IBM® DB2 Universal Database™

コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻

バージョン 8.2

SC88-9159-01
(英文原典：SC09-4849-01)

IBM® DB2 Universal Database™

コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻

バージョン 8.2

SC88-9159-01
(英文原典：SC09-4849-01)

ご注意！

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

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

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

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

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

(URL は、変更になる場合があります)

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

原　典： SC09-4849-01
IBM® DB2 Universal Database™
Call Level Interface Guide and Reference, Volume 1
Version 8.2

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2004.8

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

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

目次

第 5 部 データ変換 379

第 26 章 データ変換 381

第 6 部 付録 397

付録 A. DB2 Universal Database の技術情報の概要 399

第 1 部 CLI の背景情報

第 1 章 CLI の紹介

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、リレーショナル・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の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、 DB2 CLI は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 Driver Manager 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。これは ODBC 3.51 に準拠しています。

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。

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 ドライバーがランタイムに動的にロードされるオペレーティング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ランタイムに仲介して、それが該当する DBMS 特定の ODBC ドライバーに確実に仕向けられるようにします。 ODBC Driver Manager 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、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 Driver Manager 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入手してください。 Windows® プラットフォームの場合、 ODBC SDK は Microsoft Data Access Components (MDAC) SDK の一部として入手でき、 http://www.microsoft.com/data/ からダウンロードして使用できます。 Windows 以外のプラットフォームの場合、ODBC SDK は他のベンダーによって提供されます。 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、Microsoft 社から入手できる ODBC Programmer's Reference and SDK Guide を併用してください。

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。

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
- コンパウンド SQL
- ユーザー定義タイプ (UDT)
- ユーザー定義関数 (UDF)

DB2 CLI の更新の詳細は、以下の DB2® アプリケーション開発の Web サイトを参照してください。

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

関連概念:

- 11 ページの『DB2 CLI と Microsoft ODBC の比較』
- 5 ページの『DB2 コール・レベル・インターフェース (CLI) と組み込み動的 SQL の比較』
- 6 ページの『組み込み SQL と比較した DB2 CLI の利点』

• 8 ページの『DB2 CLI または組み込み SQL をいつ使用するか』

DB2 コール・レベル・インターフェース (CLI) と組み込み動的 SQL の比較

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ランタイムに 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移植可能な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 (ホスト・システム・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なります。ただ該当するデータベースにランタイムに接続するだけで済むようになります。

DB2 CLI と組み込み SQL の相違点と類似点を次に示します。

- 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、および位置指定 `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、普通、 SQL ステートメントとして実行されるのではなく、 `SQLEndTran()` 関数呼び出しによって発行されますが、その実行は許可されています。
- 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・ハンドルと呼ばれる情報を表す抽象オブジェクトを提供します。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* は、グローバル変数および接続特有の情報を参照する方法を提供します。*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 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、CLI アプリケーションは、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。

- 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® リレーショナル・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。)

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 はコンパウンド 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。

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直接 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、DB2 CLI により代行受信されて、適切な `SQLEndTran()` 要求として処理されます。しかし、`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ことが推奨されています。

関連資料:

- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『サポートされる SQL ステートメント』

組み込み SQL と比較した DB2 CLI 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ターゲット・データベースは不明です。アプリケーションがどのデータベース・サーバーに接続するかに関係なく、 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プリプロセス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ランタイム・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。

- 一般に、組み込み SQL アプリケーションには、アプリケーションが制御する複合データ域 (SQLDA および SQLCA など) が関連付けられており、それらはしばしば複雑になることがあります。しかし、DB2 CLI にはデータ域が必要ではありません。その代わりに、 DB2 CLI が、必要なデータ構造を割り当てて制御し、それらを参照するためのアプリケーションのハンドル を提供します。
- DB2 CLI では、マルチ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有効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、マルチスレッドの 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、主キーなど) 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、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あまり知らないプログラマーにとって、直感的に理解できるものです。
- DB2 CLI には、 DB2 Universal Database (つまり DB2 Universal Database for z/OS and OS/390 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 Universal Database のクライアントのために用意されているものです。
- DB2 CLI はスクロール可能カーソルをさらに強力にサポートします。スクロール可能カーソルを使用すると、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順方向へ
  - 1 行または複数行ごとに逆方向へ
  - 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

スクロール可能カーソルは、配列出力と組み合わせて使用できます。更新可能カーソルをスクロール可能と宣言すれば、 1 行または複数行ごとに結果セット内を順方向または逆方向に移動できます。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 または組み込み SQL をいつ使用するか

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 の方が明らかにどのよう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 を使用する方がはるかに簡単です。

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 はランタイムに準備され、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ランタイムに追加のネットワーク通信量が生じることがあります。 DB2 CLI アプリケーションが (デフォルト振る舞いの) 据え置き準備を使用する場合、追加ネットワーク・トラフィックを避けられます。

  静的 SQL が動的 SQL よりパフォーマンスが常に高いわけではない、というのも重要なことです。動的 SQL は実行時に準備され、その時点で使用可能なデータベース統計を使用しますが、静的 SQL は BIND 時に使用可能なデータベース統計を使用します。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して最適のアクセス・プランを選択することができるので、同じ SQL を静的 SQL として実行するよりも潜在的に良好なパフォーマンスが達成されます。さらに、キャッシュされていれば、動的 SQL ステートメントのプリコンパイルを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 (表やビューなど) に対するアクセス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、権限はランタイム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認可されます。

- 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。

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セキュリティー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。

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。

関連概念:

- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連タスク:

- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 27 ページの『CLI アプリケーションでの SQL ステートメントの発行』
- 205 ページの『CLI/ODBC/JDBC 静的プロファイル作成による静的 SQL の作成』

第 2 章 DB2 CLI と ODBC

DB2 CLI と Microsoft ODBC の比較

このトピックでは、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 図 1 では、DB2 CLI と DB2 ODBC ドライバーを比較しています。左側は、ODBC Driver Manager の下の ODBC ドライバーを示し、右側は、 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である DB2 CLI を示します。

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 Universal Database 製品を指します。

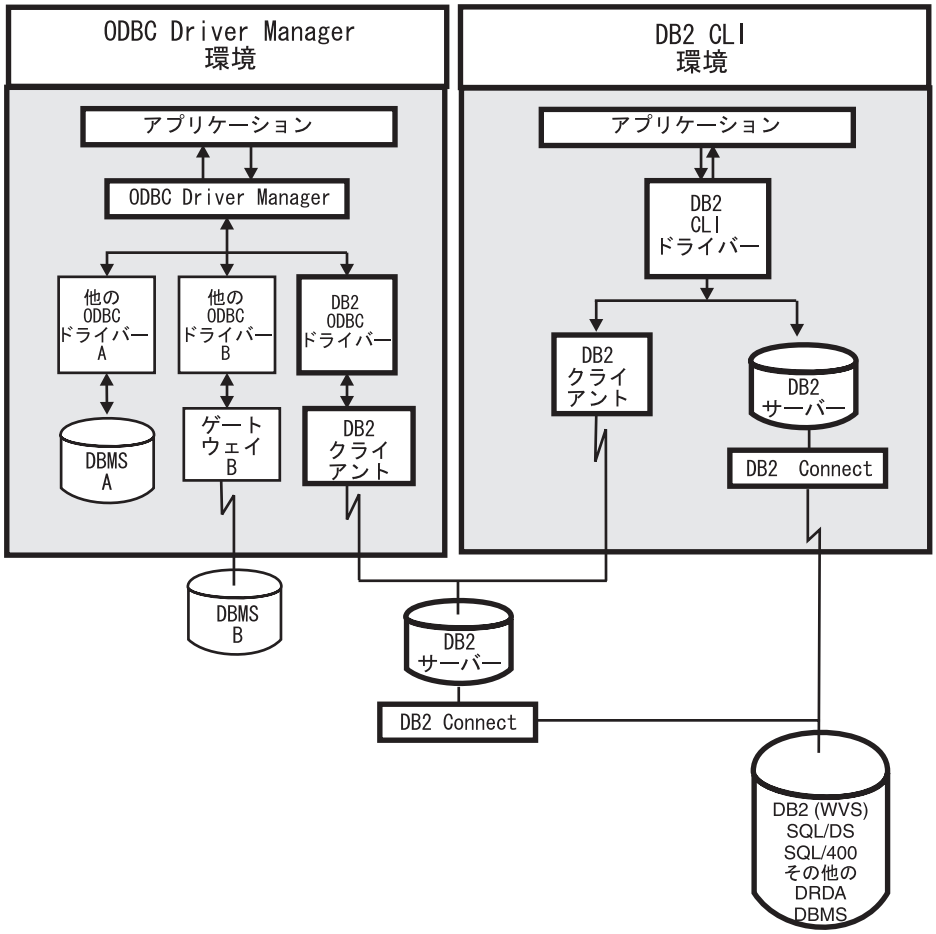


図 1. DB2 CLI と ODBC：

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 3.51 に準拠しています。

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入手してください。 Windows® プラットフォームの場合、 Microsoft® Data Access Components (MDAC) SDK の一部として、 ODBC SDK を使用することができます。これは、http://www.microsoft.com/data/ からダウンロードできます。 Windows 以外のプラットフォームの場合、ODBC SDK は他のベンダーによって提供されます。

ODBC Driver Manager 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 1 には 2 つのレベルのサポートがサマリーされており、また、CLI および ODBC 関数のサマリーには ODBC 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 1. 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。	• SQLSetConnectAttr() • SQLGetEnvAttr() • SQLSetEnvAttr() • SQLSetColAttributes() • SQLGetSQLCA() • SQLBindFileToCol() • SQLBindFileToParam() • SQLExtendedBind() • SQLExtendedPrepare() • SQLGetLength() • SQLGetPosition() • SQLGetSubString()

表 1. 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_BIGINT • SQL_BINARY • SQL_BIT • SQL_BLOB • SQL_BLOB_LOCATOR • SQL_CHAR • SQL_CLOB • SQL_CLOB_LOCATOR • SQL_DBCLOB • SQL_DBCLOB_LOCATOR • SQL_DECIMAL • SQL_DOUBLE • SQL_FLOAT • SQL_GRAPHIC • SQL_INTEGER • SQL_LONG • SQL_LONGVARBINARY • SQL_LONGVARCHAR • SQL_LONGVARGRAPHIC • SQL_NUMERIC • SQL_REAL • SQL_SHORT • SQL_SMALLINT • SQL_TINYINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_VARBINARY • SQL_VARCHAR • SQL_VARGRAPHIC • SQL_WCHAR

表 1. 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 ** • SQL_C_WCHAR **** Windows プラットフォーム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

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 2. 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 (Uncommitted read)	SQL_TXN_READ_UNCOMMITTED

表 2. ODBC での分離レベル (続き)

IBM 分離レベル	ODBC 分離レベル
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、 `SQLSetConnectAttr()` および `SQLSetStmtAttr()` は、 **HY**009 の SQLSTATE で SQL_ERROR を戻します。	

制限:

1 つのアプリケーションの中での ODBC の機能と DB2 CLI の機能を混在させることは、 Windows 64 ビット・オペレーティング・システムではサポートされていません。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『分離レベル』
- 3 ページの『CLI の紹介』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『CLI と ODBC 関数のサマリー』
- 56 ページの『CLI 関数戻りコード』

第 2 部 CLI アプリケーションのプログラミング

第 3 章 基本 CLI アプリケーションの作成

CLI アプリケーションには、初期化、トランザクション処理、および終了という 3 つのコア・コンポーネントがあります。この章では、典型的な CLI アプリケーションの重要なステップについて説明します。

初期設定

CLI でのハンドル

CLI ハンドルとは、DB2 CLI によって割り振られて管理されるデータ・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 (SQLDA など) を割り振り、管理する必要がなくなります。

CLI では、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 (属性や有効な接続など) が入っているデータ・オブジェクトを指します。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・オブジェクトを指します。そのような情報の例としては、接続に関する有効なステートメントと記述子ハンドル、トランザクション状況、および診断情報があります。

アプリケーションは、同時に複数のデータ・ソースに接続でき、同じデータ・ソースに複数の別個の接続を確立することもできます。並行した接続ご

とに、別個の接続ハンドルを割り振る必要があります。ステートメントまたは記述子ハンドルを割り振るためには、その前に接続ハンドルを割り振っておく必要があります。

接続ハンドルを使用すると、スレッドごとに 1 つの接続を利用するマルチ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。

注: 環境ハンドルごとに 512 個のアクティブ接続という制限があります。

ステートメント・ハンドル

ステートメント・ハンドルは、 1 つの SQL ステートメントの実行を追跡するのに使用されるデータ・オブジェクトを指します。これにより、エラー・メッセージのようなステートメント情報、関連付けられたカーソル名、および SQL ステートメント処理の状況情報が使用できるようになります。ステートメント・ハンドルは、SQL ステートメントを発行する前に割り振らなければなりません。

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 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、および SQL_ATTR_IMP_PARAM_DESC ステートメント属性に割り当てます。アプリケーション記述子は、記述子ハンドルを割り振ることによって、明示的に割り振ることができます。

CLI アプリケーションで使用できるステートメント・ハンドルの数は、アプリケーションが定義したラージ・パッケージによって異なり、システム・リソース全体によって制限されます (通常は、スタック・サイズ)。デフォルトでは、3 つのスモール・パッケージと 3 つのラージ・パッケージが存在します。各スモール・パッケージでは、1 つの接続につき最大で 64 のステートメント・ハンドルが許可されており、各ラージ・パッケージでは、1 つの接続につき最大で 384 のステートメント・ハンドルが許可されています。したがって、デフォルトで使用できるステートメント・ハンドル数は、 (3 * 64) + (3 * 384) = 1344 ということになります。

デフォルトの 1344 のステートメント・ハンドルよりも多くを獲得するには、 CLI/ODBC 構成キーワード CLIPkg の値を 30 までの値に設定することにより、ラージ・パッケージの数を増やします。 CLIPkg は、生成されるラージ・パッケージの数を示します。 CLIPkg を最大値の 30 に設定すると、使用できるステートメント・ハンドルの最大数は、(3 * 64) + (30 * 384) = 11,712 になります。

この制限を超過する場合には、`SQLPrepare()`、`SQLExecute()`、または `SQLExecDirect()` への呼び出しに対し、 HY014 SQLSTATE が戻される可能性があります。

パッケージはデータベースでスペースをとるため、ご使用のアプリケーションで実行する必要のあるラージ・パッケージ数だけ割り振るようお勧めします。

記述子ハンドル

記述子ハンドルは、結果セットに列についての情報が入っていて、 SQL ステートメントに動的パラメーターについての情報が入っているデータ・オブジェクトを指します。

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 リソース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。

関連概念:

- 169 ページの『CLI アプリケーションの記述子』
- 44 ページの『CLI アプリケーションでのハンドルの解放』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 26 ページの『CLI アプリケーションでのステートメント・ハンドルの割り振り』
- 43 ページの『CLI アプリケーションでのステートメント・リソースの解放』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecute 関数 (CLI) - ステートメントの実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLPrepare 関数 (CLI) - ステートメントの準備』
- 「*SQL リファレンス 第 1 巻*」の『SQLDA (SQL 記述子域)』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 227 ページの『DB2 CLI のバインド・ファイルおよびパッケージ名』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 370 ページの『TraceRefreshInterval CLI/ODBC 構成キーワード』
- 305 ページの『CLIPkg CLI/ODBC 構成キーワード』

CLI での初期化と終了の概説

22 ページの図 2 は、初期化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、 CLI でのトランザクション処理の概説 に示してあります。

初期化タスクは、環境ハンドルおよび接続ハンドルの割り振りと初期化から構成されます。接続ハンドルを作成するには、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。接続ハンドルの作成後に、アプリケーションは接続を確立できま

す。接続が存在する場合は、アプリケーションはトランザクション処理タスクに進むことができます。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。

終了タスクは、データ・ソースからの切断と、初期化フェーズで割り振られたハンドルの解放とによって構成されます。環境ハンドルを解放する前に、接続ハンドルを解放する必要があります。

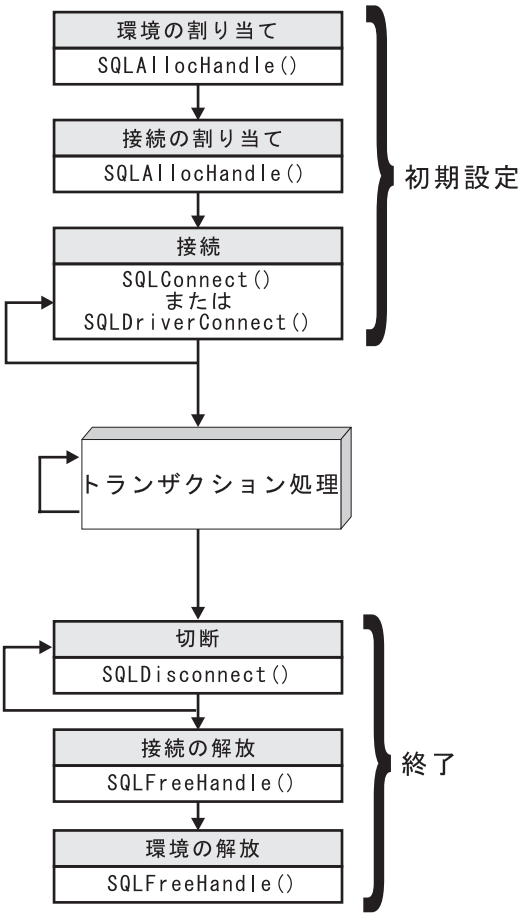


図 2. 初期化タスクと終了タスクの概念説明

関連概念:

- 19 ページの『CLI でのハンドル』
- 25 ページの『CLI でのトランザクション処理の概説』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLConnect 関数 (CLI) - データ・ソースへの接続』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDisconnect 関数 (CLI) - データ・ソースからの切断』

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDriverConnect 関数 (CLI) - データ・ソースへの (拡張) 接続』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』

CLI アプリケーションの初期設定

CLI アプリケーションを初期設定することは、CLI を使用した膨大なプログラミング作業の一部です。 CLI アプリケーションを初期設定する作業には、環境と接続ハンドルを割り振り、その後でデータ・ソースに接続することが関係します。

手順:

アプリケーションを初期設定するには、以下のようにします。

1. SQL_HANDLE_ENV の *HandleType* と SQL_NULL_HANDLE の *InputHandle* を指定した `SQLAllocHandle()` を呼び出して、環境ハンドルを割り振ります。例:

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

   これ以降に環境ハンドルが必要な呼び出しすべてに対して、 *OutputHandlePtr 引き数 (上記の例では henv) で戻された、割り振られた環境ハンドルを使用するようにします。
2. オプション: 設定する属性ごとに必要な環境属性を指定した `SQLSetEnvAttr()` を呼び出して、アプリケーションの環境属性を設定します。

   重要: アプリケーションを ODBC アプリケーションとして実行する予定の場合、 `SQLSetEnvAttr()` を使用して SQL_ATTR_ODBC_VERSION 環境属性を設定しなければなりません。厳密に DB2 CLI アプリケーションであるアプリケーションには、この属性を設定するようお勧めしますが、必須ではありません。
3. *InputHandle* 引き数としてステップ 1 で戻された環境ハンドルを使用し、SQL_HANDLE_DBC の *HandleType* を指定した `SQLAllocHandle()` を呼び出すことによって、接続ハンドルを割り振ります。例:

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

   これ以降に接続ハンドルが必要な呼び出しすべてに対して、 **OutputHandlePtr* 引き数 (上記の例では hdbc) で戻された、割り振られた接続ハンドルを使用するようにします。
4. オプション: 設定する属性ごとに必要な接続属性を指定した `SQLSetConnectAttr()` を呼び出して、アプリケーションの接続属性を設定します。
5. 接続先のデータ・ソースごとに、ステップ 3 で割り振った接続ハンドルを指定した以下のいずれかの関数を呼び出し、データ・ソースに接続します。
   - SQLConnect(): 基本データベース接続方式。例:

     ```
     SQLConnect (hdbc, server, SQL_NTS, user, SQL_NTS, password, SQL_NTS);
     ```

     ここで、SQL_NTS は、参照されるストリングが NULL 終了することを示す、特別なストリング長の値です。
   - SQLDriverConnect(): 別の接続オプションを許可し、グラフィカル・ユーザー・インターフェースをサポートする拡張された接続関数。例:

```
char * connStr = "DSN=SAMPLE;UID=;PWD=;";

SQLDriverConnect (hdbc, (SQLHWND)NULL, connStr, SQL_NTS,
                  NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
```

- SQLBrowseConnect(): データ・ソースへの接続のための属性および属性値を繰り返し戻す、あまり一般的ではない接続方式。例:

```
char * connInStr = "DSN=SAMPLE;UID=;PWD=;";
char outStr[512];

SQLBrowseConnect (hdbc, connInStr, SQL_NTS, outStr,
                  512, &strLen2Ptr);
```

これで、アプリケーションが初期設定されましたので、トランザクションの処理に進むことができます。

関連概念:

- 19 ページの『CLI でのハンドル』
- 25 ページの『CLI でのトランザクション処理の概説』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBrowseConnect 関数 (CLI) - データ・ソースへの接続に必要な属性の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLConnect 関数 (CLI) - データ・ソースへの接続』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDriverConnect 関数 (CLI) - データ・ソースへの (拡張) 接続』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetEnvAttr 関数 (CLI) - 環境属性の設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『環境属性 (CLI) リスト』

関連サンプル:

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

トランザクション処理

CLI でのトランザクション処理の概説

図 3 は、 DB2 CLI アプリケーションのトランザクション処理タスク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。

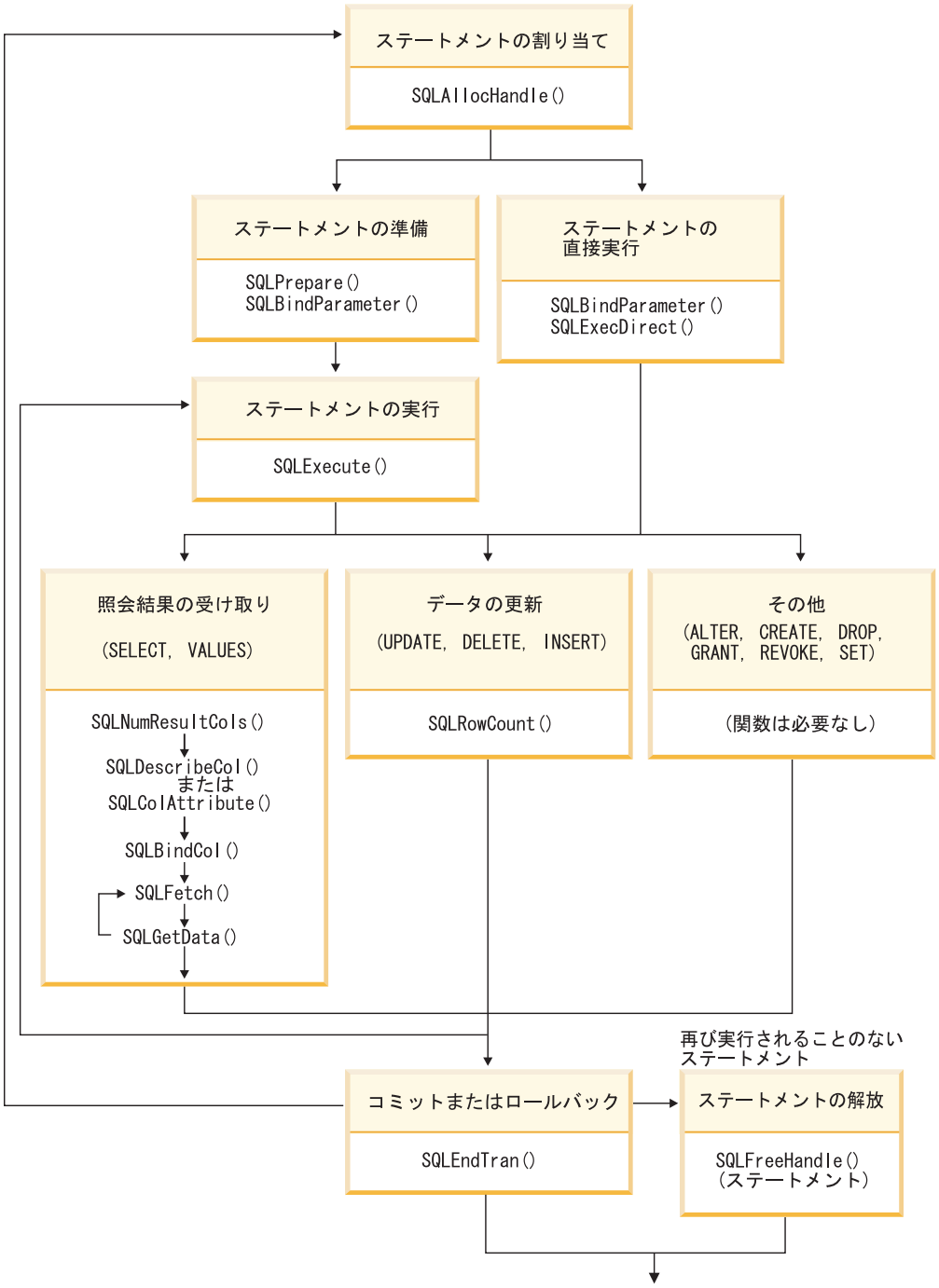


図 3. トランザクション処理

トランザクション処理タスクには、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および実行
- 結果の処理
- コミットまたはロールバック
- (オプション) ステートメントが再実行されそうにない場合のステートメント・ハンドルの解放

関連概念:

- 35 ページの『CLI アプリケーションのコミット・モード』

関連タスク:

- 26 ページの『CLI アプリケーションでのステートメント・ハンドルの割り振り』
- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 38 ページの『CLI アプリケーションでの照会結果の取り出し』
- 41 ページの『CLI アプリケーションでのデータの更新と削除』
- 43 ページの『CLI アプリケーションでのステートメント・リソースの解放』
- 59 ページの『CLI アプリケーションの終了』

CLI アプリケーションでのステートメント・ハンドルの割り振り

CLI アプリケーションで SQL ステートメントを発行するには、ステートメント・ハンドルを割り振る必要があります。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追跡するもので、接続ハンドルに関連付けられます。ステートメント・ハンドルを割り振ることは、より大きなトランザクションの処理作業の一部です。

前提条件:

ステートメント・ハンドルの割り振りを開始する前に、環境ハンドルと接続ハンドルを割り振る必要があります。これは、CLI アプリケーションを初期設定する作業の一部です。

手順:

ステートメント・ハンドルを割り振るには、以下のようにします。

1. SQL_HANDLE_STMT の *HandleType* を指定した `SQLAllocHandle()` を呼び出します。以下に例を示します。

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

2. オプション: このステートメントに属性を設定するには、必要な属性オプションごとに `SQLSetStmtAttr()` を呼び出します。

環境ハンドル、接続ハンドル、およびステートメント・ハンドルの割り振ると、SQL ステートメントを準備、発行、または実行できるようになります。

関連概念:

- 25 ページの『CLI でのトランザクション処理の概説』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 27 ページの『CLI アプリケーションでの SQL ステートメントの発行』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』

関連サンプル:

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

CLI アプリケーションでの SQL ステートメントの発行

SQL ステートメントは、SQLCHAR ストリング変数として DB2 CLI 関数に渡されます。この変数は、1 つ以上の SQL ステートメントで構成することができます。パラメーター・マーカーは、関係する処理のタイプに応じて指定されたりされなかったりします。このトピックでは、DB2 CLI アプリケーションで SQL ステートメントを発行するさまざまな方法を説明します。

SQL ステートメントを発行する前に、ステートメント・ハンドルを割り振っておく必要があります。

手順:

以下のいずれかのステップを実行して、SQL ステートメントを発行します。

- 1 つの SQL ステートメントを発行するには、その SQL ステートメントの SQLCHAR 変数を初期設定し、その変数を CLI 関数に渡すか、SQLCHAR * にキャストされたストリング引き数を直接関数に渡します。以下に例を示します。

  ```
  SQLCHAR * stmt = (SQLCHAR *) "SELECT deptname, location FROM org";
  /* ... */
  SQLExecDirect (hstmt, stmt, SQL_NTS);
  ```

  または

  ```
  SQLExecDirect (hstmt, (SQLCHAR *) "SELECT deptname, location FROM org",
                 SQL_NTS);
  ```

- 同じステートメント・ハンドルで複数の SQL ステートメントを発行するには、SQLCHAR エレメント (各エレメントは個々の SQL ステートメントを表す) の配列を初期設定するか、 ";" 文字で区切られた複数のステートメントを含む 1 つの SQLCHAR 変数を初期設定します。以下に例を示します。

  ```
  SQLCHAR * multiple_stmts[] = {
    (SQLCHAR *) "SELECT deptname, location FROM org",
    (SQLCHAR *) "SELECT id, name FROM staff WHERE years > 5",
    (SQLCHAR *) "INSERT INTO org VALUES (99,'Hudson',20,'Western','Seattle')"
  };
  ```

  または

  ```
  SQLCHAR * multiple_stmts =
    "SELECT deptname, location FROM org;
     SELECT id, name FROM staff WHERE years > 5;
     INSERT INTO org VALUES (99, 'Hudson', 20, 'Western', 'Seattle')";
  ```

注: SQL ステートメントのリストを指定する場合、一度に 1 つのステートメントだけが実行されます。この際には、リストの最初のステートメントから開始されます。その後に続く各ステートメントは、リストに示される順序で実行されます。 (後続のステートメントを実行するには、`SQLMoreResults()` を呼び出す必要があります。)

- パラメーター・マーカーを指定した SQL ステートメントを発行するには、『パラメーター・マーカーのバインド』を参照してください。
- DB2 CLI で動的に実行された SQL ステートメント (動的 SQL) を静的 SQL にキャプチャーして変換するには、『静的 SQL の作成』を参照してください。

関連概念:

- 31 ページの『CLI アプリケーションでのパラメーター・マーカー・バインディング』

関連タスク:

- 26 ページの『CLI アプリケーションでのステートメント・ハンドルの割り振り』

関連資料:

- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLMoreResults 関数 (CLI) - さらに結果セットがあるかどうかの判別』

関連サンプル:

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

CLI アプリケーションでの SQL ステートメントの準備と実行

ステートメント・ハンドルを割り振ったら、SQL ステートメントを使用して操作を実行できるようになります。 SQL ステートメントは、実行前に準備しておく必要があります。DB2 CLI では、 SQL ステートメントを準備して実行するための、以下の 2 つの方法が用意されています。

- 準備および実行操作を個別のステップで実行する
- 準備および実行操作を結合して 1 つのステップにする

前提条件:

SQL ステートメントを準備して実行する前に、そのステートメントのステートメント・ハンドルを割り振っておく必要があります。

手順:

個別のステップで SQL ステートメントを準備して実行するには、以下のようにします。

1. `SQLPrepare()` を呼び出し、*StatementText* 引き数として SQL ステートメントを渡すことにより、 SQL ステートメントを準備します。

2. `SQLBindParameter()` を呼び出して、 SQL ステートメントで使用する可能性のあるパラメーター・マーカーをすべてバインドします。
3. `SQLExecute()` を呼び出して、準備済みステートメントを実行します。

この方法は、以下の場合に使用します。

- 同じ SQL 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)。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。そうすることにより、ドライバーの効率が良くなると同時に、アプリケーションのパフォーマンスが良くなります。
- ステートメントの実行よりも前に、結果セットのパラメーターまたは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。

1 つのステップで SQL ステートメントを準備して実行するには、以下のようにします。

1. `SQLBindParameter()` を呼び出して、 SQL ステートメントで使用する可能性のあるパラメーター・マーカーをすべてバインドします。
2. *StatementText* 引き数として SQL ステートメントを指定した `SQLExecDirect()` を呼び出し、ステートメントを準備して実行します。
3. オプション: SQL ステートメントのリストを実行する予定の場合、`SQLMoreResults()` を呼び出して、次の SQL ステートメントに進みます。

1 つのステップで準備して実行するこの方法は、以下の場合に使用します。

- ステートメントが一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ことを回避します。
- 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。

関連概念:

- 30 ページの『CLI アプリケーションでの据え置き準備』
- 25 ページの『CLI でのトランザクション処理の概説』

関連タスク:

- 26 ページの『CLI アプリケーションでのステートメント・ハンドルの割り振り』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecute 関数 (CLI) - ステートメントの実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLMoreResults 関数 (CLI) - さらに結果セットがあるかどうかの判別』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*の『SQLPrepare 関数 (CLI) - ステートメントの準備』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』

関連サンプル:

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

CLI アプリケーションでの据え置き準備

据え置き準備 は、CLI 機能の名前であり、同じネットワーク・フローの中で、SQL ステートメントの準備要求と実行要求の両方を送信することにより、サーバーとの通信を最小化しようとするものです。このプロパティーのデフォルト値は、CLI/ODBC 構成キーワード DeferredPrepare を使用してオーバーライドできます。このプロパティーは、 `SQLSetStmtAttr()` を呼び出して SQL_ATTR_DEFEFERRED_PREPARE ステートメント属性を変更することにより、ステートメント・ハンドルごとに設定することができます。

据え置き準備がオンであれば、対応する実行要求が発行されるまで、準備要求はサーバーに送られません。その後、ネットワーク・フローを最小化しパフォーマンスを改善するため、 2 つの要求が 2 つではなく 1 つのコマンド/応答のフローに結合されます。この動作のため、`SQLPrepare()` によって一般に生成されるエラーは、実行時に発生します。さらに、`SQLPrepare()` は常に SQL_SUCCESS を戻します。据え置き準備が最大の利点となるのは、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。

注: 据え置き準備が使用可能な場合でも、操作の実行前にステートメントを準備しなければならない操作では、実行前に準備要求がサーバーに送信されます。記述情報は、ステートメントが準備された後にのみ使用可能になるため、`SQLDescribeParam()` または `SQLDescribeCol()` への呼び出しの結果として生じる記述操作は、据え置き準備がオーバーライドされる例と言えます。

関連タスク:

- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeParam 関数 (CLI) - パラメーター・マーカーの記述を戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLPrepare 関数 (CLI) - ステートメントの準備』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』

- 56 ページの『CLI 関数戻りコード』
- 321 ページの『DeferredPrepare CLI/ODBC 構成キーワード』

CLI アプリケーションでのパラメーター・マーカー・バインディング

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。 (静的組み込み SQL で、ホスト変数が使用される箇所では、パラメーター・マーカーが使用されます。) 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。

  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、`SQLBindParameter()` を使用します。
- 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()` 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。
- LOB 値を含むアプリケーションの環境内のファイル。

  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。

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、以下のものを示すいくつかの引き数を指定した `SQLBindParameter()` 関数を呼び出すことによって実行されます。

- パラメーターの順序を示す位置。
- パラメーターの SQL タイプ。
- パラメーターのタイプ (入力、出力、または入出力)。
- 変数の C データ・タイプ。
- アプリケーション変数へのポインター。
- 変数の長さ。

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再実行できるようになります。

各パラメーターについての情報は、以下の状況が生じるまで有効です。

- アプリケーションによってオーバーライドされる

- アプリケーションが、SQL_RESET_PARAMS オプション を指定した `SQLFreeStmt()` を呼び出して、パラメーターをアンバインドする
- アプリケーションが、SQL_HANDLE_STMT の *HandleType* を指定した `SQLFreeHandle()` か、 SQL_DROP オプション を指定した `SQLFreeStmt()` を呼び出して、ステートメント・ハンドルをドロップする

各パラメーターの情報は、オーバーライドされるまでか、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・ハンドルをドロップするまで、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、1 つ以上のパラメーターについて `SQLBindParameter()` をもう一度呼び出し、別のアプリケーション変数を指定することにより、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。そのようにすると、DB2 CLI が不要なデータを読み取ったり、無効なメモリーにアクセスしてアプリケーション・トラップになってしまう可能性があります。

