()
- SQLPrimaryKeys()
- SQLProcedureColumns()
- SQLProcedures()
- SQLPutData()
- SQLRowCount()
- SQLSetConnectAttr()
- SQLSetCursorName()
- SQLSetDescField()
- SQLSetDescRec()
- SQLSetPos()
- SQLSetStmtAttr()
- SQLSpecialColumns()
- SQLStatistics()
- SQLTablePrivileges()
- SQLTables()
- SQLTransact()

HY002 (列の番号が無効です。)

- SQLBindCol()
- SQLBindFileToCol()
- SQLDescribeCol()
- SQLGetData()

HY003 (プログラム・タイプが範囲外です。)

- SQLBindCol()
- SQLBindParameter()
- SQLExtendedBind()
- SQLGetData()
- SQLGetLength()
- SQLGetSubString()

HY004 (SQL のデータ・タイプが範囲外です。)

- SQLBindFileToParam()
- SQLGetTypeInfo()
- SQLBindParameter()
- SQLExtendedBind()

HY007 (関連したステートメントが準備されていません。)

- SQLCopyDesc()
- SQLGetDescField()
- SQLGetDescRec()

HY008 (操作が取り消されました。)

- SQLBulkOperations()
- SQLColAttribute()
- SQLColumnPrivileges()
- SQLColumns()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLExtendedPrepare()
- SQLFetch()
- SQLFetchScroll()
- SQLNumParams()
- SQLNumResultCols()
- SQLParamData()
- SQLPrepare()
- SQLPrimaryKeys()
- SQLProcedureColumns()
- SQLProcedures()
- SQLPutData()
- SQLSetPos()
- SQLSpecialColumns()
- SQLStatistics()
- SQLTablePrivileges()
- SQLTables()

HY009 (引き数値が無効です。)

- SQLBindFileToCol()
- SQLBindFileToParam()
- SQLBindParameter()
- SQLColumnPrivileges()
- SQLExecDirect()
- SQLExtendedBind()
- SQLExtendedPrepare()
- SQLForeignKeys()
- SQLGetDataLinkAttr()
- SQLGetLength()
- SQLGetPosition()
- SQLGetSubString()
- SQLNativeSql()
- SQLPrepare()
- SQLPutData()
- SQLSetConnectAttr()
- SQLSetCursorName()
- SQLSetEnvAttr()
- SQLSetStmtAttr()
- SQLSpecialColumns()
- SQLStatistics()
- SQLTablePrivileges()
- SQLTables()

HY010 (関数の順序エラーです。)

- SQLAllocHandle()
- SQLBindCol()
- SQLBindFileToCol()
- SQLBindFileToParam()
- SQLBindParameter()
- SQLBulkOperations()
- SQLCloseCursor()
- SQLColAttribute()
- SQLColumnPrivileges()
- SQLColumns()
- SQLCopyDesc()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLDisconnect()
- SQLEndTran()
- SQLExecute()
- SQLExtendedBind()
- SQLExtendedFetch()
- SQLExtendedPrepare()
- SQLFetch()
- SQLFetchScroll()
- SQLForeignKeys()
- SQLFreeConnect()
- SQLFreeEnv()
- SQLFreeHandle()
- SQLFreeStmt()
- SQLGetConnectAttr()
- SQLGetCursorName()
- SQLGetData()
- SQLGetDescField()
- SQLGetDescRec()
- SQLGetFunctions()
- SQLGetLength()
- SQLGetPosition()
- SQLGetStmtAttr()
- SQLGetSubString()
- SQLGetTypeInfo()
- SQLMoreResults()
- SQLNumParams()
- SQLNumResultCols()
- SQLParamData()
- SQLParamOptions()
- SQLPrepare()
- SQLPrimaryKeys()
- SQLProcedureColumns()
- SQLProcedures()
- SQLPutData()
- SQLRowCount()
- SQLSetConnectAttr()
- SQLSetCursorName()
- SQLSetDescField()
- SQLSetDescRec()
- SQLSetPos()
- SQLSetStmtAttr()
- SQLSpecialColumns()
- SQLStatistics()
- SQLTablePrivileges()
- SQLTables()

HY011 (この段階で操作は無効です。)

- SQLBulkOperations()
- SQLExtendedPrepare()
- SQLSetConnectAttr()
- SQLSetEnvAttr()
- SQLSetPos()
- SQLSetStmtAttr()

HY012 (トランザクション・コードが無効です。)

- SQLEndTran()
- SQLTransact()

HY013 (予期しないメモリーのハンドル・エラーが起きました。)

- SQLAllocHandle()
- SQLBindCol()
- SQLBindFileToCol()
- SQLBindFileToParam()
- SQLBindParameter()
- SQLBrowseConnect()
- SQLBulkOperations()
- SQLCancel()
- SQLCloseCursor()
- SQLConnect()
- SQLDataSources()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLDisconnect()
- SQLExecDirect()
- SQLExtendedBind()
- SQLExtendedFetch()
- SQLExtendedPrepare()
- SQLFetch()
- SQLFreeConnect()
- SQLFreeEnv()
- SQLFreeHandle()
- SQLGetCursorName()
- SQLGetData()
- SQLGetDescField()
- SQLGetDescField()
- SQLGetDescRec()
- SQLGetFunctions()
- SQLGetLength()
- SQLGetPosition()
- SQLGetStmtAttr()
- SQLGetSubString()
- SQLMoreResults()
- SQLNumParams()
- SQLNumResultCols()
- SQLParamData()
- SQLPrepare()
- SQLRowCount()
- SQLSetCursorName()
- SQLSetDescRec()
- SQLTransact()

HY014 (もはやハンドルはありません。)

- SQLAllocHandle()
- SQLColumnPrivileges()
- SQLColumns()
- SQLExecDirect()
- SQLExtendedPrepare()
- SQLForeignKeys()
- SQLPrepare()
- SQLPrimaryKeys()
- SQLProcedureColumns()
- SQLProcedures()
- SQLSpecialColumns()
- SQLStatistics()
- SQLTablePrivileges()
- SQLTables()

HY016 (インプリメンテーションの行記述子を変更することはできません。)

- SQLCopyDesc()
- SQLSetDescField()
- SQLSetDescRec()

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。)

- SQLFreeHandle()
- SQLSetStmtAttr()
- SQLExtendedPrepare()

HY018 (サーバーが取消し要求を拒否しました。)

- SQLCancel()

HY021 (記述子情報が不整合です。)

- SQLBindParameter()
- SQLCopyDesc()
- SQLExtendedBind()
- SQLGetDescField()
- SQLSetDescField()
- SQLSetDescRec()

HY024 (属性値が無効です。)

- SQLSetConnectAttr()
- SQLSetEnvAttr()
- SQLSetStmtAttr()
- SQLExtendedPrepare()

HY090 (ストリングまたはバッファー長が無効です。)

- SQLBindCol()
- SQLBindFileToCol()
- SQLBindFileToParam()
- SQLBindParameter()
- SQLBrowseConnect()
- SQLBuildDataLink()
- SQLBulkOperations()
- SQLColAttribute()
- SQLColumnPrivileges()
- SQLColumns()
- SQLConnect()
- SQLDataSources()
- SQLDescribeCol()
- SQLDriverConnect()
- SQLExecDirect()
- SQLExtendedBind()
- SQLExtendedPrepare()
- SQLForeignKeys()
- SQLGetConnectAttr()
- SQLGetCursorName()
- SQLGetData()
- SQLGetDataLinkAttr()
- SQLGetDescField()
- SQLGetInfo()
- SQLGetPosition()
- SQLGetStmtAttr()
- SQLGetSubString()
- SQLNativeSql()
- SQLPrepare()
- SQLPrimaryKeys()
- SQLProcedureColumns()
- SQLProcedures()
- SQLPutData()
- SQLSetConnectAttr()
- SQLSetCursorName()
- SQLSetEnvAttr()
- SQLSetPos()
- SQLSetStmtAttr()
- SQLSpecialColumns()
- SQLStatistics()
- SQLTablePrivileges()
- SQLTables()

HY091 (記述子タイプが範囲外です。)

- SQLColAttribute()
- SQLGetDescField()
- SQLSetDescField()

HY092 (オプション・タイプが範囲外です。)

- SQLAllocHandle()
- SQLBulkOperations()
- SQLCopyDesc()
- SQLEndTran()
- SQLExecDirect()
- SQLExtendedFetch()
- SQLExtendedPrepare()
- SQLFetch()
- SQLFreeStmt()
- SQLGetConnectAttr()
- SQLGetDataLinkAttr()
- SQLGetEnvAttr()
- SQLGetStmtAttr()
- SQLParamData()
- SQLSetConnectAttr()
- SQLSetDescField()
- SQLSetEnvAttr()
- SQLSetPos()
- SQLSetStmtAttr()

HY093 (パラメーターの数値が無効です。)

- SQLBindFileToParam()
- SQLBindParameter()
- SQLExtendedBind()

HY094 (位取り値が無効です。)

- SQLBindParameter()
- SQLExtendedBind()

HY096 (情報タイプが範囲外です。)

- SQLGetInfo()

HY097 (列タイプが範囲外です。)

- SQLSpecialColumns()

HY098 (スコープ・タイプが範囲外です。)

- SQLSpecialColumns()

HY099 (Nullable タイプが範囲外です。)

- SQLSpecialColumns()

HY100 (固有オプション・タイプが範囲外です。)

- SQLStatistics()

HY101 (正確度オプション・タイプが範囲外です。)

- SQLStatistics()

HY103 (方向オプションが範囲外です。)

- SQLDataSources()

HY104 (精度値が無効です。)

- SQLBindParameter()
- SQLExtendedBind()

HY105 (パラメーター・タイプが無効です。)

- SQLSetDescField()
- SQLBindParameter()
- SQLExtendedBind()

HY106 (フェッチ・タイプが範囲外です。)

- SQLExtendedFetch()
- SQLFetchScroll()

HY107 (行の値が範囲外です。)

- SQLFetchScroll()
- SQLParamOptions()
- SQLSetPos()

HY109 (カーソル位置が無効です。)

- SQLGetStmtAttr()
- SQLSetPos()

HY110 (ドライバーの完了が無効です。)

- SQLDriverConnect()

HY111 (ブックマーク値が無効です。)

- SQLFetchScroll()

HY501 (データ・ソース名が無効です。)

- SQLConnect()

HY503 (ファイル名の長さが無効です。)

- SQLExecDirect()

HY506 (ファイルのクローズ・エラーが起きました。)

- SQLCancel()
- SQLFreeStmt()
- SQLParamData()

HY509 (ファイルの削除エラーが起きました。)

- SQLParamData()

HYC00 (ドライバーが機能していません。)

- SQLAllocHandle()
- SQLBindCol()
- SQLBindFileToCol()
- SQLBindFileToParam()
- SQLBindParameter()
- SQLBulkOperations()
- SQLColAttribute()
- SQLColumnPrivileges()
- SQLColumns()
- SQLDescribeCol()
- SQLExtendedBind()
- SQLExtendedFetch()
- SQLExtendedPrepare()
- SQLFetch()
- SQLFetchScroll()
- SQLForeignKeys()
- SQLGetConnectAttr()
- SQLGetData()
- SQLGetInfo()
- SQLGetLength()
- SQLGetPosition()
- SQLGetStmtAttr()
- SQLGetSubString()
- SQLPrimaryKeys()
- SQLProcedureColumns()
- SQLProcedures()
- SQLSetConnectAttr()
- SQLSetEnvAttr()
- SQLSetPos()
- SQLSetStmtAttr()
- SQLSpecialColumns()
- SQLStatistics()
- SQLTablePrivileges()
- SQLTables()

HYT00 (タイムアウトになりました。)

- SQLBulkOperations()
- SQLColumnPrivileges()
- SQLColumns()
- SQLConnect()
- SQLDescribeCol()
- SQLExecDirect()
- SQLExtendedFetch()
- SQLExtendedPrepare()
- SQLFetch()
- SQLForeignKeys()
- SQLGetData()
- SQLGetTypeInfo()
- SQLMoreResults()
- SQLNumParams()
- SQLNumResultCols()
- SQLParamData()
- SQLPrepare()
- SQLPrimaryKeys()
- SQLProcedureColumns()
- SQLProcedures()
- SQLPutData()
- SQLSetPos()
- SQLSpecialColumns()
- SQLStatistics()
- SQLTablePrivileges()
- SQLTables()

HYT01 (接続タイムアウトの満了)

- SQLBulkOperations()

S0001 (データベース・オブジェクトはすでに存在しています。)

- SQLExecDirect()
- SQLPrepare()
- SQLExtendedPrepare()

S0002 (データベース・オブジェクトは存在していません。)

- SQLExecDirect()
- SQLPrepare()
- SQLExtendedPrepare()

S0011 (索引はすでに存在しています。)

- SQLExecDirect()
- SQLPrepare()
- SQLExtendedPrepare()

S0012 (索引がありません。)

- SQLExecDirect()
- SQLPrepare()
- SQLExtendedPrepare()

S0021 (列はすでに存在しています。)

- SQLExecDirect()
- SQLPrepare()
- SQLExtendedPrepare()

S0022 (列がありません。)

- SQLExecDirect()
- SQLPrepare()
- SQLExtendedPrepare()

付録F. データ変換

この節では、C と SQL データ・タイプとの間でのデータ変換に使用する表を掲載します。これには、次のものがあります。
- 精度、位取り、長さ、および各データ・タイプの表示サイズ
- SQL から C データ・タイプへの変換
- C から SQL データ・タイプへの変換

SQL データ・タイプと C データ・タイプ、その記号タイプ、および省略時変換のリストは、 33ページの表2 および 35ページの表3 を参照してください。 サポートされる変換については、39ページの表6 を参照してください。

データ・タイプ属性

以下のデータ・タイプ属性に関する情報を示します。
- 『精度』
- 876ページの『位取り』
- 877ページの『長さ』
- 878ページの『表示サイズ』

精度

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、一般には、その列またはパラメーターの最大長または定義済みの長さを指します。次の表は、各 SQL データ・タイプの精度を定義しています。

表 194. 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

表 194. 精度 (続き)

fSqlType	精度
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 形式の文字数)
SQL_TIME [b]	8 (hh:mm:ss 形式の文字数)
SQL_TIMESTAMP	TIMESTAMP データ・タイプで使用する、 "yyy-mm-dd hh:mm:ss[.fff[fff]]" 形式の文字数。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」形式の文字数)。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」形式の文字数)。
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。

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。

位取り

数値の列またはパラメーターの位取りは、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、位取り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・タイプの位取りを定義しています。

表 195. 位取り

fSqlType	位取り
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) と定義された列の位取りは 3 です。
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]]" 形式の、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" 形式を使用する場合、位取りは 3 です。
SQL_GRAPHIC SQL_VARGRAPHIC SQL_LONGVARGRAPHIC SQL_DBCLOB	該当しません。

長さ

列の長さとは、データが省略時 C データ・タイプに転送されるときにアプリケーションに戻される最大バイト 数のことです。文字データの場合、長さにはヌル終了バイトは含まれません。列の長さは、データ・ソースにデータを保管するのに必要なバイト数とは違う場合があることに注意してください。省略時 C データ・タイプのリストは、『省略時 C データ・タイプ』の節を参照してください。

次の表は、各 SQL データ・タイプの長さを定義しています。

表 196. 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_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。

表示サイズ

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 197. 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。

表 197. 表示サイズ (続き)

fSqlType	表示サイズ
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 2 進バイトは 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 形式の日付)。
SQL_TIME	8 (hh:mm:ss 形式の時刻)。
SQL_TIMESTAMP	19 (タイム・スタンプの位取りが 0 の場合)、または 20 にタイム・スタンプの位取りを加えたもの (位取りが 0 より大きい場合)。これは、"yyyy-mm-dd hh:mm:ss[fff[fff]]" 形式の文字数です。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。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。