SQL ステートメントで必要とされ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。

デフォルト設定では、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、以下のようになる可能性があります。

- DBMS による余分の変換
- DB2 CLI に実行および再実行するステートメントを記述させる DBMS でのエラー。これにより、余分のネットワーク・トラフィックが生じます。
- ステートメントを記述できないか、ステートメントを正常に再実行できない場合に、アプリケーションに戻されるエラー

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。

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。

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。

関連概念:

- 19 ページの『CLI でのハンドル』
- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 109 ページの『CLI アプリケーションでのラージ・オブジェクトの使用』
- 169 ページの『CLI アプリケーションの記述子』
- 166 ページの『CLI アプリケーションでのユーザー定義タイプ (UDT) の使用法』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindFileToParam 関数 (CLI) - LOB パラメーターへの LOB ファイル参照のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFreeStmt 関数 (CLI) - ステートメント・ハンドルの解放 (またはリセット)』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLNumParams 関数 (CLI) - SQL ステートメント内のパラメーター数の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』

関連サンプル:

- 『dbuse.c -- How to use a database』
- 『dtlob.c -- How to read and write LOB data』
- 『spclient.c -- Call various stored procedures』
- 『tbmod.c -- How to modify table data』
- 『tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement』

CLI アプリケーションでのパラメーター・マーカーのバインディング

このトピックでは、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に対してパラメーター・マーカーをバインドする方法を説明します。 SQL ステートメントのパラメーター・マーカーは、単独の値に対してバインドすることもできますし、値の配列にバインドすることも可能です。各パラメーター・マーカーをそれぞれにバインドする場合には、一連の値ごとに、サーバーへのネットワーク・フロ

ーが必要です。しかし、配列を使用する場合は、いくつかのパラメーター値のセットをバインドし、すぐにサーバーへ送信することができます。

前提条件:

パラメーター・マーカーをバインドする前に、アプリケーションを初期設定しておく必要があります。

手順:

パラメーター・マーカーをバインドするには、以下のステップのいずれかを実行します。

- パラメーター・マーカーを一度に 1 つずつアプリケーション変数へバインドする場合、バインドするアプリケーション変数ごとに SQLBindParameter() を呼び出します。必ず正確なパラメーター・タイプ (SQL_PARAM_INPUT、SQL_PARAM_OUTPUT、または SQL_PARAM_INPUT_OUTPUT) を指定するようにしてください。次の例は、2 つのパラメーター・マーカーを 2 つのアプリケーション変数にバインドする方法を示しています。

```
SQLCHAR *stmt =
   (SQLCHAR *)"DELETE FROM org WHERE deptnumb = ? AND division = ? ";
SQLSMALLINT parameter1 = 0;
char parameter2[20];

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

/* bind parameter2 to the statement */
cliRC = SQLBindParameter(hstmt,
                         2,
                         SQL_PARAM_INPUT,
                         SQL_C_CHAR,
                         SQL_VARCHAR,
                         20,
                         0,
                         parameter2,
                         20,
                         NULL);
```

- 多くの値をパラメーター・マーカーへ一度にバインドする場合は、値の配列を使用する、以下の作業のいずれかを実行します。
  - 列方向配列の入力を使用したパラメーター・マーカーのバインド
  - 行方向配列の入力を使用したパラメーター・マーカーのバインド

関連概念:

- 31 ページの『CLI アプリケーションでのパラメーター・マーカー・バインディング』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』

関連サンプル:

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

CLI アプリケーションのコミット・モード

トランザクション とは、リカバリー可能な 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。トランザクションが複数の接続にわたる場合、それは分散作業単位 (DUOW) と呼びます。

`SQLPrepare()`、`SQLExecDirect()`、`SQLGetTypeInfo()` またはカタログなどの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。

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、ステートメント実行の終了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。デフォルトのコミット・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)。

手動コミット
: 手動コミット・モードでは、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、トランザクションを開始してから `SQLEndTran()` を呼び出すまでの間に (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。 DB2 CLI が手動コミット・モードにある場合、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。

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。自動コミットは、照会専用アプリケーションの場合に便利です。なぜなら、サーバーに送信される SQL 実行要求にコミットをチェーニングできるからです。自動コミットのもう 1 つの利点として、可能

な限りロックが除去されるために並行性が向上することがあります。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。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できません。

自動コミットのオン/オフを設定する方法の例を以下に示します。

- 自動コミットをオンに設定する。

```
/* ... */

/* set AUTOCOMMIT on */
sqlrc = SQLSetConnectAttr( hdbc,
                           SQL_ATTR_AUTOCOMMIT,
                           (SQLPOINTER)SQL_AUTOCOMMIT_ON, SQL_NTS) ;

/* continue with SQL statement execution */
```

- 自動コミットをオフに設定する。

```
/* ... */

/* set AUTOCOMMIT OFF */
sqlrc = SQLSetConnectAttr( hdbc,
                           SQL_ATTR_AUTOCOMMIT,
                           (SQLPOINTER)SQL_AUTOCOMMIT_OFF, SQL_NTS) ;

/* ... */

/* execute the statement */
/* ... */
sqlrc = SQLExecDirect( hstmt, stmt, SQL_NTS ) ;

/* ... */

sqlrc = SQLEndTran( SQL_HANDLE_DBC, hdbc, SQL_ROLLBACK );
DBC_HANDLE_CHECK( hdbc, sqlrc);

/* ... */
```

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。この場合、分散作業単位接続とは違って、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。

関連概念:

- 185 ページの『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』
- 73 ページの『CLI アプリケーションのカーソル』
- 146 ページの『CLI アプリケーションでのトランザクション・マネージャーとしての DB2』
- 145 ページの『CLI アプリケーションでのマルチサイト更新 (2 フェーズ・コミット)』

関連タスク:

- 129 ページの『CLI アプリケーションからのストアード・プロシージャーの呼び出し』
- 41 ページの『CLI アプリケーションでのデータの更新と削除』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLEndTran 関数 (CLI) - 接続または環境のトランザクションの終了』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetTypeInfo 関数 (CLI) - データ・タイプ情報の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLPrepare 関数 (CLI) - ステートメントの準備』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』

関連サンプル:

- 『tut_mod.c -- How to modify table data』
- 『tut_read.c -- How to read data from tables』

CLI SQLEndTran() 関数を呼び出す時点

自動コミット・モードでは、各ステートメントの実行の終わりかカーソルのクローズ時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。

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。

- 特定の時点で個々の接続は複数の現行トランザクションを保持できないので、同一の作業単位中では従属ステートメントを保持してください。接続の下で割り振られているステートメントを、その同一の接続上で常に保持しなければならないことに注意してください。
- 接続上で現行トランザクションが実行している間は、さまざまなリソースを保持できます。トランザクションを終了すると、他のアプリケーションが使用するためにリソースが解放されます。
- 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・ログから完全にリカバリー可能になります。オープン・トランザクションはリカバリー可能ではありません。

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。
- デフォルトでは、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。デフォルトではすべてのカーソルは WITH HOLD 文節で定義されます (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場合を除きます)。

関連概念:

- 19 ページの『CLI でのハンドル』
- 31 ページの『CLI アプリケーションでのパラメーター・マーカー・バインディング』
- 35 ページの『CLI アプリケーションのコミット・モード』
- 73 ページの『CLI アプリケーションのカーソル』
- 145 ページの『CLI アプリケーションでのマルチサイト更新 (2 フェーズ・コミット)』
- 96 ページの『CLI アプリケーションでの列バインディング』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDisconnect 関数 (CLI) - データ・ソースからの切断』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLEndTran 関数 (CLI) - 接続または環境のトランザクションの終了』

関連サンプル:

- 『dbmcon.c -- How to use multiple databases』
- 『dbuse.c -- How to use a database』
- 『tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement』

CLI アプリケーションでの照会結果の取り出し

照会結果を検索することは、CLI アプリケーションにおける、より大きなトランザクションの処理作業の一部です。照会結果を検索することには、アプリケーション変数を結果セットの列にバインドしてから、データの行を取り出してアプリケーション変数に取り込むことが関係します。一般的な照会は、SELECT ステートメントです。

前提条件:

結果を検索する前に、アプリケーションを初期設定しておくと同時に、必要な SQL ステートメントを準備して実行しておくようにします。

手順:

結果セットのそれぞれの行を検索するには、以下のようにします。

1. オプション: SQLNumResultCols() および SQLDescribeCol() を呼び出すことにより、結果セットの構造、列の数、および列のタイプおよび長さを判別します。

   注: このステップを照会が実行される前に実行すると、パフォーマンスが低下する場合があります。それは、CLI に照会の列を記述させるためです。結果セットの列についての情報は、正常な実行の後に使用可能になります。さらに、結果セットを記述することが正常な実行後に行われるのであれば、記述のために余分のオーバーヘッドが生じることもありません。

2. SQLBindCol() を呼び出して、アプリケーション変数を結果セットの各列にバインドし、変数タイプが列タイプと一致するようにします。以下に例を示します。

```
struct
{
  SQLINTEGER ind;
  SQLSMALLINT val;
}
deptnumb; /* variable to be bound to the DEPTNUMB column */

struct
{
  SQLINTEGER ind;
  SQLCHAR val[15];
}
location; /* variable to be bound to the LOCATION column */

/* ... */

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

/* bind column 2 to variable */
cliRC = SQLBindCol(hstmt, 2, SQL_C_CHAR, location.val, 15,
                   &location.ind);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);
```

   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に適した 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。列は据え置き出力引き数にバインドされます。すなわち、データは、取り出されるときに、これらの記憶場所に書き込まれるということです。

   重要: 据え置き出力引き数に使用される変数の割り振り解除や廃棄を、アプリケーションが結果セットの列にバインドする時と DB2 CLI がこれらの引き数に書き込む時の間に行ってはなりません。

3. SQL_NO_DATA_FOUND が戻されるまで、SQLFetch() を呼び出して、結果セットからデータの行を繰り返し取り出します。以下に例を示します。

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

if (cliRC == SQL_NO_DATA_FOUND)
{
  printf("¥n  Data not found.¥n");
}
while (cliRC != SQL_NO_DATA_FOUND)
{
  printf("    %-8d %-14.14s ¥n", deptnumb.val, location.val);
```

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

SQLFetchScroll() を使用することにより、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。

SQLBindCol() の呼び出しのときに指定されたデータ・タイプに関してデータ変換が求められた場合には、 SQLFetch() の呼び出し時に変換が行われます。

4. オプション: 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取り出します。この方法で、すべてのアンバインドされた列を検索することができます。以下に例を示します。

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

if (cliRC == SQL_NO_DATA_FOUND)
{
  printf("¥n  Data not found.¥n");
}
while (cliRC != SQL_NO_DATA_FOUND)
{
  /* use SQLGetData() to get the results */
  /* get data from column 1 */
  cliRC = SQLGetData(hstmt,
                     1,
                     SQL_C_SHORT,
                     &deptnumb.val,
                     0,
                     &deptnumb.ind);
  STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

  /* get data from column 2 */
  cliRC = SQLGetData(hstmt,
                     2,
                     SQL_C_CHAR,
                     location.val,
                     15,
                     &location.ind);

  /* display the data */
  printf("    %-8d %-14.14s ¥n", deptnumb.val, location.val);

  /* fetch the next row */
  cliRC = SQLFetch(hstmt);
}
```

注: 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。

関連概念:

- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』

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

関連資料:

- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetData 関数 (CLI) - 列からのデータの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLNumResultCols 関数 (CLI) - 結果列の数の取得』
- 56 ページの『CLI 関数戻りコード』

関連サンプル:

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

CLI アプリケーションでのデータの更新と削除

CLI のトランザクション処理に関する大きなタスクの一部に、データの更新と削除があります。 CLI プログラミングで使用できる更新と削除の操作には、単純タイプと位置指定タイプの 2 つのタイプがあります。単純タイプの更新と削除の操作の場合は、UPDATE ステートメントや DELETE SQL ステートメントを、他の SQL ステートメントの場合と同様に発行して実行するだけで済みます。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。

位置指定タイプの更新と削除には、結果セットのデータを修正することが関係します。位置指定の更新を行うと結果セットの列が更新され、位置指定の削除を行うと結果セットの行が削除されます。位置指定の更新操作と削除操作の場合は、カーソルを使用する必要があります。本書では、最初に結果セットに関連したカーソルの名前を取得し、次に取り出したカーソル名を使用して 2 つ目のステートメント・ハンドル上で UPDATE か DELETE を発行して実行することにより、位置指定の更新操作と削除操作を実行する方法を説明します。

前提条件:

位置指定の更新操作と削除操作を実行する場合は、その前に CLI アプリケーションを初期設定してあることを確認してください。

手順:

位置指定の更新操作か削除操作を実行するには、以下のようにします。

1. SELECT SQL ステートメントを発行して実行し、これから更新か削除を実行する結果セットを生成します。
2. SELECT ステートメントを実行したハンドルと同じステートメント・ハンドルを使用し、 `SQLGetCursorName()` を呼び出してカーソルの名前を取得します。このカーソル名は、UPDATE または DELETE ステートメント中で必要になります。

   ステートメント・ハンドルが割り振られると、カーソル名が自動的に生成されます。 `SQLSetCursorName()` を使用して独自のカーソル名を定義できます。ただし、すべてのエラー・メッセージは `SQLSetCursorName()` を使用して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。
3. 位置指定の更新か削除の実行時に使用する 2 つ目のステートメント・ハンドルを割り振ります。

   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。取り出しステートメント・ハンドルを再利用して、位置指定の更新や削除を実行することはできません。それは位置指定の更新や削除の実行時にまだ使用中だからです。
4. `SQLFetch()` または `SQLFetchScroll()` を呼び出して、結果セットからデータを取り出します。
5. WHERE CURRENT 文節を使用して UPDATE または DELETE SQL ステートメントを発行し、ステップ 2 で入手したカーソル名を指定します。例:

   ```
   sprintf((char *)stmtPositionedUpdate,
           "UPDATE org SET location = 'Toronto' WHERE CURRENT of %s",
           cursorName);
   ```
6. 取り出されたデータの行にカーソルを位置指定し、位置指定の更新ステートメントか削除ステートメントを実行します。

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 27 ページの『CLI アプリケーションでの SQL ステートメントの発行』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetCursorName 関数 (CLI) - カーソル名の取得』

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLRowCount 関数 (CLI) - 行カウントの取得』
- 「*SQL リファレンス 第 2 巻*」の『DELETE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『UPDATE ステートメント』

関連サンプル:

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

CLI アプリケーションでのステートメント・リソースの解放

トランザクションの完了後に、関連したリソースを解放することによって、各ステートメント・ハンドルの処理を終了します。ステートメント・ハンドルのリソースの解放には、以下の 4 つの主なタスクが関係しています。

- オープン・カーソルのクローズ
- 列バインドのアンバインド
- パラメーター・バインドのアンバインド
- ステートメント・ハンドルの解放

ステートメント・リソースを解放するには 2 通りの方法があります。`SQLFreeHandle()` を使用する方法と `SQLFreeStmt()` を使用する方法です。

前提条件:

ステートメント・リソースを解放するには、まず CLI アプリケーションを初期化してステートメント・ハンドルを割り振っておく必要があります。

手順:

`SQLFreeHandle()` を使用してステートメント・リソースを解放するには、SQL_HANDLE_STMT の *HandleType* と、解放したいハンドルを指定して `SQLFreeHandle()` を呼び出します。これによって、このステートメント・ハンドルに関連したオープン・カーソルがクローズされ、列バインドおよびパラメーター・バインドがアンバインドされ、ステートメント・ハンドルが解放されます。このようにして、ステートメント・ハンドルが無効にされます。上記の 4 つのタスクを明示的に実行する必要はありません。

`SQLFreeStmt()` を使用してステートメント・リソースを解放する場合は、以下のようにして、タスクごとに `SQLFreeStmt()` を呼び出す必要があります (アプリケーションがインプリメントされた方法によっては、これらのタスクのすべてが必要でない場合もあります)。

- オープン・カーソルをクローズするには、`SQLCloseCursor()` を呼び出すか、SQL_CLOSE オプション と引き数にステートメント・ハンドルを指定して `SQLFreeStmt()` を呼び出す。これによって、カーソルがクローズされてペンディング結果が廃棄されます。
- 列バインドをアンバインドするには、 SQL_UNBIND オプション とステートメント・ハンドルを指定して、`SQLFreeStmt()` を呼び出す。これによって、このステートメント・ハンドルのすべての列 (ブックマーク列を除く) がアンバインドされます。

- パラメーター・バインドをアンバインドするには、 SQL_RESET_PARAMS オプション とステートメント・ハンドルを指定して、 `SQLFreeStmt()` を呼び出す。これによって、このステートメント・ハンドルのすべてのパラメーター・バインドが解放されます。
- ステートメント・ハンドルを解放するには、 SQL_DROP オプション と解放するステートメント・ハンドルを指定して、 `SQLFreeStmt()` を呼び出す。これによって、このステートメント・ハンドルが無効にされます。

注: このオプションは引き続きサポートされていますが、最新の規格に適合するよう、 DB2 CLI アプリケーションの `SQLFreeHandle()` を使用することをお勧めします。

関連概念:

- 19 ページの『CLI でのハンドル』
- 44 ページの『CLI アプリケーションでのハンドルの解放』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 26 ページの『CLI アプリケーションでのステートメント・ハンドルの割り振り』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLCloseCursor 関数 (CLI) - カーソルのクローズとペンディング結果の廃棄』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFreeStmt 関数 (CLI) - ステートメント・ハンドルの解放 (またはリセット)』

関連サンプル:

- 『tut_read.c -- How to read data from tables』
- 『tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement』
- 『utilcli.c -- Utility functions used by DB2 CLI samples』

CLI アプリケーションでのハンドルの解放

環境ハンドル:

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 を返し、環境と環境に関連した接続はすべて有効のままになります。

接続ハンドル:

ハンドル上で接続がオープンになっている場合は、 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

続に対して SQLDisconnect() を呼び出す必要があります。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:

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リソース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ペンディングにしているステートメントを解放するときに、ペンディングになっている結果は廃棄されます。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。

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() はすべて自動的にドロップするので注意してください。

記述子ハンドル:

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() はすべて自動的にドロップ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。

関連概念:

- 169 ページの『CLI アプリケーションの記述子』

関連タスク:

- 43 ページの『CLI アプリケーションでのステートメント・リソースの解放』
- 59 ページの『CLI アプリケーションの終了』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDisconnect 関数 (CLI) - データ・ソースからの切断』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『記述子ヘッダーとレコード・フィールドの初期設定値 (CLI)』
- 56 ページの『CLI 関数戻りコード』

CLI アプリケーションにおけるデータ・タイプとデータ変換

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・タイプを使用する一方で、アプリケーションは C データ・タイプを使用するので、これは避けられないことです。したがって、アプリケーションは DB2 CLI 関数を呼び出して DBMS とアプリケーションとの間でデータを転送するときに、 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。

この処理が容易になるように、DB2 CLI はさまざまなデータ・タイプにシンボル名を付け、 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。 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シンボル名を使用して両方のデータ・タイプを識別します。

データ・タイプ変換は、以下の 2 つのうちどちらかの条件が該当する場合に行われます。

- アプリケーションで指定されている C タイプが、 SQL タイプに対応するデフォルトの C タイプでない。
- アプリケーションで指定されている SQL タイプが、サーバーの基本列の SQL タイプと一致していないので、記述情報が DB2 CLI ドライバーで使用できない。

データ・タイプの使用法に関する例:

データ・ソースには SQL データ・タイプが含まれており、 CLI アプリケーションは C データ・タイプを処理するので、データを取り出す際には正しいデータ・タイプで処理される必要があります。以下の例は、アプリケーションで SQL と C のデータ・タイプを使用して、ソースからアプリケーション変数中にデータを取り出す方法を示します。この例は、`tut_read.c` サンプル・プログラムに基づくもので、サンプル・データベース中の ORG 表の DEPTNUMB 列からデータを取り出す方法について考察します。

- ORG 表の DEPTNUMB 列は、SQL データ・タイプ SMALLINT として宣言される。
- 取り出したデータを保持するアプリケーション変数は、C タイプを使用して宣言される。 DEPTNUMB 列は SQL タイプ SMALLINT なので、 C タイプ SQLSMALLINT (SQL タイプの SMALLINT と同等) を使用してアプリケーション変数を宣言する必要があります。

  ```
  struct
  {   SQLINTEGER ind;
      SQLSMALLINT val;
  } deptnumb;         /* variable to be bound to the DEPTNUMB column */
  ```

  SQLSMALLINT は短整数の基本 C タイプを表します。
- アプリケーションは、アプリケーション変数をシンボル C データ・タイプ SQL_C_SHORT にバインドする。

  ```
  sqlrc = SQLBindCol(hstmt, 1, SQL_C_SHORT, &deptnumb.val, 0,
                     &deptnumb.ind);
  ```

結果データ・タイプ SQL_C_SHORT は C タイプ SQLSMALLINT を表すので、データ・タイプが整合しました。

データ変換:

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・タイプのいずれかが、 `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、シンボル・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);
```

前の段落で述べた関数を使用して、データをデフォルトから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。

精度と位取りに関する制限を指定する規則や、タイプ変換に関する切り捨てや丸めの規則は 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。

関連概念:

- 57 ページの『DB2 CLI 用の SQLSTATE』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 381 ページの『CLI でサポートされているデータ変換』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetData 関数 (CLI) - 列からのデータの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』

関連サンプル:

- 『dtinfo.c -- How get information about data types』
- 『tut_read.c -- How to read data from tables』

CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ

下記の表では、CLI アプリケーションによって使用される各 SQL データ・タイプ、それに対応する記号名、およびデフォルトの C 記号名をリストしています。

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

記号 SQL データ・タイプ
この列には、整数値として (`sqlcli.h` で) 定義される SQL 記号名を示します。この値は、最初の列にリストした SQL データ・タイプを識別するために、さまざまな関数によって使用されます。

デフォルトの C 記号データ・タイプ
この列には C 記号名を示してあります。これも整数値として定義されています。この値は、C データ・タイプを識別するために、さまざまな関数の引き数で使用されます。記号名は、`SQLBindParameter()`、`SQLGetData()`、および `SQLBindCol()` などのさまざまな関数によってアプリケーション変数の C データ・タイプを識別する際に使用されます。これらの関数の呼び出し時に C データ・タイプを明示的に識別する代わりに、SQL_C_DEFAULT を指定することもでき、その場合 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・タイプに基づくデフォルト C データ・タイプを想定します。たとえば、SQL_DECIMAL のデフォルト C データ・タイプは SQL_C_CHAR です。

アプリケーションでは、C データ・タイプを定義するのに、 `SQL_C_DEFAULT` を使用しないことをお勧めします。この方法は、CLI ドライバーにはあまり効果的ではありません。アプリケーション内に C データ・タイプをはっきり明示すると、 `SQL_C_DEFAULT` を使用するよりもパフォーマンスが良いので、この方法を使用することが望ましいといえます。

表 3. SQL の記号データ・タイプとデフォルト・データ・タイプ

SQL データ・タイプ	記号 SQL データ・タイプ	デフォルト記号 C データ・タイプ
BIGINT	SQL_BIGINT	SQL_C_SBIGINT
BLOB	SQL_BLOB	SQL_C_BINARY
BLOB LOCATOR [a]	SQL_BLOB_LOCATOR	SQL_C_BLOB_LOCATOR
CHAR	SQL_CHAR	SQL_C_CHAR
CHAR	SQL_TINYINT	SQL_C_TINYINT
CHAR FOR BIT DATA [b]	SQL_BINARY	SQL_C_BINARY
CHAR FOR BIT DATA	SQL_BIT	SQL_C_BIT
CLOB	SQL_CLOB	SQL_C_CHAR
CLOB LOCATOR [a]	SQL_CLOB_LOCATOR	SQL_C_CLOB_LOCATOR
DATALINK	SQL_DATALINK	SQL_C_CHAR

表 3. SQL の記号データ・タイプとデフォルト・データ・タイプ (続き)

SQL データ・タイプ	記号 SQL データ・タイプ	デフォルト記号 C データ・タイプ
DATE	SQL_TYPE_DATE	SQL_C_TYPE_DATE
DBCLOB	SQL_DBCLOB	SQL_C_DBCHAR
DBCLOB LOCATOR [a]	SQL_DBCLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR
DECIMAL	SQL_DECIMAL	SQL_C_CHAR
DOUBLE	SQL_DOUBLE	SQL_C_DOUBLE
FLOAT	SQL_FLOAT	SQL_C_DOUBLE
GRAPHIC	SQL_GRAPHIC	SQL_C_DBCHAR
INTEGER	SQL_INTEGER	SQL_C_LONG
LONG VARCHAR [b]	SQL_LONGVARCHAR	SQL_C_CHAR
LONG VARCHAR FOR BIT DATA [b]	SQL_LONGVARBINARY	SQL_C_BINARY
LONG VARGRAPHIC [b]	SQL_LONGVARGRAPHIC	SQL_C_DBCHAR
LONG VARGRAPHIC [b]	SQL_WLONGVARCHAR	SQL_C_DBCHAR
NUMERIC [c]	SQL_NUMERIC [c]	SQL_C_CHAR
REAL	SQL_REAL	SQL_C_FLOAT
SMALLINT	SQL_SMALLINT	SQL_C_SHORT
TIME	SQL_TYPE_TIME	SQL_C_TYPE_TIME
TIMESTAMP	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP
VARCHAR	SQL_VARCHAR	SQL_C_CHAR
VARCHAR FOR BIT DATA [b]	SQL_VARBINARY	SQL_C_BINARY
VARGRAPHIC	SQL_VARGRAPHIC	SQL_C_DBCHAR
VARGRAPHIC	SQL_WVARCHAR	SQL_C_DBCHAR
WCHAR	SQL_WCHAR	SQL_C_WCHAR

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)。

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

c DB2 for z/OS、DB2 Server for VSE & VM および DB2 Universal Database では、NUMERIC は DECIMAL の同義語です。

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

関連概念:

- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 111 ページの『CLI アプリケーションでの LOB ロケーター』

関連資料:

- 50 ページの『CLI アプリケーション用の C データ・タイプ』

- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『SQLGetData 関数 (CLI) - 列からのデータの取得』
- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』

CLI アプリケーション用の C データ・タイプ

次の表は、CLI アプリケーションで使用されるそれぞれの記号 C タイプごとに総称型定義をリストしています。

C 記号データ・タイプ
この列には C 記号名を示してあります。これは整数値として定義されています。この値は、最後の列に示された C データ・タイプを識別するために、さまざまな関数の引き数で使用されます。

C タイプ
この列には C 定義済みタイプを示してあります。これは C `typedef` ステートメントを使用して `sqlcli.h` で定義されるものです。この列にある値を使用して、アプリケーションの可搬性を高めるために、すべての 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、 52 ページの表 6 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、 51 ページの表 5 で記述されている C 構造体です。

表 4. C データ・タイプ

C 記号データ・タイプ	C タイプ	基本 C タイプ
SQL_C_BINARY	SQLCHAR	unsigned char
SQL_C_BIT	SQLCHAR	unsigned char または char (値 1 または 0)
SQL_C_BLOB_LOCATOR [a]	SQLINTEGER	long int
SQL_C_CLOB_LOCATOR [a]	SQLINTEGER	long int
SQL_C_CHAR	SQLCHAR	unsigned char
SQL_C_DATALINK	SQLCHAR	unsigned char
SQL_C_DBCHAR	SQLDBCHAR	wchar_t
SQL_C_DBCLOB_LOCATOR	SQLINTEGER	long int
SQL_C_DOUBLE	SQLDOUBLE	double
SQL_C_FLOAT	SQLREAL	float
SQL_C_LONG	SQLINTEGER	long int
SQL_C_NUMERIC [b]	SQL_NUMERIC_STRUCT	詳細は、 51 ページの表 5 を参照
SQL_C_SBIGINT	SQLBIGINT	_int64

表 4. 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_SHORT	SQLSMALLINT	short int
SQL_C_TINYINT	SQLSCHAR	signed char (範囲: -128 以上 127 以下)
SQL_C_TYPE_DATE	DATE_STRUCT	詳細は、表 5 を参照
SQL_C_TYPE_TIME	TIME_STRUCT	詳細は、表 5 を参照
SQL_C_TYPE_TIMESTAMP	TIMESTAMP_STRUCT	詳細は、表 5 を参照
SQL_C_UBIGINT	SQLUBIGINT	unsigned _int64
SQL_C_ULONG	SQLUINTEGER	unsigned long int
SQL_C_USHORT	SQLUSMALLINT	unsigned short int
SQL_C_UTINYINT	SQLUCHAR	unsigned char
SQL_C_WCHAR	SQLWCHAR	wchar_t

- **a** LOB ロケーター・タイプ。
- **b** Windows のみ。

注: DB2 CLI では、SQL ファイル参照データ・タイプ (組み込み SQL で使用される) は必要ありません。

表 5. 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  minute;`
		`  UWORD  second;`
		`  } TIME_STRUCT;`

TIMESTAMP_STRUCT	`typedef struct TIMESTAMP_STRUCT`
	`  {`
	`  SQLUSMALLINT  year;`
	`  SQLUSMALLINT  month;`
	`  SQLUSMALLINT  day;`
	`  SQLUSMALLINT  hour;`
	`  SQLUSMALLINT  minute;`
	`  SQLUSMALLINT  second;`
	`  SQLINTEGER  fraction;`
	`  } TIMESTAMP_STRUCT;`
		`typedef struct tagTIMESTAMP_STRUCT`
		`  {`
		`  SWORD  year;`
		`  UWORD  month;`
		`  UWORD  day;`
		`  UWORD  hour;`
		`  UWORD  minute;`
		`  UWORD  second;`
		`  UDWORD  fraction;`
		`  } TIMESTAMP_STRUCT;`

表 5. C 構造体 (続き)

C タイプ	総称構造	Windows 構造

SQL_NUMERIC_STRUCT	(総称構造はありません。 Windows 構造だけです。)	`typedef struct tagSQL_NUMERIC_STRUCT`
		`  {`
		`  SQLCHAR  precision;`
		`  SQLCHAR  scale;`
		`  SQLCHAR  sign; [a]`
		`  SQLCHAR`
		`  val[SQL_MAX_NUMERIC_LEN];[b c]`
		`  } SQL_NUMERIC_STRUCT;`

SQLUSMALLINT C データ・タイプに関する詳細は、表 6 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デフォルト値を精度フィールドの値として使用し、アプリケーション記述子 (デフォルト設定は 0) の SQL_DESC_SCALE フィールドの値を位取りフィールドに使用します。アプリケーションは、アプリケーション記述子の SQL_DESC_PRECISION および SQL_DESC_SCALE フィールドを設定して、精度および位取りに指定する独自の値を提供することができます。

SQL データ・タイプにマップするデータ・タイプに加えて、ポインターやハンドルなどの他の関数の引き数に使用される 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 6. 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。
SQLRETURN	short int	DB2 CLI 関数からの戻りコード。

関連概念:

- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 111 ページの『CLI アプリケーションでの LOB ロケーター』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』

CLI アプリケーションのストリングの処理

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:

入力ストリングは、関連した長さ引き数を保持できます。この引き数は、ストリングの正確な長さ (NULL 終止符を除く)、 NULL 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 のうちのいずれかを示します。長さを SQL_NTS に設定すると、DB2 CLI は NULL 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 NULL 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、記述子 FieldIdentifier の引き数値を参照してください。

出力長さ引き数に NULL ポインターを指定して関数が呼び出される場合、DB2 CLI は長さを戻しません。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。

パフォーマンスのヒント:

長さ引き数 (*StrLen_or_IndPtr*) と出力バッファー (*TargetValuePtr*) がメモリー内で隣接していると、 DB2 CLI は両方の値をさらに効果的に返すことができ、アプリケーションのパフォーマンスは向上します。たとえば、次の構造が定義されているとします。

```
struct
{   SQLINTEGER pcbValue;
    SQLCHAR    rgbValue [BUFFER_SIZE];
} buffer;
```

さらに `&buffer.pcbValue` および `buffer.rgbValue` が `SQLBindCol()` に渡されると、 DB2 CLI は 1 回の操作で両方の値を更新します。

ストリングの NULL 終了:

デフォルトでは、 DB2 CLI が戻すすべての文字ストリングが NULL 終止符 (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 バイト文字の NULL 終止符により NULL 終了します。また、

SQL_C_WCHAR 中に取り出されるストリング・データは、Unicode NULL 終止符 0x0000 で終了します。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 NULL 終止符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 NULL 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、出力長さ引き数のストレージ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 のデフォルトは、常に NULL 終止符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 NULL 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。

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、NULL 終止符を書き込みます。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。また `SQLGetDiagField()` を呼び出して、どの列が失敗したかを検出することもできます。

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。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。

ストリングのブランク埋め込み:

DB2 のバージョン 8.1 からバージョン 8.1.4 の各リリースでは、列サイズに合わせてストリングにブランクが埋め込まれていましたが、 DB2® バージョン 8.1.4 以降、そうではなくなりました。 DB2 バージョン 8.1.4 以降では、コード・ページ変換が発生した場合に、ストリングの長さが CHAR 列で定義されている長さと違うことがあります。バージョン 8.1.4 より前の DB2 の場合、列サイズに合わせてストリングにブランクが埋め込まれていました。そのストリングが CHAR 列から取り出される際には、それらのブランクがストリング・データの一部として戻されていました。

関連概念:

- 57 ページの『DB2 CLI 用の SQLSTATE』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetEnvAttr 関数 (CLI) - 環境属性の設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『記述子ヘッダーとレコード・フィールドの初期設定値 (CLI)』
- 56 ページの『CLI 関数戻りコード』
- 345 ページの『Patch1 CLI/ODBC 構成キーワード』

CLI アプリケーションでの診断の概説

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()` と `SQLGetDiagField()` の両方で、さらに詳しい診断情報を通知します。 DBMS で診断が行われる場合は、`SQLGetSQLCA()` 関数は SQLCA へアクセスできるようにします。この調整によって、アプリケーションは、戻りコードに基づいて基本的な制御の流れを扱えるようになり、 SQLSTATE と SQLCA を一緒に使用して特定の障害原因を判別したり、特定のエラー処理を実行できるようになります。

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

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

`SQLGetSQLCA()` は、特定のフィールドにアクセスするための SQLCA を戻しますが、望んでいる情報が `SQLGetDiagRec()` または `SQLGetDiagField()` を使用して得られない場合にのみ使用してください。

関連概念:

- 57 ページの『DB2 CLI 用の SQLSTATE』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetSQLCA 関数 (CLI) - SQLCA データ構造の取得』
- 「*SQL リファレンス 第 1 巻*」の『SQLCA (SQL 連絡域)』
- 56 ページの『CLI 関数戻りコード』

CLI 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。

表 7. DB2 CLI 関数戻りコード

戻りコード	解説
SQL_SUCCESS	関数が正常に完了しました。他に利用可能な SQLSTATE 情報はありません。
SQL_SUCCESS_WITH_INFO	関数は正常に完了しましたが、警告または他の情報があります。 `SQLGetDiagRec()` または `SQLGetDiagField()` を呼び出して、 SQLSTATE およびその他の通知メッセージまたは警告を受け取ってください。 SQLSTATE のクラスは '01' です。
SQL_STILL_EXECUTING	関数は非同期に実行中で、まだ完了していません。 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。
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()` または `SQLGetDiagField()` 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。
SQL_ERROR	関数は失敗しました。 `SQLGetDiagRec()` または `SQLGetDiagField()` を呼び出して、 SQLSTATE およびその他のエラー情報を受け取ってください。
SQL_INVALID_HANDLE	関数は無効な入力ハンドル (環境、接続またはステートメント・ハンドル) のために失敗しました。これはプログラミング・エラーです。さらに情報はありません。

`tut_read.c` からの次のコード・セグメントは、関数戻りコード SQL_NO_DATA_FOUND を使用してデータ検索を停止するときを制御する方法を示しています。

```
while (cliRC != SQL_NO_DATA_FOUND)
{
  printf("    %-8d %-14.14s ¥n", deptnumb.val, location.val);

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

関連概念:

- 19 ページの『CLI でのハンドル』
- 55 ページの『CLI アプリケーションでの診断の概説』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『CLI と ODBC 関数のサマリー』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』

関連サンプル:

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

DB2 CLI 用の 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 Driver Manager によって生成されます。

注: バージョン 5 より前のバージョンの 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。

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 バージョン 3 および ISO SQL/CLI 仕様では定義されていない条件に関する 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 バージョン 3 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。アプリケーションでの依存性は、特定の SQLSTATE に基づいた論理フローの決定です。

  注: SQLSTATE のクラス (先頭 2 文字) に関する依存性を作成すると効果的な場合があります。

- 診断情報を最大限活用するために、 SQLSTATE とともにテキスト・メッセージを返してください (該当すれば、テキスト・メッセージには IBM 定義 SQLSTATE も含まれます)。アプリケーションがエラーを返した関数の名前を印刷することも効果的です。

utilcli.c からの次のコード・セグメントには、 SQLSTATE などの診断情報を検索して表示する方法が示されています。

```
void HandleDiagnosticsPrint(SQLSMALLINT htype, /* handle type identifier */
                            SQLHANDLE hndl /* handle */ )
{
  SQLCHAR message[SQL_MAX_MESSAGE_LENGTH + 1];
  SQLCHAR sqlstate[SQL_SQLSTATE_SIZE + 1];
  SQLINTEGER sqlcode;
  SQLSMALLINT length, i;

  i = 1;

  /* get multiple field settings of diagnostic record */
  while (SQLGetDiagRec(htype,
                       hndl,
                       i,
                       sqlstate,
                       &sqlcode,
                       message,
                       SQL_MAX_MESSAGE_LENGTH + 1,
                       &length) == SQL_SUCCESS)
  {
    printf("¥n  SQLSTATE          = %s¥n", sqlstate);
    printf("  Native Error Code = %ld¥n", sqlcode);
    printf("%s¥n", message);
    i++;
  }

  printf("-------------------------¥n");
}
```

アプリケーションが DB2 を呼び出す方法 (生じる何らかのエラーも含めて) をより良く理解するには、 CLI/ODBC のトレース機能を使用することができます。

関連概念:

- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』

- 56 ページの『CLI 関数戻りコード』

関連サンプル:

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

終了

CLI アプリケーションの終了

CLI アプリケーションを初期化してトランザクションを処理した後は、データ・ソースから正常に切断してリソースを解放するために、アプリケーションを終了する必要があります。

前提条件:

アプリケーションを終了する前に、 CLI アプリケーションを初期化し、すべてのトランザクションの処理を完了しておく必要があります。

手順:

CLI アプリケーションを終了するには、以下のようにします。

1. `SQLDisconnect()` を呼び出して、データ・ソースから切断する。
2. *HandleType* 引き数に SQL_HANDLE_DBC を指定して `SQLFreeHandle()` を呼び出し、接続ハンドルを解放する。

   複数のデータベース接続が存在する場合は、すべての接続がクローズされて接続ハンドルが解放されるまで、ステップ 1 と 2 を繰り返してください。
3. *HandleType* 引き数に SQL_HANDLE_ENV を指定して `SQLFreeHandle()` を呼び出し、環境ハンドルを解放する。

関連概念:

- 25 ページの『CLI でのトランザクション処理の概説』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 43 ページの『CLI アプリケーションでのステートメント・リソースの解放』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDisconnect 関数 (CLI) - データ・ソースからの切断』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』

関連サンプル:

- 『dbconn.c -- How to connect to and disconnect from a database』
- 『dbmcon.c -- How to use multiple databases』
- 『utilcli.c -- Utility functions used by DB2 CLI samples』

第 4 章 プログラミングのヒントと提案

CLI アプリケーション用のプログラミングのヒントと提案

このトピックでは、以下の対象について説明します。

DB2 CLI の更新の詳細は、以下の DB2® アプリケーション開発の Web サイトを参照してください。

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

KEEPDYNAMIC 動作とは、コミットの実行後も動的ステートメントを準備済み状態に保持するサーバーの機能のことです。この動作によって、ステートメントが次に実行されるときに、クライアントはステートメントをもう一度準備する必要がなくなります。クライアント上の一部の DB2 CLI/ODBC アプリケーションは、 DB2 UDB for z/OS™ and OS/390® バージョン 7 以降のサーバー上で KEEPDYNAMIC 動作を利用することにより、パフォーマンスが向上する場合があります。KEEPDYNAMIC 動作を使用可能にするには、以下のステップを完了します。

1. DB2 UDB for z/OS および OS/390 サーバー上で動的ステートメント・キャッシュを使用可能にします (DB2 UDB for z/OS および OS/390 サーバーの資料を参照)。

2. KEEPDYNAMIC および COLLECTION オプションを指定して、DB2 UDB for Linux、UNIX®、および Windows® クライアント上で db2clipk.bnd ファイルをバインドします。以下の例は、KEEPDYNC という名前のコレクションを作成して db2clipk.bnd をバインドする方法を示しています。
   - db2 connect to *database_name* user *userid* using *password*
   - db2 bind db2clipk.bnd SQLERROR CONTINUE BLOCKING ALL KEEPDYNAMIC YES COLLECTION KEEPDYNC GRANT PUBLIC
   - db2 connect reset
3. 以下のいずれかを実行して、KEEPDYNAMIC BIND オプションをコレクションで使用できるようになったことをクライアントに通知します。
   - db2cli.ini ファイルの中で CLI/ODBC 構成キーワード (KeepDynamic = 1、CurrentPackageSet = ステップ 2 で作成されたコレクション名) を設定する。たとえば、

     ```
     [dbname]
     KeepDynamic=1
     CurrentPackageSet=KEEPDYNC
     ```

   - DB2 CLI/ODBC アプリケーションの中で SQL_ATTR_KEEPDYNAMIC および SQL_ATTR_CURRENT_PACKAGE_SET 接続属性を設定する。例:

     ```
     SQLSetConnectAttr(hDbc,
                       SQL_ATTR_KEEP_DYNAMIC,
                       (SQLPOINTER) 1,
                        SQL_IS_UINTEGER );

     SQLSetConnectAttr(hDbc,
                       SQL_ATTR_CURRENT_PACKAGE_SET,
                        (SQLPOINTER) "KEEPDYNC",
                       SQL_NTS);
     ```

KEEPDYNAMIC 動作および構成の詳細については、DB2 UDB for OS/390 and z/OS の資料を参照してください。

共通接続属性:

以下の接続属性は、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_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_ON をオンにしておきます。

  注: デフォルト値は 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キング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、一方でデータの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。

共通ステートメント属性:

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、照会操作によって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・リソース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 (z/OS および OS/390 版) バージョン 7 以降への接続中に 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。バージョン 7 より前のバージョンの DB2 for OS/390 や、"FETCH n ROWS ONLY" 文節をサポートしない DBMS の場合、サーバー側で "OPTIMIZE FOR n ROWS" 文節を使用した完全な結果セットが生成されますが、 DB2 CLI は、クライアント上で行をカウントし、SQL_ATTR_MAX_ROWS 行までを取り出すに過ぎません。

- SQL_ATTR_CURSOR_HOLD - このステートメント属性は、 DB2 CLI がこのステートメントのカーソルを WITH HOLD 文節を使用して宣言するかどうかを決めます。

  カーソル保留動作を必要としないステートメントがこの属性を SQL_CURSOR_HOLD_OFF に設定する場合、ステートメント・ハンドルに関連するリソースはサーバーによってよりよく活用されます。この属性を適切に使用することによって得られる効率上の利点は、 OS/390 および z/OS 上で著しいものです。

  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、デフォルト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キングのレベルを決めるものです。

  すべてのステートメントがデフォルトの分離レベルのままにされているのではなく、必要とされる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リソースは 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 のロッキング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。

  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るべきです。

ステートメント・ハンドルの再利用:

CLI アプリケーションがステートメント・ハンドルを宣言するたびに、DB2 CLI ドライバーは、そのハンドルの基礎となるデータ構造を割り振って初期設定します。パフォーマンスを向上させるために、CLI アプリケーションは、別のステートメントでステートメント・ハンドルを再利用することにより、ステートメント・ハンドルの割り振りと初期設定に関連したコストを削減できます。

注: ステートメント・ハンドルを再利用する前に、以前のステートメントで使用されたメモリー・バッファーおよび他のリソースを、 `SQLFreeStmt()` 関数を呼び出すことによって解放しなければならない場合があります。また、ステートメント・ハンドルに以前に設定されたステートメント属性 (たとえば、

SQL_ATTR_PARAMSET_SIZE) を明示的にリセットする必要もあります。そうしない場合、ステートメント・ハンドルを使用する今後のすべてのステートメントに継承される可能性があります。

バインドおよび SQLGetData():

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。データが LOB 列にある場合、SQLGetData() よりも LOB 関数のほうが望ましいです (詳細は、 65 ページの『ラージ・オブジェクト・データの最適化』を参照)。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() 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。

カタログ関数の使用を制限する:

SQLTables() のようなカタログ関数により、DB2 CLI ドライバーは、情報を取り出すために DBMS カタログ表を照会します。発行される照会は複雑で、DBMS カタログ表は非常に大きくなる可能性があります。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、そのデータをアプリケーションに保管させる (キャッシュに入れる) 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千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。アプリケーションを開発する際には、この類似性を考慮に入れてください。

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 *Option* を指定して SQLCloseCursor() または 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:

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 (SQLBindCol()、SQLGetData()、および SQLDescribeCol() で使用される *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ロードされる DB2 CLI 固有の関数:

ODBC Driver Manager 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 CLI ステートメント・ハンドルにマッピングします。DB2 CLI 関数が直接に呼び出される場合、CLI ドライバーには ODBC マッピングへのアクセス権がないため、 CLI ドライバーのステートメント・ハンドルに渡す必要があります。

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、必要な箇所で HSTMT 引き数を渡すことによって、共有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。

グローバル動的ステートメント・キャッシュ:

UNIX または Windows 用のバージョン 5 以降の DB2 Universal Database サーバーには、*グローバル動的ステートメント・キャッシュ* が備えられています。このキャッシュは、準備状態の動的 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。

データ追加および検索の最適化:

パラメーターをバインドしてデータを検索するために配列を使用することを記述する方式では、コンパウンド SQL を使用してネットワーク・フローを最適化します。可能な限りそれらの方法を使用するようにしてください。

ラージ・オブジェクト・データの最適化:

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 `SQLGetPosition()` 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 ID の大文字小文字の区別:

表名、ビュー名、および列名など、データベース・オブジェクト ID はすべて、その ID 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 ID 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。

ID が SQL ステートメント内で参照されると、ID 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。

SQLDriverConnect() と SQLConnect():

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デフォルト値として使用されます。

SQL ガバナーのインプリメンテーション:

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。

この見積もりを 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_ATTR_DB2ESTIMATE 接続属性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リソース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。カタログ表のデータベース統計が最新であれば、より正確な情報が示されます。コマンド行プロセッサー・セッションから RUNSTATS コマンドを発行することにより、 DB2 Universal Database 上でデータベース統計を更新することができます。

ステートメント・スキャンをオフにする:

DB2 CLI はデフォルト設定で、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 1 つスキャンします。

アプリケーションがベンダー・エスケープ文節順序列を含む SQL ステートメント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_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スキャン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を通じてカーソルを保持する:

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同時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 1 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。

しかし、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。例:

- 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。

注: あるアプリケーションによって 1 つのデータベースに対する複数の接続が使用される場合、そのアプリケーションでは、データベース・オブジェクトに対するアクセスを注意深く同期化する必要があります。そのようにしないと、データベース・ロックはトランザクション間で共有されるものではないため、さまざまなロック競合問題が生じる可能性があります。ある接続によって更新が行われると、最初の接続が (COMMIT または ROLLBACK によって) ロックを解放するまで、他の接続も容易にロック待機状態になってしまう場合があります。

コンパウンド SQL サブステートメントの作成:

コンパウンド・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、次いでコンパウンド・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、作成エラーがコンパウンド・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およびキャスト:

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する必要があります。

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) )
```

さらに、その後で次の SQL ステートメントが発行されたとします。

```
SELECT first_name, last_name, phone_num from customer
WHERE 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。

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 )
```

ネットワーク・フロー削減のための据え置き準備:

DB2 CLI 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。その後、ネットワーク・フローを最小化しパフォーマンスを改善するため、 2 つの要求が 2 つではなく 1 つのコマンド/応答のフローに結合されます。これが最大の利点となるのは、アプリケーションが照会を生成し、その応答のセットが非常に少ない場合や、別々の要求と応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 Connect や DDCS ゲートウェイを使用する環境では、要求と応答の 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。

注: `SQLDescribeParam()`、`SQLDescribeCol()`、`SQLNumParams()`、および `SQLNumResultCols()` のような関数では、ステートメントを準備しておく必要があります。ステートメントが準備されていない場合、これらの関数は、サーバーに対して即時 PREPARE 要求を生成するため、据え置き準備の効果は表れません。

関連概念:

- 185 ページの『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』
- 189 ページの『CLI アプリケーションでのベンダー・エスケープ文節』
- 19 ページの『CLI でのハンドル』
- 73 ページの『CLI アプリケーションのカーソル』
- 109 ページの『CLI アプリケーションでのラージ・オブジェクトの使用』
- 70 ページの『CLI 配列入力チェーニングによるネットワーク・フローの削減』

関連タスク:

- 135 ページの『CLI アプリケーションでのコンパウンド SQL ステートメントの実行』
- 100 ページの『CLI アプリケーションでの列方向バインドを使用した配列データの取り出し』
- 101 ページの『CLI アプリケーションでの行方向バインドを使用した配列データの取り出し』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『CLI と ODBC 関数のサマリー』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLConnect 関数 (CLI) - データ・ソースへの接続』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDriverConnect 関数 (CLI) - データ・ソースへの (拡張) 接続』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetData 関数 (CLI) - 列からのデータの取得』

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』

CLI 配列入力チェーニングによるネットワーク・フローの削減

CLI 配列入力チェーニングをオンにした場合、チェーンが終了するまで、準備済みステートメントの実行要求が保留になりクライアント側でキューに入れられます。チェーンが終了すると、クライアント側でチェーニングされた SQLExecute() 要求が 1 回のネットワーク・フローでサーバーに送られます。

以下に示すイベント列 (疑似コードで示す) は、 CLI 配列入力チェーニングによってサーバーへのネットワーク・フローの数を削減する方法を示す例です。

```
SQLPrepare (statement1)
SQLExecute (statement1)
SQLExecute (statement1)
/* the two execution requests for statement1 are sent to the server in
two network flows */

SQLPrepare (statement2)

/* enable chaining */
SQLSetStmtAttr (statement2, SQL_ATTR_CHAINGING_BEGIN)

SQLExecute (statement2)
SQLExecute (statement2)
SQLExecute (statement2)

/* end chaining */
SQLSetStmtAttr (statement2, SQL_ATTR_CHAINING_END)

/* the three execution requests for statement2 are sent to the server
in a single network flow, instead of three separate flows */
```

SQL_ATTR_CHAINING_END を設定して SQL_ERROR または SQL_SUCCESS_WITH_INFO が戻される場合は、ステートメントのチェーンに含まれる 1 つ以上のステートメントの実行時に、 SQL_ERROR または SQL_SUCCESS_WITH_INFO が戻されたということです。エラーまたは警告の原因については、 CLI 診断関数 `SQLGetDiagRec()` および `SQLGetDiagField()` を使用してください。

関連概念:

- 61 ページの『CLI アプリケーション用のプログラミングのヒントと提案』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecute 関数 (CLI) - ステートメントの実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLPrepare 関数 (CLI) - ステートメントの準備』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 56 ページの『CLI 関数戻りコード』

第 5 章 カーソル

カーソル

CLI アプリケーションのカーソル

CLI アプリケーションは、カーソルを使用して、結果セットから行を取り出します。カーソルとは、アクティブな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。 Linux、UNIX、および Windows® プラットフォームの DB2® UDB クライアントは、 DB2 for Linux、DB2 for UNIX、DB2 for Windows、および DB2 for z/OS™ バージョン 8 に対して実行される場合、更新可能なスクロール可能カーソルをサポートします。 DB2 for z/OS または DB2 for OS/390® バージョン 7 またはそれ以上の 3 階層環境でスクロール可能カーソルにアクセスするには、クライアントおよびゲートウェイが DB2 UDB バージョン 7.2 またはそれ以上を実行している必要があります。

`SQLExecute()` または `SQLExecDirect()` によって動的 SQL SELECT ステートメントが正常に実行されると、カーソルがオープンします。一般的には、アプリケーションのカーソル操作と、カーソルのある DB2 CLI ドライバーによって実行される操作との間には、1 対 1 の相関があります。正常実行の直後に、カーソルは結果セットの先頭行の前に位置指定され、 `SQLFetch()`、`SQLFetchScroll()`、または `SQLExtendedFetch()` への呼び出しによる FETCH 操作の際に、カーソルは結果セット中を一度に 1 行ずつ進みます。カーソルが結果セットの末尾に達すると、次の取り出し操作により SQLCODE +100 が戻されます。 CLI アプリケーションの側から見ると、結果セットの末尾に達すると `SQLFetch()` により SQL_NO_DATA_FOUND が戻されます。

カーソルのタイプ:

次の 2 つのタイプのカーソルが DB2 CLI にサポートされています。

スクロール不可
順方向のみのスクロール不可カーソルは、 DB2 CLI ドライバーによって使用されるデフォルトのカーソル・タイプです。このカーソル・タイプは単一方向で、最少量のオーバーヘッド処理が必要になります。

スクロール可能
次の 3 つのタイプのスクロール可能カーソルが DB2 CLI によりサポートされています。

静的 これは読み取り専用カーソルです。一度作成されたなら、行を加え

たり削除したりできません。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。カーソルの作成に使用する分離レベルのステートメントを使用して、カーソルの行がロックされている場合にその方法を判別できます。

キー・セット主導

静的なスクロール可能カーソルとは違って、キー・セット主導のスクロール可能カーソルは基礎となるデータを検出して変更を加えることができます。キー・セット・カーソルは、行キーに基づいています。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・セットは、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データベース中の最新の値が検索されます。この値は、初めてカーソルがオープンした時点で存在していた値である必要はありません。したがって、アプリケーションが行にスクロールするまで変更内容は反映されません。

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取り出すので、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を反映します。キー・セットの生成後に行セット中の選択された行が削除されると、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。カーソルはこのタイプの変更内容を反映しません。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。挿入された行を参照するには、アプリケーションは照会を再実行しなければなりません。

注: 現在 DB2 CLI は、サーバーがキー・セット主導カーソルをサポートしている場合に限り、キー・セット主導カーソルをサポートします。現在 DB2 バージョン 8 サーバーは更新可能なスクロール可能カーソルをサポートしているので、現在 DB2 for OS/390 バージョン 6 またはそれ以前、DB2 for Unix バージョン 7 またはそれ以前、 DB2 for Windows バージョン 7 またはそれ以前にアクセスしているアプリケーションが、キー・セット・カーソル機能を必要とする場合は、クライアントを DB2 バージョン 8 に移行しないでください。

動的 動的スクロール可能カーソルでは、結果セットに対するすべての変更 (挿

入、削除、および更新) を検出し、結果セットに対して挿入、削除、および更新を実行できます。キー・セット主導カーソルとは異なり、動的カーソルでは以下を行います。

- 他のカーソルによって挿入された行を検出する
- 結果セットから削除された行を省略する (キー・セット主導カーソルは、削除された行を結果セットの中の「穴」として認識する)

現在のところ、動的スクロール可能カーソルは、 DB2 CLI において DB2 UDB for z/OS バージョン 8.1 以降のサーバーにアクセスする場合のみサポートされます。

カーソル属性:

下の表では、DB2 CLI でのカーソルのデフォルト属性をリストします。

表 8. CLI でのカーソルのデフォルト属性

カーソル・タイプ	カーソル感度	カーソル更新可能	カーソル並列処理	カーソル・スクロール可能
フォワードのみ[a]	未指定	更新不可	読み取り専用並行処理	スクロール不可
静的	反映不可	更新不可	読み取り専用並行処理	スクロール可能
キー・セット主導	反映可能	更新可能	値並行処理	スクロール可能
動的	反映可能	更新可能	値並行処理	スクロール可能

a フォワードのみは、 `FOR UPDATE` 文節を使用しないスクロール可能カーソルのデフォルトの振る舞いです。フォワードのみカーソルで `FOR UPDATE` を指定すると、更新可能、ロック並列処理、スクロール不可のカーソルが作成されます。

キー・セット主導カーソルの更新:

キー・セット主導カーソルは更新可能なカーソルです。照会が `SELECT ... FOR READ ONLY` として発行されている場合、または `FOR UPDATE` 文節がすでに指定されている場合を除いて、 CLI ドライバーは `FOR UPDATE` 文節を照会に追加します。キー・セット主導カーソルは値並行性カーソルです。値並列処理カーソルを使用するとオプティミスティック・ロッキングになります。更新または削除が試行されるまでロッキングは行われません。更新または削除が試行されると、データベース・サーバーは、アプリケーションが検索した以前の値を基本表の現行値と比較します。値が一致する場合、更新または削除は成功します。値が一致しない場合、操作は失敗します。失敗した場合、アプリケーションは値をもう一度照会して、まだ適用可能であれば更新または削除を再発行します。

アプリケーションはキー・セット主導カーソルを以下の 2 つの方法で更新することができます。

- `SQLExecute()` または `SQLExecDirect()` とともに `SQLPrepare()` を使用して、`UPDATE WHERE CURRENT OF <cursor name>` または `DELETE WHERE CURRENT OF <cursor name>` を発行します。

- SQLSetPos() または SQLBulkOperations() を使用して、結果セットに対して行の更新、削除、または追加を行います。

  注: SQLSetPos() または SQLBulkOperations() 経由で結果セットに追加された行は、サーバー上の表に挿入されますが、サーバーの結果セットには追加されません。したがって、このような行は更新されず、別のトランザクションが行った変更も反映されません。ただし、挿入された行は、クライアント側でキャッシュされるため、結果セットの一部のように見えます。挿入された行に適用されるトリガーは、アプリケーション側からは適用されていないように見えます。挿入された行を更新可能および反映可能にし、適用可能なトリガーの結果を参照するには、アプリケーションで照会を再発行して、結果セットを再生成する必要があります。

関連概念:

- 76 ページの『CLI アプリケーションのカーソルに関する考慮事項』
- 79 ページの『CLI アプリケーションにおける結果セットの用語』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecute 関数 (CLI) - ステートメントの実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExtendedFetch 関数 (CLI) - 拡張取り出し (行の配列の取り出し)』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 「*SQL リファレンス 第 2 巻*」の『DELETE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『UPDATE ステートメント』
- 313 ページの『CursorHold CLI/ODBC 構成キーワード』

CLI アプリケーションのカーソルに関する考慮事項

使用するカーソル・タイプの決定:

まず最初に、順方向カーソルとスクロール可能カーソル (両方向スクロール・カーソル) のどちらを使用するかを決める必要があります。順方向カーソルの方がスクロール可能カーソルよりオーバーヘッドが少なく、スクロール可能カーソルは並行性が低下する可能性があります。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、スクロール不可カーソル (順方向カーソル) 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、キー・セット主導カーソル、あるいは動的カーソルのいずれ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または動的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してください。

注: 現在のところ、動的カーソルは、 DB2® UDB for z/OS™ バージョン 8.1 以降のサーバーにアクセスする場合のみサポートされます。

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か動的カーソルのいずれかを使用する必要があります。動的カーソルは、より多くのオーバーヘッドを発生させ、キー・セット主導カーソルと比較して並行性が低いため、行われた変更と他のカーソルによって挿入された行の両方をアプリケーションが検出しなければならない場合は、動的カーソルだけを選択してください。

アプリケーションが特定のカーソル・タイプを指定しないで変更を検出できるスクロール可能カーソルを要求すると、 DB2 CLI は動的カーソルは不要であると見なし、キー・セット主導カーソルを提供します。この動作により、動的カーソルによって発生するオーバーヘッドの増大と並行性の低下を回避できます。

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の属性を判別するには、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

作業単位に関する考慮事項:

カーソルは明示的にも暗黙的にもクローズできます。 `SQLCloseCursor()` を呼び出すと、アプリケーションは明示的にカーソルをクローズできます。カーソルを再オープンしない限り、その後カーソルの操作を試行するとエラーになります。カーソルを暗黙クローズする方法は、カーソルの宣言方法や、 COMMIT や ROLLBACK の有無などの、複数の要素によって異なります。

デフォルトでは、DB2 CLI ドライバーはすべてのカーソルを WITH HOLD として宣言します。したがって、オープン・カーソルは複数の COMMIT 間にわたって持続するので、アプリケーションは個々のカーソルを明示的にクローズする必要があります。しかしながら、自動コミット・モードでカーソルをクローズすると、WITH HOLD オプションで定義されていない他のオープン・カーソルはクローズされ、残りのオープン・カーソルはすべて位置指定にならないことに注意してください。 (つまり、別のフェッチを発行しないと、位置指定の更新や削除を実行できません。) カーソルが WITH HOLD として宣言されているかいないかを切り替えるには、以下の 2 つの方法があります。

- ステートメント属性 SQL_ATTR_CURSOR_HOLD を SQL_CURSOR_HOLD_ON (デフォルト) または SQL_CURSOR_HOLD_OFF に設定する。この設定は、ステ

ートメント・ハンドル上の、この値が設定された後にオープンされたカーソルだけに影響します。すでにオープンしているカーソルには影響はありません。

- CLI/ODBC 構成キーワード CursorHold を設定して、デフォルトの DB2 CLI ドライバーの振る舞いを変更する。 CursorHold=1 を設定すると、 WITH HOLD として宣言されているカーソルのデフォルトの振る舞いが保持されます。 CursorHold=0 を設定すると、個々のトランザクションのコミット時にカーソルがクローズされます。前述の SQL_ATTR_CURSOR_HOLD ステートメント属性を設定すると、このキーワードをオーバーライドできます。

注: ROLLBACK は、WITH HOLD として宣言されているカーソルを含むすべてのカーソルをクローズします。

スクロール可能カーソル・サポートの前に作成されたアプリケーションのトラブルシューティング:

スクロール可能カーソル・サポートは新しい機能であるため、 DB2 for OS/390®、 DB2 for Unix、 DB2 for Windows® の前のリリースを使用していた一部の CLI/ODBC アプリケーションでは、振る舞いまたはパフォーマンスが変わる可能性があります。スクロール可能カーソルを要求したアプリケーションは、スクロール可能カーソルがサポートされる前はフォワードのみカーソルを受け取っていたために、このようなことが起こります。スクロール可能カーソル・サポート前のアプリケーションの振る舞いをリストアするには、次の構成キーワードを `db2cli.ini` ファイルに設定します。

表 9. スクロール可能カーソル・サポート前のアプリケーションの振る舞いをリストアする構成キーワード値

構成キーワード設定	説明
`Patch2=6`	スクロール可能カーソル (キー・セット主導、動的、および静的) がサポートされていないというメッセージを返します。 CLI は、スクロール可能カーソルの要求をフォワードのみカーソルに自動的にダウングレードします。
`DisableKeysetCursor=1`	キー・セット主導スクロール可能カーソルを使用不可にします。これは、キー・セット主導カーソルまたは動的カーソルが要求された場合に、 CLI ドライバーによってアプリケーションが静的カーソルを提供することを強制するために使用されます。

関連概念:

- 35 ページの『CLI アプリケーションのコミット・モード』
- 73 ページの『CLI アプリケーションのカーソル』
- 79 ページの『CLI アプリケーションにおける結果セットの用語』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetInfo 関数 (CLI) - 一般情報の取得』
- 「*SQL リファレンス 第 2 巻*」の『COMMIT ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『ROLLBACK ステートメント』
- 313 ページの『CursorHold CLI/ODBC 構成キーワード』

- 323 ページの『DisableKeysetCursor CLI/ODBC 構成キーワード』
- 345 ページの『Patch2 CLI/ODBC 構成キーワード』

結果セット

CLI アプリケーションにおける結果セットの用語

結果の処理に関する用語を以下に示します。

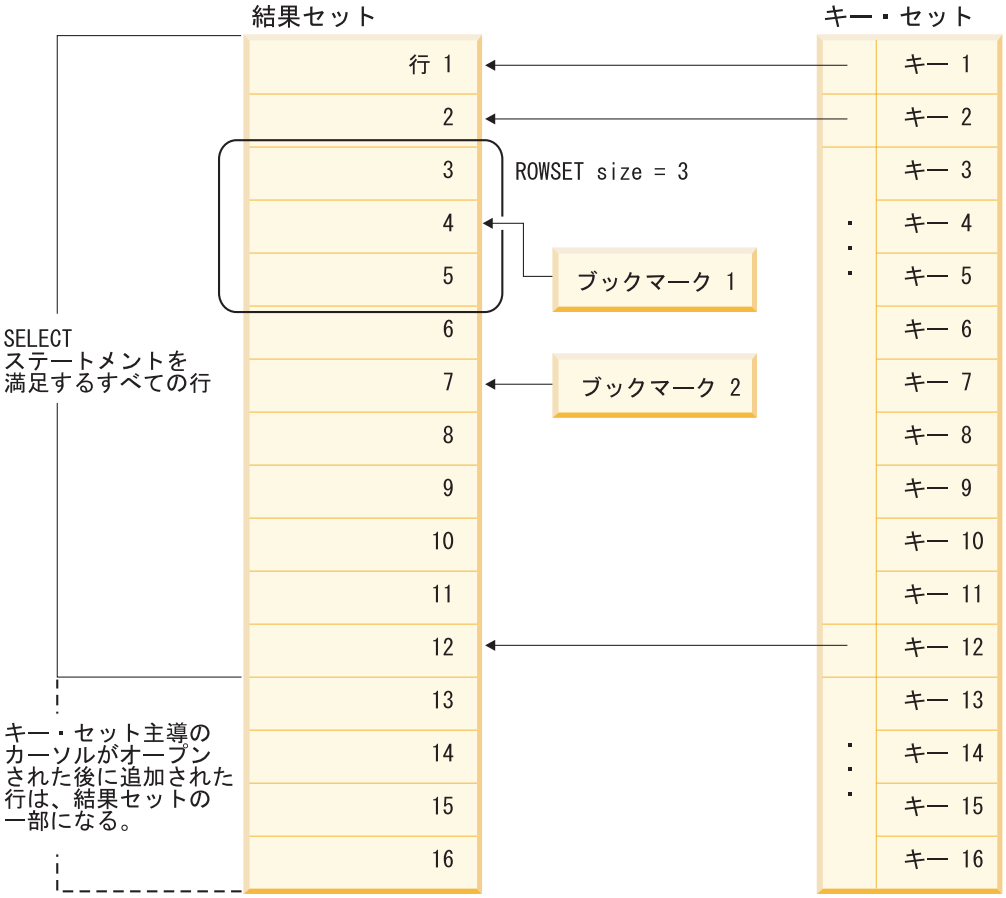
結果セット
SQL SELECT ステートメントを満たす行の完全セット。このセットから、検索行を取り出して行セットに移植します。

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。`SQLFetch()`、`SQLFetchScroll()`、または `SQLExtendedFetch()` 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
ブックマークといわれる、結果セットにある特定の行への参照を、保管することができます。一度保管すると、アプリケーションは結果セット全体を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。また `SQLBulkOperations()` による更新や削除を実行する場合にも、ブックマークを使用できます。

キー・セット
キー・セット主導カーソルに組み込まれている行のセットや順序の識別に使用する、キー値のセット。キー・セットは、初めてキー・セット主導カーソルをオープンする際に作成されます。結果セット全体をカーソル・スクロールする際に、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。

以下の図に、前述の用語の間の関係が示されています。



関連概念:

- 73 ページの『CLI アプリケーションのカーソル』
- 76 ページの『CLI アプリケーションのカーソルに関する考慮事項』
- 88 ページの『CLI アプリケーションのブックマーク』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExtendedFetch 関数 (CLI) - 拡張取り出し (行の配列の取り出し)』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 「*SQL リファレンス 第 2 巻*」の『SELECT ステートメント』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBulkOperations 関数 (CLI) - 行のセットの追加、更新、削除、または取り出し』

CLI アプリケーションでの行セット取り出しの例

一部の行セットの例:

行セットを処理する場合は、戻される結果セットのどの部分に意味のあるデータが入っているかを検証する必要があります。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、戻された行数を判別するために、行状況配列を検査する必要があります。これは行セットが行の完全セットを含んでいないという場合があるからです。たとえば、行セットのサイズが 10 に設定されている場合で、 SQL_FETCH_ABSOLUTE および -3 にセットされた *FetchOffset* を使用して `SQLFetchScroll()` を呼び出す場合を考えてください。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。

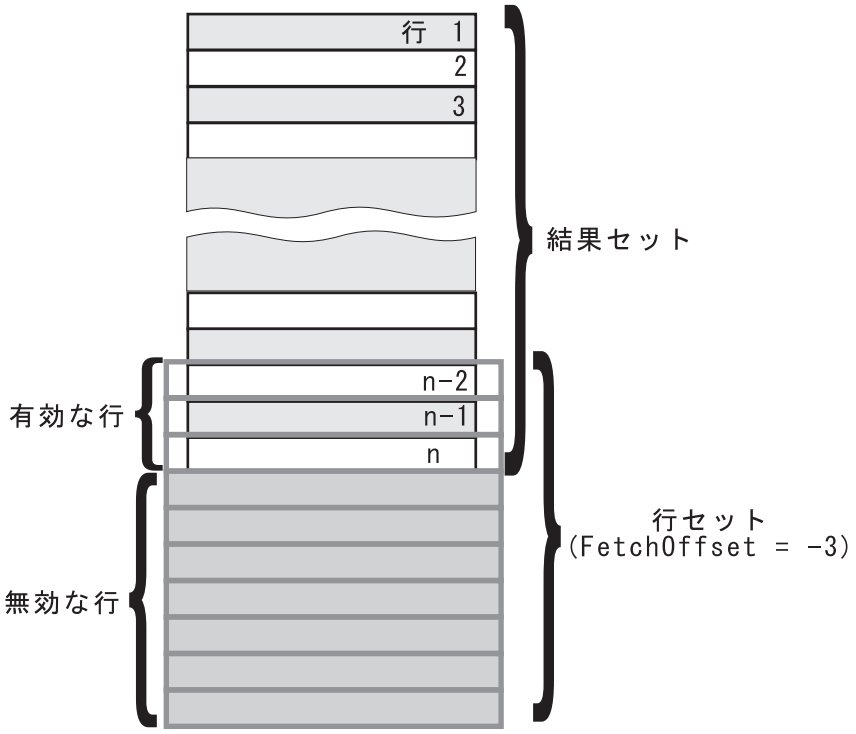


図 4. 一部の行セットの例

フェッチ・オリエンテーションの例:

以下の図は、いろいろな *FetchOrientation* 値を使用した、 `SQLFetchScroll()` への呼び出しを示しています。結果セットにはすべての行 (1 から n まで) 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。

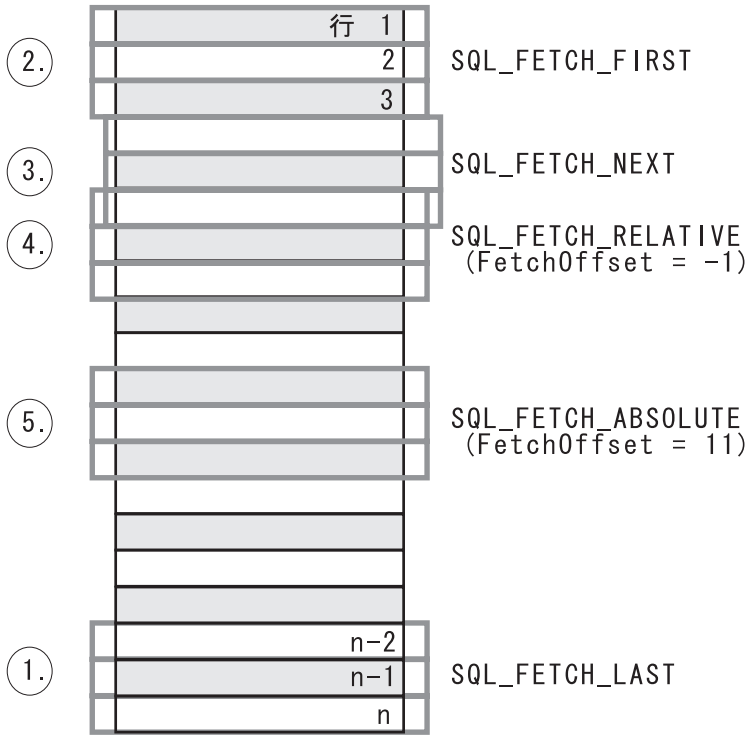


図 5. 行セットの取り出しの例

関連概念:

- 79 ページの『CLI アプリケーションにおける結果セットの用語』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』

結果セットから返される行セットの指定

データの取り出しを始める前に、返される行セットを確立する必要があります。このトピックでは、行セットのセットアップに関連したステップについて説明します。

前提条件:

行セットの指定を始める前に、CLI アプリケーションを初期設定してあることを確認してください。

手順:

DB2 CLI を使用すると、アプリケーションは、一度に複数の行にわたるスクロール不可カーソルまたはスクロール可能カーソル用に、行セットを指定できます。行セットを効果的に処理するには、アプリケーションは以下の作業を実行する必要があります。

1. ステートメント属性 SQL_ATTR_ROW_ARRAY_SIZE を行セット中の行数に設定して、 `SQLFetch()` または `SQLFetchScroll()` への呼び出しから返される行セ

ットのサイズを指定します。デフォルトの行数は 1 です。たとえば、35 行の行セットを宣言するには、以下の呼び出しを発行します。

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

2. 返される行数を保管する変数をセットアップします。タイプ SQLUINTEGER の変数を宣言し、この変数を指す SQL_ATTR_ROWS_FETCHED_PTR ステートメント属性を設定します。以下の例で、*rowsFetchedNb* には、 SQLFetchScroll() への各呼び出し後に行セットに返される行数が保持されます。