SQL から C データ・タイプへのデータ変換

指定の SQL データ・タイプについて、次の内容がリストされています。

- 表の最初の列には、 `SQLBindCol()` および `SQLGetData()` の *fCType* 引き数の有効な入力値をリストします。
- 2 番目の列では、テストの出力をリストします。このテストでは `SQLBindCol()` または `SQLGetData()` に指定されている *cbValueMax* 引き数が頻繁に使用されます。ドライバーはこのテストを実行して、データを変換できるかどうかを判別します。
- 3 番目と 4 番目の列では、ドライバーがデータ変換を試行した後の、`SQLBindCol()` または `SQLGetData()` に指定されている *rgbValue* 引き数と *pcbValue* 引き数の値を (出力ごとに) リストします。

- 最後の列では、SQLFetch()、SQLExtendedFetch()、SQLGetData()、または SQLGetSubString() によって各出力用に戻される SQLSTATE をリストします。

表では、指定の SQL データ・タイプにとって有効になるよう ODBC で定義された変換をリストします。

SQLBindCol() または SQLGetData() の *fCType* 引き数に、所定の SQL データ・タイプについて表にない値が含まれていると、 SQLFetch() または SQLGetData() は、 SQLSTATE 07006 (データ・タイプ属性制約違反) を戻します。

fCType 引き数に、表にはあっても、ドライバーでサポートされていない変換を指定する値が含まれていると、 SQLFetch() または SQLGetData() は、SQLSTATE HYC00 (ドライバー不可) を戻します。

表には示されていませんが、*pcbValue* 引き数には、 SQL データ値が NULL のときの SQL_NULL_DATA が含まれます。データを取り出す場合に複数呼び出しが行われる際の *pcbValue* の使用法に関する説明は、 SQLGetData() を参照してください。

SQL データが文字 C データに変換される際に、 *pcbValue* に戻される文字カウントにはヌル終了バイトは含まれません。 *rgbValue* が NULL ポインターの場合、 SQLBindCol() または SQLGetData() は、 SQLSTATE HY009 (無効な引き数値) を戻します。

表では、次の用語が使われています。

データの長さ
: 指定の C データ・タイプに変換された後のデータの全長 (データがストリングに変換された場合のヌル終了バイトを除く) 。これは、アプリケーションに戻される前にデータが切り捨てられる場合にも当てはまります。

有効桁 負符号 (必要な場合) および小数点の左の桁。

表示サイズ
: 文字形式でデータを表示するのに必要なバイトの合計数。

文字 SQL データから C データへの変換

文字 SQL データ・タイプは次のとおりです。

SQL_CHAR
SQL_VARCHAR

SQL_LONGVARCHAR
SQL_CLOB

表 198. 文字 SQL データから C データへの変換

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	データ長 < cbValueMax	データ	データの長さ	00000
	データ長 >= cbValueMax	切り捨てデータ	データの長さ	01004
SQL_C_BINARY	データ長 <= cbValueMax	データ	データの長さ	00000
	データ長 > cbValueMax	切り捨てデータ	データの長さ	01004
SQL_C_SHORT SQL_C_LONG SQL_C_FLOAT SQL_C_FLOAT SQL_C_TINYINT SQL_C_BIT SQL_C_UBIGINT SQL_C_SBIGINT SQL_C_NUMERIC [c]	切り捨てられずに変換されたデータ [a]	データ	C データ・タイプのサイズ	00000
	変換され切り捨てられたデータ、ただし有効桁は失われていない [a]	データ	C データ・タイプのサイズ	01004
	データの変換で、有効桁が失われる [a]	影響なし	C データ・タイプのサイズ	22003
	データは数値でない [a]	影響なし	C データ・タイプのサイズ	22005
SQL_C_DATE	データ値は有効な時刻 [a]	データ	6 [b]	00000
	データ値は有効な時刻ではない [a]	影響なし	6 [b]	22007
SQL_C_TIME	データ値は有効な時刻 [a]	データ	6 [b]	00000
	データ値は有効な時刻ではない [a]	影響なし	6 [b]	22007
SQL_C_TIMESTAMP	データ値は有効なタイム・スタンプ [a]	データ	16 [b]	00000
	データ値は有効なタイム・スタンプではない [a]	影響なし	16 [b]	22007

表 198. 文字 SQL データから C データへの変換 (続き)

fCType	Test	rgbValue	pcbValue	SQLSTATE

注:

[a] *cbValueMax* の値はこの変換では無視されます。ドライバーは、 *rgbValue* のサイズは C データ・タイプのサイズであると想定します。

[b] これは、対応する C データ・タイプのサイズです。

[c] SQL_C_NUMERIC は 32 ビット Windows プラットフォーム上でのみサポートされます。

SQLSTATE **00**000 は `SQLError()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

グラフィック SQL データから C データへの変換

グラフィック SQL データ・タイプは、以下のとおりです。

SQL_GRAPHIC
SQL_VARGRAPHIC
SQL_LONGVARGRAPHIC
SQL_DBCLOB

表 199. SQL グラフィック・データを C データに変換

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	2 バイト文字数 * 2 <= cbValueMax	データ	データの長さ (オクテット)	00000
	2 バイト文字数 * 2 > cbValueMax	*cbValueMax* 未満の最大偶数バイトに切り捨てられたデータ	データの長さ (オクテット)	01004
SQL_C_DBCHAR	2 バイト文字数 * 2 < cbValueMax	データ	データの長さ (オクテット)	00000
	2 バイト文字数 * 2 >= cbValueMax	*cbValueMax* 未満の最大偶数バイトに切り捨てられたデータ	データの長さ (オクテット)	01004

注: SQLSTATE **00**000 は SQLError() では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

浮動小数点値への変換時に、結果値の非有効桁が失われていると、 SQLSTATE 22003 は戻されません。

SQL 数値データを C データに変換

SQL 数値データ・タイプは、以下のとおりです。

SQL_DECIMAL
SQL_NUMERIC
SQL_SMALLINT
SQL_INTEGER
SQL_BIGINT
SQL_REAL
SQL_FLOAT
SQL_DOUBLE

表 200. SQL 数値データを C データに変換

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	表示サイズ < cbValueMax	データ	データの長さ	00000
	有効桁数 < cbValueMax	切り捨てデータ	データの長さ	01004
	有効桁数 >= cbValueMax	影響なし	データの長さ	22003
SQL_C_SHORT SQL_C_LONG SQL_C_FLOAT SQL_C_DOUBLE SQL_C_TINYINT SQL_C_BIT SQL_C_UBIGINT SQL_C_SBIGINT SQL_C_NUMERIC [b]	切り捨てられずに変換されたデータ [a]	データ	C データ・タイプのサイズ	00000
	変換され切り捨てられたデータ、ただし有効桁は失われていない [a]	切り捨てデータ	C データ・タイプのサイズ	01004
	データの変換で、有効桁が失われる [a]	影響なし	C データ・タイプのサイズ	22003

注:

a *cbValueMax* の値はこの変換では無視されます。ドライバーは、 *rgbValue* のサイズは C データ・タイプのサイズであると想定します。

b SQL_C_NUMERIC は 32 ビット Windows プラットフォーム上でのみサポートされます。

SQLSTATE **00**000 は `SQLError()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

2 進 SQL データから C データへの変換

2 進 SQL データ・タイプは次のとおりです。

SQL_BINARY
SQL_VARBINARY
SQL_LONGVARBINARY
SQL_BLOB

表 201. 2 進 SQL データから C データへの変換

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	(データ長) < cbValueMax	データ	データの長さ	なし
	(データ長) >= cbValueMax	切り捨てデータ	データの長さ	01004
SQL_C_BINARY	データ長 <= cbValueMax	データ	データの長さ	なし
	データ長 > cbValueMax	切り捨てデータ	データの長さ	01004

日付 SQL データから C データへの変換

日付 SQL データ・タイプは次のとおりです。

SQL_DATE

表 202. 日付 SQL データから C データへの変換

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	cbValueMax >= 11	データ	10	00000
	cbValueMax < 11	影響なし	10	22003
SQL_C_DATE	なし[a]	データ	6 [b]	00000
SQL_C_TIMESTAMP	なし[a]	データ[c]	16 [b]	00000

注:

a *cbValueMax* の値はこの変換では無視されます。ドライバーは、 *rgbValue* のサイズは C データ・タイプのサイズであると想定します。

b これは、対応する C データ・タイプのサイズです。

c TIMESTAMP_STRUCT 構造の時刻フィールドはゼロに設定されます。

SQLSTATE **00**000 は `SQLError()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

日付 SQL データ・タイプが文字の C データ・タイプに変換される場合、ストリングは「yyyy-mm-dd」形式になります。

時刻 SQL データから C データへの変換

時刻 SQL データ・タイプは次のとおりです。

SQL_TIME

表 203. 時刻 SQL データから C データへの変換

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	cbValueMax >= 9	データ	8	00000
	cbValueMax < 9	影響なし	8	22003
SQL_C_TIME	なし[a]	データ	6 [b]	00000
SQL_C_TIMESTAMP	なし[a]	データ[c]	16 [b]	00000

注:

a *cbValueMax* の値はこの変換では無視されます。ドライバーは、 *rgbValue* のサイズは C データ・タイプのサイズであると想定します。

b これは、対応する C データ・タイプのサイズです。

c TIMESTAMP_STRUCT 構造の日付フィールドは、アプリケーションが実行しているマシンの現行システム日付に設定され、時刻小数部はゼロに設定されます。

SQLSTATE **00**000 は `SQLError()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

時刻 SQL データ・タイプが文字の C データ・タイプに変換される場合、ストリングは「hh:mm:ss」形式になります。

タイム・スタンプ SQL データから C データへの変換

タイム・スタンプ SQL データ・タイプは次のとおりです。

SQL_TIMESTAMP

表 204. タイム・スタンプ SQL データから C データへの変換

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	表示サイズ < cbValueMax	データ	データの長さ	00000
	19 <= cbValueMax <= 表示サイズ	切り捨てデータ [b]	データの長さ	01004
	cbValueMax < 19	影響なし	データの長さ	22003
SQL_C_DATE	なし[a]	切り捨てデータ [c]	6 [e]	01004
SQL_C_TIME	なし[a]	切り捨てデータ [d]	6 [e]	01004
SQL_C_TIMESTAMP	なし[a]	データ	16 [e]	00000

注:

a *cbValueMax* の値はこの変換では無視されます。ドライバーは、 *rgbValue* のサイズは C データ・タイプのサイズであると想定します。

b タイム・スタンプの小数秒は切り捨てられます。

c タイム・スタンプの時刻部分は削除されます。

d タイム・スタンプの日付部分は削除されます。

e これは、対応する C データ・タイプのサイズです。

SQLSTATE **00**000 は `SQLError()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

タイム・スタンプ SQL データ・タイプが文字の C データ・タイプに変換される場合、ストリングは「yyyy-mm-dd hh:mm:ss[.fff[fff]]」形式になります。

SQL から C へのデータ変換例

表 205. SQL から C へのデータ変換例

SQL データ・タイプ	SQL データ値	C データ・タイプ	cbValue Max	rgbValue	SQL STATE
SQL_CHAR	abcdef	SQL_C_CHAR	7	abcdef¥0 [a]	00000
SQL_CHAR	abcdef	SQL_C_CHAR	6	abcde¥0 [a]	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	8	1234.56¥0 [a]	00000
SQL_DECIMAL	1234.56	SQL_C_CHAR	5	1234¥0 [a]	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	4	---	22003

表 205. SQL から C へのデータ変換例 (続き)

SQL データ・タイプ	SQL データ値	C データ・タイプ	cbValue Max	rgbValue	SQL STATE
SQL_DECIMAL	1234.56	SQL_C_FLOAT	無視されます	1234.56	00000
SQL_DECIMAL	1234.56	SQL_C_SHORT	無視されます	1234	01004
SQL_DATE	1992-12-31	SQL_C_CHAR	11	1992-12-31¥0 [a]	00000
SQL_DATE	1992-12-31	SQL_C_CHAR	10	---	22003
SQL_DATE	1992-12-31	SQL_C_ TIMESTAMP	無視されます	1992,12,31, 0,0,0,0 [b]	00000
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	23	1992-12-31 23:45:55.12¥0 [a]	00000
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	22	1992-12-31 23:45:55.1¥0 [a]	01004
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	18	---	22003

注:

[a] 「¥0」はヌル終了文字を表します。

[b] このリストの数値は、TIMESTAMP_STRUCT 構造体のフィールドに保管される数値です。

SQLSTATE **00**000 は SQLError() では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

C から SQL データ・タイプへのデータ変換

指定の C データ・タイプについて、次の内容がリストされています。

- 表の最初の列には、SQLBindParameter() または SQLSetParam() の *fSqlType* 引き数の有効な入力値をリストします。
- 2 番目の列では、テストの出力をリストします。このテストでは SQLBindParameter() または SQLSetParam の *pcbValue* 引き数で指定されたパラメーター・データの長さが頻繁に使用されます。ドライバーはこのテストを実行して、データを変換できるかどうかを判別します。
- 3 番目の列では、 SQLExecDirect() または SQLExecute() によって各出力に返される SQLSTATE をリストします。

注: SQLSTATE が 00000 (正常実行) の場合に限り、データがデータ・ソースに送られます。

表では、指定の SQL データ・タイプにとって有効になるよう ODBC で定義された変換をリストします。

SQLBindParameter() または SQLSetParam() の *fSqlType* 引き数に、指定の C データ・タイプについて表にない値が含まれている場合、 SQLSTATE 07006 (データ・タイプ属性制約違反) が戻されます。

fSqlType 引き数に、表にはあっても、ドライバーでサポートされていない変換を指定する値が含まれていると、 SQLBindParameter() または SQLSetParam() は、 SQLSTATE HYC00 (ドライバー不可) を戻します。

SQLBindParameter() または SQLSetParam() に指定された *rgbValue* および *pcbValue* 引き数が両方ともヌル・ポインターである場合、その関数は SQLSTATE HY009 (引き数値が無効です) を戻します。

データの長さ
指定された SQL データ・タイプに変換された後のデータの全長 (データがストリングに変換された場合のヌル終了バイトを除く) 。これは、データ・ソースに送られる前にデータが切り捨てられる場合にも当てはまります。

列の長さ
データが省略時 C データ・タイプへ転送されるときにアプリケーションに戻されるバイトの最大数。文字データの場合、長さにはヌル終了バイトは含まれません。

表示サイズ
データを文字書式で表示するために必要な最大バイト数。

有効桁 負符号 (必要な場合) および小数点の左の桁。

C 文字データを SQL データに変換

文字 C データ・タイプは次のとおりです。

SQL_C_CHAR

表 206. 文字 C データから SQL データへの変換

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR SQL_CLOB	データ長 <= 列長	00000
	データ長 > 列長	22001
SQL_DECIMAL SQL_NUMERIC SQL_SMALLINT SQL_INTEGER SQL_REAL SQL_FLOAT SQL_DOUBLE	切り捨てられずに変換されたデータ	00000
	変換され切り捨てられたデータ、ただし有効桁は失われていない	22001
	データ変換の結果、有効桁が消失する	22003
	データ値は数値ではない	22005
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_BLOB	(データ長) < 列長	なし
	(データ長) >= 列長	22001
	データ値は 16 進値ではない	22005
SQL_DATE	データ値は有効な日付	00000
	データ値は有効な日付ではない	22007
SQL_TIME	データ値は有効な時刻	00000
	データ値は有効な時刻ではない	22007
SQL_TIMESTAMP	データ値は有効なタイム・スタンプ	00000
	データ値は有効なタイム・スタンプではない	22007
SQL_GRAPHIC SQL_VARGRAPHIC SQL_LONGVARGRAPHIC SQL_DBCLOB	データ長 / 2 <= 列長	00000
	データ長 / 2 < 列長	22001

注: SQLSTATE **00**000 は `SQLError()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

C 数値データから SQL データへの変換

数値 C データ・タイプは次のとおりです。

SQL_C_SHORT
SQL_C_LONG
SQL_C_FLOAT
SQL_C_DOUBLE
SQL_C_TINYINT

SQL_C_BIT

表 207. 数値 C データから SQL データへの変換

fSQLType	Test	SQLSTATE
SQL_DECIMAL SQL_NUMERIC	切り捨てられずに変換されたデータ	00000
SQL_SMALLINT SQL_INTEGER SQL_REAL	変換され切り捨てられたデータ、ただし有効桁は失われていない	22001
SQL_FLOAT SQL_DOUBLE	データ変換の結果、有効桁が消失する	22003
SQL_CHAR SQL_VARCHAR	切り捨てられずに変換されたデータ	00000
	データの変換で有効桁が失われる	22003

注: SQLSTATE **00**000 は `SQLError()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

浮動小数点値への変換時に、結果値の非有効桁が失われていると、 SQLSTATE 22003 は戻されません。

2 進 C データから SQL データへの変換

2 進 C データ・タイプは次のとおりです。

SQL_C_BINARY

表 208. 2 進 C データから SQL データへの変換

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	データ長 <= 列長	なし
SQL_LONGVARCHAR SQL_CLOB	データ長 > 列長	22001
SQL_BINARY SQL_VARBINARY	データ長 <= 列長	なし
SQL_LONGVARBINARY SQL_BLOB	データ長 > 列長	22001

DBCHAR C データから SQL データへの変換

2 バイト C データ・タイプは次のとおりです。

SQL_C_DBCHAR

表 209. DBCHAR C データから SQL データへの変換

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	データ長 <= 列長 x 2	なし
SQL_LONGVARCHAR SQL_CLOB	データ長 > 列長 x 2	22001
SQL_BINARY SQL_VARBINARY	データ長 <= 列長 x 2	なし
SQL_LONGVARBINARY SQL_BLOB	データ長 > 列長 x 2	22001

日付 C データから SQL データへの変換

日付 C データ・タイプは次のとおりです。

SQL_C_DATE

表 210. 日付 C データから SQL データへの変換

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	列長 >= 10	00000
	列長 < 10	22003
SQL_DATE	データ値は有効な日付	00000
	データ値は有効な日付ではない	22007
SQL_TIMESTAMP [a]	データ値は有効な日付	00000
	データ値は有効な日付ではない	22007

注: SQLSTATE **00**000 は `SQLError()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

注: **a**、TIMESTAMP の時刻構成要素はゼロに設定されます。

時刻 C データから SQL データへの変換

時刻 C データ・タイプは次のとおりです。

SQL_C_TIME

表 211. 時刻 C データから SQL データへの変換

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	列長 >= 8	00000
	列長 < 8	22003
SQL_TIME	データ値は有効な時刻	00000
	データ値は有効な時刻ではない	22007
SQL_TIMESTAMP [a]	データ値は有効な時刻	00000
	データ値は有効な時刻ではない	22007

注: SQLSTATE **00**000 は SQLError() では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

注: a、TIMESTAMP の日付構成要素は、アプリケーションが実行しているマシンのシステム日付に設定されます。

タイム・スタンプ C データから SQL データへの変換

タイム・スタンプ C データ・タイプは次のとおりです。

SQL_C_TIMESTAMP

表 212. タイム・スタンプ C データから SQL データへの変換

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	列長 >= 表示サイズ	00000
	19 <= 列長 < 表示サイズ [a]	22001
	列長 < 19	22003
SQL_DATE	データ値は有効な日付 [b]	22001
	データ値は有効な日付ではない	22007
SQL_TIME	データ値は有効な時刻 [c]	22001
	データ値は有効な時刻ではない	22007
SQL_TIMESTAMP	データ値は有効なタイム・スタンプ	00000
	データ値は有効なタイム・スタンプではない	22007

注:

a タイム・スタンプの小数秒は切り捨てられます。

b タイム・スタンプの時刻部分は削除されます。

c タイム・スタンプの日付部分は削除されます。

SQLSTATE **00**000 は SQLError() では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

C から SQL へのデータ変換例

表 213. C から SQL へのデータ変換例

C データ・タイプ	C データ値	SQL データ・タイプ	列長	SQL データ値	SQL STATE
SQL_C_CHAR	abcdef¥0	SQL_CHAR	6	abcdef	00000
SQL_C_CHAR	abcdef¥0	SQL_CHAR	5	abcde	22001
SQL_C_CHAR	1234.56¥0	SQL_DECIMAL	6	1234.56	00000
SQL_C_CHAR	1234.56¥0	SQL_DECIMAL	5	1234.5	22001
SQL_C_CHAR	1234.56¥0	SQL_DECIMAL	3	---	22003
SQL_C_FLOAT	1234.56	SQL_FLOAT	該当せず	1234.56	00000
SQL_C_FLOAT	1234.56	SQL_INTEGER	該当せず	1234	22001

注: SQLSTATE **00**000 は `SQLError()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

付録G. ストアード・プロシージャーのカタログ表示

DB2 CLI 関数 SQLProcedures() および SQLProcedureColumns() は、次の 2 つのカタログ表を使用して、ストアード・プロシージャーとその属性に関する情報を検索します。

- SYSCAT.PROCEDURES
- SYSCAT.PROCPARMS

上記のカタログ表、および CREATE PROCEDURE コマンドを使用してストアード・プロシージャーをこの表に登録する方法の詳細については、 *SQL 解説書* を参照してください。

付録H. ストアード・プロシージャー登録の疑似カタログ表

DB2 ユニバーサル・データベース以前の DB2 CLI バージョンでは、SQLProcedures() と SQLProcedureColumns() が呼び出され、ストアード・プロシージャーとその属性に関する情報を検索する前に、サーバーで DB2CLI.PROCEDURES 表を作成し、データを入れておかなければなりません。

DB2 ユニバーサル・データベースはストアード・プロシージャーの情報が入った SYSCAT.PROCEDURES および SYSCAT.PROCPARAMS カタログを使用するようになったので、 DB2CLI.PROCEDURES 表は不要になりました。

バージョン 5 以前の DB2 をお使いの場合は、コマンド行プロセッサー入力ファイル STORPROC.DDL を使って DB2CLI.PROCEDURES 表を作成できます。次いで、サンプルの STORPROC.XMP ファイルを修正して、この PROCEDURE 表に行を挿入することができます。これら 2 つのファイルは両方とも、 sqllib ディレクトリーの misc サブディレクトリー内にあります。入力ファイルを使用して表を作成するには、コマンド行から次のコマンドを実行します。

```
db2 -f STORPROC.DDL -z STORPROC.LOG -t
```

情報が正しく表の中に入力されたことを確認し、表を最新のものに更新していくのは、データベース管理担当者の役割です。最初は、すべてのユーザーにこの表に対する SELECT (選択) 特権があり、 DBADM 権限を持つユーザーだけに、この表の中の行について INSERT (挿入)、 DELETE (削除) または UPDATE (更新) を行うことができます。それ以外の表については、DBADM 権限を持つユーザーが他のユーザーに各特権を付与することができます。

DB2CLI.PROCEDURES 表の凡例

列名　列の名前

データ・タイプ
　列のデータ・タイプ

ヌル可

　はい - ヌルを使用できる
　いいえ - ヌルは使用できない

キー

キー - 列は基本キーの一部である
いいえ - 列はキーの一部では**ない**

説明　列の説明

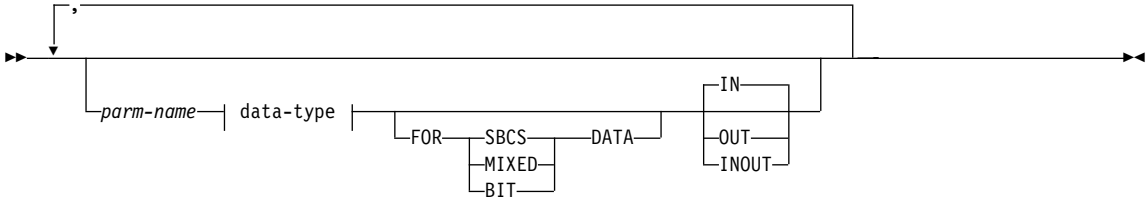
表 214. DB2CLI スキーマでの PROCEDURES 表の列

列名	データ・タイプ	ヌル可	キー	説明
PROCSCHEMA	VARCHAR(18)	いいえ	PK	プロシージャーのスキーマ名。
PROCNAME	VARCHAR(18)	いいえ	PK	SQL CALL ステートメントで指定されるストアード・プロシージャーの名前。
DEFINER	VARCHAR(8)	いいえ	いいえ	ストアード・プロシージャーの定義者。 (表にこの行を挿入したデータベース管理者。)
PKGSCHEMA	VARCHAR(18)	いいえ	いいえ	ストアード・プロシージャーが実行されるときに使用されるパッケージのスキーマ名。
PKGNAME	VARCHAR(18)	いいえ	いいえ	ストアード・プロシージャーが実行されるときにロードされるパッケージの名前。
PROC_LOCATION	VARCHAR(254)	いいえ	いいえ	プロシージャーの外部 (完全パス) 名。
PARM_STYLE	CHAR(1)	いいえ	いいえ	ストアード・プロシージャーにパラメーターを渡すのに使用する規則。 • *D* は、データベース・アプリケーション・リモート・インターフェース (DARI) 規則で、 DB2 共通サーバー用サーバーが使用するものです。
LANGUAGE	CHAR(8)	いいえ	いいえ	ストアード・プロシージャーを作成するのに使用するプログラミング言語。 DB2 共通サーバー用に可能な値としては、COBOL、 C、REXX、および FORTRAN があります。 (値 C は、 C プログラムと C++ プログラムの両方を指して用いられます。) 他の製品では PL/I と BASIC のような他の言語が使用できる場合があります。

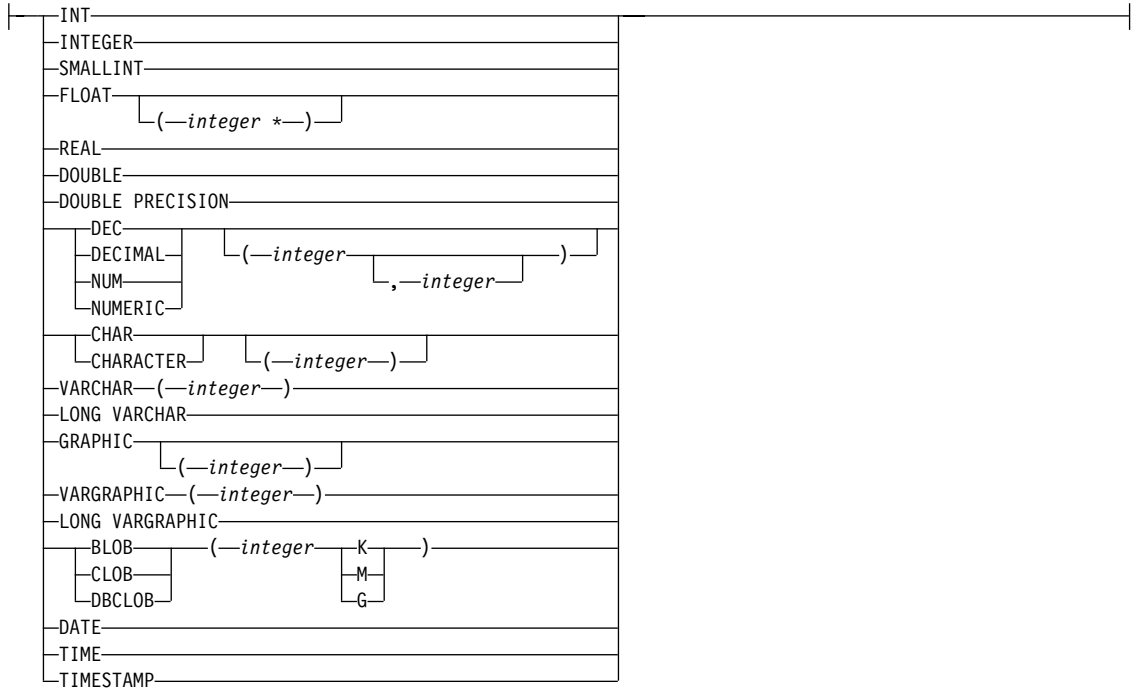
表 214. DB2CLI スキーマでの PROCEDURES 表の列 (続き)

列名	データ・タイプ	ヌル可	キー	説明
STAYRESIDENT	CHAR(1)	いいえ	いいえ	ストアード・プロシージャーが終了するときにプロシージャーのロード・モジュールがメモリーから削除されるかどうかを判別します。 • *Y* は、ストアード・プロシージャー終了後にもロード・モジュールがメモリーに残っていることを示します。 これには、いくつかの呼び出しを残しておくためにストアード・プロシージャーが SQLZ_HOLD_PROC を戻してから、ストアード・プロシージャーが SQLZ_DISCONNECT_PROC を戻して終了したという場合が含まれます。 • ブランク 項目は、ストアード・プロシージャーが終了すると、ロード・モジュールがメモリーから削除されることを示します。
RUNOPTS	VARCHAR(254)	いいえ	いいえ	予約済み (空ストリング)。
PARM_LIST	VARCHAR(3000)	いいえ	いいえ	ストアード・プロシージャーのパラメーター・リスト。このパラメーター・リストの形式については、この表の次にある構文図を参照してください。
FENCED	CHAR(1)	いいえ	いいえ	プロシージャーが「分離された状態で」実行するかどうかを示します。 • *Y* はストアード・プロシージャーが分離されることを示します。 • *N* はストアード・プロシージャーが分離されないことを示します。
REMARKS	VARCHAR(254)	はい	いいえ	ストアード・プロシージャーの説明。
RESULT_SETS	SMALLINT	いいえ	いいえ	戻すことができる結果セットの数。

パラメーター・リスト列 (PARM_LIST) の入力形式は、図19 に定義されています。この列の内容に構文エラーがある場合、SQLProcedureColumns() への呼び出しはエラーとなります。



data-type



注: * - DB2 共通サーバー版ではサポートされていません。

図 19. PARMLIST ストリング構文. この PARMLIST 構文図は、 DB2 (MVS/ESA 版) と DB2 DB2 共通サーバー版の両方でサポートされているデータ・タイプを結合したものです。

付録I. サポートされている SQL ステートメント

表 215. SQL ステートメント (DB2 ユニバーサル・データベース)

SQL ステートメント	動的[1]	コマンド行プロセッサー (CLP)	コール・レベル・インターフェース[3] (CLI)	SQL プロシージャー
ALLOCATE CURSOR				X
割り当てステートメント				X
ASSOCIATE LOCATORS				X
ALTER { BUFFERPOOL, NICKNAME,[10] NODEGROUP, SERVER,[10] TABLE, TABLESPACE, USER MAPPING,[10] TYPE, VIEW }	X	X	X	
BEGIN DECLARE SECTION[2]				
CALL		X[9]	X[4]	X
CASE ステートメント				X
CLOSE		X	SQLCloseCursor(), SQLFreeStmt()	X
COMMENT ON	X	X	X	X
COMMIT	X	X	SQLEndTran, SQLTransact()	X
複合 SQL (組み込み)			X[4]	
複合ステートメント				X
CONNECT (タイプ 1)		X	SQLBrowseConnect(), SQLConnect(), SQLDriverConnect()	
CONNECT (タイプ 2)		X	SQLBrowseConnect(), SQLConnect(), SQLDriverConnect()	

表 215. SQL ステートメント (DB2 ユニバーサル・データベース) (続き)