```
/* ... */

SQLUINTEGER rowsFetchedNb;

/* ... */

rc = SQLSetStmtAttr(hstmt,
                    SQL_ATTR_ROWS_FETCHED_PTR,
                    &rowsFetchedNb,
                    0);
```

3. 行状況の配列をセットアップします。行セットのサイズ (ステップ 1 で指定) と同じ行数を指定して、SQLUSMALLINT タイプの配列を宣言します。それから、ステートメント属性 SQL_ATTR_ROW_STATUS_PTR によりこの配列のアドレスを指定します。例:

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

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。SQLFetch() または SQLFetchScroll() への各呼び出し後に、配列は更新されます。 SQLFetch() または 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、行状況の配列の内容が未定義です。定義されている場合には、行状況の配列の値が返されます (値の完全なリストについては、 SQLFetchScroll() の資料中の、行状況の配列の項を参照してください)。

4. 行セットの開始位置を指示して、結果セット中の行セットの位置を指定します。この位置を指定するには、*FetchOrientation* および *FetchOffset* 値を指定して、SQLFetch() または SQLFetchScroll() を呼び出します。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。

```
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数に対する行セット・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。

注: アプリケーションが表示画面中のデータをバッファーに入れ、結果セットを再生成して更新を参照できる場合は、代わりに順方向カーソルを使用してください。こうすると、結果セットが小さくなり、パフォーマンスが向上します。

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
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

5. それぞれの行セットが作成された後、行フェッチ・ポインターをチェックして、返される行数を判別してください。それぞれの行の状況について、行状況の配列をチェック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場合があるからです。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。

   たとえば、行セットのサイズが 10 に設定されている場合で、SQL_FETCH_ABSOLUTE および -3 にセットされた *FetchOffset* を使用して `SQLFetchScroll()` を呼び出す場合を考えてください。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。

関連概念:

- 76 ページの『CLI アプリケーションのカーソルに関する考慮事項』
- 79 ページの『CLI アプリケーションにおける結果セットの用語』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』

- 85 ページの『CLI アプリケーションでのスクロール可能カーソルによるデータの取り出し』
- 89 ページの『CLI アプリケーションでのブックマークによるデータの取り出し』

関連資料:

- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 56 ページの『CLI 関数戻りコード』

関連サンプル:

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

CLI アプリケーションでのスクロール可能カーソルによるデータの取り出し

スクロール可能カーソルを使用すると、結果セットのどこにでも移動することができます。データを取り出す際に、この機能を使用できます。このトピックでは、スクロール可能カーソルを使用してデータを取り出す方法について説明します。

前提条件:

スクロール可能カーソルを使用してデータを取り出す場合は、その前に CLI アプリケーションを初期設定してあることを確認してください。

手順:

スクロール可能カーソルを使用してデータを取り出すには、以下のようにします。

1. ステートメント属性 SQL_ATTR_ROW_ARRAY_SIZE を行セット中の行数に設定して、返される行セットのサイズを指定します。デフォルトの行数は 1 です。たとえば、35 行の行セットを宣言するには、以下の呼び出しを発行します。

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

2. 使用するスクロール可能カーソルのタイプを指定します。 `SQLSetStmtAttr()` を使用して、静的な使用カーソルの場合は SQL_ATTR_CURSOR_TYPE ステートメント属性を SQL_CURSOR_STATIC に設定し、キー・セット主導カーソルの場合は SQL_CURSOR_KEYSET_DRIVEN に設定してください。例:

```
sqlrc = SQLSetStmtAttr (hstmt,
                        SQL_ATTR_CURSOR_TYPE,
                        (SQLPOINTER) SQL_CURSOR_STATIC,
                        0);
```

カーソルのタイプを設定しないと、デフォルトの順方向のみのスクロール不可カーソルが使用されます。

3. 返される行数を保管する変数をセットアップします。タイプ SQLUINTEGER の変数を宣言し、この変数を指す SQL_ATTR_ROWS_FETCHED_PTR ステートメント属性を設定します。以下の例で、*rowsFetchedNb* には、SQLFetchScroll() への各呼び出し後に行セットに返される行数が保持されます。

```
/* ... */

SQLUINTEGER rowsFetchedNb;

/* ... */

rc = SQLSetStmtAttr(hstmt,
                    SQL_ATTR_ROWS_FETCHED_PTR,
                    &rowsFetchedNb,
                    0);
```

4. 行状況の配列をセットアップします。行セットのサイズ (ステップ 1 で指定) と同じ行数を指定して、SQLUSMALLINT タイプの配列を宣言します。それから、ステートメント属性 SQL_ATTR_ROW_STATUS_PTR によりこの配列のアドレスを指定します。例:

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

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。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、行状況の配列の内容が未定義です。定義されている場合には、行状況の配列の値が返されます (値の完全なリストについては、 SQLFetchScroll() の資料中の、行状況の配列の項を参照してください)。

5. オプション: スクロール可能カーソルと共にブックマークを使用したい場合は、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。例:

```
sqlrc = SQLSetStmtAttr (hstmt,
                        SQL_ATTR_USE_BOOKMARKS,
                        (SQLPOINTER) SQL_UB_VARIABLE,
                        0);
```

6. SQL SELECT ステートメントを発行します。
7. SQL SELECT ステートメントを実行します。
8. 列方向または行方向のバインドを使用して、結果セットをバインドします。

9. 結果セットから行の行セットを取り出します。
   a. SQLFetchScroll() を呼び出して、結果セットからデータの行セットを取り出します。行セットの開始位置を指示して、結果セット中の行セットの位置を指定します。この位置を指定するには、*FetchOrientation* および *FetchOffset* 値を指定して、 SQLFetchScroll() を呼び出します。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。

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

   b. それぞれの行セットが作成された後、行状況の配列をチェックして、返される行数を判別してください。それは行セットが行の完全セットを含んでいない場合があるからです。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。

      たとえば、行セットのサイズが 10 に設定されている場合で、SQL_FETCH_ABSOLUTE および -3 にセットされた *FetchOffset* を使用して SQLFetchScroll() を呼び出す場合を考えてください。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。
   c. 返される行のデータを表示または操作する。
10. SQLCloseCursor() を呼び出してカーソルをクローズするか、SQL_HANDLE_STMT の *HandleType* を指定して SQLFreeHandle() を呼び出してステートメント・ハンドルを解放します。

    取り出しが終了するたびにステートメント・ハンドルを解放する必要はありません。後でアプリケーションが他のハンドルを解放する際に、ステートメント・ハンドルも解放することができます。

関連概念:

- 73 ページの『CLI アプリケーションのカーソル』
- 76 ページの『CLI アプリケーションのカーソルに関する考慮事項』
- 88 ページの『CLI アプリケーションのブックマーク』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 27 ページの『CLI アプリケーションでの SQL ステートメントの発行』

関連資料:

- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLCloseCursor 関数 (CLI) - カーソルのクローズとペンディング結果の廃棄』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』

- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『ステートメント属性 (CLI) のリスト』
- 56 ページの『CLI 関数戻りコード』

関連サンプル:

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

ブックマーク

CLI アプリケーションのブックマーク

スクロール可能カーソルの使用時に、ブックマークを使用して、結果セットにある任意の行への参照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たり、キー・セット・カーソルの使用時に行の更新や削除を行ったりします。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。

関連概念:

- 76 ページの『CLI アプリケーションのカーソルに関する考慮事項』
- 79 ページの『CLI アプリケーションにおける結果セットの用語』

関連資料:

- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『SQLGetData 関数 (CLI) - 列からのデータの取得』
- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『SQLSetPos 関数 (CLI) - 行セット (Rowset) 内のカーソル位置の設定』

CLI アプリケーションでのブックマークによるデータの取り出し

ブックマークは、スクロール可能カーソルの使用時に限り使用できますが、これを使用すると結果セット中の行に対する参照を保管できます。データを検索する際に、この機能の利点を活用できます。このトピックでは、ブックマークを使用してデータを検索する方法について説明します。

前提条件:

ブックマークを使用してデータを検索する場合は、その前に CLI アプリケーションを初期設定してあることを確認してください。 『CLI アプリケーションでのスクロール可能カーソルによるデータの取り出し』で説明されているステップに加えて、以下のステップを実行する必要があります。

手順:

ブックマークとスクロール可能カーソルを使用してデータを検索するには、以下のようにします。

1. SQL_ATTR_USE_BOOKMARKS ステートメント属性を SQL_UB_VARIABLE に設定して、ブックマークを使用することを指示します (まだ指示していない場合)。例:

```
sqlrc = SQLSetStmtAttr (hstmt,
                        SQL_ATTR_USE_BOOKMARKS,
                        (SQLPOINTER) SQL_UB_VARIABLE,
                        0);
```

2. SELECT ステートメントを実行し、SQLFetchScroll() を使用して行セットを検索した後に、行セット中のご希望の行からブックマーク値を取得します。取得するには、SQLSetPos() を呼び出して、行セット内のカーソルの位置を指定します。それから SQLGetData() を呼び出して、ブックマーク値を検索します。例:

```
sqlrc = SQLFetchScroll(hstmt, SQL_FETCH_ABSOLUTE, 15);
/* ... */
sqlrc = SQLSetPos(hstmt, 3, SQL_POSITION, SQL_LOCK_NO_CHANGE);
/* ... */
sqlrc = SQLGetData(hstmt, 0, SQL_C_LONG, bookmark.val, 4,
                   &bookmark.ind);
```

   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。

3. 次の SQLFetchScroll() への呼び出しに関するブックマーク位置を保管します。SQL_ATTR_FETCH_BOOKMARK ステートメント属性を、ブックマーク値を含む変数に設定します。たとえば、前述の例では *bookmark.val* にブックマーク値が保管されるので、呼び出し SQLSetStmtAttr() は以下のようになります。

```
sqlrc = SQLSetStmtAttr(hstmt,
                       SQL_ATTR_FETCH_BOOKMARK_PTR,
                       (SQLPOINTER) bookmark.val,
                       0);
```

4. ブックマークに基づいて行セットを検索します。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。

以下の SQLFetchScroll() への呼び出しは、ブックマークの付いた行から始まる行セットを検索します。

```
sqlrc = 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。たとえば、以下の呼び出しは、ブックマークの付いた行の 4 行後から始まる行セットを検索します。

```
sqlrc = SQLFetchScroll(hstmt, SQL_FETCH_BOOKMARK, 4);
```

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。

関連概念:

- 76 ページの『CLI アプリケーションのカーソルに関する考慮事項』
- 79 ページの『CLI アプリケーションにおける結果セットの用語』
- 88 ページの『CLI アプリケーションのブックマーク』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 85 ページの『CLI アプリケーションでのスクロール可能カーソルによるデータの取り出し』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetData 関数 (CLI) - 列からのデータの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetPos 関数 (CLI) - 行セット (Rowset) 内のカーソル位置の設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*の『ステートメント属性 (CLI) のリスト』

関連サンプル:

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

第 6 章 配列の入力および出力

配列の入力

列方向配列の入力を使用した CLI アプリケーションでのパラメーター・マーカーのバインド

別の値を指定しながら繰り返される SQL ステートメントを処理する場合、列方向配列の入力を使用して、大量の挿入、削除、または更新を実現できます。このようにすると、同じ SQL ステートメントで値ごとに `SQLExecute()` を繰り返し呼び出す必要はなくなるため、サーバーへのネットワーク・フローは少なくなります。列方向配列の入力を使用すると、記憶場所の配列をパラメーター・マーカーにバインドできます。別の配列が各パラメーターに対してバインドされます。

前提条件:

パラメーター・マーカーを列方向バインドでバインドする前に、 CLI アプリケーションを初期設定しておくようにします。

制限:

文字およびバイナリー入力データの場合は、アプリケーションが `SQLBindParameter()` 呼び出しの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、配列内の各エレメントの長さは C データ・タイプのサイズであると見なされます。

手順:

列方向配列の入力を使用してパラメーター・マーカーをバインドするには、以下のようにします。

1. SQL_ATTR_PARAMSET_SIZE ステートメント属性を指定した `SQLSetStmtAttr()` を呼び出して、配列のサイズ (挿入する行数) を指定します。
2. バインドするパラメーター・マーカーごとに、配列を初期設定して取り込みます。

注: 各配列には、少なくとも SQL_ATTR_PARAMSET_SIZE エレメントが含まれていなければなりません。含まれていない場合、メモリー・アクセス違反が生じる可能性があります。

3. オプション: SQL_ATTR_BIND_TYPE ステートメント属性を SQL_PARAMETER_BIND_BY_COLUMN に設定することにより (これは、デフォルト設定です)、列方向バインドを使用することを示します。
4. パラメーター・マーカーごとに `SQLBindParameter()` を呼び出すことにより、各パラメーター・マーカーを対応する入力値の配列にバインドします。

関連概念:

- 31 ページの『CLI アプリケーションでのパラメーター・マーカー・バインディング』
- 93 ページの『CLI アプリケーションでのパラメーター診断情報』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecute 関数 (CLI) - ステートメントの実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』

行方向配列の入力を使用した CLI アプリケーションでのパラメーター・マーカーのバインド

別の値を指定しながら繰り返される SQL ステートメントを処理する場合、行方向配列の入力を使用して、大量の挿入、削除、または更新を実現できます。このようにすると、同じ SQL ステートメントで値ごとに `SQLExecute()` を繰り返し呼び出す必要はなくなるため、サーバーへのネットワーク・フローは少なくなります。行方向配列の入力を使用すると、構造の配列をパラメーターにバインドできます。

前提条件:

パラメーター・マーカーを行方向バインドでバインドする前に、 CLI アプリケーションを初期設定しておくようにします。

手順:

行方向配列の入力を使用してパラメーター・マーカーをバインドするには、以下のようにします。

1. パラメーターごとに、2 つのエレメントを含む構造の配列を初期設定して取り込みます。最初のエレメントでは、長さ/標識バッファーを保持し、2 番目のエレメ

ントはその値を保持します。配列のサイズは、各パラメーターに適用される値の数に対応しています。たとえば、次の配列には、3 つのパラメーターの長さと値が入ります。

```
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];
```

2. SQLSetStmtAttr() を使用して、SQL_ATTR_PARAM_BIND_TYPE ステートメント属性を、前のステップで作成された構造の長さに設定することにより、行方向バインドを使用することを示します。
3. SQLSetStmtAttr() を使用し、ステートメント属性 SQL_ATTR_PARAMSET_SIZE を配列の行数に設定します。
4. SQLBindParameter() を使用し、各パラメーターを、ステップ 1 で作成した配列の最初の行にバインドします。たとえば、以下のようになります。

```
/* 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);
```

関連概念:

- 31 ページの『CLI アプリケーションでのパラメーター・マーカー・バインディング』
- 93 ページの『CLI アプリケーションでのパラメーター診断情報』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecute 関数 (CLI) - ステートメントの実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』

CLI アプリケーションでのパラメーター診断情報

パラメーター状況配列 とは、 CLI アプリケーションによって割り振られる 1 つ以上の SQLSMALLINT の配列のことです。配列中の個々のエレメントは、入力 (または出力) パラメーターの配列中のエレメントに対応します。 DB2 CLI ドライバー

を指定すると、 SQLExecute() または SQLExecDirect() 呼び出しに組み込まれているパラメーター・セットごとの処理状況に関する情報で、パラメーター状況配列が更新されます。

DB2 CLI は、パラメーター状況配列中のエレメントを以下の値で更新します。

- SQL_PARAM_SUCCESS: SQL ステートメントは、このパラメーターのセットに対して正常に実行されました。
- SQL_PARAM_SUCCESS_WITH_INFO: SQL ステートメントは、このパラメーターのセットに対して正常に実行されました。ただし、診断データ構造体の中に警告情報があります。
- SQL_PARAM_ERROR: このパラメーターのセットの処理中にエラーが生じました。診断データ構造体の中に追加のエラー情報があります。
- SQL_PARAM_UNUSED: このパラメーター・セットは使用できませんでした。前のパラメーター・セットのいずれかでエラーが発生し、処理が打ち切られたことが原因とみられます。
- SQL_PARAM_DIAG_UNAVAILABLE: 診断情報は使用できません。パラメーター・セットの使用前にエラーが検出されたことが原因とみられます (SQL ステートメント構文エラーなど)。

DB2 CLI がパラメーター状況配列を更新する前に、 CLI アプリケーションが SQLSetStmtAttr() 関数を呼び出して SQL_ATTR_PARAM_STATUS_PTR 属性を設定しなければなりません。その代わりに、アプリケーションは SQLSetDescField() 関数を呼び出して、パラメーター状況配列を指す IPD 記述子中の SQL_DESC_ARRAY_STATUS_PTR フィールドを設定することもできます。

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。

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。

関連タスク:

- 33 ページの『CLI アプリケーションでのパラメーター・マーカーのバインディング』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecute 関数 (CLI) - ステートメントの実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetDescField 関数 (CLI) - 記述子レコードの単一フィールドの設定』

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『記述子 FieldIdentifier 引き数の値 (CLI)』

オフセットを使用した CLI アプリケーションでのパラメーター・バインドの変更

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。

前提条件:

パラメーターのバインドを変更する前に、アプリケーションを初期設定するようにします。

手順:

オフセットを使用してパラメーターのバインドを変更するには、次のようにします。

1. パラメーターをバインドしたときに、`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. ステートメントを実行したときに、`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初期設定します。

   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. もう一度 `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 `SQLBindParameter()` 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メモリーのどこに保管されるかを判別します。
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。

関連概念:

- 73 ページの『CLI アプリケーションのカーソル』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 91 ページの『列方向配列の入力を使用した CLI アプリケーションでのパラメーター・マーカーのバインド』
- 92 ページの『行方向配列の入力を使用した CLI アプリケーションでのパラメーター・マーカーのバインド』

関連資料:

- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecute 関数 (CLI) - ステートメントの実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』

配列の出力

CLI アプリケーションでの列バインディング

列を以下の位置にバインドすることができます。

- アプリケーション・ストレージ

  `SQLBindCol()` は、アプリケーション・ストレージ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、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。

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。

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。通常は、SQLGetData() を使用するより、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。データが LOB 列に存在する場合は、SQLGetData() よりも LOB 関数のほうが望ましいでしょう。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() 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。

SQLBindCol() への複数の呼び出しの代わりに、 DB2 CLI は列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。

可変長列をバインドするときに、DB2 CLI は、 *StrLen_or_IndPtr* と *TargetValuePtr* を隣接して割り振る場合は、両方に一操作で書き込むことができます。たとえば、以下のようにします。

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

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。

関連概念:

- 111 ページの『CLI アプリケーションでの LOB ロケーター』

関連タスク:

- 103 ページの『CLI アプリケーションでの列バインドの相対位置による列バインドの変更』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindFileToCol 関数 (CLI) - LOB 列への LOB ファイル参照のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetData 関数 (CLI) - 列からのデータの取得』
- 「*SQL リファレンス 第 2 巻*」の『FREE LOCATOR ステートメント』

CLI アプリケーションでの結果セットの配列への取り出し

アプリケーションが行う最も一般的なタスクの 1 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。結果セットの各列または各行を配列内に保管することがアプリケーションで必要とされる場合は、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ストレージ域を割り当てなくてはなりません。

もう 1 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ストレージを割り当てることも行います。デフォルトでは、行のバインドは列方向です。これは `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同様です。 図 6 は、列方向バインドの論理ビューです。

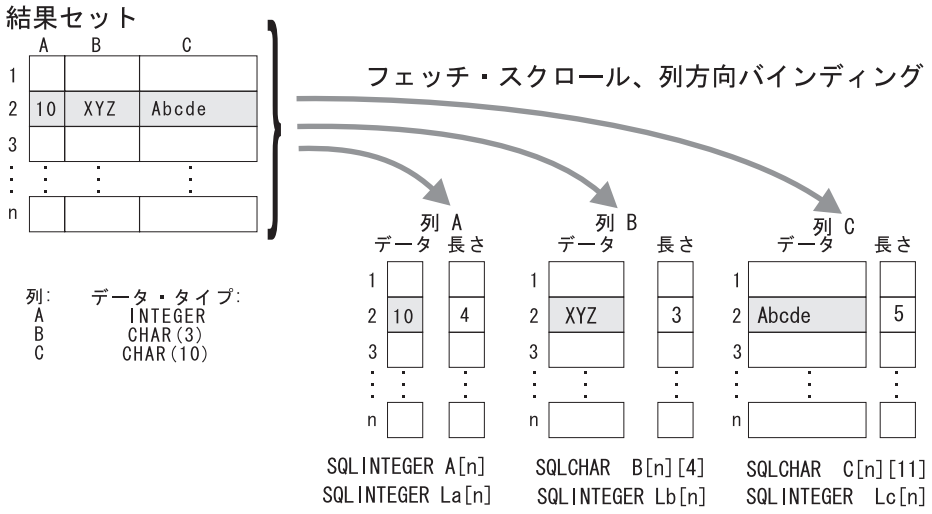


図 6. 列方向バインド

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。 図 7 は、行方向バインドを図示しています。

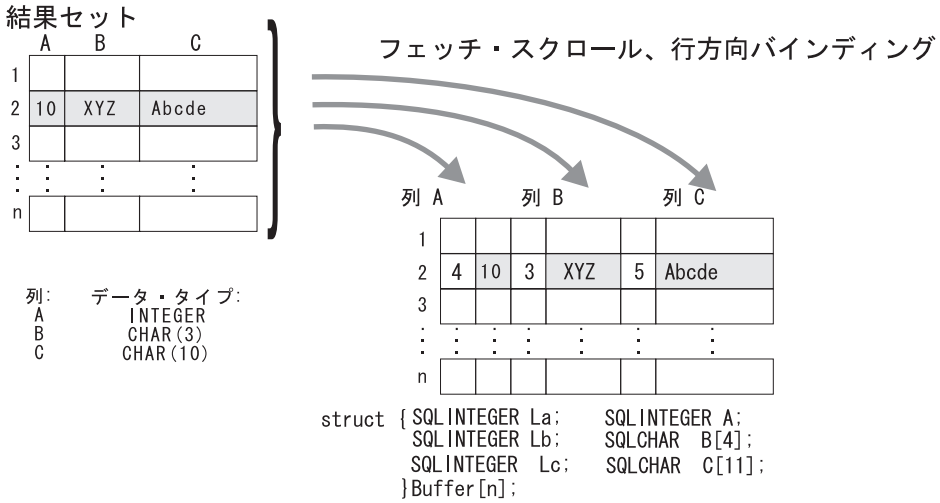


図 7. 行方向バインド

関連タスク:

- 100 ページの『CLI アプリケーションでの列方向バインドを使用した配列データの取り出し』
- 101 ページの『CLI アプリケーションでの行方向バインドを使用した配列データの取り出し』
- 103 ページの『CLI アプリケーションでの列バインドの相対位置による列バインドの変更』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』

CLI アプリケーションでの列方向バインドを使用した配列データの取り出し

データを取り出す際には、一度に複数の行を取り出して、配列中にデータを保管することもできます。配列中の個々のデータ行を取り出してコピーしたり、新しいストレージ域にバインドしたりする代わりに、列方向のバインドを使用して、一度に複数のデータ行を取り出せます。列方向のバインドは、個々のデータ値とその長さを配列中に保管するデフォルトの行バインド方式です。

前提条件:

列方向バインドを使用した配列中へのデータの取り出しを始める前に、 CLI アプリケーションを初期設定してあることを確認してください。

手順:

列方向バインドを使用してデータを取り出すには、以下のようにします。

1. 列データ値ごとに該当するデータ・タイプの配列を割り振ります。この配列は、取り出されたデータ値を保持します。
2. 列ごとに SQLINTEGER の配列を割り振ります。個々の配列は、個々の列のデータ値を保管します。
3. `SQLSetStmtAttr()` を使用し、 SQL_ATTR_ROW_BIND_TYPE ステートメント属性を SQL_BIND_BY_COLUMN に設定して、列方向の配列取り出しを使用することを指定します。
4. `SQLSetStmtAttr()` を使用し、SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性を設定して、取り出される行数を指定します。

   SQL_ATTR_ROW_ARRAY_SIZE 属性の値が 1 よりも大きいと、DB2 CLI は、据え置き出力データ・ポインターと長さポインターが、結果セットの列のデータと長さのある 1 つのエレメント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と認識します。
5. データの取り出しに使用する SQL ステートメントを準備して実行します。
6. 列ごとに `SQLBindCol()` を呼び出して、個々の配列をその列にバインドします。
7. `SQLFetch()` または `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。各エレメント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。

関連概念:

- 98 ページの『CLI アプリケーションでの結果セットの配列への取り出し』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 101 ページの『CLI アプリケーションでの行方向バインドを使用した配列データの取り出し』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』

関連サンプル:

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

CLI アプリケーションでの行方向バインドを使用した配列データの取り出し

データを取り出す際には、一度に複数の行を取り出して、配列中にデータを保管することもできます。配列中の個々のデータ行を取り出してコピーしたり、新しいストレージ域にバインドしたりする代わりに、行方向のバインドを使用して、複数のデータ行を取り出せます。行方向バインドは、結果セットの 1 行全体を 1 つの構造に関連付けます。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。

前提条件:

行方向バインドを使用した配列中へのデータの取り出しを始める前に、 CLI アプリケーションを初期設定してあることを確認してください。

手順:

行方向バインドを使用してデータを取り出すには、以下のようにします。

1. 取り出される行数に相当するサイズの構造体の配列を割り振ります。この構造体の個々のエレメントは、個々の行のデータ値と個々のデータ値の長さから成ります。

   たとえば、結果セットの個々の行が、タイプ INTEGER の Column A、タイプ CHAR(3) の Column B、タイプ CHAR(10) の Column C から成る場合には、以下の構造体を割り振ります (n は結果セット中の行数を表す)。

   ```
   struct { SQLINTEGER La; SQLINTEGER A;
            SQLINTEGER Lb; SQLCHAR B[4];
            SQLINTEGER Lc; SQLCHAR C[11];
          } buffer[n];
   ```

2. `SQLSetStmtAttr()` を使用し、SQL_ATTR_ROW_BIND_TYPE ステートメント属性を、結果列がバインドされる構造のサイズに設定して、行方向の配列取り出しを使用することを指定します。
3. `SQLSetStmtAttr()` を使用し、SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性を設定して、取り出される行数を指定します。
4. データの取り出しに使用する SQL ステートメントを準備して実行します。
5. 行の列ごとに `SQLBindCol()` を呼び出して、個々の構造体を行にバインドします。

   DB2 CLI は、`SQLBindCol()` の据え置き出力データ・ポインターを、構造の配列の先頭エレメント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われます。
6. `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 ステートメント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。

関連概念:

- 98 ページの『CLI アプリケーションでの結果セットの配列への取り出し』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 100 ページの『CLI アプリケーションでの列方向バインドを使用した配列データの取り出し』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』

関連サンプル:

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

CLI アプリケーションでの列バインドの相対位置による列バインドの変更

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を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。

前提条件:

列バインドの相対位置を使用して結果セットのバインドを変更する場合は、その前に CLI アプリケーションを初期設定してあることを確認してください。

制限:

この方式は、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。

手順:

列バインドの相対位置を使用して結果セットのバインドを変更するには、以下のようにします。

1. 通常どおり、`SQLBindCol()` を呼び出して、結果セットをバインドします。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()` を呼び出して、データを取り出します。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、次のデータ・セットを保持できるだけのスペースがある有効なメモリー・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ます。これにより、結果を保管するのがどのメモリーかが判別されます。
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。

関連概念:

- 96 ページの『CLI アプリケーションでの列バインディング』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 101 ページの『CLI アプリケーションでの行方向バインドを使用した配列データの取り出し』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』

第 7 章 大量データの操作

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

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ストレージ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ます。

前提条件:

実行時パラメーター値の指定を行う場合は、その前に CLI アプリケーションを初期設定してあることを確認してください。

制限:

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。

- 下記の `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、下記のループを強制終了します。
- `SQLGetDiagRec()` 関数。

手順:

実行時データ・パラメーターとは、`SQLExecute()` または `SQLExecDirect()` が呼び出される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です。 `SQLBindParameter()` 呼び出しでそのようなパラメーターを指定するには、次のようにします。

1. 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。例:

```
/* dtlob.c */
/* ... */
SQLINTEGER      blobInd ;
/* ... */
blobInd = SQL_DATA_AT_EXEC;
sqlrc = SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
                         SQL_BLOB, BUFSIZ, 0, (SQLPOINTER)inputParam,
                         BUFSIZ, &blobInd);
```

2. 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。
3. 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、下記のステップのように応答します。
4. 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。
5. 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。
6. 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。
7. 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 4 および 5 を繰り返します。

例:

```
/* dtlob.c */
/* ... */
    else
    {   sqlrc = SQLParamData( hstmt, (SQLPOINTER *) &valuePtr);
        /* ... */

        while ( sqlrc == SQL_NEED_DATA)
        {   /*
            if more than 1 parms used DATA_AT_EXEC then valuePtr would
            have to be checked to determine which param needed data
            */
            while ( feof( pFile ) == 0 )
            {   n = fread( buffer, sizeof(char), BUFSIZ, pFile);
                sqlrc = SQLPutData(hstmt, buffer, n);
                STMT_HANDLE_CHECK( hstmt, sqlrc);
                fileSize = fileSize + n;
                if ( fileSize > 102400u)
                {   /* BLOB column defined as 100K MAX */
                    /* ... */
                    break;
                }
            }
            /* ... */
            sqlrc = SQLParamData( hstmt, (SQLPOINTER *) &valuePtr);
            /* ... */
        }
    }
```

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() が返すはずであった戻り値および診断を作成します。

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLCancel 関数 (CLI) - ステートメントの取り消し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLParamData 関数 (CLI) - データ値が必要な次のパラメーターの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLPutData 関数 (CLI) - パラメーターのデータ値の引き渡し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』

関連サンプル:

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

CLI アプリケーションでの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情報 (SQLDescribeCol() への呼び出しなど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、文字およびバイナリー・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、 108 ページの図 8 のフロー・チャートの左側の分岐に示したように、 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。例:

```
/* dtlob.c */
/* ... */
sqlrc = SQLGetData(hstmt, 1, SQL_C_BINARY, (SQLPOINTER) buffer,
                   BUFSIZ, &bufInd);
/* ... */
while( sqlrc == SQL_SUCCESS_WITH_INFO || sqlrc == SQL_SUCCESS )
{   if ( bufInd > BUFSIZ)  /* full buffer */
    {   fwrite( buffer, sizeof(char), BUFSIZ, pFile);
    }
    else /* partial buffer on last GetData */
    {   fwrite( buffer, sizeof(char), bufInd, pFile);
```

```
        }

        sqlrc = SQLGetData( hstmt, 1, SQL_C_BINARY, (SQLPOINTER)buffer,
                            BUFSIZ, &bufInd);
        /* ... */
    }
```

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。長形式データを取り出す他の方式については、ラージ・オブジェクトの使用法に関する資料を参照してください。

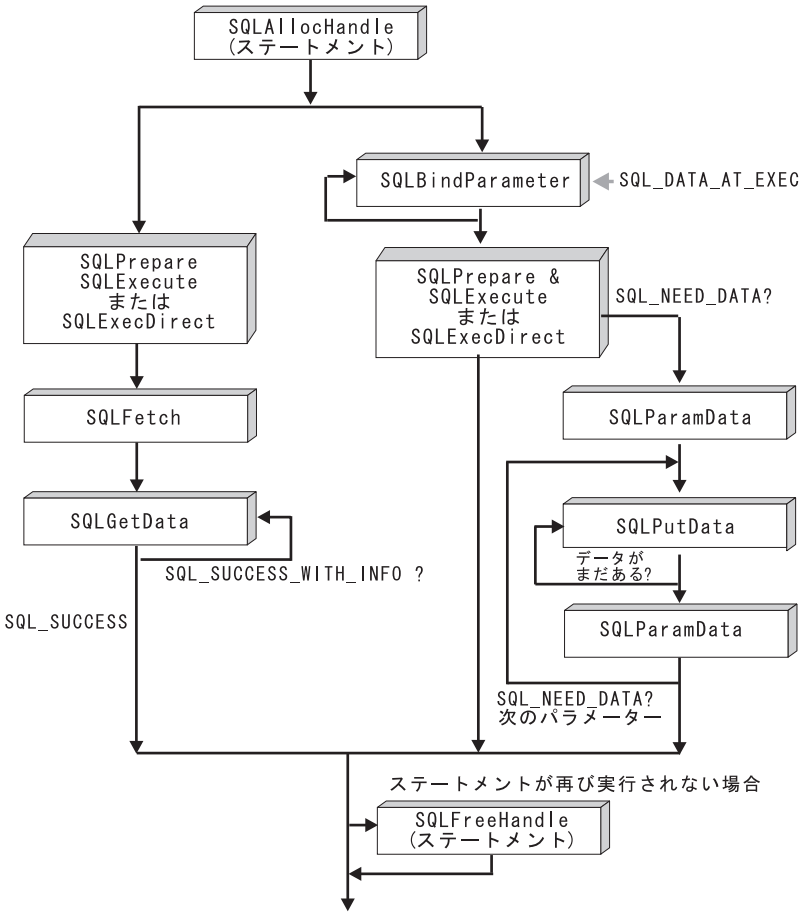


図 8. 分割入力および取り出し

関連概念:

- 109 ページの『CLI アプリケーションでのラージ・オブジェクトの使用』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetData 関数 (CLI) - 列からのデータの取得』

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetSubString 関数 (CLI) - ストリング値の部分的な取り出し』
- 56 ページの『CLI 関数戻りコード』

関連サンプル:

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

CLI アプリケーションでのラージ・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、ラージ・オブジェクトの任意のタイプを参照するのに使用されます。 3 つの LOB データ・タイプがあります。それはバイナリー・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・タイプはシンボルでそれぞれ、SQL_BLOB、SQL_CLOB、SQL_DBCLOB と表されます。 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 シンボリック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 LOB ロケーターを使ってセグメント単位で、または直接ファイル入出力を使って転送を行います。

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()` を呼び出すか、特定の LOB データ・タイプを指定して `SQLGetTypeInfo()` を呼び出してください。

110 ページの図 9 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。

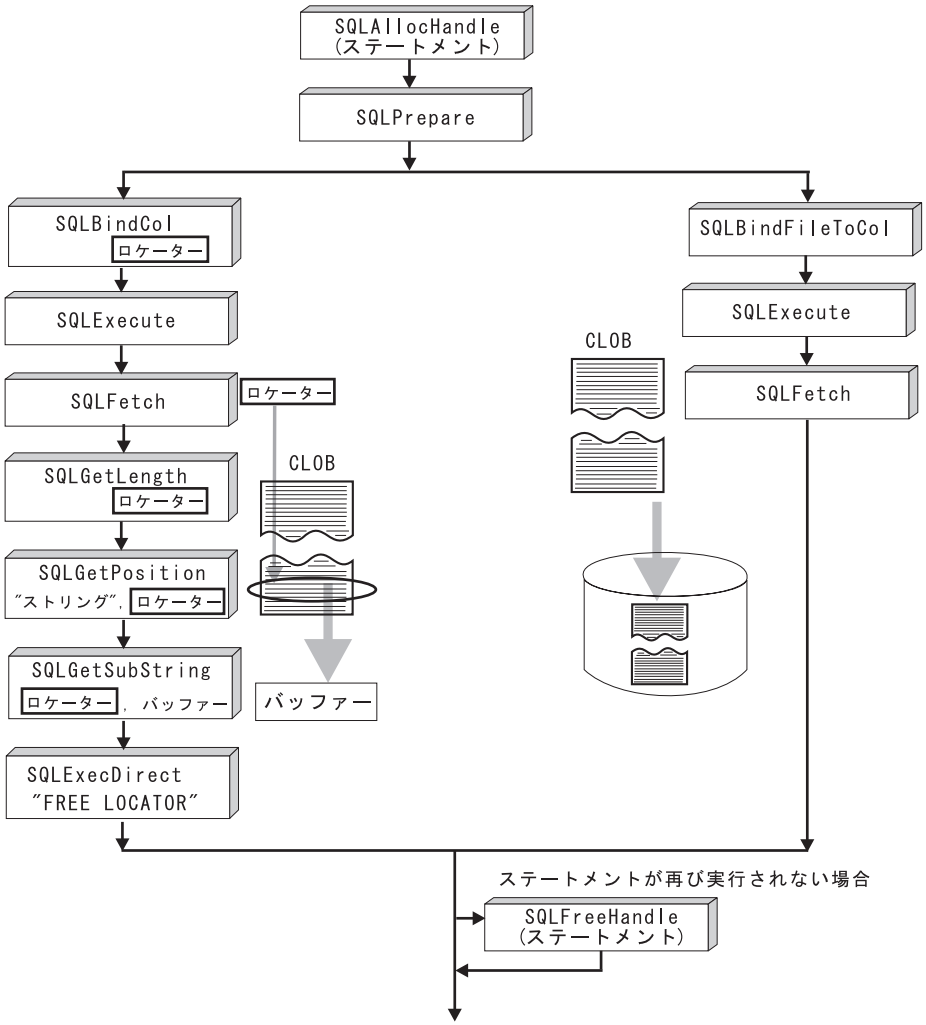


図 9. CLOB データの取り出し

関連概念:

- 107 ページの『CLI アプリケーションでのデータの分割取り出し』
- 111 ページの『CLI アプリケーションでの LOB ロケーター』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetData 関数 (CLI) - 列からのデータの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetFunctions 関数 (CLI) - 関数の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetTypeInfo 関数 (CLI) - データ・タイプ情報の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLPutData 関数 (CLI) - パラメーターのデータ値の引き渡し』

CLI アプリケーションでの LOB ロケーター

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、タイプ SQLINTEGER として定義される、ラージ・オブジェクトに効率よくランダム・アクセスするためのトークン値です。 LOB ロケーターを使用すると、サーバーは照会を実行し、結果セット中に LOB 列の値を入れる代わりに、 LOB の値に対応する整数で LOB ロケーターを更新します。その後アプリケーションが結果を要求する際にはサーバーにロケーターを渡し、サーバーは LOB 結果を返します。

LOB ロケーターはデータベース中に保管されません。 LOB ロケーターはトランザクション中に LOB 値を参照し、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。 LOB ロケーターは単純なトークン値で、行中の列ではなく、 1 つのラージ・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。

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。

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、データベース中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))
```

正規のデータ・タイプと LOB ロケーターとの間の違い:

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()` に指定されているか、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ストレージ・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)。

関連概念:

- 31 ページの『CLI アプリケーションでのパラメーター・マーカー・バインディング』
- 109 ページの『CLI アプリケーションでのラージ・オブジェクトの使用』

関連タスク:

- 113 ページの『CLI アプリケーションでの LOB ロケーターによる LOB データの取り出し』

関連資料:

- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetData 関数 (CLI) - 列からのデータの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetLength 関数 (CLI) - ストリング値の長さの取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetPosition 関数 (CLI) - ストリングの開始位置を戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetSubString 関数 (CLI) - ストリング値の部分的な取り出し』

CLI アプリケーションでの LOB ロケーターによる LOB データの取り出し

LOB ロケーターを使用して LOB データを取り出す場合の代表的なステップを以下に示します。個々のステップ中の例は、ロケーターを使用して CLOB データを検索することにより、 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方法を示しています。 LOB ロケーターが取り出され、それから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ます。それからこのサブストリングが検索されます。

前提条件:

LOB ロケーターを使用して LOB データを取り出す場合は、その前に CLI アプリケーションを初期設定してあることを確認してください。

手順:

LOB ロケーターを使用して LOB データを取り出すには、次のようにします。

1. SQLBindCol() または SQLGetData() 関数を使用して、 LOB ロケーターをアプリケーション変数中に取り出します。例:

```
SQLINTEGER  clobLoc ;
SQLINTEGER  pcbValue ;

/* ... */
sqlrc = SQLBindCol( hstmtClobFetch, 1, SQL_C_CLOB_LOCATOR,
                    &clobLoc, 0, &pcbValue);
```

2. SQLFetch() を使用してロケーターを取り出します。

```
sqlrc = SQLFetch( hstmtClobFetch );
```

3. SQLGetLength() を呼び出して、 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。例:

```
sqlrc = SQLGetLength( hstmtLocUse, SQL_C_CLOB_LOCATOR,
                      clobLoc, &clobLen, &ind ) ;
```

4. SQLGetPosition() を呼び出して、 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。例:

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

5. SQLGetSubString() を呼び出して、サブストリングを検索します。例:

```
sqlrc = SQLGetSubString( hstmtLocUse,
                         SQL_C_CLOB_LOCATOR,
                         clobLoc,
                         clobPiecePos,
                         clobLen - clobPiecePos,
                         SQL_C_CHAR,
```

```
                                buffer,
                                clobLen - clobPiecePos + 1,
                                &clobPieceLen,
                                &ind ) ;
```

6. ロケーターを解放します。トランザクションの終了時には、すべての LOB ロケーターが暗黙的に解放されます。 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。

   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 シンボル・データ・タイプに設定して、 SQLBindParameter() を使用します。たとえば、以下のようになります。

```
sqlrc = SQLSetParam( hstmtLocFree,
                     1,
                     SQL_C_CLOB_LOCATOR,
                     SQL_CLOB_LOCATOR,
                     0,
                     0,
                     &clobLoc,
                     NULL ) ;

/* ... */
sqlrc = SQLExecDirect( hstmtLocFree, stmtLocFree, SQL_NTS ) ;
```

関連概念:

- 111 ページの『CLI アプリケーションでの LOB ロケーター』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetData 関数 (CLI) - 列からのデータの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetLength 関数 (CLI) - ストリング値の長さの取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetPosition 関数 (CLI) - ストリングの開始位置を戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetSubString 関数 (CLI) - ストリング値の部分的な取り出し』
- 「*SQL リファレンス 第 2 巻*」の『FREE LOCATOR ステートメント』

関連サンプル:

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

CLI アプリケーションでの LOB 処理のための直接ファイル入出力

LOB ロケーターを使用するもう 1 つの方法として、アプリケーションで LOB 列の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 列の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。

例:

```
SQLUINTEGER     fileOption = SQL_FILE_OVERWRITE;
SQLINTEGER      fileInd = 0;
SQLSMALLINT     fileNameLength = 14;
/* ... */
SQLCHAR         fileName[14] = "";

/* ... */
rc = SQLBindFileToCol(hstmt, 1, fileName, &fileNameLength,
                      &fileOption, 14, NULL, &fileInd);
```

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。

例:

```
SQLUINTEGER     fileOption = SQL_FILE_OVERWRITE;
SQLINTEGER      fileInd = 0;
SQLSMALLINT     fileNameLength = 14;
/* ... */
SQLCHAR         fileName[14] = "";

/* ... */

rc = SQLBindFileToParam(hstmt, 3, SQL_BLOB, fileName,
        &fileNameLength, &fileOption, 14, &fileInd);
```

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 (オペレーティング環境の) 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。

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。 LOB をサポートしているサーバーに対しては、 `SQLBindFileToParam()` はサポートされていないからです。

関連概念:

- 109 ページの『CLI アプリケーションでのラージ・オブジェクトの使用』
- 111 ページの『CLI アプリケーションでの LOB ロケーター』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindFileToCol 関数 (CLI) - LOB 列への LOB ファイル参照のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindFileToParam 関数 (CLI) - LOB パラメーターへの LOB ファイル参照のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetFunctions 関数 (CLI) - 関数の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLPutData 関数 (CLI) - パラメーターのデータ値の引き渡し』

関連サンプル:

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

ODBC アプリケーションでの LOB の使用法

既存の ODBC 準拠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 構成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、引き続きこれらの ODBC 準拠アプリケーションから LOB 列にアクセスすることもできます。こうすると、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。 LOBMaxColumnSize 構成キーワードを使用すると、 LOB データ・タイプのデフォルトの COLUMN_SIZE をオーバーライドできます。

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 のデフォルトは、SQL_LD_COMPAT_NO です。マッピングは有効ではありません。

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 データの取り出しや入力を行うことができます。

関連概念:

- 109 ページの『CLI アプリケーションでのラージ・オブジェクトの使用』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetData 関数 (CLI) - 列からのデータの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLPutData 関数 (CLI) - パラメーターのデータ値の引き渡し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 334 ページの『LOBMaxColumnSize CLI/ODBC 構成キーワード』
- 335 ページの『LongDataCompat CLI/ODBC 構成キーワード』

バルク・データの操作

CLI アプリケーションでのバルク挿入およびバルク更新用の長いデータ

`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。

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 を戻して次のイベントに進みます。これについては、次の項目で説明します。 (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 (関数シーケンス・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。

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetInfo 関数 (CLI) - 一般情報の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLParamData 関数 (CLI) - データ値が必要な次のパラメーターの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLPutData 関数 (CLI) - パラメーターのデータ値の引き渡し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBulkOperations 関数 (CLI) - 行のセットの追加、更新、削除、または取り出し』
- 56 ページの『CLI 関数戻りコード』

CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの検索

ブックマークおよび DB2 CLI `SQLBulkOperations()` 関数を使用して、バルク・データを検索する (取り出す) ことができます。

前提条件:

ブックマークおよび `SQLBulkOperations()` を使用してバルク・データを取り出す前に、 CLI アプリケーションを初期化しておいてください。

制限:

DB2 CLI でのブックマークは、カーソルのクローズ操作後も保持されることはありません。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。

手順:

`SQLBulkOperations()` を使用してブックマークによるバルク・フェッチを実行するには、以下のようにします。

1. `SQLSetStmtAttr()` を使用して、 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. `SQLSetStmtAttr()` を使用して、 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. 取り出したいデータをバインドするために、`SQLBindCol()` を呼び出す。

   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。
5. 列 0 (ブックマーク列) をバインドするために、`SQLBindCol()` を呼び出す。
6. 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。

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 NULL ポインターでなければなりません。

7. *Operation* 引き数に SQL_FETCH_BY_BOOKMARK を指定して `SQLBulkOperations()` を呼び出し、データを取り出す。

   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。

関連概念:

- 88 ページの『CLI アプリケーションのブックマーク』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 120 ページの『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの挿入』
- 124 ページの『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの削除』
- 122 ページの『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの更新』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBulkOperations 関数 (CLI) - 行のセットの追加、更新、削除、または取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』

CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの挿入

`SQLBulkOperations()` を使用して、ブックマークによるバルク・データの挿入を実行できます。

前提条件:

`SQLBulkOperations()` を使用してバルク・データを挿入する前に、CLI アプリケーションを初期化しておいてください。

制限:

DB2 CLI でのブックマークは、カーソルのクローズ操作後も保持されることはありません。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。

手順:

`SQLBulkOperations()` を使用してバルク・データ挿入を実行するには、以下のようにします。

1. `SQLSetStmtAttr()` を使用して、 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. `SQLSetStmtAttr()` を使用して、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。
4. 挿入したいデータをバインドするために、`SQLBindCol()` を呼び出す。

   データは、直前のステップで設定した 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 NULL ポインターでなければなりません。
5. *Operation* 引き数に SQL_ADD を指定して `SQLBulkOperations()` を呼び出し、データを挿入する。

   CLI は、新しく挿入された行のブックマーク値を使用して、バインドされた列 0 のバッファーを更新します。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。

   注: *Operation* 引き数に SQL_ADD を指定した `SQLBulkOperations()` を、重複した列を含むカーソルに対して呼び出した場合、エラーが戻されます。

関連概念:

- 88 ページの『CLI アプリケーションのブックマーク』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 119 ページの『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの検索』
- 124 ページの『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの削除』
- 122 ページの『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの更新』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBulkOperations 関数 (CLI) - 行のセットの追加、更新、削除、または取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』

CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの更新

`SQLBulkOperations()` を使用して、ブックマークによるバルク・データの更新を実行できます。

前提条件:

バルク・データを更新する前に、CLI アプリケーションを初期化しておいてください。

制限:

DB2 CLI でのブックマークは、カーソルのクローズ操作後も保持されることはありません。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。

手順:

バルク・データを更新するには、以下のようにします。

1. `SQLSetStmtAttr()` を使用して、 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. `SQLSetStmtAttr()` を使用して、 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。
4. 更新したいデータをバインドするために、`SQLBindCol()` を呼び出す。

   データは、直前のステップで設定した 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。
5. `SQLBindCol()` を呼び出して、ブックマーク列を列 0 にバインドする。
6. 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。
7. 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 NULL ポインターでなければなりません。

8. *Operation* 引き数に SQL_UPDATE_BY_BOOKMARK を指定して `SQLBulkOperations()` を呼び出し、データを更新する。

   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。

9. オプション: *Operation* 引き数に SQL_FETCH_BY_BOOKMARK を指定した `SQLBulkOperations()` を呼び出すことによって更新が行われたことを確認する。これによって、バインドされたアプリケーション・バッファーにデータが取り出されます。

   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。

   注: 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。

関連概念:

- 88 ページの『CLI アプリケーションのブックマーク』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 120 ページの『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの挿入』
- 119 ページの『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの検索』
- 124 ページの『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの削除』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBulkOperations 関数 (CLI) - 行のセットの追加、更新、削除、または取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』

CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの削除

SQLBulkOperations() とブックマークを使用して、データを大量に削除できます。

前提条件:

バルク・データを削除する前に、CLI アプリケーションを初期化しておいてください。

制限:

DB2 CLI でのブックマークは、カーソルのクローズ操作後も保持されることはありません。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。

手順:

ブックマークと SQLBulkOperations() を使用してバルク削除を実行するには、以下のようにします。

1. SQLSetStmtAttr() を使用して、 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、ブックマーク列を列 0 にバインドする。
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 NULL ポインターでなければなりません。
6. *Operation* 引き数に SQL_DELETE_BY_BOOKMARK を指定して SQLBulkOperations() を呼び出し、削除を実行する。

   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。

関連概念:

- 88 ページの『CLI アプリケーションのブックマーク』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 120 ページの『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの挿入』
- 119 ページの『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの検索』

- 122 ページの『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの更新』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBulkOperations 関数 (CLI) - 行のセットの追加、更新、削除、または取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』

CLI アプリケーションでの CLI LOAD ユーティリティーによるデータのインポート

CLI LOAD 機能は、CLI から IBM DB2 LOAD ユーティリティーへのインターフェースを設けます。この機能を使用すると、配列を挿入する代わりに、LOAD を使用して CLI 中のデータを挿入できます。大量のデータを挿入する必要がある場合に、このオプションを使用するとパフォーマンスの面で大きな利点が生じます。このインターフェースは LOAD を呼び出すので、LOAD を使用する際の考慮事項が、 CLI LOAD インターフェースを使用する際にも考慮される必要があります。

前提条件:

CLI LOAD ユーティリティーを使用してデータをインポートする場合は、その前に CLI アプリケーションを初期設定してあることを確認してください。

制限:

- IBM DB2 LOAD ユーティリティーとは違って、 CLI LOAD ユーティリティーは入力ファイルから直接データをロードしない。その代わりに、必要に応じて、アプリケーションはデータを入力ファイルから取り出して、準備したステートメント中のパラメーター・マーカーに対応する当該アプリケーション・パラメーター中に挿入する必要があります。
- ロード・ユーティリティーはアトミシティを排除するので、データの追加は非アトミックである。 LOAD は、渡された行をすべて正常に挿入できる訳ではありません。たとえば、行を挿入するとユニーク・キーの制約に対する違反が生じる場合は、 LOAD はこの行を挿入しませんが、残りの行のロードを続行します。
- データを挿入するための準備済み SQL ステートメントにおいて、 INSERT ステートメント中で VALUES 文節の代わりに全選択を使用する場合、その SQL ステートメントには、ターゲット表内のすべての列のパラメーター・マーカーが含まれていなければなりません。
- COMMIT が LOAD によって発行される。したがって、データの挿入が正常に完了したら、 LOAD やトランザクション中の他のステートメントをロールバックできません。

- CLI LOAD インターフェースに関して報告されるエラーは、配列を挿入する際のエラーとは違う。特定の行に関するエラーなどの重大でないエラーや警告は、LOAD メッセージ・ファイルだけに示されます。

手順:

CLI LOAD ユーティリティーを使用してデータをインポートするには、以下のようにします。

1. 以下のサポートされている値のいずれかを指定して、 `SQLSetStmtAttr()` 中にステートメント属性 SQL_ATTR_USE_LOAD_API を指定します。

   SQL_USE_LOAD_INSERT
   LOAD ユーティリティーを使用して、表中の既存のデータに追加します。

   SQL_USE_LOAD_REPLACE
   LOAD ユーティリティーを使用して、表中の既存のデータを置き換えます。

   たとえば、以下の呼び出しは、 CLI LOAD ユーティリティーを使用して表中の既存のデータに追加することを指示します。

   ```
   SQLSetStmtAttr (hStmt, SQL_ATTR_USE_LOAD_API,
                   (SQLPOINTER) SQL_USE_LOAD_INSERT, 0);
   ```

   注: SQL_USE_LOAD_INSERT または SQL_USE_LOAD_REPLACE を設定し、SQL_USE_LOAD_OFF を設定しないと、以下を除く CLI 関数は呼び出せません (下記のステップ 3 を参照)。

   - SQLBindParameter()
   - SQLExecute()
   - SQLExtendedBind()
   - SQLParamOptions()
   - SQLSetStmtAttr()

2. タイプ db2LoadStruct の構造体を作成し、この構造体を使用してご希望のロード・オプションを指定します。 SQL_ATTR_LOAD_INFO ステートメント属性をこの構造体を指すポインターに設定します。

3. 挿入するデータのために準備した SQL ステートメントに対して、`SQLExecute()` を発行します。その INSERT SQL ステートメントとしては、SELECT ステートメントを使用して表からデータをロードする全選択を使用できます。 INSERT ステートメントの 1 回の実行で、SELECT のすべてのデータがロードされます。以下の例は、全選択ステートメントを使用して 1 つの表のデータを別の表にロードする方法を示すものです。

   ```
   SQLPrepare (hStmt,
               (SQLCHAR *) "INSERT INTO tableB SELECT * FROM tableA",
        SQL_NTS);
   SQLExecute (hStmt);
   ```

4. SQL_USE_LOAD_OFF を指定して `SQLSetStmtAttr()` を呼び出します。呼び出すと、LOAD ユーティリティーを使用したデータの処理が終了します。以後 SQL_ATTR_USE_LOAD_API を再設定しない限り、正規の CLI 配列の挿入が有効になります (ステップ 1 を参照)。

5. オプション: 以下のステートメント属性のいずれかを指定した `SQLGetStmtAttr()` を呼び出すことによって、完了した CLI LOAD 操作の結果を照会します。

- SQL_ATTR_LOAD_ROWS_COMMITTED_PTR: 処理された合計行数を表す整数へのポインター。この値は、正常にロードされ、データベースにコミットされた行数と、スキップおよび拒否された行数の合計と等しくなります。
- SQL_ATTR_LOAD_ROWS_DELETED_PTR: 削除された重複行数を表す整数へのポインター。
- SQL_ATTR_LOAD_ROWS_LOADED_PTR: ターゲット表にロードされた行数を表す整数へのポインター。
- SQL_ATTR_LOAD_ROWS_READ_PTR: 読み取られた行数を表す整数へのポインター。
- SQL_ATTR_LOAD_ROWS_REJECTED_PTR: ロードできなかった行数を表す整数へのポインター。
- SQL_ATTR_LOAD_ROWS_SKIPPED_PTR: CLI LOAD 操作が開始される前にスキップされた行数を表す整数へのポインター。

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 91 ページの『列方向配列の入力を使用した CLI アプリケーションでのパラメーター・マーカーのバインド』
- 92 ページの『行方向配列の入力を使用した CLI アプリケーションでのパラメーター・マーカーのバインド』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 「*コマンド・リファレンス*」の『LOAD コマンド』
- 「*管理 API リファレンス*」の『db2Load - ロード』

関連サンプル:

- 『tbload.c -- How to insert data using the CLI LOAD utility 』

第 8 章 ストアード・プロシージャー

CLI アプリケーションからのストアード・プロシージャーの呼び出し

CLI アプリケーションは、CALL プロシージャー SQL ステートメントを実行することにより、ストアード・プロシージャーを呼び出します。このトピックでは、CLI アプリケーションからストアード・プロシージャーを呼び出す方法を説明します。

前提条件:

ストアード・プロシージャーを呼び出す前に、CLI アプリケーションを初期設定しておくようにします。

制限:

呼び出されるストアード・プロシージャーがカタログされていない場合、 CLI スキーマ関数のいずれも呼び出さないことを確認してください。カタログされていないストアード・プロシージャーからの CLI スキーマ関数の呼び出しはサポートされていません。

CLI スキーマ関数は、以下のとおりです。 `SQLColumns()`、`SQLColumnPrivileges()`、`SQLForeignKeys()`、`SQLPrimaryKeys()`、`SQLProcedureColumns()`、`SQLProcedures()`、`SQLSpecialColumns()`、`SQLStatistics()`、`SQLTables()`、および `SQLTablePrivileges()`。

手順:

ストアード・プロシージャーを呼び出すには、以下のようにします。

1. ストアード・プロシージャーの IN、INOUT、および OUT パラメーターにそれぞれ対応するアプリケーション・ホスト変数を宣言します。アプリケーションの変数データのタイプと長さが、ストアード・プロシージャーのシグニチャーのデータ・タイプと引き数の長さに一致することを確認します。 DB2 CLI は、すべての SQL タイプをパラメーター・マーカーとして使用して、ストアード・プロシージャーを呼び出すことをサポートしています。
2. IN、INOUT、および OUT パラメーターのアプリケーション変数を初期設定します。
3. CALL SQL ステートメントを発行します。以下に例を示します。

   ```
   SQLCHAR *stmt = (SQLCHAR *)"CALL OUT_LANGUAGE (?)";
   ```

   パフォーマンスを最高にするために、アプリケーションでは、 CALL プロシージャー・ストリングの中でストアード・プロシージャー引き数のパラメーター・マーカーを使用してから、ホスト変数をこれらのパラメーター・マーカーにバインドする必要があります。ただし、インバウンド・ストアード・プロシージャー引き数を、パラメーター・マーカーではなく、ストリング・リテラルとして指定しなければならない場合、CALL プロシージャー・ステートメントに、 ODBC 呼び出しエスケープ文節の区切り文字 { } を含めます。以下に例を示します。

```
SQLCHAR *stmt = (SQLCHAR *)"{CALL IN_PARAM (123, 'Hello World!')}";
```

CALL プロシージャー・ステートメントでストリング・リテラルおよび ODBC エスケープ文節が使用される場合、 IN モード・ストアード・プロシージャー引き数として、ストリング・リテラルだけを指定できます。 INOUT および OUT モード・ストアード・プロシージャー引き数は、引き続きパラメーター・マーカーを使用して指定する必要があります。

4. オプション: SQLPrepare() を呼び出して CALL ステートメントを準備します。
5. SQLBindParameter() を呼び出して、CALL プロシージャー・ステートメントの各パラメーターをバインドします。

   注: 各パラメーターが (SQL_PARAM_INPUT、SQL_PARAM_OUTPUT、または SQL_PARAM_INPUT_OUTPUT に対して) 正しくバインドされたことを確認します。正しくバインドされていないと、CALL プロシージャー・ステートメントが実行されるときに、予期しない結果が生じる可能性があります。たとえば、入力パラメーターが、SQL_PARAM_OUTPUT の *InputOutputType* を使用して、不正確にバインドされる場合に、このことが生じます。
6. SQLExecDirect() を使用して CALL プロシージャー・ステートメントを実行するか、ステップ 4 で CALL プロシージャー・ステートメントを準備済みの場合には、 SQLExecute() を使用して実行します。

   注: ストアード・プロシージャーを呼び出したアプリケーションかスレッドが、そのストアード・プロシージャーの完了前に終了する場合、ストアード・プロシージャーの実行も終了します。ストアード・プロシージャーが早めに終了してしまう場合にも、データベースは一貫した状態と望ましい状態を保つようなロジックを、そのストアード・プロシージャーに含めることは大切です。
7. 関数が戻されるときに SQLExecDirect() または SQLExecute() の戻りコードを調べ、 CALL プロシージャー・ステートメントまたはストアード・プロシージャーのいずれかの実行時に、何らかのエラーが発生していないかを判別します。戻りコードが SQL_SUCCESS_WITH_INFO か SQL_ERROR である場合、 CLI 診断関数 SQLGetDiagRec() および SQLGetDiagField() を使用して、エラーが発生した理由を判別します。

   ストアード・プロシージャーを正常に実行した場合、OUT パラメーターとしてバインドされた変数には、そのストアード・プロシージャーが CLI アプリケーションに戻したデータが含まれる可能性があります。該当する場合には、ストアード・プロシージャーは、スクロール不可カーソルを使用して、 1 つ以上の結果セットを戻す場合もあります。 CLI アプリケーションでは、SELECT ステートメントの実行によって生成された結果セットを処理するときに、ストアード・プロシージャーの結果セットを処理する必要があります。

   注: CLI アプリケーションが、ストアード・プロシージャーによって戻された結果セットに示された、パラメーターの番号またはタイプが分からない場合、その結果セットに対して、SQLNumResultCols()、 SQLDescribeCol()、および SQLColAttribute() 関数を (この順序で) 呼び出して、この情報を判別することができます。

CALL ステートメントを実行したら、該当する場合には、ストアード・プロシージャーから結果セットを検索できます。

注:

データベースの作成または移行時に、DB2 CLI パッケージは、自動的にデータベースにバインドされます。ただし、FixPak がクライアントまたはサーバーのいずれかに適用される場合、以下のコマンドを発行することによって db2cli.lst を再バインドする必要があります。

UNIX

```
db2 bind <BNDPATH>/@db2cli.lst blocking all grant public
```

Windows

```
db2 bind "%DB2PATH%¥bnd¥@db2cli.lst" blocking all grant public
```

関連概念:

- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『ルーチン: プロシージャー』
- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『DB2 ストアード・プロシージャー』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 233 ページの『CLI 環境のセットアップ』
- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 33 ページの『CLI アプリケーションでのパラメーター・マーカーのバインディング』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecute 関数 (CLI) - ステートメントの実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 「*SQL リファレンス 第 2 巻*」の『CALL ステートメント』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』
- 56 ページの『CLI 関数戻りコード』
- 227 ページの『DB2 CLI のバインド・ファイルおよびパッケージ名』
- 132 ページの『DB2 CLI ストアード・プロシージャー・コミット動作』

関連サンプル:

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

- 『spclient.c -- Call various stored procedures』
- 『spclires.c -- Contrast stored procedure multiple result set handling methods』
- 『spserver.c -- Definition of various types of stored procedures』

DB2 CLI ストアード・プロシージャー・コミット動作

DB2 サーバーで実行されている DB2 CLI クライアント・アプリケーションとコールされたストアード・プロシージャーの両方での SQL ステートメントのコミット動作は、そのアプリケーションおよびストアード・プロシージャーで適用されるコミットの組み合わせによります。可能な組み合わせおよび、その結果のコミット動作が、以下の表に説明されています。

表 10. DB2 CLI ストアード・プロシージャー・コミット動作

CLI クライアント	ストアード・プロシージャー	コミット動作
自動コミット ON	自動コミット ON	ストアード・プロシージャー内の正常に実行されたすべての SQL ステートメントは、ストアード・プロシージャー内の他の SQL ステートメントが失敗し、 CALL ステートメントにエラーまたは警告の SQLCODE が返された場合でも、コミットされます。
自動コミット ON	自動コミット OFF	ストアード・プロシージャーが SQLCODE >= 0 を返した場合、ストアード・プロシージャー内のすべての正常に実行された SQL ステートメントはコミットされます。そうでない場合、ストアード・プロシージャー内のすべての SQL ステートメントはロールバックされます。
自動コミット ON	マニュアル・コミット	手動でコミットされた、ストアード・プロシージャー内の正常に実行されたすべての SQL ステートメントは、 CALL ステートメントにエラー SQLCODE が戻された場合でも、ロールバックされません。 **注:** ストアード・プロシージャーが SQLCODE >= 0 を戻した場合、最後の手動コミットの後に発生したストアード・プロシージャー内のすべての正常に実行された SQL ステートメントは、コミットされます。そうでない場合は、手動コミット時点までロールバックされます。
自動コミット OFF	自動コミット ON	ストアード・プロシージャー内の正常に実行されたすべての SQL ステートメントは、 CALL ステートメントにエラー SQLCODE が戻された場合でもコミットされ、ロールバックはされません。さらに、CALL ステートメントを含む、それまでの CLI クライアント・アプリケーション内の、正常に実行されコミットされていないすべての SQL ステートメントはコミットされます。 **注:** CALL ステートメントの発行後は、トランザクションを完全にロールバックすることはできないため、このコミットの組み合わせをマルチ SQL ステートメント・クライアント・サイド・トランザクションで使用する場合は、注意してください。
自動コミット OFF	自動コミット OFF	ストアード・プロシージャーが SQLCODE >= 0 を戻した場合、ストアード・プロシージャー内のすべての正常に実行された SQL ステートメントは、 CALL ステートメントを含むトランザクションがコミットされるとコミットされます。そうでない場合、ストアード・プロシージャー内のすべての SQL ステートメントは、 CALL ステートメントを含むトランザクションがロールバックされたときにロールバックされます。

表 10. DB2 CLI ストアード・プロシージャー・コミット動作 (続き)

CLI クライアント	ストアード・プロシージャー	コミット動作
自動コミット OFF	マニュアル・コミット	手動でコミットされた、ストアード・プロシージャー内の正常に実行されたすべての SQL ステートメントは、 CALL ステートメントにエラー SQLCODE が戻された場合でも、ロールバックされません。さらに、CALL ステートメントまでの、 CLI クライアント・アプリケーション内のすべての正常に実行された、コミットされていない SQL ステートメントはコミットされます。 **注:** ストアード・プロシージャーが SQLCODE >= 0 を戻した場合、最後の手動コミットの後に発生したストアード・プロシージャー内のすべての正常に実行された SQL ステートメントは、コミットされます。そうでない場合は、手動コミット時点までロールバックされます。 **注:** CALL ステートメントの発行後は、トランザクションを完全にロールバックすることはできないため、このコミットの組み合わせをマルチ SQL ステートメント・クライアント・サイド・トランザクションで使用する場合は、注意してください。

関連概念:

- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『ルーチン: プロシージャー』
- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『DB2 ストアード・プロシージャー』

関連タスク:

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

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CALL ステートメント』

第 9 章 コンパウンド SQL

CLI アプリケーションでのコンパウンド SQL ステートメントの実行

コンパウンド SQL を使用すると、複数の SQL ステートメントをグループ化して単一の実行可能ブロックに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。

制限:

- コンパウンド 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、サブステートメント間に依存性があってはなりません。
- コンパウンド SQL ステートメントはネストすることはできません。
- BEGIN COMPOUND および END COMPOUND ステートメントは、同じステートメント・ハンドルで実行する必要があります。
- BEGIN COMPOUND SQL ステートメントの STOP AFTER FIRST ? STATEMENTS 文節で指定される値は、タイプ SQL_INTEGER でなければならず、この値に対して、タイプ SQL_C_INTEGER または SQL_C_SMALLINT のアプリケーション・バッファーだけをバインドできます。
- 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。
- アトミック配列入力は、 SQL ステートメントの BEGIN COMPOUND および END COMPOUND ブロック内ではサポートされていません。アトミック配列入力とは、挿入が 1 回でも失敗すると、すべての挿入を取り消す動作を指します。
- END COMPOUND ステートメントが実行されるまで、ステートメント・ハンドルはすべて割り振られている状態でなければなりません。
- `SQLEndTran()` は、同一接続で、または BEGIN COMPOUND および END COMPOUND 間の接続要求で呼び出すことはできません。
- コンパウンド・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のは、以下の関数だけです。
  - `SQLAllocHandle()`
  - `SQLBindParameter()`
  - `SQLBindFileToParam()`
  - `SQLExecute()`
  - `SQLParamData()`
  - `SQLPrepare()`
  - `SQLPutData()`

手順:

CLI アプリケーションでコンパウンド SQL ステートメントを実行するには、以下のようにします。

1. 親ステートメント・ハンドルを割り振ります。以下に例を示します。

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

2. コンパウンド・サブステートメントごとにステートメント・ハンドルを割り振ります。以下に例を示します。

```
SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmtsub1);
SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmtsub2);
SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmtsub3);
```

3. サブステートメントを準備します。以下に例を示します。

```
SQLPrepare (hstmtsub1, stmt1, SQL_NTS);
SQLPrepare (hstmtsub2, stmt2, SQL_NTS);
SQLPrepare (hstmtsub3, stmt3, SQL_NTS);
```

4. 親ステートメント・ハンドルを使用して BEGIN COMPOUND ステートメントを実行します。以下に例を示します。

```
SQLExecDirect (hstmtparent, (SQLCHAR *) "BEGIN COMPOUND NOT ATOMIC STATIC",
               SQL_NTS);
```