SQL ステートメント	動的[1]	コマンド行プロセッサー (CLP)	コール・レベル・インターフェース[3] (CLI)	SQL プロシージャー
CREATE { ALIAS, BUFFERPOOL, DISTINCT TYPE, EVENT MONITOR, FUNCTION, FUNCTION MAPPING,[10] INDEX, INDEX EXTENSION, METHOD, NICKNAME,[10] NODEGROUP, PROCEDURE, SCHEMA, TABLE, TABLESPACE, TRANSFORM TYPE MAPPING,[1] TRIGGER, USER MAPPING,[10] TYPE, VIEW, WRAPPER[10] }	X	X	X	X[11]
DECLARE CURSOR[2]		X	SQLAllocStmt()	X
DECLARE GLOBAL TEMPORARY TABLE	X	X	X	X
DELETE	X	X	X	X
DESCRIBE[8]		X	SQLColAttributes(), SQLDescribeCol(), SQLDescribParam()[6]	
DISCONNECT		X	SQLDisconnect()	
DROP	X	X	X	X[11]
END DECLARE SECTION[2]				
EXECUTE			SQLExecute()	X
EXECUTE IMMEDIATE			SQLExecDirect()	X
EXPLAIN	X	X	X	X
FETCH		X	SQLExtendedFetch()[7], SQLFetch(), SQLFetchScroll()[7]	X
FLUSH EVENT MONITOR	X	X	X	
FOR ステートメント				X
FREE LOCATOR			X[4]	X
GET DIAGNOSTICS				X
GOTO ステートメント				X
GRANT	X	X	X	X

表 215. SQL ステートメント (DB2 ユニバーサル・データベース) (続き)

SQL ステートメント	動的[1]	コマンド行プロセッサー (CLP)	コール・レベル・インターフェース[3] (CLI)	SQL プロシージャー
IF ステートメント				X
INCLUDE[2]				
INSERT	X	X	X	X
ITERATE				X
LEAVE ステートメント				X
LOCK TABLE	X	X	X	X
LOOP ステートメント				X
OPEN		X	SQLExecute(), SQLExecDirect()	X
PREPARE			SQLPrepare()	X
REFRESH TABLE	X	X	X	
RELEASE		X		X
RELEASE SAVEPOINT	X	X	X	X
RENAME TABLE	X	X	X	
RENAME TABLESPACE	X	X	X	
REPEAT ステートメント				X
RESIGNAL ステートメント				X
RETURN ステートメント				X
REVOKE	X	X	X	
ROLLBACK	X	X	SQLEndTran(), SQLTransact()	X
SAVEPOINT	X	X	X	X
select-statement	X	X	X	X
SELECT INTO				X
SET CONNECTION		X	SQLSetConnection()	
SET CURRENT DEFAULT TRANSFORM GROUP	X	X	X	X
SET CURRENT DEGREE	X	X	X	X
SET CURRENT EXPLAIN MODE.	X	X	X, SQLSetConnectAttr()	X
SET CURRENT EXPLAIN SNAPSHOT	X	X	X, SQLSetConnectAttr()	X
SET CURRENT PACKAGESET				

表 215. SQL ステートメント (DB2 ユニバーサル・データベース) (続き)

SQL ステートメント	動的[1]	コマンド行プロセッサー (CLP)	コール・レベル・インターフェース[3] (CLI)	SQL プロシージャー
SET CURRENT QUERY OPTIMIZATION	X	X	X	X
SET CURRENT REFRESH AGE	X	X	X	X
SET EVENT MONITOR STATE	X	X	X	X
SET INTEGRITY	X	X	X	
SET PASSTHRU[10]	X	X	X	X
SET PATH	X	X	X	X
SET SCHEMA	X	X	X	X
SET SERVER OPTION[10]	X	X	X	X
SET 変換変数[5]	X	X	X	X
SIGNAL ステートメント				X
SIGNAL SQLSTATE[5]	X	X	X	
UPDATE	X	X	X	X
VALUES INTO				X
WHENEVER[2]				
WHILE ステートメント				X

表 215. SQL ステートメント (DB2 ユニバーサル・データベース) (続き)

SQL ステートメント	動的[1]	コマンド行プロセッサー (CLP)	コール・レベル・インターフェース[3] (CLI)	SQL プロシージャー

注:

1. このリストのすべてのステートメントは静的 SQL としてコーディングできますが、動的 SQL としてコーディングできるのは X マークの付いたステートメントだけです。
2. このステートメントは実行できません。
3. X は、該当するステートメントが SQLExecDirect() または SQLPrepare() と SQLExecute() のどちらによっても実行できるという意味です。等価の DB2 CLI 関数があると、その関数名がリストされます。
4. このステートメントは動的ではないものの、 DB2 CLI によって SQLExecDirect() または SQLPrepare() と SQLExecute() のどちらかを呼び出すときに、このステートメントが指定されます。
5. これは CREATE TRIGGER ステートメントでのみ使用することができます。
6. SQL DESCRIBE ステートメントは、出力の記述にのみ使用できます。 DB2 CLI では、入力も記述できます (SQLDescribeParam() 関数を使用)。
7. SQL FETCH ステートメントは、一度に 1 列を 1 方向に取り出す場合にのみ使用できます。 DB2 CLI の SQLExtendedFetch() および SQLFetchScroll() 関数では、配列から取り出すことができます。さらに、どちらの方向にも、また結果セットのどの位置でも取り出し可能です。
8. DESCRIBE SQL ステートメントの構文は、CLP DESCRIBE コマンドとは違っています。 DESCRIBE SQL ステートメントについては、*SQL 解説書* を参照してください。 DESCRIBE CLP コマンドについては、*コマンド解説書* を参照してください。
9. コマンド行プロセッサーから CALL を発行する場合は、特定のプロシージャーと個別パラメーターだけがサポートされます。
10. ステートメントは、連合データベース・サーバーでのみサポートされます。
11. SQL プロシージャーは、索引、表、視点に対しては、 CREATE および DROP ステートメントのみ発行できます。

付録J. CLI サンプル・コード

この付録では、本書に記したプログラムをはじめとする、 DB2 CLI プログラムのサンプルすべてをリストします。サンプルは、 OS/2 および Windows 32 ビット・オペレーティング・システムでは sqllib¥samples¥cli に、 UNIX プラットフォームでは sqllib/samples/cli にあります。 README ファイルには各ファイルの完全な説明が入っており、makefile を使用し、スクリプトを作成してすべてのサンプルを作成する方法について説明されています。

CLI 関数を呼び出すサンプルの完全なリストの他に、各 CLI 関数をリストしている HTML の相互参照表もあります。 OS/2 および 32 ビット Windows オペレーティング・システムの場合は sqllib¥doc¥html¥db2hs¥cli¥ に、UNIX プラットフォームの場合は sqllib/doc/html/db2hs/cli/ にある、ファイル funcxref.htm を参照してください。

次の表に、すべての DB2 CLI サンプル・プログラムをリストします。

表 216. DB2 ユニバーサル・データベースでのサンプル CLI プログラム

サンプル・プログラム名	プログラムの説明
共通ユーティリティー・ファイル	
utilcli.c	CLI サンプルで使用するユーティリティー関数。
utilapi.c	DB2 API を呼び出すユーティリティー関数。
アプリケーション・レベル - DB2 および CLI のアプリケーション・レベルを扱うサンプル。	
apinfo.c	アプリケーション・レベルの情報を入手および設定する方法。
aphndls.c	ハンドルを割り当ておよび解放する方法。
apsqlca.c	SQLCA データを処理する方法。
インストール・イメージ・レベル - DB2 および CLI のインストール・イメージ・レベルを扱うサンプル。	
ilinfo.c	インストール・レベル情報 (CLI ドライバーのバージョンなど) を入手および設定する方法。
インスタンス・レベル - DB2 および CLI のインスタンス・レベルを扱うサンプル。	
ininfo.c	インスタンス・レベルの情報を入手および設定する方法。
データベース・レベル - DB2 のデータベース・オブジェクトを扱うサンプル。	
dbconn.c	データベースへの接続および切断を行う方法。
dbinfo.c	データベース・レベルでの情報を入手および設定する方法。

表 216. DB2 ユニバーサル・データベースでのサンプル CLI プログラム (続き)

サンプル・プログラム名	プログラムの説明
dbmconn.c	複数のデータベースへの接続および切断を行う方法 (DB2 API を使用して、 2 番目のデータベースの作成および除去を行います)。
dbmuse.c	複数のデータベースとトランザクションをやり取りする方法 (DB2 API を使用して、 2 番目のデータベースの作成および除去を行います)。
dbnative.c	ODBC エスケープ文節を含んでいるステートメントを、データ・ソース固有の形式に変換する方法。
dbuse.c	データベース・オブジェクトを処理する方法。
dbusemx.sqc	組み込み SQL と組み合わせて 1 つのデータベースを使用する方法。
表レベル - DB2 の表オブジェクトを扱うサンプル。	
tbconstr.c	表の制約を処理する方法。
tbconstr.c	表の作成、変更、および除去を行う方法。
tbinfo.c	表レベルでの情報を入手および設定する方法。
tbmod.c	表内の情報を変更する方法。
tbread.c	表内の情報を読み取る方法。
データ・タイプ・レベル - データ・タイプを扱うサンプル。	
dtinfo.c	データ・タイプに関する情報を入手する方法。
dtlob.c	LOB データの読み取りおよび書き込みを行う方法。
dtudt.c	ユーザー定義特殊タイプの作成、使用、および除去を行う方法。
UDF レベル - ユーザー定義関数を例示しているサンプル。	
udfcli.c	udfsrv.c のユーザー定義関数を呼び出すクライアント・アプリケーション。
udfsrv.c	udfcli.c サンプルで呼び出されるユーザー定義関数 ScalarUDF。
ストアード・プロシージャー・レベル - CLI のストアード・プロシージャーを例示しているサンプル。	
spcreate.db2	CREATE PROCEDURE ステートメントを発行する CLP スクリプト。
spdrop.db2	カタログからストアード・プロシージャーを除去する CLP スクリプト。
spclient.c	spserver.c で宣言されているサーバー関数を呼び出すのに使用するクライアント・プログラム。
spserver.c	サーバーで作成および実行されるストアード・プロシージャー関数。
spcall.c	任意のストアード・プロシージャーを呼び出すプログラム。

注: samples/cli ディレクトリーの上記以外のファイルには、以下のものがあります。

- README - サンプル・ファイルをすべてリストします。
- makefile - すべてのファイルの MAKE ファイル
- アプリケーションおよびストアード・プロシージャーのビルド・ファイル

この節には、以下の 2 つのサンプル・ファイルがリストされています。

- embedded.c - DB2 CLI と組み込み呼び出しを比較する
- adhoc.c - 完全に対話式の SQL の例

組み込み SQL の例

この例は、X/Open SQL CLI の資料に入っている例を修正したバージョンです。注釈で組み込みステートメントを示し、等価の DB2 CLI 関数呼び出しも示しています。

```
/* From CLI sample embedded.c */
/* ... */
#include <string.h>
#include <stdlib.h>
#include <sqlcli1.h>
#include "samputil.h"          /* Header file for CLI sample code */
/* ... */
/*
 Global Variables for user id and password.
 To keep samples simple, not a recommended practice.
*/
extern SQLCHAR server[SQL_MAX_DSN_LENGTH + 1] ;
extern SQLCHAR uid[MAX_UID_LENGTH + 1] ;
extern SQLCHAR pwd[MAX_PWD_LENGTH + 1] ;
int main( int argc, char * argv[] ) {
    SQLHANDLE henv, hdbc, hstmt ;
    SQLRETURN rc ;
    SQLINTEGER id ;
    SQLCHAR name[51] ;
    SQLCHAR * create = "CREATE TABLE NAMEID (ID integer, NAME varchar(50))" ;
    SQLCHAR * insert = "INSERT INTO NAMEID VALUES (?, ?)" ;
    SQLCHAR * select = "select ID, NAME from NAMEID" ;
    SQLCHAR * drop   = "DROP TABLE NAMEID" ;
/* ... */
    /* EXEC SQL CONNECT TO :server USER :uid USING :authentication_string; */
    /* macro to initalize server, uid and pwd */
    INIT_UID_PWD ;
    /* allocate an environment handle */
    rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv ) ;
    if ( rc != SQL_SUCCESS ) return( terminate( henv, rc ) ) ;
    /* allocate a connect handle, and connect */
    rc = DBconnect( henv, &hdbc ) ;
    if ( rc != SQL_SUCCESS ) return( terminate( henv, rc ) ) ;
    /* allocate a statement handle */
    rc = SQLAllocHandle( SQL_HANDLE_STMT, hdbc, &hstmt ) ;
    CHECK_HANDLE( SQL_HANDLE_DBC, hdbc, rc ) ;
    /* EXEC SQL CREATE TABLE NAMEID (ID integer, NAME varchar(50)); */
    /* execute the sql statement */
    rc = SQLExecDirect( hstmt, create, SQL_NTS ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    /* EXEC SQL COMMIT WORK; */
    /* commit create table */
```

```
rc = SQLEndTran( SQL_HANDLE_DBC, hdbc, SQL_COMMIT ) ;
CHECK_HANDLE( SQL_HANDLE_DBC, hdbc, rc ) ;
/* EXEC SQL INSERT INTO NAMEID VALUES ( :id, :name ); */
/* show the use of SQLPrepare/SQLExecute method */
/* prepare the insert */
rc = SQLPrepare( hstmt, insert, SQL_NTS ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
/* Set up the first input parameter "id" */
rc = SQLBindParameter( hstmt,
                       1,
                       SQL_PARAM_INPUT,
                       SQL_C_LONG,
                       SQL_INTEGER,
                       0,
                       0,
                       (SQLPOINTER) & id,
                       0,
                       NULL
                     ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
/* Set up the second input parameter "name" */
rc = SQLBindParameter( hstmt,
                       2,
                       SQL_PARAM_INPUT,
                       SQL_C_CHAR,
                       SQL_VARCHAR,
                       51,
                       0,
                       name,
                       51,
                       NULL
                     ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
/* now assign parameter values and execute the insert */
id = 500 ;
strcpy( name, "Babbage" ) ;
rc = SQLExecute( hstmt ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
/* EXEC SQL COMMIT WORK; */
/* commit inserts */
rc = SQLEndTran( SQL_HANDLE_DBC, hdbc, SQL_COMMIT ) ;
CHECK_HANDLE( SQL_HANDLE_DBC, hdbc, rc ) ;
/*  Reset input parameter.  */
rc = SQLFreeStmt( hstmt, SQL_RESET_PARAMS ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
/* EXEC SQL DECLARE c1 CURSOR FOR SELECT ID, NAME FROM NAMEID; */
/* EXEC SQL OPEN c1; */
/* The application doesn't specify "declare c1 cursor for" */
rc = SQLExecDirect( hstmt, select, SQL_NTS ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
/* EXEC SQL FETCH c1 INTO :id, :name; */
/* Binding first column to output variable "id" */
SQLBindCol( hstmt, 1, SQL_C_LONG, ( SQLPOINTER ) & id, 0, NULL ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
/* Binding second column to output variable "name" */
```

```
    SQLBindCol( hstmt, 2, SQL_C_CHAR, name, 51, NULL ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    /* now execute the fetch        */
    while ( ( rc = SQLFetch( hstmt ) ) == SQL_SUCCESS )
       printf( "Result of Select: id = %ld name = %s¥n", id, name ) ;
    if (rc != SQL_NO_DATA_FOUND)
       CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    /* finally, we should commit, discard hstmt, disconnect */
    /* EXEC SQL COMMIT WORK; */
    /*  Close cursor and free bound columns.  */
    /*  Free statement resources  */
    rc = SQLFreeStmt( hstmt, SQL_UNBIND ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    rc = SQLFreeStmt( hstmt, SQL_CLOSE ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    /*  Drop table.  */
    rc = SQLExecDirect( hstmt, drop, SQL_NTS ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    /* commit the transaction */
    rc = SQLEndTran( SQL_HANDLE_DBC, hdbc, SQL_COMMIT ) ;
    CHECK_HANDLE( SQL_HANDLE_DBC, hdbc, rc ) ;
    /* EXEC SQL CLOSE c1; */
    /* free the statement handle */
    rc = SQLFreeHandle( SQL_HANDLE_STMT, hstmt ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    /* EXEC SQL DISCONNECT; */
    /* disconnect from the database */
    printf( "¥n>Disconnecting .....¥n" ) ;
    rc = SQLDisconnect( hdbc ) ;
    CHECK_HANDLE( SQL_HANDLE_DBC, hdbc, rc ) ;
    /* free the connection handle */
    rc = SQLFreeHandle( SQL_HANDLE_DBC, hdbc ) ;
    CHECK_HANDLE( SQL_HANDLE_DBC, hdbc, rc ) ;
    /* free the environment handle */
    rc = SQLFreeHandle( SQL_HANDLE_ENV,  henv ) ;
    if ( rc != SQL_SUCCESS ) return( terminate( henv, rc ) ) ;
    return( SQL_SUCCESS ) ;
}
```