5. これがアトミック・コンパウンド SQL 操作の場合は、 `SQLExecute()` 関数を使用してのみ、サブステートメントを実行してください。以下に例を示します。

```
SQLExecute (hstmtsub1);
SQLExecute (hstmtsub2);
SQLExecute (hstmtsub3);
```

注: アトミック・コンパウンド・ブロック内で実行されるすべてのステートメントを、最初に準備する必要があります。アトミック・コンパウンド・ブロック内で `SQLExecDirect()` 関数を使用しようとすると、エラーになります。

6. 親ステートメント・ハンドルを使用して END COMPOUND ステートメントを実行します。以下に例を示します。

```
SQLExecDirect (hstmtparent, (SQLCHAR *) "END COMPOUND NOT ATOMIC STATIC",
               SQL_NTS);
```

7. オプション: 入力パラメーター値の配列を使用した場合、親ステートメント・ハンドルを指定した `SQLRowCount()` を呼び出して、入力配列のすべてのエレメントに影響を受ける行数をまとめて検索します。以下に例を示します。

```
SQLRowCount (hstmtparent, &numRows);
```

8. サブステートメントのハンドルを解放します。以下に例を示します。

```
SQLFreeHandle (SQL_HANDLE_STMT, hstmtsub1);
SQLFreeHandle (SQL_HANDLE_STMT, hstmtsub2);
SQLFreeHandle (SQL_HANDLE_STMT, hstmtsub3);
```

9. 親ステートメント・ハンドルの使用が終了したら、その親ステートメント・ハンドルを解放します。以下に例を示します。

```
SQLFreeHandle (SQL_HANDLE_STMT, hstmtparent);
```

アプリケーションが自動コミット・モードで動作しておらず、COMMIT オプションが指定されていない場合、サブステートメントはコミットされません。しかし、アプリケーションが自動コミット・モードで動作している場合には、 COMMIT オプションが指定されていなくても、サブステートメントは END COMPOUND 時にコミットされます。

関連タスク:

- 26 ページの『CLI アプリケーションでのステートメント・ハンドルの割り振り』
- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 91 ページの『列方向配列の入力を使用した CLI アプリケーションでのパラメーター・マーカーのバインド』
- 92 ページの『行方向配列の入力を使用した CLI アプリケーションでのパラメーター・マーカーのバインド』
- 43 ページの『CLI アプリケーションでのステートメント・リソースの解放』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecute 関数 (CLI) - ステートメントの実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLPrepare 関数 (CLI) - ステートメントの準備』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLRowCount 関数 (CLI) - 行カウントの取得』
- 「*SQL リファレンス 第 2 巻*」の『COMMIT ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『ROLLBACK ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『コンパウンド SQL (動的) ステートメント』
- 137 ページの『CLI アプリケーションでのコンパウンド SQL の戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』

CLI アプリケーションでのコンパウンド SQL の戻りコード

戻りコードは、END COMPOUND ステートメントの `SQLExecute()` または `SQLExecDirect()` への呼び出し時に生成されます。以下に ATOMIC および NOT ATOMIC コンパウンド・ステートメントの戻りコードをリストします。

ATOMIC

- SQL_SUCCESS: すべてのサブステートメントは実行され、警告やエラーはありませんでした。
- SQL_SUCCESS_WITH_INFO: すべてのサブステートメントは正常に実行されましたが、 1 つ以上の警告がありました。 `SQLGetDiagRec()` または `SQLGetDiagField()` を呼び出して、エラーまたは警告についての追加情報を調べてください。 `SQLGetDiagRec()` または `SQLGetDiagField()` を処理するために使用されるハンドルは、 BEGIN COMPOUND および END COMPOUND ステートメントを処理するのに使用するのと同じハンドルでなければなりません。

- SQL_NO_DATA_FOUND: BEGIN COMPOUND および END COMPOUND ステートメントがサブステートメントなしで実行されたか、サブステートメントが行に影響を与えることはありませんでした。
- SQL_ERROR: 1 つ以上のサブステートメントが失敗し、すべてのサブステートメントがロールバックされました。

NOT ATOMIC

- SQL_SUCCESS: すべてのサブステートメントは実行され、エラーはありませんでした。
- SQL_SUCCESS_WITH_INFO: COMPOUND ステートメントは実行されましたが、1 つ以上の警告が 1 つ以上のサブステートメントによって戻されました。 `SQLGetDiagRec()` または `SQLGetDiagField()` を呼び出して、エラーまたは警告についての追加情報を調べてください。 `SQLGetDiagRec()` または `SQLGetDiagField()` を処理するために使用されるハンドルは、 BEGIN COMPOUND および END COMPOUND ステートメントを処理するのに使用するのと同じハンドルでなければなりません。
- SQL_NO_DATA_FOUND: BEGIN COMPOUND および END COMPOUND ステートメントがサブステートメントなしで実行されたか、サブステートメントが行に影響を与えることはありませんでした。
- SQL_ERROR: COMPOUND ステートメントは失敗しました。少なくとも 1 つのサブステートメントがエラーを戻しました。 SQLCA を調べて、失敗したステートメントを判別してください。

関連タスク:

- 135 ページの『CLI アプリケーションでのコンパウンド SQL ステートメントの実行』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLError 関数 (CLI) - エラー情報の取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetSQLCA 関数 (CLI) - SQLCA データ構造の取得』
- 「*SQL リファレンス 第 1 巻*」の『SQLCA (SQL 連絡域)』
- 「*SQL リファレンス 第 2 巻*」の『コンパウンド SQL (動的) ステートメント』

第 10 章 マルチスレッド CLI アプリケーション

マルチスレッド CLI アプリケーション

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
- Linux
- Solaris™
- 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 はデータベースに対するすべてのスレッド化アクセス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。つまり、DB2 CLI を使用するアプリケーションやストアード・プロシージャーを、複数回呼び出したり、同時に呼び出したりできます。

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、マルチスレッド混合アプリケーションに関する資料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・マシン上で) 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 つの接続から共有アプリケーション・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、並行して上記共有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 1 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るからです。

たいていの GUI-based 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、いく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。

プログラミングのヒント:

DB2 CLI で割り振られるリソースは、スレッドの安全が保証されています。これは、共有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。

オブジェクトがスレッド間で共有されている場合は、アプリケーションのタイミングに関する問題が生じることがあります。たとえば、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:

1. ハンドルに関するスレッド保護は 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、そのポインターは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することができます。この理由のために、ODBC アプリケーションを作成する際には、マルチスレッド CLI アプリケーションのアプリケーション・モデルに従うのが最善です。
2. 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。詳細については、コンパイラーの資料をご覧ください。

関連概念:

- 141 ページの『マルチスレッド CLI アプリケーションのアプリケーション・モデル』
- 142 ページの『混合マルチスレッド CLI アプリケーション』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLCancel 関数 (CLI) - ステートメントの取り消し』
- 56 ページの『CLI 関数戻りコード』

マルチスレッド CLI アプリケーションのアプリケーション・モデル

以下の一般的なマルチスレッド CLI アプリケーションのモデルは、例として示されています。

- 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。
- 子スレッドがマスター・スレッドに接続を返すまで、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。

注: マルチスレッド DB2 CLI ストアード・プロシージャーは、そのストアード・プロシージャーが現在実行しているデータベースだけに接続できます。

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 はそのリソース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・リソースは 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、あるスレッドがデータベース A に接続して、 1 つの接続から共有アプリケーション・バッファー中にデータを読み取る一方で、他のスレッドがデータベース B に接続して、並行して共有バッファーから読み取りを行いデータをデータベース B に挿入するというデータベース間のコピー・シナリオにおいて、共有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有リソース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・リソース「A」を保留して、データベース・リソース「B」を待っている。そして、
- アプリケーション・リソース「A」を待っている間に、他のスレッドがデータベース・リソース「B」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 構成キーワードの設定が設定されない限り、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・リソースを共有することはありません。

関連概念:

- 139 ページの『マルチスレッド CLI アプリケーション』
- 142 ページの『混合マルチスレッド CLI アプリケーション』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『locktimeout - 「ロック・タイムアウト」構成パラメーター』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLCancel 関数 (CLI) - ステートメントの取り消し』

混合マルチスレッド CLI 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実行する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 呼び出しを実行する場合

アプリケーションが CLI 関数の前に DB2 API 関数または組み込み SQL 関数を呼び出す場合は、 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 関数または組み込み SQL 関数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 を使用して、 DB2 CLI 関数を呼び出す前にコンテキストを割り当ててアタッチします (`SQLAllocEnv()` は、既存のコンテキストをデフォルト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。

注: デフォルトのアプリケーションのスタック・サイズを使用せずに、スタック・サイズを少なくとも 256,000 に増やすことをお勧めします。 DB2 では、DB2 関数の呼び出し時に必要な最小アプリケーション・スタック・サイズは 256,000 です。したがって、お使いのアプリケーションと、 DB2 関数呼び出し時の最小要件の両方を十分に満たす合計スタック・サイズが割り当てられていることを確認する必要があります。

関連概念:

- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『DB2 CLI と組み込み動的 SQL の比較』
- 139 ページの『マルチスレッド CLI アプリケーション』
- 141 ページの『マルチスレッド CLI アプリケーションのアプリケーション・モデル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLAllocEnv 関数 (CLI) - 環境ハンドルの割り振り』
- 「*管理 API リファレンス*」の『sqleAttachToCtx - コンテキストへのアタッチ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』

第 11 章 マルチサイト更新 (2 フェーズ・コミット)

CLI アプリケーションでのマルチサイト更新 (2 フェーズ・コミット)

一般的な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が取り上げられます。並行トランザクションには同時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。

マルチサイト更新、2 フェーズ・コミット (2PC) プロトコル、および整合分散トランザクションを使用すると、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。

注: マルチサイト更新のことを、分散作業単位 (DUOW) ともいいます。

マルチ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。マルチ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。

マルチサイト更新によっては、トランザクション・マネージャー (TM) 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。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できます。

- 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
- 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
- ホストおよび AS/400® データベース・サーバー

注: ホストまたは iSeries™ データベース・サーバーに接続している場合は、特定の DB2 CLI/ODBC クライアント構成は必要ありませんが、DB2 Connect™ を実行しているマシンが、ホストに対してマルチサイト更新モードを実行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。

関連概念:

- 「*DB2 Connect ユーザーズ・ガイド*」の『マルチサイト更新』
- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『マルチサイト更新アプリケーションの構成パラメーターに関する考慮事項』
- 146 ページの『CLI アプリケーションでのトランザクション・マネージャーとしての DB2』
- 156 ページの『CLI アプリケーションに関するプロセス・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) のプログラミングの考慮事項』

関連タスク:

- 「*DB2 Connect ユーザーズ・ガイド*」の『コントロール・センターを使ったマルチサイト更新の使用可能化』

CLI アプリケーションでのトランザクション・マネージャーとしての DB2

トランザクション・マネージャーとしての DB2 の構成:

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 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、 DB2 トランザクション・マネージャーの構成に関する資料中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、次の構成を適用する必要があります。

- `db2cli.ini` ファイル内の [COMMON] セクションで、 DisableMultiThread CLI/ODBC 構成キーワードを 1 に設定します。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・レベルでシリアライズされます。

  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。
- SQL_ATTR_CONNECTTYPE 環境属性を設定します。この属性は、アプリケーションを整合分散環境で実行するか、それとも非整合分散環境で実行するかを制御します。整合分散環境では、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 - 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。各接続には、それぞれのコミット範囲があります。トランザクションの調整の実施は試みられません。これはデフォル

トで、組み込み SQL 中の Type 1 CONNECT に対応します。SQL_ATTR_SYNC_POINT 環境属性の現行設定は無視されます。

– 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。整合トランザクションとは、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) されるトランザクションのことです。 SQL_ATTR_CONNECTTYPE をこの値に設定することは、組み込み SQL 中の Type 2 CONNECT に対応します。

環境ハンドルが割り振られたら、必要に応じて、アプリケーションはできる限り即時に `SQLSetEnvAttr()` への呼び出しを行って、この環境属性を設定することをお勧めします。 ODBC アプリケーションは `SQLSetEnvAttr()` にアクセスできないので、個々の接続ハンドルが割り振られてから確立されるまでの間に、`SQLSetConnectAttr()` を使用してこの属性を設定しなければなりません。

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。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、接続アクティブに接続タイプを変更しようとすると、エラーが返されます。

- 前述の SQL_ATTR_CONNECTTYPE が SQL_COORDINATED_TRANS に設定されていた場合は、 SQL_ATTR_SYNC_POINT 環境属性を SQL_TWOPHASE に設定します。複数データベース・トランザクションにおいて、各データベースによって実行された作業をコミットする場合には、 2 フェーズ・コミットが使用されます。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。

  アプリケーションは、環境ハンドルが作成されたら即時にこの環境属性を設定する必要があります。この属性を設定するには、`SQLSetEnvAttr()` への呼び出しを行う必要があります。 ODBC アプリケーションは、接続が確立される前に、`SQLSetConnectAttr()` を使用して、この環境中の接続ハンドルごとにこの属性を設定しなければなりません。

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

- 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、マルチサイト更新環境で関数 `SQLEndTran()` を使用しなければなりません。

並行および整合トランザクションでのアプリケーション・フロー:

148 ページの図 10 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。

149 ページの図 11 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。

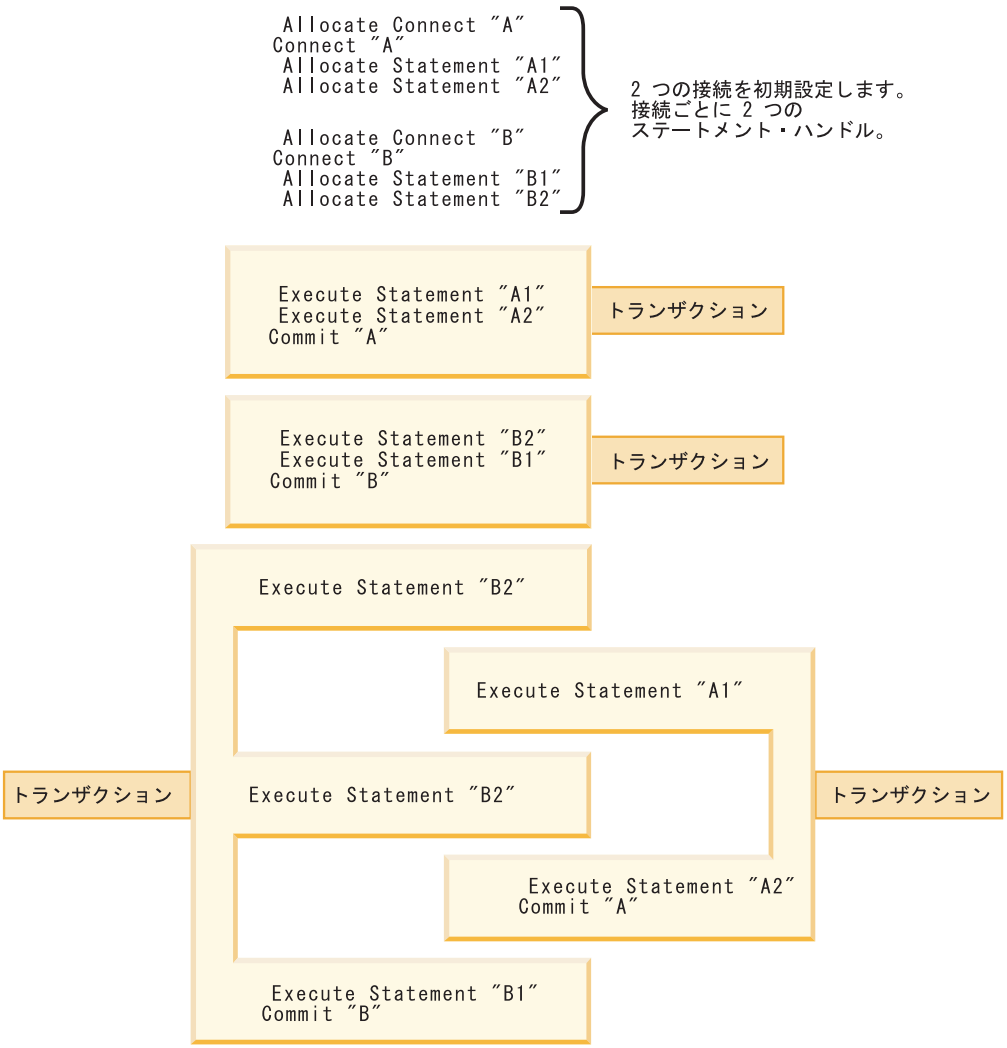


図 10. 並行トランザクションを使用した複数接続

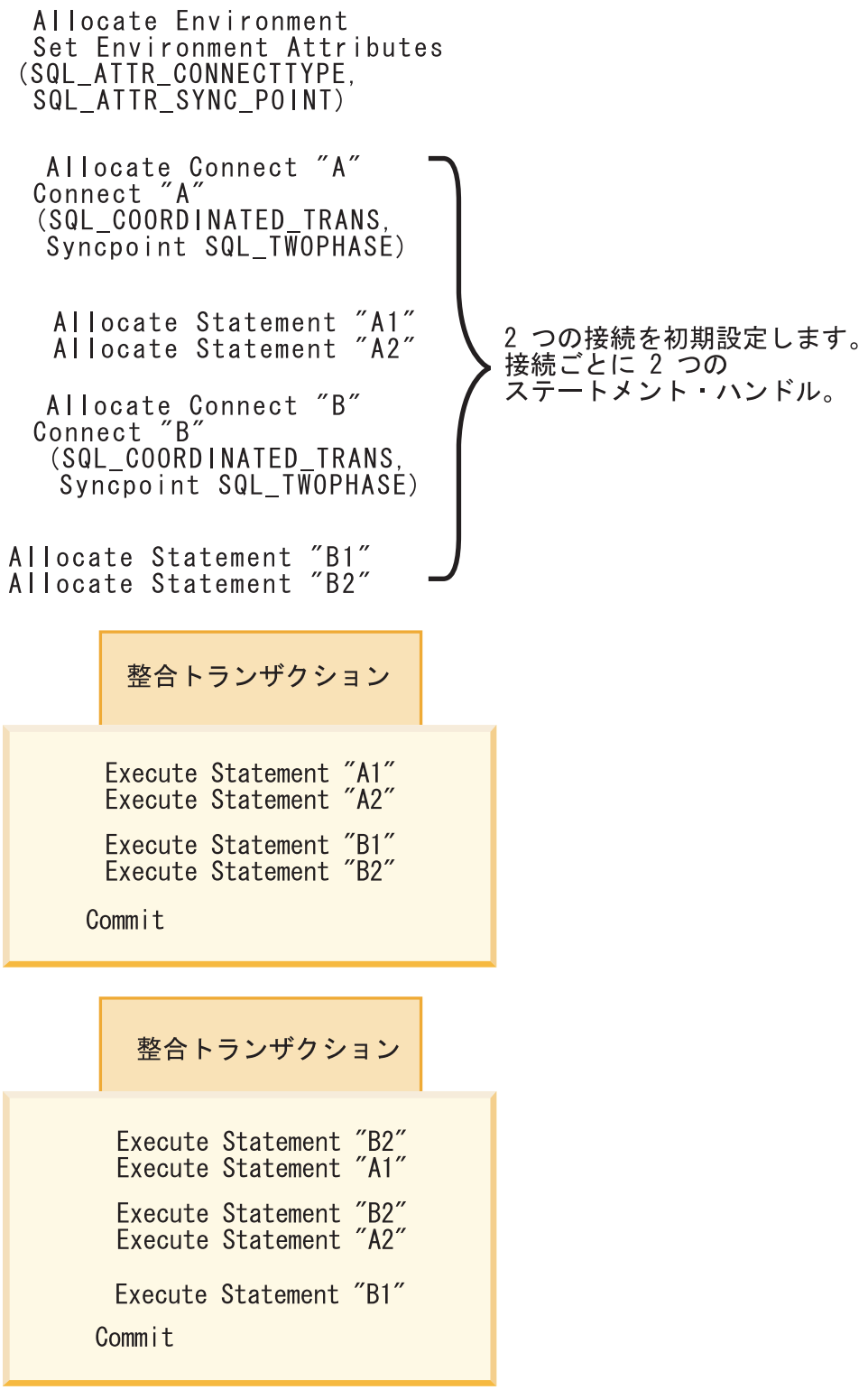


図 11. 整合トランザクションを使用した複数接続

制限:

マルチ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。

関連概念:

- 203 ページの『組み込み SQL と DB2 CLI の混合に関する考慮事項』
- 「*管理ガイド: プランニング*」の『DB2 Universal Database トランザクション・マネージャーの構成』
- 145 ページの『CLI アプリケーションでのマルチサイト更新 (2 フェーズ・コミット)』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLEndTran 関数 (CLI) - 接続または環境のトランザクションの終了』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetEnvAttr 関数 (CLI) - 環境属性の設定』
- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 1) ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 2) ステートメント』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『環境属性 (CLI) リスト』
- 308 ページの『ConnectType CLI/ODBC 構成キーワード』
- 324 ページの『DisableMultiThread CLI/ODBC 構成キーワード』

関連サンプル:

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

トランザクション・モニターとしての Microsoft Transaction Server (MTS)

トランザクション・マネージャーとしての Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+)

DB2® UDB は Microsoft® Transaction Server (MTS) バージョン 2.0 (Windows® NT の場合) または Microsoft Component Services (COM+) (Windows 2000 および Windows XP の場合) と完全に統合され、複数の DB2 UDB、zSeries™、および iSeries™ データベース・サーバー、さらに MTS や COM+ の仕様に準拠する他のリソース・マネージャーとの間で、 2 フェーズ・コミットを整合できるようになりました。

前提条件:

MTS または COM+ の分散トランザクション・サポートを使用するには、 DB2 クライアントがインストールされている Windows マシンが以下の要件を満たしている必要があります。

- Windows NT® で MTS バージョン 2.0 を使用する場合: Microsoft Hotfix 0772 以降

  MTS Version 2.0 for Windows NT は Windows NT 4.0 Option Pack の一部として入手できます。 Option Pack は以下のサイトからダウンロードできます。

  http://www.microsoft.com/ntserver/nts/downloads/recommended/NT4OptPk/

- Windows 2000: Service Pack 3 以降

MTS または COM+ を使用する DB2 CLI アプリケーションについて:

- SQL_ATTR_CONNECTION_POOLING CLI 環境属性のデフォルト値 (SQL_CP_OFF) を変更しないでください。
- DB2 ODBC ドライバーを Windows オペレーティング・システムにインストールすると、以下の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
```

サポートされる DB2 データベース・サーバー:

MTS または COM+ を使用してトランザクションを整合する場合、以下のサーバーがマルチサイト更新用にサポートされます。

- DB2 Universal Database™ Enterprise Server Edition (ESE)

  注: MTS または COM+ 用の疎結合グローバル・トランザクションは、超並列処理 (MPP) 環境ではサポートされません。疎結合のグローバル・トランザクションとは、複数のアプリケーション・プロセスがトランザクション・マネージャーの調整下にあるにもかかわらず、個々のアプリケーション・プロセスがあたかも別々のグローバル・トランザクションに属するかのように複数のリソース・マネージャーにアクセスする状況です。個々のアプリケーション・プロセスごとに、リソース・マネージャー内にそれぞれ固有のトランザクション分岐があります。いずれかのアプリケーション・プロセス、トランザクション・マネージャー、またはリソース・マネージャーによってコミットまたはロールバックが要求されると、トランザクション分岐は完了します。分岐間でリソース・デッドロックが発生しないように担当するのは、アプリケーションです。

  (複数のアプリケーション・プロセスが 1 つのリソース・マネージャー内の同じトランザクション分岐の下で操作を分担している状況は、密結合グローバル・トランザクションです。これら 2 つのアプリケーション・プロセスは、リソース・マネージャーからは単一のエンティティーと見なされます。リソース・マネージャーは、トランザクション分岐の中でリソースのデッドロックが発生しないようにします。)

- DB2 Universal Database for z/OS™
- DB2 Universal Database for iSeries
- DB2 Server for VSE & VM

インストールおよび構成の考慮事項:

MTS を使用する場合のインストールおよび構成の考慮事項は、以下のとおりです (COM+ はデフォルトで Windows 2000 の一部としてインストールされます)。

- MTS と DB2 クライアントの両方を、 MTS アプリケーションが実行される同じマシン上にインストールします。
- マルチサイト更新でホストまたは iSeries データベース・サーバーを使用する場合、以下のようにします。
  1. ローカル・マシンまたはリモート・マシン上に、DB2 Connect™ 機能 (DB2 Connect Enterprise Edition (EE)、または DB2 Connect 機能がインストールされている DB2 UDB Enterprise Server Edition (ESE) のいずれか) をインストールします。 DB2 Connect 機能によって、ホストまたは iSeries サーバーはマルチサイト更新トランザクションに参加できるようになります。
  2. DB2 Connect サーバーでマルチサイト更新が使用可能になっていることを確認します。

関連概念:

- 「*管理ガイド: プランニング*」の『X/Open 分散トランザクション処理のモデル』
- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『MTS および COM+ 分散トランザクションのサポートと IBM OLE DB Provider』
- 153 ページの『Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) のトランザクション・タイムアウト』
- 152 ページの『Microsoft Component Services (COM+) での疎結合サポート』
- 154 ページの『Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) を使用した ODBC および ADO 接続プール』

関連タスク:

- 「*DB2 Connect Enterprise Edition 概説およびインストール*」の『DB2 Connect Enterprise Edition のインストール (Windows)』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 「*DB2 Connect ユーザーズ・ガイド*」の『DB2 Connect の製品オファリング』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『環境属性 (CLI) リスト』

Microsoft Component Services (COM+) での疎結合サポート

疎結合のグローバル・トランザクションとは、複数のアプリケーション・プロセスがトランザクション・マネージャーの調整下にあるにもかかわらず、個々のアプリケーション・プロセスがあたかも別々のグローバル・トランザクションに属するかのように複数のリソース・マネージャーにアクセスする状況です。個々のアプリケーション・プロセスごとに、リソース・マネージャー内にそれぞれ固有のトランザクション分岐があります。いずれかのアプリケーション・プロセス、トランザクション・マネージャー、またはリソース・マネージャーによってコミットまたはロールバックが要求されると、トランザクション分岐は完了します。分岐間でリソース・デッドロックが発生しないように担当するのは、アプリケーションです。

DB2® Universal Database バージョン 8 は、 COM+ オブジェクトに対して疎結合グローバル・トランザクションをサポートし、ロック・タイムアウトやデッドロックはありません。ただし、以下のような制約事項があります。

- データ定義言語 (DDL) は、単一の分岐で実行されており、他の疎結合トランザクションがアクティブでない場合にのみサポートされます。 DDL を実行している単一分岐がアクティブ状態のとき、他の疎結合分岐を開始しようとすると、その疎結合分岐は拒否されます。反対に、アクティブな疎結合のトランザクションが少なくとも 1 つ存在する場合、別の分岐において DDL を実行しようとしても拒否されます。
- 疎結合グローバル・トランザクションは、超並列処理 (MPP) 環境ではサポートされません。 MPP 環境ではそれぞれのグローバル・トランザクションが独立して処理され、デッドロックやタイムアウトが発生する可能性があります。
- セーブポイントの処理および SQL ステートメントは、複数の接続の間で連続的に実行されます。
- ある接続において暗黙的にロールバックが実行された場合、疎結合トランザクションに参加しているその他の接続のすべての分岐は、 SQL0998N とともに理由コード 225、サブコード 4「このトランザクションには、ロールバックのみが許されています」を戻します。

関連概念:

- 「*管理ガイド: プランニング*」の『X/Open 分散トランザクション処理のモデル』
- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『MTS および COM+ 分散トランザクションのサポートと IBM OLE DB Provider』
- 150 ページの『トランザクション・マネージャーとしての Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+)』
- 153 ページの『Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) のトランザクション・タイムアウト』
- 154 ページの『Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) を使用した ODBC および ADO 接続プール』

Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) のトランザクション・タイムアウト

MTS または COM+ を使用している場合、以下のツールを使用してトランザクション・タイムアウトを設定できます。

- MTS (Microsoft Windows® NT): MTS Explorer ツール
- COM+ (Microsoft Windows 2000 および XP): Component Services (Windows コントロール パネルの管理ツールの下にある)

トランザクションの時間がトランザクション・タイムアウト値 (デフォルトは 60 秒) を超えた場合、 MTS または COM+ は関係するすべてのリソース・マネージャーに対してアボートを非同期的に出し、トランザクション全体がアボートされます。

アボートは、サーバー側では DB2® ロールバック要求として解釈されます。 DB2 for z/OS™ および DB2 for iSeries™ 以外のサーバーでは、データベース・サーバー上のデータ保全性を保証するために、ロールバック要求が接続においてシリアライズされます。 DB2 for z/OS サーバーまたは DB2 for iSeries サーバーの場合、DCS カタログ・エントリーの INTERRUPT_ENABLED オプションを使って接続を定義します。これによって、タイムアウト発生時に DB2 Connect サーバーから z/OS または iSeries サーバーへの接続が切断され、 z/OS または iSeries サーバー上でロールバックが強制的に実行されます。

その結果、以下のようになります。

- 接続がアイドル状態の場合、ただちにロールバックが実行されます。
- SQL ステートメントが長時間にわたって処理されている場合、その SQL ステートメントの終了までロールバック要求は実行されません。

関連概念:

- 「*管理ガイド: プランニング*」の『X/Open 分散トランザクション処理のモデル』
- 「*DB2 Connect ユーザーズ・ガイド*」の『DCS ディレクトリーの値』
- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『MTS および COM+ 分散トランザクションのサポートと IBM OLE DB Provider』
- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『割り込み要求の処理』
- 150 ページの『トランザクション・マネージャーとしての Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+)』
- 152 ページの『Microsoft Component Services (COM+) での疎結合サポート』
- 154 ページの『Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) を使用した ODBC および ADO 接続プール』

Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) を使用した ODBC および ADO 接続プール

接続プールを使用すれば、アプリケーションは複数の接続からなるプールの中から 1 つの接続を使用できます。これによって、接続を使用するたびに再確立する必要がなくなります。いったん接続が作成されてプールに入れられると、アプリケーションは完全な接続プロセスを実行しなくても、その接続を再使用できます。アプリケーションがデータ・ソースから切断したときに接続はプールに入れられ、同じ属性の新しい接続で使用されます。

ODBC 接続プール:

接続プールは ODBC 2.x 以来、ODBC Driver Manager 機能に含まれてきました。Microsoft® Data Access Components (MDAC) ダウンロードの一部として利用できる最新の ODBC Driver Manager (バージョン 3.5) では、接続プールの構成の一部が変更され、トランザクション MTS COM+ オブジェクトの ODBC 接続に関する新しい機能が追加されました。

ODBC Driver Manager 3.5 では、接続プールを活動化する前に、 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しなければなりません。以下のようなキーワードです。

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

Windows® オペレーティング・システム用の DB2 ODBC ドライバーは接続プールを完全にサポートするので、このキーワードは登録されます。

デフォルト値 60 は、接続が切断される前に 60 秒間プールされることを意味しています。

通信量の多い環境では、物理的な接続および切断の回数が多くなりすぎないよう、CPTimeout 値をより大きくする必要があります。接続や切断が多数発生した場合、システム・メモリーや通信スタック・リソースなどのシステム・リソースが大量に消費されるためです。

さらに、マルチプロセッサー・マシンの同じトランザクションに含まれるオブジェクトの間で同じ接続が使用されるようにするために、「プロセッサーあたりの複数プール」サポートをオフにする必要があります。これを行うには、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、それをプレーン・テキスト・ファイルとして保存した後、 **odbcpool.reg** コマンドを発行します。 Windows オペレーティング・システムでは、このレジストリー設定値がインポートされます。

```
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 や COM+ は同一トランザクション用の複数の接続をさまざまなプールに入れる可能性があり、その場合、同じ接続が再使用されません。

ADO 接続プール:

MTS または 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、 OLE DB リソース・プールをオフにする必要があります。こうすれば、Microsoft OLE DB Provider for ODBC (MSDASQL) が ODBC 接続プールと競合しなくなります。この機能は ADO 2.0 では初期設定で OFF でしたが、 ADO 2.1 では初期設定が ON になっています。 OLE DB リソース・プールをオフにするには、以下の行を oledb.reg というファイルにコピーし、それをプレーン・テキスト・ファイルとして保存した後、 **oledb.reg** コマンドを発行します。 Windows オペレーティング・システムでは、これらのレジストリー設定値がインポートされます。

```
REGEDIT4

[HKEY_CLASSES_ROOT¥CLSID¥{c8b522cb-5cf3-11ce-ade5-00aa0044773d}]
@="MSDASQL"
"OLEDB_SERVICES"=dword:fffffffc
```

関連概念:

- 「*管理ガイド: プランニング*」の『X/Open 分散トランザクション処理のモデル』
- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『MTS および COM+ 分散トランザクションのサポートと IBM OLE DB Provider』
- 150 ページの『トランザクション・マネージャーとしての Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+)』
- 153 ページの『Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) のトランザクション・タイムアウト』
- 152 ページの『Microsoft Component Services (COM+) での疎結合サポート』

CLI アプリケーションに関するプロセス・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) のプログラミングの考慮事項

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 DB2 CLI/ODBC アプリケーションを実行することに関する考慮事項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、 XA トランザクション・マネージャーの構成に関する考慮事項に従わなければなりません。

注: XA トランザクション処理環境に入ると、CLI/ODBC 構成キーワードの接続に関する設定は必要なくなります。

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・ハンドルをすべてドロップする。
  - `SQLDisconnect()` および `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:

マルチ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。

関連概念:

- 203 ページの『組み込み SQL と DB2 CLI の混合に関する考慮事項』
- 「*管理ガイド: プランニング*」の『XA トランザクション・マネージャーの構成に関する考慮事項』
- 146 ページの『CLI アプリケーションでのトランザクション・マネージャーとしての DB2』
- 145 ページの『CLI アプリケーションでのマルチサイト更新 (2 フェーズ・コミット)』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLConnect 関数 (CLI) - データ・ソースへの接続』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDisconnect 関数 (CLI) - データ・ソースからの切断』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDriverConnect 関数 (CLI) - データ・ソースへの (拡張) 接続』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』

第 12 章 Unicode

Unicode CLI アプリケーション

DB2® CLI Unicode アプリケーションのサポート領域には次の 2 つがあります。

- ANSI ストリング引き数の代わりに Unicode ストリング引き数を受け入れる関数のセットの追加。
- Unicode データを記述する、新しい C および SQL データ・タイプの追加。

アプリケーションが Unicode アプリケーションであるためには、そのアプリケーションがデータベースに接続する際に、 `SQLConnectW()` か `SQLDriverConnectW()` のいずれかを使用する必要があります。こうすると、確実に CLI が Unicode を CLI 自体とデータベースの間の優先的な通信方式と見なすことができます。

ODBC は既存の C タイプと SQL タイプのセットにタイプを追加して Unicode に適合させ、それに応じて CLI は追加されたタイプを使用します。新しい C タイプの SQL_C_WCHAR は、 C バッファーに Unicode データが含まれていることを指示します。 DB2 CLI/ODBC ドライバーは、アプリケーションと交換するすべての Unicode データを、ネイティブ・エンディアン形式の UCS-2 と見なします。新しい SQL タイプの SQL_WCHAR、 SQL_WVARCHAR、および SQL_WLONGVARCHAR は、特定の列やパラメーター・マーカーに Unicode データが含まれていることを指示します。 DB2 Unicode データベースの場合、GRAPHIC 列は新しいタイプを使用して記述されます。 GRAPHIC データの場合と同様に、SQL_C_WCHAR と SQL_CHAR、SQL_VARCHAR、SQL_LONGVARCHAR と SQL_CLOB の間で変換が実行されます。