対話式 SQL の例

この例は、X/Open SQL CLI の資料に入っている例を修正したバージョンです。対話式 SQL ステートメントの実行を示し、11ページの『第2章 DB2 CLI アプリケーションの作成』で説明されている流れにしたがっています。

```
/* From CLI sample adhoc.c */
/* ... */
/******************************************************************
** process_stmt
** - allocates a statement resources
** - executes the statement
** - determines the type of statement
**   - if there are no result columns, therefore non-select statement
```

```
**      - if rowcount > 0, assume statement was UPDATE, INSERT, DELETE
**     else
**      - assume a DDL, or Grant/Revoke statement
**   else
**      - must be a select statement.
**      - display results
** - frees the statement resources
*****************************************************************/
int process_stmt( SQLHANDLE hstmt, SQLCHAR * sqlstr ) {
    SQLSMALLINT     nresultcols;
    SQLINTEGER      rowcount;
    SQLRETURN       rc;
    /* execute the SQL statement in "sqlstr"    */
    rc = SQLExecDirect(hstmt, sqlstr, SQL_NTS);
    if (rc != SQL_SUCCESS)
        if (rc == SQL_NO_DATA_FOUND) {
            printf("¥nStatement executed without error, however,¥n");
            printf("no data was found or modified¥n");
            return (SQL_SUCCESS);
        }
        else CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    rc = SQLNumResultCols(hstmt, &nresultcols);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    /* determine statement type */
    if (nresultcols == 0) {     /* statement is not a select statement */
        rc = SQLRowCount(hstmt, &rowcount);
        if (rowcount > 0) /* assume statement is UPDATE, INSERT, DELETE */
            printf("Statement executed, %ld rows affected¥n", rowcount);
        else /* assume statement is GRANT, REVOKE or a DLL statement */
           printf( "Statement completed successful¥n" ) ;
    }
    else print_results( hstmt ) ; /* display the result set */
    /* end determine statement type */
    /* free statement resources */
    rc = SQLFreeStmt( hstmt, SQL_UNBIND ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    rc = SQLFreeStmt( hstmt, SQL_RESET_PARAMS ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    rc = SQLFreeStmt( hstmt, SQL_CLOSE ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
    return( 0 ) ;
}                           /* end process_stmt */


/* From the CLI sample utilcli.c */
/* ... */
        sqlrc = SQLDescribeCol( hstmt,
                        ( SQLSMALLINT ) ( i + 1 ),
                        colName,
                        sizeof(colName),
                        &colNameLen,
                        &colType,
```

```
&colSize,
&colScale,
NULL ) ;
```

付録K. DB2 CLI/ODBC/JDBC トレース機能の使用

アプリケーションで CLI、ODBC、または SQLJ と、DB2 CLI ドライバーを使用すれば、すべての関数呼び出しを平文ファイルにトレースさせることができます。これは、問題判別、データベース調整、アプリケーション調整などに役立つほか、サード・パーティーによるアプリケーションの内容を理解する上でも役に立ちます。

トレースは、実行時に、クライアント構成アシスタントからアクセスして CLI/ODBC 設定ノートブックを使用する (可能な場合) か、または db2cli.ini ファイルを直接編集することによって、使用可能にすることができます。また、CLI/ODBC/JDBC アプリケーションは、 SQL_ATTR_TRACE および SQL_ATTR_TRACEFILE 環境または接続属性を設定することによっても使用可能にできます。これらは、Microsoft ODBC ドライバー・マネージャーのトレース機能で使用している同じ属性です。

db2cli.ini ファイルを使用してトレースを使用可能にする

省略時設定では db2cli.ini ファイルは、 Intel プラットフォームでは ¥sqllib¥ パスに、 UNIX プラットフォームでは /sqllib/cfg/ パスにあります。トレース機能で使用される CLI/ODBC/JDBC 構成キーワードを以下に示します。

- 223ページの『TRACE』
- 224ページの『TRACECOMM』
- 225ページの『TRACEFILENAME』
- 226ページの『TRACEFLUSH』
- 227ページの『TRACEPATHNAME』

トレースを使用可能にするには、以下の行を追加する必要があります。キーワードは大文字小文字の区別がありません。

1. [COMMON]
2. Trace=1
3. TraceFileName= (完全なファイル名)

   または

   TracePathname= (完全パス名)
4. TraceFlush=(0 または 1)、任意選択
5. TraceComm=(0 または 1)、任意選択

たとえば、

```
[COMMON]
trace=1
tracefilename=d:¥temp¥clitrace.txt
```

TRACE を 0 に設定すると、トレースがオフになります。トレース・ファイル情報は、次回の必要に備えて構成ファイルに残ります。詳しくは 223ページの『TRACE』 を参照してください。

アプリケーションが終了していなかったり、存在していても異常だったりする場合、トレース・ファイルは完了しない可能性があります。 **TRACEFLUSH** を 1 に設定すると、関数呼び出しのたびにディスクをフラッシュします (これはトレースのオーバーヘッドを著しく増加させます)。詳しくは、226ページの『TRACEFLUSH』 を参照してください。

トレース・ファイルに含まれる各ネットワーク要求に関する情報を得るには、**TRACECOMM** を 1 に設定します。詳しくは 224ページの『TRACECOMM』を参照してください。

結果ファイルの位置決め

TRACEFILENAME キーワードで完全修飾ファイル名を使用すれば、ファイルの位置決めは容易です。相対パス名を使用すると、アプリケーションの現行パスをオペレーティング・システムがどのように認識するかで、違いが出てきます。

TRACEPATHNAME キーワードでファイル名の代わりにパス名を使用した場合、ディレクトリーをチェックして、アプリケーションの処理 ID に設定されている名前と、各固有スレッドごとの順序番号となる拡張子を持つファイル・セットが作成されているか調べる必要があります (たとえば、 65397.0、65397.1、 65397.2 など)。目当てのファイルを突き止めるのに、ファイルの日付やタイム・スタンプを使用すれば役立ちます。

相対パス名を使用すると、アプリケーションの現行パスをオペレーティング・システムがどのように認識するかで、違いが出てきます。

出力ファイルがない場合は、以下のようにします。

- キーワードが正しく db2cli.ini に設定されていることを確認します。
- アプリケーションが再始動されたことを確かめます。再始動して SQLAllocEnv() を呼び出さなければ、 db2cli.ini ファイルを読んで、トレースを初期設定することができないため。

- 指定のファイル名に対してアプリケーションが書き込みアクセスを持っていることを確かめます。
- DB2CLIINIPATH 環境変数が指定されているか確認します。この環境変数は、db2cli.ini ファイルの読み取り先を変更します。
- 最初の接続呼び出しを行うまで、ODBC アプリケーションは IBM CLI/ODBC/JDBC ドライバーにアクセスしません。アプリケーションがこの接続呼び出しを行うまで、トレース項目はファイルに書き込まれません。詳細については、922ページの『ODBC ドライバー・マネージャー・トレース』を参照してください。

トレース情報の読み取り

トレースの目的は、呼び出し順序、呼び出された各関数ごとの入出力引き数、および戻りコードを表示することにあります。トレースは、CLI/ODBC/JDBC 関数呼び出しに習熟している方のために用意されています。特に重要な 2 つの項目は、実行中の SQL ステートメント・テキストと、アプリケーションではおそらく報告されないエラー・メッセージです。

下記のそれぞれを探索するには、以下のようにします。

- SQL ステートメント

  トレース・ファイルを検索して、ストリング ″SQLExecDirect″ と ″SQLPrepare″ を探します。 SQL ステートメントが、テキストと ″──>″ 入力矢印を含んだ同じ行にあります (エディターでは、行が折り返しになる場合もあります)。
- エラー (アプリケーションにより照会されたもの)

  トレース・ファイルを検索して ″SQLError″ を探します。メッセージ・テキストが、ストリングと出力矢印 ″<──″ を含んだ同じ行にあります。
- エラー (アプリケーションにより無視されたもの)

  ″Unretrieved error message=″ (未検索エラー・メッセージ) を検索します。これは、前の呼び出しで SQL_ERROR または SQL_SUCCESS_WITH_INFO 戻りコードが戻されたのに、アプリケーションがエラー情報を照会しなかったことを示します。

  注: アプリケーションは、何らかのエラー・メッセージをあらかじめ予期していることもあります。トレース・ファイル中のすべてのエラー・メッセージを見て、重大なエラーを判別してください。

詳細トレース・ファイルの形式

下記のサンプル・トレース・ファイルを参照してください。ここでは行番号が追加されていますが、トレースでは表示されないことに留意してください。

- 行 1: IBM のサービスを補助するための作成日とプロダクト署名があります。
- 行 2〜3: 続きの 2 行が一緒になった最初のもので、関数呼び出しの入力引き数 (—>) を示します。整数引き数は、″SQL_HANDLE_ENV″ のような定義値にマップされることがあります。出力引き数は、通常、″&″ 接頭部のあるポインターとして示されます。
- 行 4〜5: 続きの 2 行で、関数呼び出しの出力結果 (<—) を示します。出力引き数だけが示されてから、2 行目に (—>) に続いて戻りコードが示されます。これと先にある入力行とを突き合わせます。
- 行 7: 入力時の経過時間の例。これは、アプリケーションにおける CLI/ODBC/JDBC 関数呼び出しと CLI/ODBC/JDBC 関数呼び出しの間の時間を秒で示したものです。

  注: このような時間の細分性や正確度は、プラットフォームによって異なります。
- 行 8: 出力時の経過時間の例。これは、CLI/ODBC/JDBC で関数の実行にかかった時間です。
- 行 18〜20: `SQLDriverConnect()` と `SQLConnect()` は、両方とも入力接続ストリングと db2cli.ini ファイルで設定されているキーワードを表示します。
- 行 23: 出力ステートメント・ハンドルは 1:1 のように示されます。最初の数は接続ハンドルで、2 番目はその接続でのステートメント・ハンドルを表します。これは記述子ハンドルにも該当しますが、接続ハンドルや環境ハンドルには該当しません。これらでは、最初の数は必ずゼロです。
- 行 29: `SQLPrepare()` の SQL ステートメント・テキストの例。
- 行 43〜44: `SQLBindParameter()` 呼び出し (行 33 〜 40) からの据え置き引き数。これは、準備ステートメント (行 29) の各 SQL パラメーター・マーカー (?) に送られるデータです。
- 行 79〜81: `SQLFetch()` 呼び出しからの出力。 (iCol = 列、rgbValue = 文字形式のデータ、pcbValue=長さ)。
- 行 110: メッセージ・テキストを示す `SQLError()` 出力。エラーがデータベース・サーバーではなく CLI/ODBC/JDBC に起因する場合、 pfNativeError は DB2 SQLCODE か -9999 です。
- 行 123: 未検索エラー・メッセージを示します。これは、前にエラーが発生しながら、アプリケーションでは検索されたことがないハンドルを使用して