注: UCS-2 は、1 文字を 2 バイトで表記する固定長文字コード化スキームです。UCS-2 エンコードのストリングに含まれる文字数は、単にそのストリングを格納するのに必要な SQLWCHAR の数としてカウントされます。

廃止された CLI/ODBC キーワード値:

Unicode アプリケーションがサポートされるようになるまでは、Graphic=1、2、または 3 や Patch2=7 といった一連の DB2 CLI 構成キーワードによって、単一バイト文字データの操作用に作成されたアプリケーションが 2 バイト GRAPHIC データを操作できるようにしていました。これらの対処法により、GRAPHIC データが文字データとして表示され、報告されるデータの長さにも影響していました。これらのキーワードは、Unicode アプリケーションの場合には不要であり、さらに潜在的な副次作用を持つ危険性があるため、使用しないようにしてください。あるアプリケーションが Unicode アプリケーションかどうかがわからない場合は、 GRAPHIC データの処理に影響するキーワードなしで試してください。

unicode データベースのリテラル:

非 Unicode データベースでは、 LONG VARGRAPHIC および LONG VARCHAR 列のデータは比較できません。 GRAPHIC/VARGRAPHIC および CHAR/VARCHAR 列のデータは、比較のみが可能か、または暗黙的コード・ページ変換がサポートされていないため、明示的な cast 関数を使用して相互に割り当てることができます。これには、GRAPHIC/VARGRAPHIC リテラルが G 接頭部によって CHAR/VARCHAR と区別される、 GRAPHIC/VARGRAPHIC および CHAR/VARCHAR が含まれます。 Unicode データベースについては、GRAPHIC/VARGRAPHIC リテラルと CHAR/VARCHAR リテラルの間のキャストは不要です。また、GRAPHIC/VARGRAPHIC リテラルの前に G 接頭部は必要ありません。少なくとも 1 つの引き数がリテラルの場合、暗黙的変換が実行されます。これにより、リテラルは G 接頭部を持っていても持っていなくても、 `SQLPrepareW()` または `SQLExecDirect()` を使用するステートメント内で使用することができます。LONG VARGRAPHIC のリテラルには G 接頭部が必要です。

関連概念:

- 160 ページの『Unicode 関数 (CLI)』
- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『Unicode データベースに接続されるアプリケーション』
- 161 ページの『Unicode 関数から ODBC Driver Manager への呼び出し』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 345 ページの『Patch2 CLI/ODBC 構成キーワード』

Unicode 関数 (CLI)

DB2 CLI Unicode 関数は、ANSI ストリング引き数の代わりに Unicode ストリング引き数を受け入れます。 Unicode ストリング引き数は、 UCS-2 エンコード (ネイティブ・エンディアン形式) でなければなりません。 ODBC API 関数には、それぞれのストリング引き数の形式を示す接尾部があります。すなわち、 W での Unicode 終了を受け入れるストリング引き数と、ANSI を受け入れるストリング引き数には、接尾部がありません (ODBC は、名前が A で終わる同等の関数を追加しますが、これらは DB2 CLI によって提供されるものではありません)。次に示すのは、ANSI バージョンと Unicode バージョンの両方を持つ、 DB2 CLI で使用できる関数のリストです。

```
SQLBrowseConnect        SQLForeignKeys          SQLPrimaryKeys
SQLColAttribute         SQLGetConnectAttr       SQLProcedureColumns
SQLColAttributes        SQLGetConnectOption     SQLProcedures
SQLColumnPrivileges     SQLGetCursorName        SQLSetConnectAttr
SQLColumns              SQLGetDescField         SQLSetConnectOption
SQLConnect              SQLGetDescRec           SQLSetCursorName
SQLDataSources          SQLGetDiagField         SQLSetDescField
SQLDescribeCol          SQLGetDiagRec           SQLSetStmtAttr
SQLDriverConnect        SQLGetInfo              SQLSpecialColumns
SQLError                SQLGetStmtAttr          SQLStatistics
SQLExecDirect           SQLNativeSQL            SQLTablePrivileges
SQLExtendedPrepare      SQLPrepare              SQLTables
```

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

たとえば、`SQLGetInfoW()` は長さをバイト数として解釈しますが、`SQLExecDirectW()` は SQLWCHAR エレメント数を使用します。 UTF-16 拡張文字セットの 1 文字について考慮してみましょう (UTF-16 は UCS-2 の拡張文字セットの 1 つです。 Microsoft® Windows® 2000 および Microsoft Windows XP では UTF-16 が使用されています)。 Microsoft Windows 2000 では、その 1 文字を格納するために 2 個の SQL_C_WCHAR、したがって 4 バイトが使用されます。それで、この文字の表示サイズは 1、ストリング長は 2 (SQL_C_WCHAR を使用した場合)、そしてバイト・カウントは 4 になります。 CLI は結果セットからのデータを、アプリケーションのバインドに応じて Unicode または ANSI で返します。アプリケーションが SQL_C_CHAR にバインドする場合、ドライバーは SQL_WCHAR データを SQL_CHAR に変換します。 ODBC Driver Manager は (使用されている場合)、 ANSI ドライバーについては SQL_C_WCHAR を SQL_C_CHAR にマップしますが、 Unicode ドライバーについてはマッピングを行いません。

ANSI から Unicode 関数へのマッピング:

DB2 CLI Unicode 関数の構文は、それに対応する ANSI 関数の構文と同じですが、SQLCHAR パラメーターが SQLWCHAR として定義されている点は異なります。ANSI 構文で SQLPOINTER と定義されている文字バッファーは、 Unicode 関数では、SQLCHAR か SQLWCHAR のいずれかとして定義できます。 ANSI 構文の詳細は、ANSI バージョンの CLI Unicode 関数を参照してください。

関連概念:

- 159 ページの『Unicode CLI アプリケーション』
- 161 ページの『Unicode 関数から ODBC Driver Manager への呼び出し』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『CLI と ODBC 関数のサマリー』

Unicode 関数から ODBC Driver Manager への呼び出し

ODBC 準拠アプリケーションでは、 DB2 CLI/ODBC を使用することによって DB2® データベースにアクセスできます。それには、DB2 CLI/ODBC ドライバー・ライブラリーをリンクする方法と、 ODBC Driver Manager ライブラリーをランクする方法の 2 種類の方法があります。ここでは、 ODBC Driver Manager ライブラリーをリンクする CLI アプリケーションについて説明します。

- 直接アクセス - アプリケーションは、 DB2 CLI/ODBC ドライバー・ライブラリーにリンクし、エクスポートされた CLI/ODBC 関数を呼び出します。 DB2 CLI/ODBC ドライバーに直接アクセスする Unicode アプリケーションでは、データベースに対するトランザクションのアクセスと実行において CLI Unicode 関数を使用しなければなりません。また、Unicode データはすべて UCS-2 であるということを理解した上で、SQLWCHAR バッファーを使用する必要があります。アプリケーションが Unicode アプリケーションであるためには、そのアプリケーションがデータベースに接続する際に、 `SQLConnectW()` か `SQLDriverConnectW()` のいずれかを使用する必要があります。
- 間接アクセス - アプリケーションは、 ODBC Driver Manager ライブラリーにリンクし、標準の ODBC 関数を呼び出します。 ODBC Driver Manager がアプリケーションのために DB2 CLI/ODBC ドライバーをロードし、エクスポートされた ODBC 関数を呼び出します。アプリケーションから DB2 CLI/ODBC ドライバーに渡されるデータは、 ODBC Driver Manager によって変換されることがあります。 ODBC Driver Manager がアプリケーションを Unicode アプリケーションとして認識するためには、 `SQLConnectW()` または `SQLDriverConnectW()` を呼び出します。

データ・ソースに接続する際、ODBC Driver Manager は、要求されたドライバーが `SQLConnectW()` 関数をエクスポートしているかどうかを調べます。その関数がサポートされているなら、その ODBC ドライバーは Unicode ドライバーと見なされ、それ以降、アプリケーションで ODBC 関数を呼び出すと、それらは ODBC Driver Manager により、すべてその関数の Unicode 版 (末尾にサフィックス W が付いているもの、たとえば `SQLConnectW()`) への呼び出しとして処理されることになります。アプリケーションが Unicode 関数を呼び出す場合、ストリング変換は不要であり、ODBC Driver Manager が直接 Unicode 関数を呼び出します。アプリケーションが ANSI 関数を呼び出す場合、 ODBC Driver Manager は、 ANSI ストリングをすべて Unicode ストリングに変換してから、対応する Unicode 関数を呼び出します。

アプリケーションが Unicode 関数を呼び出したが、ドライバーが `SQLConnectW()` をエクスポートしていない場合、 ODBC Driver Manager は、Unicode 関数呼び出しを、対応する ANSI 版の呼び出しとして処理します。対応する ANSI 関数を呼び出す前にすべての Unicode ストリングは、 ODBC Driver Manager によって、アプリケーションのコード・ページの ANSI ストリングに変換されます。そのため、アプリケーションのコード・ページに変換できない Unicode 文字がアプリケーションの中で使用している場合、データが失われる可能性があります。

Unicode ストリングのために使用されるコード化スキームは、オペレーティング・システムおよび各 ODBC Driver Manager ごとに異なります。

表 11. オペレーティング・システムごとの Unicode ストリング・コード化スキーム

ドライバー・マネージャー	オペレーティング・システム		
	Microsoft® Windows® 98、Windows ME、および Windows NT®	**Microsoft Windows 2000 および Windows XP**	**UNIX® 系**
Microsoft ODBC Driver Manager	UCS-2	UTF-16*	該当せず

表 11. オペレーティング・システムごとの Unicode ストリング・コード化スキーム (続き)

ドライバー・マネージャー	オペレーティング・システム		
	Microsoft® Windows® 98、Windows ME、および Windows NT®	Microsoft Windows 2000 および Windows XP	UNIX® 系
unixODBC Driver Manager	UCS-2	UCS-2	UCS-2
DataDirect Connect for ODBC Driver Manager	UCS-2	UTF-16*	UTF-8
* UTF-16 は UCS-2 のスーパーセットであり、それらには互換があります。			

UNIX での DataDirect Connect for ODBC Driver Manager に関する制限:

UNIX 環境で、 DB2 CLI/ODBC ドライバーと共に DataDirect Connect for ODBC Driver Manager を使用すると、ドライバー・マネージャーで UTF-8 文字エンコードが使用されているため、問題が発生します。 UTF-8 は、文字を格納するのに 1 バイト以上 6 バイト以下のいずれかを使用する可変長文字コード化スキームです。 UTF-8 と UCS-2 は本質的に互換ではなく、 UCS-2 を予期している DB2 CLI/ODBC ドライバーに UTF-8 データを渡すと、アプリケーション・エラー、データ破壊、またはアプリケーション例外が発生する可能性があります。

この問題を回避するため、 DataDirect Connect for ODBC Driver Manager 4.2 Service Pack 2 では、 DB2 CLI/ODBC ドライバーを認識して Unicode 関数を使用しないようにし、実質的に DB2 CLI/ODBC ドライバーを ANSI 専用ドライバーとして扱うようにしています。 Release 4.2 Service Pack 2 より前の DataDirect Connect for ODBC Driver Manager では、 SQLConnectW() 関数をエクスポートしていない DB2 CLI/ODBC ドライバーについて、その "_36" バージョンをリンクしなければなりませんでした。

関連概念:

- 11 ページの『DB2 CLI と Microsoft ODBC の比較』
- 160 ページの『Unicode 関数 (CLI)』
- 159 ページの『Unicode CLI アプリケーション』

関連タスク:

- 234 ページの『UNIX ODBC 環境のセットアップ』
- 236 ページの『unixODBC Driver Manager のセットアップ』

第 13 章 ユーザー定義タイプ (UDT)

CLI アプリケーションでの特殊タイプの使用

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。この種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見なされます。特殊タイプは、CREATE DISTINCT TYPE SQL ステートメントを使用して作成します。

特殊タイプは、オブジェクト指向プログラミングで必要な強い型定義の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特殊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、すべて特殊タイプに適用されます。
- 特殊タイプは、組み込みタイプと同じデフォルト 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、明示的に特殊タイプ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特殊タイプ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。

関連概念:

- 166 ページの『CLI アプリケーションでのユーザー定義タイプ (UDT) の使用法』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLColAttribute 関数 (CLI) - 列属性を戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』
- 「*SQL リファレンス 第 2 巻*」の『CREATE DISTINCT TYPE ステートメント』

- 383 ページの『CLI での SQL から C へのデータ変換例』

関連サンプル:

- 『dtudt.c -- How to create, use, and drop user-defined distinct types.』

CLI アプリケーションでのユーザー定義タイプ (UDT) の使用法

ユーザー定義タイプ (UDT) とは、従来の SQL タイプでは使用できない構造または強い型定義を提供する、ユーザーによって定義されるデータベース・タイプです。UDT には、特殊タイプ、構造タイプ、および参照タイプという 3 つの種類があります。

CLI アプリケーションは、特定のデータベース列が UDT であるかどうか、もしそうであるならどの種類の UDT であるかを判別できます。記述子フィールド SQL_DESC_USER_DEFINED_TYPE_CODE を使用して、この情報を入手できます。SQL_DESC_USER_DEFINED_TYPE_CODE が `SQLColAttribute()` を使用して検索されるか、 `SQLGetDescField()` を使用して IPD から直接検索される場合は、以下のいずれかの数値が含まれています。

```
SQL_TYPE_BASE  (これは正規 SQL タイプであり、UDT ではない)
SQL_TYPE_DISTINCT  (this value indicates that the column
                          is a distinct type)
SQL_TYPE_STRUCTURED  (this value indicates that the column
                           is a structured type)
SQL_TYPE_REFERENCE  (this value indicates that the column
                           is a reference type)
```

さらに、以下の記述子フィールドを使用してタイプ名を入手できます。

- SQL_DESC_REFERENCE_TYPE。参照タイプの名前または空ストリング (列が参照タイプではない場合) が入っています。
- SQL_DESC_STRUCTURED_TYPE。構造タイプの名前または空ストリング (列が構造タイプではない場合) が入っています。
- SQL_DESC_USER_TYPE または SQL_DESC_DISTINCT_TYPE。特殊タイプまたは空ストリング (列が特殊タイプではない場合) が入っています。

上記の記述子フィールドは、スキーマを名前の一部として戻します。スキーマが 8 文字より少ない場合は、ブランクが埋め込まれます。

接続属性 SQL_ATTR_TRANSFORM_GROUP を使用すると、アプリケーションはトランスフォーム・グループを設定できるようになります。また、これは SQL ステートメント SET CURRENT DEFAULT TRANSFORM GROUP の代替属性です。

CLI アプリケーションにとって、列に UDT が含まれているかどうかを判別するために SQL_DESC_USER_DEFINED_TYPE_CODE 記述子フィールドの値を繰り返し取得するのは望ましくない場合があります。このため、接続レベルとステートメント処理レベルの両方で、 SQL_ATTR_RETURN_USER_DEFINED_TYPES という属性があります。 `SQLSetConnectAttr()` を使用して SQL_TRUE に設定すると、CLI は SQL_DESC_USER_DEFINED_TYPE を戻します。この場合、`SQLColAttribute()`、`SQLDescribeCol()`、および `SQLGetDescField()` への呼び出しの結果に、通常は SQL タイプが含まれています。この設定により、アプリケーションはこの特殊なタイプがないかどうかを調べ、 UDT 用の特殊な処理を実行するようになります。この属性のデフォルト値は SQL_FALSE です。

SQL_ATTR_RETURN_USER_DEFINED_TYPES 属性を SQL_TRUE に設定すると、記述子フィールド SQL_DESC_TYPE は UDT の「基本」SQL タイプ (つまり、UDT の基礎となるまたは UDT の変換後の SQL タイプ) を戻さなくなります。このため、記述子フィールド SQL_DESC_BASE_TYPE は、 UDT の基本タイプと、通常列の SQL タイプを常に戻します。このフィールドにより、UDT を特別に処理するわけではないプログラムのモジュールが単純化されます。このフィールドを使用しない場合は、モジュールで接続属性を変更する必要があります。

`SQLBindParameter()` では、タイプが SQL_USER_DEFINED_TYPE のパラメーターをバインドできないことに注意してください。パラメーターをバインドするには、基本 SQL タイプを使用する必要がありますが、これは、記述子フィールド SQL_DESC_BASE_TYPE を使用して取得できます。たとえば、SQL_VARCHAR に基づく特殊タイプの列にバインドする場合に使用される `SQLBindParameter()` 呼び出しは、以下のとおりです。

```
sqlrc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
            SQL_VARCHAR, 30, 0, &c2, 30, NULL);
```

関連概念:

- 165 ページの『CLI アプリケーションでの特殊タイプの使用』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLColAttribute 関数 (CLI) - 列属性を戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 「*SQL リファレンス 第 2 巻*」の『CREATE DISTINCT TYPE ステートメント』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』

関連サンプル:

- 『dtudt.c -- How to create, use, and drop user-defined distinct types.』
- 『udfcli.c -- How to work with different types of user-defined functions (UDFs)』

第 14 章 記述子

CLI アプリケーションの記述子

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* は上記情報の論理ビューであり、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()` を呼び出すことと同じように、 `SQLSetStmtAttr()` を呼び出すことは、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 (IRD) 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から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 12. 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]	

表 12. ヘッダー・フィールド (続き)

注:

[a] このヘッダー・フィールドはステートメント属性に対応します。

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 (列やパラメーターの数のカウントではありません)。 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 13 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 13. 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
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

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 14. 据え置きフィールド

フィールド	NULL 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、ドライバーはストリングが NULL 値であると見なす。出力パラメーターの場合、このフィールドの NULL 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コンポーネント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、NULL をデフォルト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。

関連概念:

- 173 ページの『CLI アプリケーションの記述子の整合性検査』
- 174 ページの『記述子の割り当てと解放』
- 177 ページの『CLI アプリケーションでの記述子ハンドルによる記述子の操作』
- 179 ページの『CLI アプリケーションでの記述子ハンドルを使用しない記述子の操作』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetDescField 関数 (CLI) - 記述子レコードの単一フィールドの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』

- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『ステートメント属性 (CLI) のリスト』
- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『記述子ヘッダーとレコード・フィールドの初期設定値 (CLI)』
- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『記述子 FieldIdentifier 引き数の値 (CLI)』

CLI アプリケーションの記述子の整合性検査

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。 `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。

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。

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チェックします。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。

関連概念:

- 169 ページの『CLI アプリケーションの記述子』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetDescRec 関数 (CLI) - 列またはパラメーター・データ用の複数の記述子フィールドの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『記述子ヘッダーとレコード・フィールドの初期設定値 (CLI)』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『記述子 FieldIdentifier 引き数の値 (CLI)』

記述子の割り当てと解放

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)

以下の例では、ステートメントの暗黙的に割り当てられる実装パラメーター記述子に対するアクセス権が付与されます。

```
/* dbuse. c */
/* ... */
sqlrc = SQLGetStmtAttr ( hstmt,
                         SQL_ATTR_IMP_PARAM_DESC,
                         &hIPD,
                         SQL_IS_POINTER,
                         NULL);
```

注: この方法で取得されるハンドルに対する記述子はやはり、割り当て対象のステートメントが解放された場合に同様に解放されます。

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。

アプリケーションがデータベースに接続する際に、いつでもアプリケーション記述子を明示的に割り当てることができます。アプリケーション記述子を明示的に割り当てるには、 SQL_HANDLE_DESC の *HandleType* を指定して、SQLAllocHandle() を呼び出してください。たとえば、以下の呼び出しはアプリケーション行記述子を割り当てます。

```
rc = SQLAllocHandle( SQL_HANDLE_DESC, hdbc, &hARD ) ;
```

ステートメントの暗黙的に割り当てられる記述子の代わりに、明示的に割り当てられるアプリケーション記述子を使用するには、 SQLSetStmtAttr() を呼び出し、ステートメント・ハンドル、記述子ハンドル、および以下のどちらかの*属性* 値を渡してください。

- SQL_ATTR_APP_PARAM_DESC (APD)、または
- SQL_ATTR_APP_ROW_DESC (ARD)

明示的に割り当てられた記述子と暗黙的に割り当てられた記述子が両方ともある場合は、明示的に指定された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、記述子ヘッダーとレコード・フィールドの初期設定値に関する資料中にリストされている値に初期設定されます。 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 ARD 中のこの値を行セット内の行数にセットします。

IRD のフィールドについてはデフォルトがありません。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。

SQLPrepare() への呼び出しにより自動的に移植されるまでは、以下の IPD のフィールドは未定義です。

- 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。動的に生成された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。

IPD の自動移植がアクティブなとき、`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ドロップする `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()` を呼び出すことにより解放はできません。

関連概念:

- 169 ページの『CLI アプリケーションの記述子』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLPrepare 関数 (CLI) - ステートメントの準備』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『記述子ヘッダーとレコード・フィールドの初期設定値 (CLI)』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『記述子 FieldIdentifier 引き数の値 (CLI)』

関連サンプル:

- 『dbuse.c -- How to use a database』

CLI アプリケーションでの記述子ハンドルによる記述子の操作

記述子を操作するには、記述子ハンドルを使用するか、または記述子ハンドルを使わない DB2 CLI 関数を使用します。このトピックでは、記述子ハンドルを使用した記述子へのアクセスについて説明します。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。

記述子フィールド値の取り出し:

DB2 CLI 関数 `SQLGetDescField()` を使用して、記述子レコードの単一フィールドを得ることができます。 `SQLGetDescRec()` は、列またはパラメーター・データのストレージとデータ・タイプに影響する複数の記述子フィールドの設定値を取り出します。

記述子フィールド値の設定:

記述子フィールドを設定するには、一度に 1 つずつフィールドを設定する方式と、一度に複数のフィールドを設定する方式の 2 つの方式があります。

個々のフィールドの設定:

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。記述子 FieldIdentifier 値に関する資料中の、ヘッダーとレコードのフィールドのリストを参照してください。

次のように、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、SQLSetDescField() に関する資料で説明されている、記述子フィールドの設定に関する手順に従ってください。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。

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。

複数のフィールドの設定:

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。)

たとえば、以下の呼び出しを使用すると、前述の記述子フィールドがすべて設定されます。

```
/* dbuse.c */
/* ... */
rc = SQLSetDescRec(hARD, 1, type, 0,
                   length, 0, 0, &id_no, &datalen, NULL);
```

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。コピー元記述子およびコピー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)。

関連概念:

- 19 ページの『CLI でのハンドル』
- 169 ページの『CLI アプリケーションの記述子』
- 173 ページの『CLI アプリケーションの記述子の整合性検査』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetStmtAttr 関数 (CLI) - ステートメント属性の現行設定値の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetDescField 関数 (CLI) - 記述子レコードの単一フィールドの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetDescRec 関数 (CLI) - 列またはパラメーター・データ用の複数の記述子フィールドの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『記述子ヘッダーとレコード・フィールドの初期設定値 (CLI)』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『記述子 FieldIdentifier 引き数の値 (CLI)』

関連サンプル:

- 『dbuse.c -- How to use a database』

CLI アプリケーションでの記述子ハンドルを使用しない記述子の操作

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。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・データのストレージ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。詳しくは、記述子に関する資料の『記述子に保管される値』を参照してください。

関連概念:

- 19 ページの『CLI でのハンドル』
- 169 ページの『CLI アプリケーションの記述子』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetStmtAttr 関数 (CLI) - ステートメント属性の現行設定値の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetDescField 関数 (CLI) - 記述子レコードの単一フィールドの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetDescRec 関数 (CLI) - 列またはパラメーター・データ用の複数の記述子フィールドの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』

第 15 章 環境、接続、およびステートメントの属性

CLI アプリケーションでの環境、接続、およびステートメントの属性

環境、接続、およびステートメントには、それぞれ定義済みの属性 (またはオプション) の集まりがあります。アプリケーションですべての属性を照会することができますが、デフォルト値を変更できるのは一部の属性だけです。アプリケーションで属性値を変更して、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、環境ハンドルに接続ハンドルを割り振る前にのみ呼び出すことができます。それぞれの環境属性の詳細は、CLI 環境属性のリストを参照してください。

接続ハンドルには、その接続での 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
- 接続が確立されたら、いつでも設定できるもの
- 接続が確立された後で、未解決のトランザクションまたはオープン・カーソルがない場合に限り設定できるもの

アプリケーションでは、 `SQLSetConnectAttr()` を呼び出して接続属性の値を変更したり、 `SQLGetConnectAttr()` を呼び出して属性の現行値を得ることができます。ハンドルが割り振られた後であればいつでも設定できる接続属性の例としては、自動コミット・オプション SQL_ATTR_AUTOCOMMIT があります。それぞれの接続属性の詳細は、CLI 接続属性のリストを参照してください。

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

アプリケーションでは、`SQLSetStmtAttr()` を呼び出して設定可能なステートメント属性の値を指定したり、 `SQLGetStmtAttr()` を呼び出して属性の現行値を得ることができます。それぞれのステートメント属性の詳細は、CLI ステートメント属性のリストを参照してください。

`SQLSetConnectAttr()` 関数を、ステートメント属性を設定するために使用することはできません。この関数は、バージョン 5 より前の DB2 CLI でサポートされていました。

アプリケーションの多くは、デフォルトの属性設定値だけを使用します。しかし、これらのデフォルト値の一部が、アプリケーションの特定のユーザーには適さない状況もありえます。デフォルト値によっては、CLI/ODBC 構成キーワードを設定することで変更できるものもあります。 DB2 CLI では、エンド・ユーザーのために、いくつかの構成キーワードを設定する 2 つの方式が備えられています。 1 番目の方式は、`SQLDriverConnect()` および `SQLBrowseConnect()` 関数に接続ストリングを入力するときにキーワードと新しいデフォルト属性値を指定するものです。 2 番目の方式は、CLI/ODBC 構成キーワードを使用した DB2 CLI 初期設定ファイル内の新しいデフォルト属性値の仕様にかかわるものです。

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デフォルト値を変更することができます。 `SQLDriverConnect()` 接続ストリング中にデフォルト属性値を指定する方法がアプリケーションで使用できない場合は、エンド・ユーザーにとってこの方法がデフォルト値を変更する唯一の方法になります。 `SQLDriverConnect()` に指定されるデフォルト属性値によって、特定の接続に関する DB2 CLI 初期設定ファイル内の値がオーバーライドされます。

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

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。

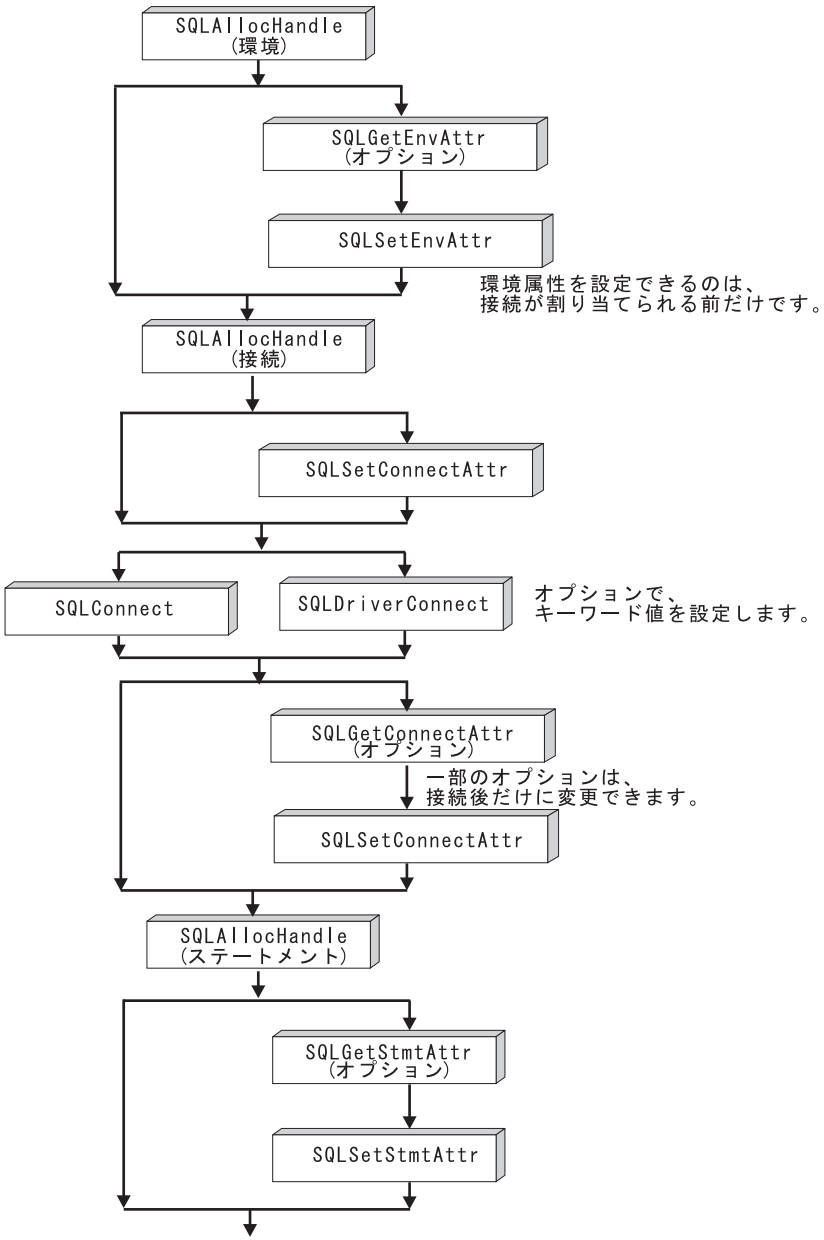


図 12. 属性 (オプション) の設定と取り出し

関連概念:

- 61 ページの『CLI アプリケーション用のプログラミングのヒントと提案』
- 19 ページの『CLI でのハンドル』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetConnectAttr 関数 (CLI) - 現在の属性設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetEnvAttr 関数 (CLI) - 現行の環境属性値の検索』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』

- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『SQLSetEnvAttr 関数 (CLI) - 環境属性の設定』
- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『ステートメント属性 (CLI) のリスト』
- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『接続属性 (CLI) リスト』
- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『環境属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

関連サンプル:

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

第 16 章 システム・カタログ情報の照会

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

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 1 つ以上の表を選択します。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、そのような DB2 コマンドのために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 (スキーマ関数とも呼ばれる) 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。

- `SQLColumnPrivileges()`
- `SQLColumns()`
- `SQLForeignKeys()`
- `SQLGetTypeInfo()`
- `SQLPrimaryKeys()`
- `SQLProcedureColumns()`
- `SQLProcedures()`
- `SQLSpecialColumns()`
- `SQLStatistics()`
- `SQLTablePrivileges()`
- `SQLTables()`

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。

関連概念:

- 186 ページの『CLI アプリケーションのカタログ関数の入力引き数』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『CLI と ODBC 関数のサマリー』

CLI アプリケーションのカタログ関数の入力引き数

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。

カタログ関数の入力引き数は、普通の引き数として処理される場合と、パターン値引き数として処理される場合があります。普通の引き数はリテラルとして扱われるので、大文字小文字の違いが有効です。これらの引き数は、対象オブジェクトを識別することにより、照会の範囲を制限します。この引き数にアプリケーションが NULL ポインターを渡すとエラーになります。

カタログ関数によっては、いくつかの入力引き数でパターン値を受け入れるものがあります。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。特定の入力引き数がパターン値を受け入れるかどうかについては、各カタログ関数の「関数引き数」のセクションを参照してください。

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、基本照会の WHERE 文節に LIKE 述部が含まれている場合と同じです。パターン値の入力についてアプリケーションが NULL 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、対応する WHERE 文節中の LIKE がない)。カタログ関数に複数のパターン値の入力引き数があると、基本照会内の WHERE 文節の LIKE 述部が 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、LIKE 述部のすべての条件を満たした場合に限り行が現れます。

各パターン値の引き数には、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 NULL 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、特殊な意味のない文字。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。

たとえば、以下の呼び出しは先頭が「ST」の表をすべて取り出します。

```
/* tbinfo.c */
/* ... */
struct
{   SQLINTEGER ind ;
    SQLCHAR    val[129] ;
} tbQualifier, tbSchema, tbName, tbType;

struct
{   SQLINTEGER ind ;
    SQLCHAR val[255] ;
} tbRemarks;

SQLCHAR tbSchemaPattern[] = "%";
SQLCHAR tbNamePattern[] = "ST%";  /* all the tables starting with ST */

/* ... */
sqlrc = SQLTables( hstmt, NULL, 0,
                   tbSchemaPattern, SQL_NTS,
                   tbNamePattern, SQL_NTS,
                   NULL, 0);
/* ... */

/* bind columns to variables */
sqlrc = SQLBindCol( hstmt, 1, SQL_C_CHAR, tbQualifier.val, 129,
                    &tbQualifier.ind ) ;
STMT_HANDLE_CHECK( hstmt, sqlrc);
sqlrc = SQLBindCol( hstmt, 2, SQL_C_CHAR, tbSchema.val, 129,
                    &tbSchema.ind ) ;
STMT_HANDLE_CHECK( hstmt, sqlrc);
sqlrc = SQLBindCol( hstmt, 3, SQL_C_CHAR, tbName.val, 129,
                    &tbName.ind ) ;
STMT_HANDLE_CHECK( hstmt, sqlrc);
sqlrc = SQLBindCol( hstmt, 4, SQL_C_CHAR, tbType.val, 129,
                    &tbType.ind ) ;
STMT_HANDLE_CHECK( hstmt, sqlrc);
sqlrc = SQLBindCol( hstmt, 5, SQL_C_CHAR, tbRemarks.val, 255,
                    &tbRemarks.ind ) ;
STMT_HANDLE_CHECK( hstmt, sqlrc);

/* ... */
sqlrc = SQLFetch( hstmt );
/* ... */
while (sqlrc != SQL_NO_DATA_FOUND)
{   /* ... */
    sqlrc = SQLFetch( hstmt );
    /* ... */
}
```

関連概念:

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

関連サンプル:

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

第 17 章 ベンダー・エスケープ文節

CLI アプリケーションでのベンダー・エスケープ文節

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。

短縮された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:

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ことを考えてみます。

```
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:

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。 SQL 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、SQL ESCAPE 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。

"escape" ODBC エスケープ文節を使用する方法の一例として、列 Name および Growth を備えた表 Customers があるとします。 Growth 列には、メタキャラクター '%' を持つデータが含まれます。次のステートメントでは、Growth の中に 10% から 19% までの間の値だけを持ち、 100% より上の値は持たない Name から、すべての値を選択することになります。

```
SELECT Name FROM Customers WHERE Growth LIKE '1_¥%'{escape '¥'}
```

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、 SQL 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:

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]]...)]***}**

説明:
- [?=] は、戻り値のためのオプショナル・パラメーター・マーカーを指定します。
- *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。

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 はプロシージャーの戻り値として戻りコードを戻します。 **?=** がない場合にストアード・プロシージャーの戻りコードが SQL_SUCCESS でないなら、アプリケーションは `SQLGetDiagRec()` 関数と `SQLGetDiagField()` 関数を使用することによって、 SQLCODE を含む診断情報を取り出すことができます。 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いますが、ベンダーのエスケープ文節を使用する必要があります。たとえば、`CALL storedproc ('aaaa', 1)` というステートメントは失敗しますが、 `{CALL storedproc ('aaaa', 1)}` というステートメントは成功することになります。パラメーターが出力パラメーターである場合、パラメーター・マーカーでなければなりません。

たとえば、DB2 CLI が次のステートメントを変換することを考えてみます。

```
{CALL NETB94(?,?,?)}
```

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* は拡張スカラー関数のリスト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ことを考えてみます。