関数を呼び出した場合に必ず示されます。これは、この時点でアプリケーションにとっては実質的に「消失」しますが、トレースでは取り込まれます。

サンプル・トレース・ファイル

この例では、便宜上行番号が追加されていますが、トレースでは表示されません。

```
 1  Build Date: 97/05/13-Product: QDB2/6000 (4) - Driver Version: 05.00.0000
 2  SQLAllocHandle( fHandleType=SQL_HANDLE_ENV, hInput=0:0,
          phOutput=&2ff7f388 )
 3      ---> Time elapsed - +1.399700E-002 seconds
 4  SQLAllocHandle( phOutput=0:1 )
 5      <--- SQL_SUCCESS   Time elapsed - +6.590000E-003 seconds
 6  SQLAllocHandle( fHandleType=SQL_HANDLE_DBC, hInput=0:1,
          phOutput=&2ff7f378 )
 7      ---> Time elapsed - +1.120000E-002 seconds
 8  SQLAllocHandle( phOutput=0:1 )
 9      <--- SQL_SUCCESS   Time elapsed - +8.979000E-003 seconds
10  SQLSetConnectOption( hDbc=0:1, fOption=SQL_ATTR_AUTOCOMMIT, vParam=0 )
11      ---> Time elapsed - +6.638000E-003 seconds
12  SQLSetConnectOption( )
13      <--- SQL_SUCCESS   Time elapsed - +1.209000E-003 seconds
14  SQLDriverConnect( hDbc=0:1, hwnd=0:0, szConnStrIn="DSN=loopback;
          uid=clitest1;pwd=*******", cbConnStrIn=-3, szConnStrOut=&2ff7e7b4,
          cbConnStrOutMax=250, pcbConnStrOut=&2ff7e7ae,
          fDriverCompletion=SQL_DRIVER_NOPROMPT )
15      ---> Time elapsed - +1.382000E-003 seconds
16  SQLDriverConnect( szConnStrOut="DSN=LOOPBACK;UID=clitest1;PWD=*******;",
          pcbConnStrOut=38 )
17      <--- SQL_SUCCESS   Time elapsed - +7.675910E-001 seconds
18  ( DSN="LOOPBACK" )
19  ( UID="clitest1" )
20  ( PWD="*******" )
21  SQLAllocHandle( fHandleType=SQL_HANDLE_STMT, hInput=0:1,
          phOutput=&2ff7f378 )
22      ---> Time elapsed - +1.459900E-002 seconds
23  SQLAllocHandle( phOutput=1:1 )
24      <--- SQL_SUCCESS   Time elapsed - +7.008300E-002 seconds
25  SQLExecDirect( hStmt=1:1, pszSqlStr="create table test(id integer,
          name char(20), created date)", cbSqlStr=-3 )
26      ---> Time elapsed - +1.576899E-002 seconds
27  SQLExecDirect( )
28      <--- SQL_SUCCESS   Time elapsed - +1.017835E+000 seconds
29  SQLPrepare( hStmt=1:1, pszSqlStr="insert into test
          values (?, ?, current date)", cbSqlStr=-3 )
30      ---> Time elapsed - +5.008000E-003 seconds
31  SQLPrepare( )
32      <--- SQL_SUCCESS   Time elapsed - +7.896000E-003 seconds
33  SQLBindParameter( hStmt=1:1, iPar=1, fParamType=SQL_PARAM_INPUT,
          fCType=SQL_C_LONG, fSQLType=SQL_INTEGER, cbColDef=4, ibScale=0,
          rgbValue=&20714d88, cbValueMax=4, pcbValue=&20714d54 )
34      ---> Time elapsed - +2.870000E-003 seconds
35  SQLBindParameter( )
```

```
36      <--- SQL_SUCCESS   Time elapsed - +3.803000E-003 seconds
37  SQLBindParameter( hStmt=1:1, iPar=2, fParamType=SQL_PARAM_INPUT,
          fCType=SQL_C_CHAR, fSQLType=SQL_CHAR, cbColDef=20, ibScale=0,
          rgbValue=&20714dd8, cbValueMax=21, pcbValue=&20714da4 )
38      ---> Time elapsed - +2.649000E-003 seconds
39  SQLBindParameter( )
40      <--- SQL_SUCCESS   Time elapsed - +3.882000E-003 seconds
41  SQLExecute( hStmt=1:1 )
42      ---> Time elapsed - +3.681000E-003 seconds
43  ( iPar=1, fCType=SQL_C_LONG, rgbValue=10, pcbValue=4, piIndicatorPtr=4 )
44  ( iPar=2, fCType=SQL_C_CHAR, rgbValue="-3", pcbValue=2, piIndicatorPtr=2 )
45  SQLExecute( )
46      <--- SQL_SUCCESS   Time elapsed - +4.273490E-001 seconds
47  SQLExecute( hStmt=1:1 )
48      ---> Time elapsed - +5.483000E-003 seconds
49  ( iPar=1, fCType=SQL_C_LONG, rgbValue=10, pcbValue=4, piIndicatorPtr=4 )
50  ( iPar=2, fCType=SQL_C_CHAR, rgbValue="-3", pcbValue=2, piIndicatorPtr=2 )
51  SQLExecute( )
52      <--- SQL_SUCCESS   Time elapsed - +1.299300E-002 seconds
53  SQLExecute( hStmt=1:1 )
54      ---> Time elapsed - +3.702000E-003 seconds
55  ( iPar=1, fCType=SQL_C_LONG, rgbValue=10, pcbValue=4, piIndicatorPtr=4 )
56  ( iPar=2, fCType=SQL_C_CHAR, rgbValue="-3", pcbValue=2, piIndicatorPtr=2 )
57  SQLExecute( )
58      <--- SQL_SUCCESS   Time elapsed - +1.265700E-002 seconds
59  SQLExecDirect( hStmt=1:1, pszSqlStr="select * from test", cbSqlStr=-3 )
60      ---> Time elapsed - +2.983000E-003 seconds
61  SQLExecDirect( )
62      <--- SQL_SUCCESS   Time elapsed - +2.469180E-001 seconds
63  SQLBindCol( hStmt=1:1, iCol=1, fCType=SQL_C_LONG, rgbValue=&20714e38,
          cbValueMax=4, pcbValue=&20714e04 )
64      ---> Time elapsed - +5.069000E-003 seconds
65  SQLBindCol( )
66      <--- SQL_SUCCESS   Time elapsed - +2.660000E-003 seconds
67  SQLBindCol( hStmt=1:1, iCol=2, fCType=SQL_C_CHAR, rgbValue=&20714e88,
          cbValueMax=21, pcbValue=&20714e54 )
68      ---> Time elapsed - +2.492000E-003 seconds
69  SQLBindCol( )
70      <--- SQL_SUCCESS   Time elapsed - +2.795000E-003 seconds
71  SQLBindCol( hStmt=1:1, iCol=3, fCType=SQL_C_CHAR, rgbValue=&20714ee8,
          cbValueMax=21, pcbValue=&20714eb4 )
72      ---> Time elapsed - +2.490000E-003 seconds
73  SQLBindCol( )
74      <--- SQL_SUCCESS   Time elapsed - +2.749000E-003 seconds
75  SQLFetch( hStmt=1:1 )
76      ---> Time elapsed - +2.660000E-003 seconds
77  SQLFetch( )
78      <--- SQL_SUCCESS   Time elapsed - +9.200000E-003 seconds
79  ( iCol=1, fCType=SQL_C_LONG, rgbValue=10, pcbValue=4 )
80  ( iCol=2, fCType=SQL_C_CHAR, rgbValue="-3                   ",
          pcbValue=20 )
81  ( iCol=3, fCType=SQL_C_CHAR, rgbValue="1997-05-23", pcbValue=10 )
82  SQLFetch( hStmt=1:1 )
83      ---> Time elapsed - +4.942000E-003 seconds
84  SQLFetch( )
```

```
 85      <--- SQL_SUCCESS   Time elapsed - +7.860000E-003 seconds
 86  ( iCol=1, fCType=SQL_C_LONG, rgbValue=10, pcbValue=4 )
 87  ( iCol=2, fCType=SQL_C_CHAR, rgbValue="-3                  ",
         pcbValue=20 )
 88  ( iCol=3, fCType=SQL_C_CHAR, rgbValue="1997-05-23", pcbValue=10 )
 89  SQLFetch( hStmt=1:1 )
 90      ---> Time elapsed - +4.872000E-003 seconds
 91  SQLFetch( )
 92      <--- SQL_SUCCESS   Time elapsed - +7.669000E-003 seconds
 93  ( iCol=1, fCType=SQL_C_LONG, rgbValue=10, pcbValue=4 )
 94  ( iCol=2, fCType=SQL_C_CHAR, rgbValue="-3                  ",
         pcbValue=20 )
 95  ( iCol=3, fCType=SQL_C_CHAR, rgbValue="1997-05-23", pcbValue=10 )
 96  SQLFetch( hStmt=1:1 )
 97      ---> Time elapsed - +5.103000E-003 seconds
 98  SQLFetch( )
 99      <--- SQL_NO_DATA_FOUND   Time elapsed - +6.044000E-003 seconds
100  SQLCloseCursor( hStmt=1:1 )
101      ---> Time elapsed - +2.682000E-003 seconds
102  SQLCloseCursor( )
103      <--- SQL_SUCCESS   Time elapsed - +6.794000E-003 seconds
104  SQLExecDirect( hStmt=1:1, pszSqlStr="select * foo bad sql", cbSqlStr=-3 )
105      ---> Time elapsed - +2.967000E-003 seconds
106  SQLExecDirect( )
107      <--- SQL_ERROR   Time elapsed - +1.103700E-001 seconds
108  SQLError( hEnv=0:0, hDbc=0:0, hStmt=1:1, pszSqlState=&2ff6f19c,
           pfNativeError=&2ff6ed00, pszErrorMsg=&2ff6ed9c, cbErrorMsgMax=1024,
           pcbErrorMsg=&2ff6ed0a )
109      ---> Time elapsed - +2.267000E-003 seconds
110  SQLError( pszSqlState="42601", pfNativeError=-104,
           pszErrorMsg="[IBM][CLI Driver][DB2/6000] SQL0104N  An unexpected
           token "foo bad sql" was found following "select * ".
           Expected tokens may include:  "<space>".  SQLSTATE=42601
111  ", pcbErrorMsg=163 )
112      <--- SQL_SUCCESS   Time elapsed - +5.299000E-003 seconds
113  SQLError( hEnv=0:0, hDbc=0:0, hStmt=1:1, pszSqlState=&2ff6f19c,
           pfNativeError=&2ff6ed00, pszErrorMsg=&2ff6ed9c, cbErrorMsgMax=1024,
           pcbErrorMsg=&2ff6ed0a )
114      ---> Time elapsed - +2.753000E-003 seconds
115  SQLError( )
116      <--- SQL_NO_DATA_FOUND   Time elapsed - +2.502000E-003 seconds
117  SQLExecDirect( hStmt=1:1, pszSqlStr="select * foo bad sql", cbSqlStr=-3 )
118      ---> Time elapsed - +3.292000E-003 seconds
119  SQLExecDirect( )
120      <--- SQL_ERROR   Time elapsed - +6.012500E-002 seconds
121  SQLFreeHandle( fHandleType=SQL_HANDLE_STMT, hHandle=1:1 )
122      ---> Time elapsed - +2.867000E-003 seconds
123  ( Unretrieved error message="SQL0104N  An unexpected token "foo bad sql"
           was found following "select * ".  Expected tokens may
           include:  "<space>".  SQLSTATE=42601
124  " )
125  SQLFreeHandle( )
126      <--- SQL_SUCCESS   Time elapsed - +4.936600E-002 seconds
127  SQLEndTran( fHandleType=SQL_HANDLE_DBC, hHandle=0:1, fType=SQL_ROLLBACK )
128      ---> Time elapsed - +2.968000E-003 seconds
```

```
129  SQLEndTran( )
130       <--- SQL_SUCCESS   Time elapsed - +1.643370E-001 seconds
131  SQLDisconnect( hDbc=0:1 )
132       ---> Time elapsed - +2.559000E-003 seconds
133  SQLDisconnect( )
134       <--- SQL_SUCCESS   Time elapsed - +8.253310E-001 seconds
135  SQLFreeHandle( fHandleType=SQL_HANDLE_DBC, hHandle=0:1 )
136       ---> Time elapsed - +4.247000E-003 seconds
137  SQLFreeHandle( )
138       <--- SQL_SUCCESS   Time elapsed - +4.742000E-003 seconds
139  SQLFreeHandle( fHandleType=SQL_HANDLE_ENV, hHandle=0:1 )
140       ---> Time elapsed - +2.023000E-003 seconds
141  SQLFreeHandle( )
142       <--- SQL_SUCCESS   Time elapsed - +4.420000E-003 seconds
```

マルチスレッド・アプリケーションまたはマルチプロセス・アプリケーションのトレース

マルチスレッド・アプリケーションまたはマルチプロセス・アプリケーションにとってトレースが役立つ場合、 **TRACEPATHNAME** キーワードを使用する必要があります。使用しないと、マルチスレッドまたはマルチプロセスが同時にトレースに書き込みする場合に、トレースが不明瞭になります。詳しくは 227ページの『TRACEPATHNAME』 を参照してください。

アプリケーションの処理 ID に設定されている名前と、各固有スレッドごとの順序番号となる拡張子が指定されたパスで作成されます (たとえば、 65397.0、65397.1、65397.2 など).

各スレッドが独自のファイルに書き込むようにすると、セマフォーがなくてもトレース・ファイルへのアクセスを制御できます。つまり、マルチスレッド・アプリケーションの動作をトレースが変更しないということです。

ODBC ドライバー・マネージャー・トレース

ODBC トレースでも、ODBC ドライバー・マネージャーが提供するものと、DB2 CLI/ODBC ドライバー (IBM ODBC ドライバー・トレース) が提供するものとでは違いがあることを覚えておくと有益です。

出力ファイル形式が異なります。 ODBC トレースでは、アプリケーションからドライバー・マネージャーへの呼び出しを表示するという点が異なります。DB2 CLI トレースでは、ODBC ドライバー・マネージャーから受けた呼び出しを表示します。

ODBC ドライバー・マネージャーは、アプリケーション関数呼び出しを、別の関数や別の引き数にマップしたり、あるいは呼び出しを遅らせたりします。

以下のうちの 1 つまたは複数が適用される場合があります。

- ODBC 3.0 では置き換えられている ODBC 2.0 関数を使用して書いたアプリケーションでは、 ODBC ドライバー・マネージャーによって、旧関数が新関数にマップされます。
- 関数引き数によっては、ODBC 2.0 の値から ODBC 3.0 の同等値に値がマップされるものもあります。
- Microsoft カーソル・ライブラリーでは、`SQLExtendedFetch()` などの呼び出しを、取り出したい複数の呼び出しやその他のサポート関数にマップします。

そのような理由で、どのようになっているかを理解するには、両方のトレースを使用したり比較したりできることが必要となります。

詳しくは、*Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference* を参照してください。

付録L. DB2 ライブラリーの使用法

DB2 ユニバーサル・データベース ライブラリーは、オンライン・ヘルプ、ブック (PDF および HTML)、および HTML 形式のサンプル・プログラムから成っています。このセクションでは、ユーザーに提供される情報について紹介し、その入手方法を示します。

オンライン製品情報をご利用になるには、インフォメーション・センターを使用することができます。詳細については、941ページの『インフォメーション・センターを使用した情報へのアクセス』を参照してください。ここではタスク情報、DB2 ブック、トラブルシューティング情報、サンプル・プログラム、および Web の DB2 情報を見ることができます。

DB2 PDF ファイルおよびハードコピー版資料

DB2 情報

以下に示す表では、DB2 ブックを 4 つのカテゴリーに分類しています。

DB2 の手引きおよび解説書
これらの資料は、すべてのプラットフォームに共通の DB2 情報を含んでいます。

DB2 のインストールおよび構成の情報
これらの資料は、特定のプラットフォーム上の DB2 ごとに用意されています。たとえば、 OS/2、Windows、および UNIX ベースのプラットフォームで稼働するそれぞれの DB2 用に、別個の*概説*および*インストール* 資料が用意されています。

プラットフォーム共通のサンプル・プログラム (HTML 形式)
これらのサンプルは、アプリケーション開発クライアントとともにインストールされるサンプル・プログラムの HTML 版です。これらのサンプルは参考用であり、実際のプログラムに代わるものではありません。

リリース情報
これらのファイルには、DB2 ブックには含められなかった最新の情報が記載されています。

インストール情報、リリース情報、およびチュートリアルは、製品 CD-ROM から HTML 形式で参照することができます。ほとんどの資料は、製品

CD-ROM から HTML 形式で表示できますし、 DB2 の資料 CD-ROM から Adobe Acrobat (PDF) 形式で表示し印刷することができます。 IBM にハードコピー版の資料を注文したい場合は、 937ページの『印刷資料の注文方法』を参照してください。注文可能な資料については、以下の表をご覧ください。

OS/2 および Windows プラットフォームの場合、 HTML ファイルは sqllib¥doc¥html ディレクトリーにインストールできます。 DB2 情報はいくつかの言語で提供されています。しかし、すべての言語に翻訳されているわけではありません。ある言語で情報が提供されていない場合は、英語版の情報が提供されます。

UNIX プラットフォームの場合、言語ごとに異なる複数の HTML ファイルを doc/%*L*/html ディレクトリーにインストールできます。ここで、*%L* は地域を表しています。詳細については、適切な「概説およびインストールの手引き」を参照してください。

DB2 ブックを入手して情報を利用するには、次のようなさまざまな方法があります。

- 940ページの『オンライン情報の表示』
- 945ページの『オンライン情報の検索』
- 937ページの『印刷資料の注文方法』
- 937ページの『PDF 資料の印刷』

表 217. DB2 情報

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
	DB2 の手引きおよび解説書情報		
管理の手引き	*管理の手引き: 計画* は、データベース概念について概説し、設計 (たとえば、論理および物理データベース設計) に関する情報を提供し、高い可用性について解説しています。	第 1 巻 SC88-8513 db2d1x70	db2d0
	管理の手引き: インプリメンテーション は、設計、データベースへのアクセス、監査、バックアップ、および回復などのインプリメンテーションについて説明しています。	第 2 巻 SC88-8511 db2d2x70	
	管理の手引き: パフォーマンス は、データベース環境について解説し、さらにアプリケーションのパフォーマンスの評価と調整の方法について説明しています。	第 3 巻 SC88-8512 db2d3x70	
管理 API 解説書	データベースの管理に使用できる DB2 アプリケーション・プログラミング・インターフェース (API) およびデータ構造について説明します。また、この資料は、アプリケーションから API を呼び出す方法も示します。	SC88-8514 db2b0x70	db2b0
アプリケーション構築の手引き	環境設定に関する情報を提供し、Windows、OS/2、および UNIX ベースのプラットフォームでの DB2 アプリケーションのコンパイル、リンク、実行の各ステップについて説明します。	SC88-8515 db2axx70	db2ax
APPC, CPI-C, and SNA Sense Codes	DB2 ユニバーサル・データベース製品をご使用中に発生する可能性のあるセンス・コード APPC、CPI-C、および SNA についての一般情報を提供します。 HTML 形式でのみご利用いただけます。	資料番号なし db2apx70	db2ap

表 217. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
アプリケーション開発の手引き	DB2 データベースにアクセスするアプリケーションを、組み込み SQL または Java (JDBC および SQLJ) を使用して開発する方法について説明します。さらに、ストアード・プロシージャーの作成方法、ユーザー定義関数の作成方法、ユーザー定義タイプの作成方法、トリガーの使用方法、区画化されている環境または統合されているシステムでのアプリケーションの開発方法などについて解説されています。	SC88-8516 db2a0x70	db2a0
コール・レベル・インターフェースの手引きおよび解説書	DB2 データベースにアクセスするアプリケーションを、 DB2 コール・レベル・インターフェース (Microsoft ODBC 仕様互換の呼び出し可能 SQL) を使用して開発する方法について説明します。	SC88-8517 db2l0x70	db2l0
コマンド解説書	コマンド行プロセッサーの使用法について説明し、データベースの管理に使用できる DB2 コマンドについて解説しています。	SC88-8518 db2n0x70	db2n0
コネクティビティー 補足	DB2 (AS/400 版)、DB2 (OS/390 版)、DB2 (MVS 版)、または DB2 (VM 版) を DRDA アプリケーション・リクエスターとして DB2 ユニバーサル・データベースとともに使用するためのセットアップ情報および参照情報を提供します。また、この資料は DRDA アプリケーション・サーバーを DB2 コネクト アプリケーション・リクエスターとともに使用する方法の詳細を示します。 HTML と PDF でのみ利用可能	資料番号なし db2h1x70	db2h1
データ移動ユーティリティー 手引きおよび解説書	データの移動を行う DB2 ユーティリティー (インポート、エクスポート、ロード、AutoLoader、および DPROP など) の使用法について説明しています。	SC88-8522 db2dmx70	db2dm

表 217. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
データウェアハウスセンター 管理の手引き	データウェアハウスセンターを使用してデータウェアハウスを構築および保守する方法を説明します。	SC88-8545 db2ddx70	db2dd
データウェアハウスセンター アプリケーション統合の手引き	プログラマーがアプリケーションをデータウェアハウスセンターおよび情報カタログ・マネージャーと統合するのに役立つ情報を提供します。	SC88-8546 db2adx70	db2ad
DB2 コネクト 使用者の手引き	DB2 コネクト製品の概念、プログラミング、および一般的な使用方法に関する情報を提供します。	SC88-8521 db2c0x70	db2c0
DB2 クエリー・パトローラー 管理の手引き	DB2 クエリー・パトローラー・システムの運用の概説を行い、運用および管理に関する詳細情報、および管理用グラフィカル・ユーザー・インターフェース・ユーティリティーについてのタスク情報を提供します。	SC88-8525 db2dwx70	db2dw
DB2 クエリー・パトローラー 使用者の手引き	DB2 クエリー・パトローラーのツールや関数の使用方法を説明します。	SC88-8527 db2wwx70	db2ww
用語集	DB2 およびその構成要素で使用される用語の定義を示します。 HTML 形式と SQL 解説書 で利用可能	資料番号なし db2t0x70	db2t0
イメージ、オーディオ、およびビデオ・エクステンダー 管理およびプログラミングの手引き	DB2 エクステンダーの一般情報について提供し、画像、音声、およびビデオ (IAV) エクステンダーの管理と構成について、および IAV エクステンダーを使用したプログラミングについて説明しています。さらに、参照情報、診断情報 (メッセージ解説)、およびサンプルも収録されています。	SC88-8609 dmbu7x70	dmbu7
情報カタログ・マネージャー 管理の手引き	情報カタログを管理するためのガイドです。	SC88-8547 db2dix70	db2di
情報カタログ・マネージャー プログラミングの手引きおよび解説書	情報カタログ・マネージャー用の体系化されたインターフェースの定義を示します。	SC88-8549 db2bix70	db2bi

表 217. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
情報カタログ・マネージャー 使用者の手引き	情報カタログ・マネージャー・ユーザー・インターフェースの使用に関する情報を提供します。	SC88-8548 db2aix70	db2ai
インストールおよび構成補足	プラットフォーム固有の DB2 クライアントの計画、インストール、およびセットアップのガイドです。この補足資料には、バインド、クライアント / サーバー通信の設定、 DB2 GUI ツール、DRDA AS、分散インストール、分散要求の構成、および異種データ・ソースへのアクセスについても説明されています。	GC88-8524 db2iyx70	db2iy
メッセージ解説書	DB2、情報カタログ・マネージャー、およびデータウェアハウスセンターから出されるメッセージとコードをリストし、取るべき処置を解説しています。	第 1 巻 GC88-8543 db2m1x70 第 2 巻 GC88-8544 db2m2x70	db2m0
OLAP Integration Server Administration Guide	OLAP Integration Server の Administration Manager 構成要素の使用方法を説明します。	SC27-0782 db2dpx70	n/a
OLAP Integration Server Metaoutline User's Guide	標準の OLAP Metaoutline インターフェースを使用して (Metaoutline Assistant を使用するのではなく) OLAP metaoutline を作成しデータを取り込む方法を説明しています。	SC27-0784 db2upx70	n/a
OLAP Integration Server Model User's Guide	(Model Assistant ではなく) 標準的な OLAP Model Interface を使用して OLAP モデルを作成する方法を説明します。	SC27-0783 db2lpx70	n/a
OLAP Setup and User's Guide	OLAP Starter Kit の構成およびセットアップに関する情報を提供します。	SC27-0702 db2ipx70	db2ip

表 217. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
OLAP Spreadsheet Add-in User's Guide for Excel	Excel 作表計算プログラムを使用して OLAP データを分析する方法を説明します。	SC27-0786 db2epx70	db2ep
OLAP Spreadsheet Add-in User's Guide for Lotus 1-2-3	ロータス 1-2-3 作表計算プログラムを使用して OLAP データを分析する方法を説明します。	SC27-0785 db2tpx70	db2tp
レプリケーションの手引きおよび解説書	DB2 に付属の IBM レプリケーション・ツールの計画、構成、管理、および使用方法に関する情報を提供します。	SC88-8550 db2e0x70	db2e0
地理情報エクステンダー使用者の手引きおよび解説書	地理情報エクステンダーのインストール、構成、管理、プログラミング、およびトラブルシューティングに関する情報を提供します。また、地理情報データの概念についての重要事項を示し、地理情報エクステンダー固有の参照情報 (メッセージおよび SQL) を提供します。	SC88-8624 db2sbx70	db2sb
SQL 概説	SQL の概念を紹介し、構造体とタスクの例を多数提供しています。	SC88-8539 db2y0x70	db2y0
SQL 解説書	SQL の構文、セマンティクス、および言語規則について説明します。また、この資料には、各リリース間の互換性、製品の制限事項、およびカタログ・ビューも含まれます。	第 1 巻 SC88-8540 db2s1x70 第 2 巻 SC88-8657 db2s2x70	db2s0
システム・モニター 手引きおよび解説書	データベースおよびデータベース・マネージャーに関連したさまざまな情報を収集する方法を示します。この資料は、この情報を利用して、データベース活動の把握、パフォーマンス向上、および問題原因の判別を行う方法を説明しています。	SC88-8523 db2f0x70	db2f0

表 217. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
テキスト・エクステンダー *管理およびプログラミング*	DB2 エクステンダーの一般情報、テキスト・エクステンダーの管理および構成情報、およびテキスト・エクステンダーを使用したプログラミングの方法について解説します。この資料には、参照情報、診断情報 (メッセージ解説)、およびサンプルが含まれています。	SC88-8610 desu9x70	desu9
問題判別の手引き	エラーの原因の判別、問題からの回復、および DB2 カスタマー・サービスの支援の下での診断ツールの使用法を記載しています。	GD88-7271 db2p0x70	db2p0
新機能	DB2 ユニバーサル・データベース バージョン 7 の新しい機能および拡張機能について説明します。	SC88-8541 db2q0x70	db2q0
DB2 のインストールおよび構成の情報			
DB2 コネクト エンタープライズ・エディション (OS/2 および Windows 版) 概説およびインストール	OS/2 および Windows 32 ビット オペレーティング・システム版の DB2 コネクト エンタープライズ・エディションで、計画、移行、インストール、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8520 db2c6x70	db2c6
DB2 コネクト エンタープライズ・エディション (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 コネクト エンタープライズ・エディションの計画、移行、インストール、構成、およびタスクに関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8519 db2cyx70	db2cy

表 217. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
DB2 コネクト パーソナル・エディション 概説およびインストール	OS/2 および Windows 32 ビット オペレーティング・システムの DB2 コネクト パーソナル・エディションで、計画、移行、インストール、および構成を行う場合のタスク情報を提供します。また、この資料はサポートされているすべてのクライアントのインストールおよびセットアップについても説明します。	GC88-8533 db2c1x70	db2c1
DB2 コネクト パーソナル・エディション (Linux 版) 概説およびインストール	サポートされる Linux 配布プログラムの DB2 コネクト パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8528 db2c4x70	db2c4
DB2 データ・リンク・マネージャー (Windows 版) 概説およびインストール	AIX および Windows 32 ビット・オペレーティング・システムの DB2 データ・リンク・マネージャーで、計画、インストール、構成を行う場合の情報を提供します。	GC88-8532 db2z6x70	db2z6
DB2 エンタープライズ拡張エディション (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 エンタープライズ拡張エディションの計画、インストール、および構成に関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8530 db2v3x70	db2v3
DB2 エンタープライズ拡張エディション (Windows 版) 概説およびインストール	Windows 32 ビット・オペレーティング・システムの DB2 エンタープライズ拡張エディションで、計画、インストール、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8529 db2v6x70	db2v6

表 217. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
DB2 ユニバーサル・データベース (OS/2 版) 概説およびインストール	OS/2 オペレーティング・システムでの DB2 ユニバーサル・データベースの計画、インストール、移行、および構成に関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8534 db2i2x70	db2i2
DB2 ユニバーサル・データベース (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 ユニバーサル・データベースの計画、インストール、移行、および構成に関する情報を提供します。 また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8536 db2ixx70	db2ix
DB2 ユニバーサル・データベース (Windows 版) 概説およびインストール	Windows 32 ビット オペレーティング・システムの DB2 ユニバーサル・データベースで、計画、インストール、移行、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8537 db2i6x70	db2i6
DB2 パーソナル・エディション 概説およびインストール	OS/2 および Windows 32 ビット オペレーティング・システム版の DB2 ユニバーサル・データベース パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8535 db2i1x70	db2i1
DB2 パーソナル・エディション (Linux 版) 概説およびインストール	サポートされる Linux 配布プログラムの DB2 ユニバーサル・データベース・パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8538 db2i4x70	db2i4
DB2 クエリー・パトローラー インストールの手引き	DB2 クエリー・パトローラーのインストール情報を提供します。	GC88-8526 db2iwx70	db2iw

表 217. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
データウェアハウス・マネージャー インストールの手引き	ウェアハウス・エージェント、ウェアハウス・トランスフォーマー、および情報カタログ・マネージャーのインストール情報を提供します。	GC88-8572 db2idx70	db2id
プラットフォーム共通のサンプル・プログラム (HTML 形式)			
サンプル・プログラム (HTML)	DB2 のサポートするすべてのプラットフォームでのプログラム言語用に、サンプル・プログラム (HTML 形式) を提供します。これらのサンプル・プログラムは、参照用としてのみ提供されています。サンプルは、すべてのプログラミング言語で利用できるわけではありません。HTML サンプルが利用できるのは、 DB2 アプリケーション開発クライアントがインストールされている場合だけです。 プログラムの詳細については、*アプリケーション構築の手引き* を参照してください。	資料番号なし	db2hs
リリース情報			
DB2 コネクト リリース情報	DB2 コネクトの資料には含められなかった最新の情報が収録されています。	注 #2 を参照してください。	db2cr
DB2 インストール情報	DB2 ブックには含められなかったインストールに関する最新の情報が収録されています。	製品 CD-ROM からのみ利用できます。	
DB2 リリース情報	DB2 ブックには含められなかった DB2 製品とその機能に関する最新の情報が収録されています。	注 #2 を参照してください。	db2ir

注:

1. ファイル名の 6 桁目の文字 *x* は、その資料の言語を表します。たとえば、ファイル名 `db2d0e70` は、*管理の手引き* の英語版であることを示し、ファイル名 `db2d0f70` は同じ資料のフランス語版を示します。資料の言語を表すためにファイル名の 6 桁目で使用されている文字は以下のとおりです。

言語	識別子
ブラジル・ポルトガル語	b
ブルガリア語	u
チェコ語	x
デンマーク語	d
オランダ語	q
英語	e
フィンランド語	y
フランス語	f
ドイツ語	g
ギリシャ語	a
ハンガリー語	h
イタリア語	i
日本語	j
韓国語	k
ノルウェー語	n
ポーランド語	p
ポルトガル語	v
ロシア語	r
簡体字中国語	c
スロベニア語	l
スペイン語	z
スウェーデン語	s
繁体字中国語	t
トルコ語	m

2. DB2 ブックには含められなかった最新の情報が、「リリース情報」で HTML 形式および ASCII ファイルとして利用できます。 HTML 版は、インフォメーション・センターおよび製品 CD-ROM からご利用になれます。 ASCII ファイルの参照方法:
   - UNIX ベースのプラットフォームでは、ファイル Release.Notes を参照してください。このファイルは DB2DIR/Readme/*%L* ディレクトリーにあります。ここで *%L* は地域名を、 DB2DIR は以下のものを表します。
     - /usr/lpp/db2_07_01 (AIX の場合)
     - /opt/IBMdb2/V7.1 (HP-UX、DYNIX/ptx、Solaris、および Silicon Graphics IRIX の場合)
     - /usr/IBMdb2/V7.1 (Linux の場合)
   - これ以外のプラットフォームでは、ファイル RELEASE.TXT を参照してください。このファイルは、製品がインストールされているディレクトリーにあります。 OS/2 プラットフォームでは、 **IBM DB2** フォルダーをダブルクリックし、 **Release Notes** アイコンをダブルクリックすることもできます。

PDF 資料の印刷

資料のハードコピー版が必要な場合、 DB2 の資料 CD-ROM にある PDF ファイル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することができます。ライブラリー内の各資料のファイルについては、927ページの表217 を参照してください。

Adobe Acrobat Reader の最新版は、Adobe の Web サイト http://www.adobe.com から入手できます。

PDF ファイルは、DB2 の資料 CD-ROM に収録されており、ファイル拡張子 PDF が付いています。 PDF ファイルにアクセスするには以下のようにします。

1. DB2 の資料 CD-ROM を挿入します。 UNIX ベースのプラットフォームの場合は、 DB2 資料 CD-ROM をマウントします。マウントの手順については、*概説およびインストール* を参照してください。
2. Acrobat Reader を起動します。
3. 以下に示すいずれかの位置から必要な PDF ファイルを開きます。
   - OS/2 および Windows プラットフォームでは:

     x:¥doc¥*language* ディレクトリー。ここで、*x* は CD-ROM ドライブを、*language* は 2 桁の言語を表す国コード (たとえば、EN は英語) を示します。
   - UNIX ベースのプラットフォームでは:

     CD-ROM の */cdrom*/doc/*%L* ディレクトリー。ここで、*/cdrom* は CD-ROM のマウント・ポイントを、 *%L* は地域名を表します。

さらに、PDF ファイルを CD-ROM からローカル・ドライブまたはネットワーク・ドライブにコピーし、そこから参照することもできます。

印刷資料の注文方法

ハードコピー版の DB2 ブックは、個別に注文することができます。資料を注文するには、IBM 承認の販売業者または営業担当員に連絡してください。

オンライン・ヘルプへのアクセス

すべての DB2 構成要素で、オンライン・ヘルプを利用できます。以下の表に、さまざまな種類のヘルプを示します。

ヘルプの種類	内容	利用方法
コマンド・ヘルプ	コマンド行プロセッサーのコマンド構文について説明します。	コマンド行プロセッサーの対話モードから、次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 コマンドのヘルプが表示されます。
クライアント構成アシスタントのヘルプ *コマンド・センターのヘルプ* *コントロール・センターのヘルプ* *データウェアハウスセンターのヘルプ* *イベント・アナライザーのヘルプ* *情報カタログ・マネージャーのヘルプ* *サテライト管理センターのヘルプ* *スクリプト・センターのヘルプ*	そのウィンドウまたはノートブックで実行できるタスクについて説明します。このヘルプは、知っておく必要のある概説および前提条件に関する情報を含みます。また、ウィンドウやノートブックの制御の使用方法を示します。	ウィンドウまたはノートブックから、**「ヘルプ (Help)」**押しボタンをクリックするか、または **F1** キーを押します。

ヘルプの種類	内容	利用方法

メッセージ・ヘルプ	メッセージの原因、および取るべき処置を説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。
		`? XXXnnnnn`
		ここで、*XXXnnnnn* は有効なメッセージ識別子を表します。
		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。
		一度に 1 画面分のメッセージ・ヘルプを表示させるには、次のように入力します。
		`? XXXnnnnn | more`
		メッセージ・ヘルプをファイルに保管するには、次のように入力します。
		`? XXXnnnnn > filename.ext`
		ここで、*filename.ext* はメッセージ・ヘルプを保管するファイルを表します。

SQL ヘルプ	SQL ステートメントの構文について説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。 `help statement` ここで、*statement* は SQL ステートメントを表します。 たとえば、help SELECT と入力すると、 SELECT ステートメントのヘルプが表示されます。 **注:** UNIX ベースのプラットフォームでは、SQL ヘルプを利用できません。
SQLSTATE ヘルプ	SQL 状態およびクラス・コードについて説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。 `? sqlstate or ? 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` によって SQL 状態 08003 のヘルプが表示され、 `? 08` によってクラス・コード 08 のヘルプが表示されます。

オンライン情報の表示

この製品に付属のブックは、ハイパーテキスト・マークアップ言語 (HTML) ソフトコピー形式です。ソフトコピー形式では情報を検索または表示したり、ハイパーテキスト・リンクを利用して関連情報に移動したりすることができます。また、1 つの端末を超えてライブラリーを容易に共用することができます。

オンライン・ブックやサンプル・プログラムは、 HTML バージョン 3.2 仕様に準拠するすべてのブラウザーを使って表示できます。

オンライン・ブックまたはサンプル・プログラムは、次のようにして表示します。

- DB2 管理ツールを実行している場合、インフォメーション・センターを使用します。
- ブラウザーで、**ファイル (File) —>ページを開く (Open Page)** をクリックします。次のようなページを開いて、DB2 情報に関する説明とリンクを表示してください。
  - UNIX ベースのプラットフォームでは、以下のページを開きます。

    INSTHOME/sqllib/doc/*%L*/html/index.htm

    ここで *%L* はロケール名です。
  - その他のプラットフォームでは、以下のページを開きます。

    sqllib¥doc¥html¥index.htm

    パスは DB2 がインストールされているドライブです。

    インフォメーション・センターをインストールしていない場合、 **DB2 Information** アイコンをダブルクリックしてページを開くことができます。このアイコンは、ご使用のシステムに応じて、製品のメイン・フォルダー内または Windows 「スタート」メニューにあります。

Netscape ブラウザーのインストール

システムに Web ブラウザーがインストールされていない場合、製品の箱の中にある Netscape CD-ROM から Netscape をインストールすることができます。インストールに関する詳細な説明については、以下を参照してください。

1. Netscape CD-ROM を挿入します。
2. UNIX ベースのプラットフォームでは、CD-ROM をマウントします。マウントの手順については、*概説およびインストール* を参照してください。

3. インストールの手順については、 CDNAV*nn*.txt ファイルを参照します。ここで、*nn* は 2 桁の言語識別子を表します。ファイルは CD-ROM のルート・ディレクトリーにあります。

インフォメーション・センターを使用した情報へのアクセス

インフォメーション・センターを使用すると、DB2 製品情報にすばやくアクセスすることができます。インフォメーション・センターは、DB2 管理ツールを使用できるすべてのプラットフォームで利用できます。

インフォメーション・センターは「インフォメーション・センター (Information Center)」アイコンをダブルクリックすることによってオープンできます。このアイコンのある場所はシステムによって異なります。メイン・プロダクト・フォルダーか Windows の**「スタート」**メニューのどちらかです。

Windows プラットフォームの DB2 では、ツールバーおよび**ヘルプ**・メニューを使用して、インフォメーション・センターにアクセスすることもできます。

インフォメーション・センターは 6 種類の情報を提供します。適切なタブをクリックすると、種類ごとに提供されているトピックが表示されます。

タスク (Tasks)
DB2 を使用して実行できる主要なタスク。

参照 (Reference)
DB2 参照情報 (キーワード、コマンド、API など)。

ブック (Books)
DB2 ブック。

トラブルシューティング (Troubleshooting)
エラー・メッセージのカテゴリーと、メッセージに対する回復処置。

サンプル・プログラム (Sample Programs)
DB2 アプリケーション開発クライアントに付属のサンプル・プログラム。 DB2 アプリケーション開発クライアントをインストールしていない場合、このタブは表示されません。

Web
WWW 上にある DB2 情報。この情報にアクセスするには、ご使用のシステムから Web への接続が必要です。

リストから項目を 1 つ選択すると、インフォメーション・センターはビューアーを立ち上げて情報を表示します。選択した情報の種類に応じて、ビューアーはシステム・ヘルプ・ビューアー、エディター、または Web ブラウザーです。

インフォメーション・センターには検索機能が備わっており、リストを参照せずに特定のトピックを探すことができます。

テキストの全検索を行うには、インフォメーション・センター内のハイパーテキスト・リンク「**DB2 オンライン情報の検索 (Search DB2 Online Information)**」検索フォームに従います。

通常、HTML 検索サーバーは自動的に始動します。 HTML 情報の検索がうまくいかない場合は、以下の方法の 1 つを使用して、検索サーバーを始動しなければならない場合もあります。

Windows では

「スタート」をクリックし、**「プログラム」—>「IBM DB2」—>「Information」—>「Start HTML Search Server」**を選択します。

OS/2 では

「DB2 (OS/2 版)」フォルダーをダブルクリックして、**「Start HTML Search Server」**アイコンをダブルクリックします。

HTML 情報の検索でこの他の問題が発生した場合は、リリース情報を参照してください。

注: 検索機能は、Linux、DYNIX/ptx、および Silicon Graphics IRIX 環境では利用できません。

DB2 ウィザードの使用

ウィザードを使用すると、各タスクをステップごとに進めることによって、さまざまな管理タスクを遂行することができます。ウィザードは、コントロール・センターおよびクライアント構成アシスタントを通して使用できます。以下の表では、ウィザードとその目的をリストしています。

注: データベース作成、索引作成、複数サイト更新の構成、およびパフォーマンス構成ウィザードは、区分データベース環境で使用できます。

ウィザード	内容	利用方法
データベース追加 (Add Database)	クライアント・ワークステーション上にデータベースのカタログを作成します。	クライアント構成アシスタントから、「**追加 (Add)**」をクリックします。

ウィザード	内容	利用方法
データベース・バックアップ *(Back up Database)*	バックアップ計画を決定、作成、およびスケジュールします。	「コントロール・センター (Control Center)」からバックアップするデータベースを右クリックし、**「バックアップ (Backup)」—>「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。
複数サイト更新の構成 (Configure Multisite Update)	複数サイト更新、分散トランザクション、または 2 フェーズ・コミットを構成します。	「コントロール・センター (Control Center)」から、**「データベース (Databases)」**フォルダーを右クリックして、**「複数サイト更新 (Multisite Update)」**を選択します。
データベース作成 (Create Database)	データベースを作成し、いくつかの基本的な構成タスクを実行します。	「コントロール・センター (Control Center)」から、**「データベース (Databases)」**フォルダーを右クリックして、**「作成 (Create)」—> 「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。
表作成 (Create Table)	基本的なデータ・タイプを選択して、表の基本キーを作成します。	「コントロール・センター (Control Center)」から、**「表 (Tables)」**アイコンを右クリックして、**「作成 (Create)」—>「ウィザードを使用する表 (Table Using Wizard)」**を選択します。
表スペース作成 (Create Table Space)	新しい表スペースを作成します。	「コントロール・センター (Control Center)」から、**「表スペース (Table Spaces)」**アイコンを右クリックして、**「作成 (Create)」—> 「ウィザードを使用する表スペース (Table Space Using Wizard)」**を選択します。
索引作成 (Create Index)	すべての照会について、作成すべき索引および除去すべき索引を提案します。	「コントロール・センター (Control Center)」から、**「索引 (Index)」**アイコンを右クリックして、**「作成 (Create)」—>「ウィザードを使用する索引 (Index Using Wizard)」**を選択します。

ウィザード	内容	利用方法
パフォーマンス構成 (Performance Configuration)	ビジネス要件に適合するように構成パラメーターを更新して、データベースのパフォーマンスを調整します。	「コントロール・センター (Control Center)」から、調整したいデータベースを右クリックして、**「ウィザードを使用するパフォーマンスの構成 (Configure Performance Using Wizard)」**を選択します。 区分データベース環境では、「Database Partitions」視点から、調整したい最初のデータベース区画を右クリックして、**「ウィザードを使用するパフォーマンスの構成 (Configure Performance Using Wizard)」**を選択します。
データベース復元 (Restore Database)	障害の後、データベースを回復します。どのバックアップを使用し、どのログを再生するかの判別を支援します。	「コントロール・センター (Control Center)」から復元するデータベースを右クリックし、**「復元 (Restore)」—>「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。

文書サーバーのセットアップ

デフォルトでは、DB2 情報はローカル・システムにインストールされます。つまり、DB2 情報にアクセスする必要のある各担当者が同じファイルをインストールする必要があります。 DB2 情報を 1 か所に格納するには、次のようにします。

1. ¥sqllib¥doc¥html のすべてのファイルとサブディレクトリーを、ローカル・システムから Web サーバーにコピーします。各ブックには独自のサブディレクトリーがあり、そのブックを構成する必要な HTML および GIF ファイルが入っています。ディレクトリー構造は常に同じ状態に保つ必要があります。
2. Web サーバーを構成して、ファイルを新しい場所で検索するようにします。さらに詳しい情報については、 *インストールおよび構成 補足* の NetQuestion 付録を参照してください。
3. インフォメーション・センターの Java バージョンをご使用の場合は、すべての HTML ファイルのベース URL を指定できます。この URL はブックのリストに使用してください。

4. 資料ファイルが表示されるようになったなら、よく使うトピックにはブックマークを付けておいてください。ブックマークを付けるページは、たとえば以下のものがあります。
   - ブックのリスト
   - 頻繁に使用されるブックの目次
   - 頻繁に参照する情報 (たとえば、ALTER TABLE トピックなど)
   - 検索フォーム

中央のマシンから DB2 ユニバーサル・データベース オンライン文書ファイルを提供する方法については、 *インストールおよび構成 補足* の NetQuestion 付録を参照してください。

オンライン情報の検索

HTML ファイルの情報を検索するには、以下の方法のどれか 1 つを使用してください。

- 最上部にある**「検索 (Search)」**をクリックします。検索フォームを使用して特定のトピックを見つけます。この機能は、 Linux、DYNIX/ptx、または Silicon Graphics IRIX 環境ではご利用になれません。
- 最上部にある**「索引 (Index)」**をクリックします。索引を使用して、ブック内の特定のトピックを見つけます。
- HTML 資料またはヘルプの目次あるいは索引を表示してから、 Web ブラウザーの検索機能を利用して資料内の特定のトピックを見つけます。
- Web ブラウザーのブックマーク機能を使用して、特定のトピックにすばやく戻ります。
- インフォメーション・センターの検索機能を使用して、特定のトピックを検索します。詳しくは、941ページの『インフォメーション・センターを使用した情報へのアクセス』を参照してください。

付録M. 特記事項

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミングまたはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミングまたはサービスを、日本で発表する意図があることを必ずしも示すものではありません。本書で、 IBM ライセンス・プログラムまたは他の IBM 製品に言及している部分があっても、このことは当該プログラムまたは製品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。これらのプログラムまたは製品に代えて、 IBM の知的所有権を侵害することのない機能的に同等な他社のプログラム、製品またはサービスを使用することができます。ただし、 IBM によって明示的に指定されたものを除き、これらのプログラムまたは製品に関連する稼働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 および他社は、本書で説明する主題に関する特許権 (特許出願を含む)、商標権、または著作権を所有している場合があります。本書は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用権等を許諾することを意味するものではありません。実施権、使用権等の許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
AP 事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

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
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

本プログラムに関する上記の情報は、適切な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が、これは説明に具体性を与えるために記載されたものであり、それらの例には、個人、企業、ブランドの、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。それらの名前はすべて架空のものであり、また名称や住所が類似する企業が実在しても、それは偶然に過ぎません。

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。該当するテキストは、許可のもとに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。該当するテキストは、許可のもとに下記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

次のものは、IBM Corporation の米国およびその他の国における商標です。

ACF/VTAM
ADSTAR
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
Extended Services
FFST
First Failure Support Technology
IBM
IMS
LAN Distance
MVS/ESA
MVS/XA
OS/400
OS/390
OS/2
PowerPC
QMF
RACF
RISC System/6000
SP
SQL/DS
SQL/400
S/370
System370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WIN-OS/2

他社の商標

次のものは、他の商標または登録商標です。

C-bus は Corollary Inc. 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。

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。

PC Direct は Ziff Communications Company の米国およびその他の国における商標であり、 IBM Corporation がライセンスを得て使用しています。

ActionMedia、 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。

UNIX は、 The Open Group がライセンスしている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等はそれぞれ各社の商標または登録商標です。

参考文献

- *DB2 SQL 概説 バージョン 7* (SC88-8539)
- *ODBC 2.0 Programmer's Reference and SDK Guide* (ISBN: 1-55615-658-8)
- *ODBC 3.0 Software Development Kit and Programmer's Reference* (ISBN: 1-57231-516-4)
- 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).
- *DB2 SQL 解説書 第 1 巻と第 2 巻* (SC88-8540)
- *DB2 (MVS/ESA 版) SQL 解説書* (SC88-7381-00)
- *SQL/DS SQL 解説書 (VM, VSE)* (SH88-7024-01)
- *DB2 for OS/400 SQL Reference Version 3* (SC41-9608-00)

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]

[ヤ行]

[ラ行]

[ワ行]

[数字]

A

B

C

D

P

Q

R

S

T

U

V

W

X

Y

［特殊文字］

IBM と連絡をとる

技術上の問題がある場合は、時間をとって「問題判別の手引き」に定義されている処置を検討し、それらの提案を実行した後で、DB2 顧客サービスに連絡をとってください。この資料には、DB2 顧客サービスがお客さまを支援するために必要とする情報が説明されています。

製品情報

以下の情報は英語で提供されます。内容は英語版製品に関する情報です。

http://www.ibm.com/software/data/

DB2 World Wide Web ページには、ニュース、製品説明、研修スケジュールなどの DB2 に関する最新情報が提供されています。ただし、提供されている情報は英語です。

http://www.ibm.com/software/data/db2/library/

「DB2 Product and Service Technical Library」では、よくされる質問 (FAQ)、修正内容、資料、および最新の DB2 技術情報などの情報へのアクセスが提供されています。

注: この情報のご提供は英語のみとなりますのでご注意ください。

http://www.elink.ibmlink.ibm.com/pbl/pbl/

「International Publications」注文用 Web サイトでは、マニュアルの注文方法についての情報を提供しています。ただし、提供されている情報は英語です。

http://www.ibm.com/education/certify/

IBM の「Professional Certification Program」Web サイトでは、 DB2 を含むさまざまな IBM 製品の認証テストの情報を提供しています。ただし、提供されている情報は英語です。

ftp.software.ibm.com

匿名でログオンしてください。ディレクトリー `/ps/products/db2` には、 DB2 および多数の他製品に関連したデモ、修正プログラム、情報、およびツールがあります。ただし、提供されている情報は英語です。

comp.databases.ibm-db2, bit.listserv.db2-l

これらのインターネット・ニュースグループは、ユーザーが DB2 製品に関する自分の経験について話し合うために利用できます。ただし、提供されている情報は英語です。

Compuserve: GO IBMDB2

このコマンドを入力すると、IBM DB2 Family forum にアクセスできます。すべての DB2 製品が、このフォーラムでサポートされています。ただし、提供されている情報は英語です。

米国以外の国で IBM に連絡する方法については、 *IBM Software Support Handbook* の Appendix A を参照してください。この資料にアクセスするには、 Web ページ: http://www.ibm.com/support/ にアクセスし、ページの最下部にある「IBM Software Support Handbook」リンク・ボタンを選択します。

注: 国によっては、 IBM が承認している販売業者が、 IBM サポート・センターの代わりにそれら販売業者のサポート・センターに連絡する場合があります。

Printed in Japan

SC88-8517-00

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