```
SELECT {fn CONCAT(FIRSTNAME,LASTNAME)} FROM EMPLOYEE
```

以下のように変更します。

```
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、オプション SQL_NUMERIC_FUNCTIONS、 SQL_STRING_FUNCTIONS、SQL_SYSTEM_FUNCTIONS、および SQL_TIMEDATE_FUNCTIONS を指定して呼び出してください。

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetInfo 関数 (CLI) - 一般情報の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLNativeSql 関数 (CLI) - ネイティブ SQL テキストの取得』
- 192 ページの『CLI アプリケーション用の拡張スカラー関数』
- 「*SQL リファレンス 第 1 巻*」の『LIKE 述部』
- 「*SQL リファレンス 第 2 巻*」の『CALL ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『SELECT ステートメント』

関連サンプル:

- 『dbnative.c -- How to translate a statement that contains an ODBC escape clause』

CLI アプリケーション用の拡張スカラー関数

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。

- 193 ページの『ストリング関数』

- 195 ページの『数値関数』
- 197 ページの『日時関数』
- 200 ページの『システム関数』
- 201 ページの『変換関数』

以下の節にでてくる表には、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 5 以降の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 算術オーバーフローまたはアンダーフロー。
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 無効なインターバル・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 にあると見なされます。

表 15. 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	ワークステーション			
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。				

表 15. ストリング・スカラー関数 (続き)

DB2	ワークステーション			
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	ワークステーション	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	ワークステーション			
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	ワークステーション	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	ワークステーション		VM/VSE	
LEFT(*string_exp,count* **)** *string_exp* の文字の左端の *count* を戻します。				
DB2	ワークステーション	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:** DB2 for MVS/ESA には後書きブランクが組み込まれています。				
DB2	ワークステーション	MVS	VM/VSE	AS/400
LOCATE(*string_exp1*, *string_exp2* **[** ,*start* **])** *string_exp2* 内で *string_exp1* が最初に現れた開始位置を戻します。 *string_exp1* が最初に現れた位置の検索は、オプションの引き数 *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	ワークステーション			
LTRIM(*string_exp* **)** 先行ブランクを除いて *string_exp* の文字を戻します。				
DB2	ワークステーション		VM/VSE	AS/400
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	ワークステーション	MVS	VM/VSE	AS/400
REPLACE(*string_exp1*, *string_exp2*, *string_exp3* **)** *string_exp1* 内で出現したすべての *string_exp2* を *string_exp3* で置換します。				
DB2	ワークステーション			
RIGHT(*string_exp*, *count* **)** *string_exp* の文字の右端の count を戻します。				
DB2	ワークステーション	MVS	VM/VSE	AS/400

表 15. ストリング・スカラー関数 (続き)

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	ワークステーション		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	ワークステーション			
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	ワークステーション			
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	ワークステーション	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。				
DB2	ワークステーション		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 16. 数値スカラー関数

ABS(*numeric_exp* **)** *numeric_exp* の絶対値を戻します。				
DB2	ワークステーション			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	ワークステーション			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	ワークステーション			AS/400

表 16. 数値スカラー関数 (続き)

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	ワークステーション			AS/400
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	ワークステーション			
CEILING(*numeric_exp* **)** *numeric_exp* 以上の最短整数を戻します。				
DB2	ワークステーション			
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	ワークステーション			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	ワークステーション			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	ワークステーション			AS/400 3.6
EXP(*double_exp* **)** *double_exp* の指数値を戻します。				
DB2	ワークステーション			AS/400
FLOOR(*numeric_exp* **)** *numeric_exp* 以下の最大整数を戻します。				
DB2	ワークステーション			AS/400 3.6
LOG(*double_exp* **)** *double_exp* の自然対数を戻します。				
DB2	ワークステーション			AS/400
LOG10(*double_exp* **)** *double_exp* の 10 を底とする対数 (常用対数) を戻します。				
DB2	ワークステーション			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	ワークステーション			AS/400
PI() πの定数値を浮動小数点値として戻します。				
DB2	ワークステーション			AS/400
POWER(*numeric_exp*, *integer_exp* **)** 値 *numeric_exp* の *integer_exp* 乗を戻します。				

表 16. 数値スカラー関数 (続き)

DB2	ワークステーション			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	ワークステーション			
RAND([*integer_exp* **])** *integer_exp* をオプション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	ワークステーション			
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	ワークステーション			
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。				
DB2	ワークステーション			
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	ワークステーション			AS/400
SQRT(*double_exp* **)** *double_exp* の平方根を戻します。				
DB2	ワークステーション			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	ワークステーション			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	ワークステーション			

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 17. 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	ワークステーション	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	ワークステーション	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	ワークステーション			
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	ワークステーション	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	ワークステーション			AS/400 3.6
DAYOFWEEK_ISO(*date_exp* **)** 1 週間の曜日を、1 から 7 の範囲の整数値として *date_exp* で戻します。 1 は月曜を表します。この関数と DAYOFWEEK() 関数との間の相違に注意してください。				
DB2	ワークステーション			
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	ワークステーション			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	ワークステーション	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。				
DB2	ワークステーション			
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	ワークステーション	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	ワークステーション	MVS	VM/VSE	AS/400

表 17. 日時スカラー関数 (続き)

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	ワークステーション			
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	ワークステーション	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	ワークステーション			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	ワークステーション	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* に小数秒のコンポーネントがある場合、小数秒は廃棄されます。				
DB2	ワークステーション			
TIMESTAMPADD(*interval*, *integer_exp*, *timestamp_exp* **)** タイプ *interval* の *integer_exp* インターバルを *timestamp_exp* に加算して計算されたタイム・スタンプを戻します。インターバルの有効値は、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、どのインターバルがサポートされているかを判別します。				
DB2	ワークステーション			

表 17. 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* のインターバルの整数を戻します。インターバルの有効値は、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、どのインターバルがサポートされているかを判別します。				
DB2	ワークステーション			
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	ワークステーション			AS/400 3.6
WEEK_ISO(*date_exp* **)** 1 年の週を、1 から 53 までの範囲の整数値として *date_exp* に戻します。 Week 1 は年の最初の週で、木曜を含みます。そのため、月曜日が週の最初の日であると考えられているため、 Week1 は Jan 4 が含まれる最初の週と同じことになります。 WEEK_ISO() は、54 までの値を返す WEEK() の現行定義とは異なることに注意してください。 WEEK() 関数では、Week 1 は、最初の日曜を含む週となります。これは、週に 1 日しか含まれていなくても、Jan. 1 が含まれる週と同じことになります。				
DB2	ワークステーション			
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	ワークステーション	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。

DAYOFWEEK_ISO() および WEEK_ISO() は、 DB2 バージョン 7 以降で作成されたデータベースで自動的に使用できます。データベースがバージョン 7 以前に作成された場合、これらの関数は使用できない可能性があります。 DAYOFWEEK_ISO() および WEEK_ISO() 関数をそのようなデータベースで使用可能にするには、**db2updb** システム・コマンドを使用してください。

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 18. 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	ワークステーション	MVS	VM/VSE	AS/400
IFNULL(*exp*, *value* **)** *exp* が NULL の場合、*value* が戻されます。 *exp* が NULL 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	ワークステーション	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	ワークステーション	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リジェクトされます。

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 19. 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。				
DB2	ワークステーション			

関連概念:

- 189 ページの『CLI アプリケーションでのベンダー・エスケープ文節』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』

第 18 章 組み込み SQL と DB2 CLI の混合

組み込み SQL と DB2 CLI の混合に関する考慮事項

アプリケーションの中で、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。この 2 つは混在させないようにします。アプリケーションでは、以下の 2 つのオプションを使用できます。
  - DB2 CLI 呼び出しを使用してすべての接続およびコミット/ロールバックを実行してから、組み込み SQL を使用して作成された関数を呼び出すもの。
  - 組み込み SQL を使用してすべての接続およびコミット/ロールバックを実行してから、 DB2 CLI API を使用する関数を呼び出す、特に NULL 接続を呼び出すもの。
- 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 を使用して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 を実行する前に、`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 処理を実行するときの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。

関連概念:

- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『DB2 CLI と組み込み動的 SQL の比較』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetConnection 関数 (CLI) - 接続ハンドルの設定』

関連サンプル:

- 『dbmconx.c -- How to use multiple databases with embedded SQL.』
- 『dbusemx.sqc -- How to execute embedded SQL statements in CLI』

第 19 章 CLI/ODBC/JDBC 静的プロファイル

CLI/ODBC/JDBC 静的プロファイル作成による静的 SQL の作成

CLI/ODBC/JDBC 静的プロファイル作成機能により、アプリケーションのエンド・ユーザーは、動的 SQL の代わりに静的 SQL を使用することができるようになります。その結果、実行時のパフォーマンスが改善され、パッケージ・ベースの認証メカニズムによって、セキュリティーがより堅固になります。

制限:

- 事前バインドの静的 SQL ステートメントがあるアプリケーションを実行する際、動的ステートメントの振る舞いを制御するレジスターは静的に変換されたステートメントの影響を受けません。
- アプリケーションが後続の DML (データ操作言語) ステートメントを参照するオブジェクトに DDL (データ定義言語) を発行する場合、取り込んだファイルの中でこれらのステートメントをすべて検索できます。 CLI/ODBC/JDBC 静的プロファイル・バインド・ツールである db2cap が、それらをバインドしようとします。バインドの試みは、VALIDATE(RUN) BIND オプションをサポートする DBMS では成功しますが、そうでないものは失敗します。このケースでは、アプリケーションは静的プロファイルを使用するべきではありません。
- データベース管理者 (DBA) は、アプリケーション固有の要件に応じて、SQL ステートメントを追加、変更、除去するためのキャプチャー・ファイルを編集することができます。

プロファイル作成セッションでアプリケーションを実行する前に、以下の条件を確認しておいてください。

- SQL ステートメントがプロファイル・セッション中にキャプチャーされるためには、正常に実行されている必要があります (生成される SQLCODE が正数)。マッチング・セッションでは、アンマッチの動的ステートメントは、動的 CLI/ODBC/JDBC 呼び出しとして実行が継続します。
- SQL ステートメントがステートメント・マッチングで有効な候補であるにはキャプチャーされたり、バインドされたりしたステートメントが文字単位で等しくなければなりません。スペースは有効です。たとえば、″COL = 1″ は ″COL=1″ と異なると見なされます。一致するヒット数を増やすため、リテラルの代わりにパラメーター・マーカーを使用します。

すべての動的 CLI/ODBC 呼び出しを取り込んで静的パッケージにグループ化できるとは限らないので、注意してください。考えられる理由は、以下のとおりです。

- アプリケーションが定期的に環境ハンドルを解放していない。キャプチャー・セッション中、特定の環境ハンドルの下でキャプチャーされるステートメントは、その環境ハンドルが解放されて初めてキャプチャー・ファイルに書き込まれます。
- アプリケーションが複雑な制御フローを持っているために、一度のアプリケーションの実行で、すべての実行時条件を網羅することが難しい。

- アプリケーションが SET ステートメントを実行して登録変数を変更する。これらのステートメントは記録されません。一致モードには、動的 SET SQLID および SET SCHEMA ステートメントを検出するための限定された機能が存在し、その動作に応じて静的ステートメントの実行が中断されることに注意してください。しかし、他の SET ステートメントの場合、設定される登録変数に依存した後続の SQL ステートメントは、適切に動作しない可能性があります。
- アプリケーションが DML (データ操作言語) ステートメントを発行する。アプリケーションの複雑さと、これらのステートメントの性質に応じ、 (1) 一致しないか、(2) 実行時に正しく実行されないかのいずれかになります。

動的 SQL と静的 SQL はまったく異なるため、エンド・ユーザーに使用できるようにする前に、 DBA は、必ず静的一致モードでのアプリケーションの動作を確認する必要があります。さらに、静的 SQL は動的 SQL に比べて実行時のパフォーマンスが高いことがあるとはいえ、すべてのステートメントについてそうとは限りません。特定のステートメントに関して静的実行がかえってパフォーマンスを低下させることがテストによって明らかになった場合、 DBA は、キャプチャー・ファイルからそのステートメントを削除することによって、そのステートメントが強制的に動的実行されるようにする場合があります。さらに、動的 SQL とは違って静的 SQL の場合は、パフォーマンスを維持するためにパッケージの再バインドが時々必要になることがあります。特に、パッケージの中で参照されるデータベース・オブジェクトが頻繁に変更される場合には、それが必要になります。 CLI/ODBC/JDBC 静的プロファイルが、実行しているアプリケーションのタイプに適していない場合には、静的 SQL の利点を利用できる別のプログラミング方式 (たとえば、組み込み SQL やストアード・プロシージャーなど) があります。

手順:

既存の動的 SQL ステートメントから静的 SQL ステートメントを作成するには、以下のステップを実行します。

1. アプリケーションによって発行されたすべての動的 SQL ステートメントを取り込み、アプリケーションのプロファイルを作成します。このプロセスは、静的キャプチャー・モードでのアプリケーションの実行というものです。静的キャプチャー・モードをオンにするには、アプリケーションを実行する前に、`db2cli.ini` 構成ファイルの中で CLI/ODBC/JDBC データ・ソースに関して、以下の CLI/ODBC 構成キーワードを設定します。
   - `StaticMode = CAPTURE`
   - `StaticPackage =` *修飾パッケージ名*
   - `StaticCapFile =` *キャプチャー・ファイル名*

   以下に例を示します。

   ```
   [DSN1]
   StaticMode = CAPTURE
   StaticPackage = MySchema.MyPkg
   StaticCapFile = E:¥Shared¥MyApp.cpt
   ```

   重要: `StaticPackage` キーワードについては、必ずスキーマ名を指定するようにします (上記のサンプルでは MySchema)。スキーマが指定されていない場合、指定する名前は、パッケージ名ではなく、コンテナー名であると見なされます。パッケージ名はブランクになります。

結果の静的プロファイルは、テキスト・ベースのキャプチャー・ファイル の形式になり、キャプチャーされた SQL ステートメントについての情報がそこに入れられます。

上記の例のファイルでは、以下の結果が生じます。 Data Source Name 1 (DSN1) はキャプチャー・モードに設定されます。そのパッケージの名前は `MySchema.MyPkg` です。さらに、キャプチャー・ファイル `MyApp.cpt` は、`E:¥Shared¥` ディレクトリーに保管されます。 `StaticMode` キーワードが `CAPTURE` 以外の値 (たとえば、静的キャプチャー・モードをオフにするときに使用する `DISABLED`) に変更されるまでは、これ以降にこのアプリケーションを実行するたびに、SQL ステートメントがキャプチャーされ、キャプチャー・ファイル `MyApp.cpt` に追加されることになります。しかし、ユニークな SQL ステートメントだけがキャプチャーされるため、重複した実行は無視されます。

2. オプション: CLI/ODBC 構成キーワード StaticLogFile を設定し、CLI/ODBC/JDBC 静的プロファイルのログ・ファイルを生成します。ここには、ステートメント取り込みプロセスの状態を判別する、有益な情報が含まれています。
3. アプリケーションを実行します。

   これで、ユニークな SQL ステートメントがキャプチャー・ファイルにキャプチャーされます。重複したステートメントは無視されます。
4. CLI/ODBC 構成キーワード StaticMode を DISABLED に設定して静的キャプチャー・モードを使用不可にするか、最初のステップで設定したキーワードを `db2cli.ini` ファイルから除去します。
5. コマンド行プロセッサーで db2cap コマンドを発行します。 db2cap ユーティリティーは、キャプチャー・ファイルに基づいて静的パッケージを生成します。db2cap ユーティリティーが正常終了したことを示すメッセージを戻さない場合、それはキャプチャー・ファイルの中のステートメントを静的にバインドできなかったということです。 DBA は、障害のあるステートメントをキャプチャー・ファイルから除去してから、 db2cap ユーティリティーを再実行する必要があります。
6. db2cap で処理したキャプチャー・ファイルのコピーを、アプリケーションの各エンド・ユーザーに配布します。すべてのユーザーが同じクライアント・プラットフォームに存在する場合、別の方法として、すべてのユーザーがアクセスできるネットワーク・ディレクトリーの中に、このキャプチャー・ファイルの読み取り専用コピーを配置します。
7. アプリケーションで、動的 SQL ステートメントから静的 SQL ステートメントへのマッピング (静的一致モードとして知られる) を可能にします。このことは、以下の CLI/ODBC 構成キーワードを設定して行います。

   - `StaticMode = MATCH`
   - `StaticCapFile =` *キャプチャー・ファイル名*

   以下に例を示します。

   ```
   [DSN1]
   StaticMode = MATCH
   StaticCapFile = E:¥Shared¥MyApp.cpt
   ```
8. オプション: CLI/ODBC 構成キーワード StaticLogFile を設定することにより、突き合わせセッションにおいて、一致した (したがって静的に実行される) ステ

ートメントの数や一致しなかった (したがって動的に実行される) ステートメントの数などの有用な情報を記録するようにします。 DBA は、その情報を使用することにより、静的プロファイル作成をエンド・ユーザーから利用できるようにする前に、突き合わせモードでの静的プロファイル作成での一致率が受け入れ可能なものになるようにします。

9. アプリケーションを実行します。

関連概念:

- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『静的 SQL を使用する場合の特性とそれを使用する理由』
- 208 ページの『CLI/ODBC/JDBC 静的プロファイル作成のためのキャプチャー・ファイル』

関連資料:

- 「コマンド・リファレンス」の『db2cap - CLI/ODBC 静的パッケージ・バインディング・ツール・コマンド』
- 355 ページの『StaticCapFile CLI/ODBC 構成キーワード』
- 356 ページの『StaticLogFile CLI/ODBC 構成キーワード』
- 357 ページの『StaticMode CLI/ODBC 構成キーワード』
- 357 ページの『StaticPackage CLI/ODBC 構成キーワード』

CLI/ODBC/JDBC 静的プロファイル作成のためのキャプチャー・ファイル

静的プロファイル作成時に生成されるキャプチャー・ファイルは、テキスト・ファイルです。ここには、SQL ステートメントと静的キャプチャー・モードで入手される他の関連情報のテキストが示されています。さらにこれは、さまざまな構成可能な BIND オプションを追跡します。キャプチャーの実行によって入手された特定の値がすでに含まれているものや、ブランクのままのものもあります。ブランクのままの場合、プリコンパイラーは、パッケージのバインド時にデフォルト値を使用します。パッケージ (複数可) をバインドする前に、DBA は、キャプチャー・ファイルを調べ、テキスト・エディターを使用して、これらの BIND オプションに必要な変更を加えることができます。

SQL ステートメントの編集方法を理解しやすくするため、ここで、ステートメント内のフィールドを説明します。

フィールド	説明
SQLID	存在する場合、ステートメントを取り込んだときの SCHEMA または SQLID が、パッケージ (複数可) のデフォルトの QUALIFIER とは異なることを示します。
SECTNO	ステートメントのバインド先である静的パッケージのセクション番号。
ISOLATION	ステートメントの分離レベル。これは、ステートメントが 5 つのパッケージのどれに属するかを決定します。
STMTTEXT	ステートメント・ストリング

フィールド	説明
STMTTYPE	以下の 3 つの値が可能です。 • SELECT_CURSOR_WITHHOLD: 保留カーソルを使用した SELECT ステートメント • SELECT_CURSOR_NOHOLD: 非保留カーソルを使用した SELECT ステートメント • OTHER: SELECT 以外のステートメント
CURSOR	SELECT ステートメントで宣言されたカーソル名
INVARnn	n 番目の入力変数の説明 7 つあるコンマ区切りのフィールドは、以下のものを示します。 1. SQL データ・タイプ 2. データの長さ。 10 進数または浮動小数点タイプの場合、これは精度です。 3. 10 進数または浮動小数点タイプの場合に限り、これは位取りです。 4. 文字データが FOR BIT DATA タイプの場合は TRUE で、それ以外は FALSE です。 5. 変数が NULL 可能な場合は TRUE で、それ以外は FALSE です。 6. 列名 7. この変数が実際の列名を指す場合は SQL_NAMED で、変数がシステム生成名の場合は SQL_UNNAMED です。
OUTVARn	SELECT ステートメントの n 番目の出力変数の説明。コンマ区切りのフィールドは、INVAR と同じ規則に従います。

関連概念:

- 3 ページの『CLI の紹介』

関連タスク:

- 205 ページの『CLI/ODBC/JDBC 静的プロファイル作成による静的 SQL の作成』

第 20 章 CLI/ODBC/JDBC トレース機能

CLI/ODBC/JDBC トレース機能

このトピックでは、以下の対象について説明します。

- 『DB2 CLI および DB2 JDBC トレースの構成』
- 212 ページの『DB2 CLI トレース・オプションと db2cli.ini ファイル』
- 213 ページの『DB2 JDBC トレース・オプションと db2cli.ini ファイル』
- 215 ページの『DB2 CLI ドライバーのトレースと ODBC Driver Manager のトレース』
- 216 ページの『DB2 CLI ドライバー、CLI ベースの Legacy Type 2 JDBC Driver、および DB2 トレース』
- 216 ページの『DB2 CLI と DB2 JDBC のトレースおよび CLI または Java のストアード・プロシージャー』

DB2 CLI および CLI ベースの Legacy Type 2 JDBC Driver for Linux、 UNIX®、および Windows® では、包括的なトレース機能が提供されています。デフォルトでは、これらの機能は使用不可になっており、付加的なコンピューター・リソースを使用しません。これらのトレース機能を使用可能にすると、アプリケーションが適切なドライバー (DB2 CLI または CLI ベースの Legacy Type 2 JDBC Driver) にアクセスしたときに、 1 つ以上のテキスト・ログ・ファイルが生成されます。これらのログ・ファイルには、以下のものに関する詳細情報が記録されています。

- CLI または JDBC 関数がアプリケーションによって呼び出された順序
- CLI または JDBC 関数との間でやり取りされた入出力パラメーターの内容
- CLI または JDBC 関数によって生成された戻りコードおよびエラーまたは警告メッセージ

DB2 CLI および DB2® JDBC トレース・ファイルの分析は、いろいろな仕方でアプリケーション開発者の益になります。まず、プログラム・ロジックおよびパラメーター初期化に関する微妙なエラーが、しばしばトレースで明確になります。 2 番目に、DB2 CLI および DB2 JDBC トレースから、アクセス先のアプリケーションやデータベースをより良く調整する方法が分かる場合があります。たとえば、DB2 CLI トレースで、ある表が特定の属性セットに基づいて何度も照会されていることが示されている場合は、それらの属性に対応する索引を表で作成することによって、アプリケーションのパフォーマンスを向上させることができます。最後に、DB2 CLI および DB2 JDBC トレース・ファイルの分析は、サード・パーティーのアプリケーションやインターフェースがどのように動作しているかをアプリケーション開発者が理解するのに役立ちます。

DB2 CLI および DB2 JDBC トレースの構成:

DB2 CLI および DB2 JDBC トレース機能の構成パラメーターは、いずれも DB2 CLI 構成ファイル `db2cli.ini` から読み取られます。デフォルトでは、このファイルは Windows プラットフォームでは `¥sqllib` パスにあり、 UNIX プラットフォ

ームでは /sqllib/cfg パスにあります。このデフォルト・パスは、DB2CLIINIPATH 環境変数を設定することによってオーバーライドできます。Windows プラットフォームでは、ユーザー定義のデータ・ソースが ODBC Driver Manager によって定義されている場合に、付加的な db2cli.ini ファイルがユーザーのプロファイル (またはホーム) ディレクトリーに存在することがあります。この db2cli.ini ファイルは、デフォルト・ファイルをオーバーライドします。

現在の db2cli.ini トレース構成パラメーターをコマンド行プロセッサーから表示するには、以下のコマンドを発行します。

```
db2 GET CLI CFG FOR SECTION COMMON
```

以下の 3 つの方法で db2cli.ini ファイルを変更すれば、DB2 CLI および DB2 JDBC トレース機能を構成できます。

- DB2 構成アシスタント (使用可能な場合) を使用する
- テキスト・エディターを使用して、 db2cli.ini ファイルを手動で編集する
- UPDATE CLI CFG コマンドをコマンド行プロセッサーから発行する

たとえば、以下のコマンドをコマンド行プロセッサーから発行すると、 db2cli.ini ファイルが更新され、 JDBC トレース機能が使用可能になります。

```
db2 UPDATE CLI CFG FOR SECTION COMMON USING jdbctrace 1
```

注:

1. 通常、DB2 CLI および DB2 JDBC トレース構成オプションは、アプリケーションが初期化されるときにのみ、 db2cli.ini 構成ファイルから読み取られます。ただし、特殊な db2cli.ini トレース・オプションである TraceRefreshInterval を使用すれば、特定の DB2 CLI トレース・オプションが db2cli.ini ファイルから再読み取りされるインターバルを指定できます。
2. DB2 CLI トレース機能は、 SQL_ATTR_TRACE および SQL_ATTR_TRACEFILE 環境属性を設定することにより、動的に構成することもできます。これらの設定は、db2cli.ini ファイルに含まれている設定をオーバーライドします。

重要: 必要でなければ、DB2 CLI および DB2 JDBC トレース機能を使用不可にしてください。不必要なトレースを行うと、アプリケーションのパフォーマンスが低下し、不要なトレース・ログ・ファイルが生成される場合があります。 DB2 では、生成されたトレース・ファイルは削除されず、新しいトレース情報は既存のトレース・ファイルに付加されます。

DB2 CLI トレース・オプションと db2cli.ini ファイル:

DB2 CLI ドライバーを使用するアプリケーションが実行を開始すると、このドライバーは、 db2cli.ini ファイルの [COMMON] セクションにトレース機能オプションがないかどうかをチェックします。これらのトレース・オプションは、db2cli.ini ファイルの [COMMON] セクションで特定の値に設定された特定のトレース・キーワードです。

注: DB2 CLI トレース・キーワードは db2cli.ini ファイルの [COMMON] セクションに存在するので、それらの値は DB2 CLI ドライバーを介したすべてのデータベース接続に適用されます。

定義可能な DB2 CLI トレース・キーワードは以下のとおりです。

- Trace
- TraceComm
- TraceErrImmediate
- TraceFileName
- TraceFlush
- TraceFlushOnError
- TraceLocks
- TracePathName
- TracePIDList
- TracePIDTID
- TraceRefreshInterval
- TraceStmtOnly
- TraceTime
- TraceTimeStamp

注: DB2 CLI トレース・キーワードは、 TraceRefreshInterval キーワードが設定されていない限り、アプリケーションの初期化時に `db2cli.ini` ファイルから一度だけ読み取られます。このキーワードが設定されていると、指定されたインターバルで Trace および TracePIDList キーワードが `db2cli.ini` ファイルから再読み取りされ、適切であれば、現在実行中のアプリケーションに適用されます。

これらの DB2 CLI キーワードおよび値を使用する `db2cli.ini` ファイル・トレース構成の例を以下に示します。

```
[COMMON]
trace=1
TraceFileName=¥temp¥clitrace.txt
TraceFlush=1
```

注:

1. CLI トレース・キーワードでは、大文字小文字の区別がありません。ただし、パスおよびファイル名のキーワード値は、一部のオペレーティング・システム (UNIX など) で大文字小文字の区別がある場合があります。
2. `db2cli.ini` にある DB2 CLI トレース・キーワードか関連値が無効な場合、DB2 CLI トレース機能はそれを無視し、そのトレース・キーワードにデフォルト値を使用します。

DB2 JDBC トレース・オプションと db2cli.ini ファイル:

CLI ベースの Legacy Type 2 JDBC Driver を使用するアプリケーションが実行を開始すると、このドライバーも、db2cli.ini ファイルにトレース機能オプションがないかどうかをチェックします。 DB2 CLI トレース・オプションと同様、DB2 JDBC トレース・オプションは、 `db2cli.ini` ファイルの [COMMON] セクションで、キーワード/値の対として指定されます。

注: DB2 JDBC トレース・キーワードは `db2cli.ini` ファイルの [COMMON] セクションに存在するので、それらの値は CLI ベースの Legacy Type 2 JDBC Driver を介したすべてのデータベース接続に適用されます。

定義可能な DB2 JDBC トレース・キーワードは以下のとおりです。

- JDBCTrace
- JDBCTracePathName
- JDBCTraceFlush

JDBCTrace = 0 | 1

JDBCTrace キーワードは、他の DB2 JDBC トレース・キーワードがプログラム実行に影響するかどうかを制御します。 JDBCTrace をデフォルト値の 0 に設定すると、DB2 JDBC トレース機能が使用不可になります。 JDBCTrace を 1 に設定すると、この機能が使用可能になります。

JDBCTrace キーワードは、JDBCTracePathName キーワードも指定されていない限り、それだけではほとんど効果がなく、トレース出力を生成しません。

JDBCTracePathName = <fully_qualified_trace_path_name>

JDBCTracePathName の値は、すべての DB2 JDBC トレース情報が書き込まれるディレクトリーの完全修飾パスです。 DB2 JDBC トレース機能は、JDBC アプリケーションが CLI ベースの Legacy Type 2 JDBC Driver を使用して実行されるたびに、新しいトレース・ログ・ファイルを生成しようとします。マルチスレッド・アプリケーションの場合は、各スレッドごとに別個のトレース・ログ・ファイルが生成されます。トレース・ログ・ファイルの名前は、アプリケーション・プロセス ID、スレッド・シーケンス番号、およびスレッド識別ストリングを連結したものにより、自動的に付けられます。 DB2 JDBC トレース出力ログ・ファイルが書き込まれるデフォルト・パス名は存在しません。

JDBCTraceFlush = 0 | 1

JDBCTraceFlush キーワードは、トレース情報が DB2 JDBC トレース・ログ・ファイルに書き込まれる頻度を指定します。デフォルトでは、JDBCTraceFlush は 0 に設定されており、各 DB2 JDBC トレース・ログ・ファイルは、トレース対象アプリケーションまたはスレッドが正常終了するまでオープンされたままになっています。アプリケーションが異常終了すると、トレース・ログ・ファイルに書き込まれていない一部のトレース情報が失われる可能性があります。

DB2 JDBC トレース・ログ・ファイルに書き込まれるトレース情報の保全性と完全性を保証するため、 JDBCTraceFlush キーワードを 1 に設定できます。各トレース入力後がトレース・ログ・ファイルに書き込まれると、DB2 JDBC ドライバーはこのファイルをクローズしてから再オープンし、新しいトレース入力後をこのファイルの末尾に付加します。これにより、トレース情報が失われないことが保証されます。

注: *DB2 JDBC ログ・ファイルのクローズおよび再オープン操作が行われるたびに有効な入出力オーバーヘッドが発生するため、パフォーマンスがかなり低下する可能性があります。*

これらの DB2 JDBC キーワードおよび値を使用する db2cli.ini ファイル・トレース構成の例を以下に示します。

```
[COMMON]
jdbctrace=1
JdbcTracePathName=¥temp¥jdbctrace¥
JDBCTraceFlush=1
```

注:

1. JDBC トレース・キーワードでは、大文字小文字の区別がありません。ただし、パスおよびファイル名のキーワード値は、一部のオペレーティング・システム (UNIX など) で大文字小文字の区別がある場合があります。
2. db2cli.ini にある DB2 JDBC トレース・キーワードか関連値が無効な場合、DB2 JDBC トレース機能はそれを無視し、そのトレース・キーワードにデフォルト値を使用します。
3. DB2 JDBC トレースを使用可能にしても、DB2 CLI トレースは使用可能になりません。 CLI ベースのLegacy Type 2 JDBC Driver は、 DB2 CLI ドライバーを使用してデータベースにアクセスします。現在、Java™ 開発者は DB2 CLI トレースを使用可能にすることにより、自分のアプリケーションがさまざまなソフトウェア層を介してデータベースと対話する方法に関する追加情報を得ることができます。 DB2 JDBC および DB2 CLI トレース・オプションは相互に依存しておらず、 db2cli.ini ファイルの [COMMON] セクションで、任意の順序で指定できます。

DB2 CLI ドライバーのトレースと ODBC Driver Manager のトレース:

ODBC Driver Manager トレースと DB2 CLI ドライバー・トレースの違いを理解することは重要です。 ODBC Driver Manager トレースは、 ODBC アプリケーションが ODBC Driver Manager に対して行った ODBC 関数呼び出しを示します。対照的に、DB2 CLI ドライバー・トレースは、 ODBC Driver Manager がアプリケーションに代わって DB2 CLI ドライバーに対して行った関数呼び出しを示します。

ODBC Driver Manager は、一部の関数呼び出しをアプリケーションから DB2 CLI ドライバーに直接転送することがあります。しかし、ODBC Driver Manager は、ドライバーへの一部の関数呼び出しの転送を遅らせたり回避したりすることもあります。また、ODBC Driver Manager は、DB2 CLI ドライバーに呼び出しを転送する前に、アプリケーション関数引き数を変更したり、アプリケーション関数を他の関数にマップしたりすることもあります。

アプリケーション関数呼び出しに ODBC Driver Manager が干渉する理由には、以下のものがあります。

- ODBC 3.0 では使用すべきでなくなった ODBC 2.0 関数を使用して作成されたアプリケーションでは、以前の関数が新しい関数にマップされます。
- ODBC 3.0 で使用すべきでなくなった ODBC 2.0 関数引き数は、等価の ODBC 3.0 引き数にマップされます。
- Microsoft® カーソル・ライブラリーでは、 SQLExtendedFetch() などの呼び出しが、同じ終了結果を実現するように、 SQLFetch() や他のサポート関数への複数の呼び出しにマップされます。
- ODBC Driver Manager の接続プールでは、通常、 SQLDisconnect() 要求が据え置かれます (または、接続が再利用されたときに回避されます)。

これらおよび他の理由により、アプリケーション開発者は ODBC マネージャー・トレースを、 DB2 CLI ドライバー・トレースを補完するものとして活用できます。

ODBC Driver Manager トレースの取り込みと解釈について詳しくは、 ODBC Driver Manager の資料を参照してください。 Windows プラットフォームでは、Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference を参照してください。これはオンライン (http://www.msdn.microsoft.com/) でも使用可能です。

DB2 CLI ドライバー、CLI ベースの Legacy Type 2 JDBC Driver、および DB2 トレース:

CLI ベースの Legacy Type 2 JDBC Driver は、内部で DB2 CLI ドライバーを使用してデータベースにアクセスします。たとえば、Java の getConnection() メソッドは、内部で CLI ベースの Legacy Type 2 JDBC Driver によって DB2 CLI `SQLConnect()` 関数にマップされます。その結果、Java 開発者は DB2 CLI トレースを、DB2 JDBC トレースを補完するものとして活用できます。

DB2 CLI ドライバーは、多くの内部関数および DB2 固有の関数を使用して作業を行います。これらの内部関数および DB2 固有の関数は、DB2 トレースに記録されます。 DB2 トレースは、IBM® サービスによる問題判別および解決を支援するためにのみ存在しているので、アプリケーション開発者にとって役に立ちません。

DB2 CLI と DB2 JDBC のトレースおよび CLI または Java のストアード・プロシージャー:

どのワークステーション・プラットフォームでも、 DB2 CLI および DB2 JDBC トレース機能を使用すれば、 DB2 CLI および DB2 JDBC ストアード・プロシージャーをトレースできます。

これ以前のセクションに載せられている DB2 CLI および DB2 JDBC トレース情報および指示は汎用的なものであり、アプリケーションとストアード・プロシージャーの両方に等しく当てはまります。しかし、データベース・サーバーのクライアントである (また、通常はデータベース・サーバーとは別個のマシンで実行される) アプリケーションとは異なり、ストアード・プロシージャーはデータベース・サーバーで実行されます。したがって、DB2 CLI または DB2 JDBC ストアード・プロシージャーをトレースするときは、以下の付加的なステップを行う必要があります。

- トレース・キーワード・オプションが、 DB2 サーバーに存在する `db2cli.ini` ファイルで指定されていることを確かめる。
- TraceRefreshInterval キーワードがゼロ以外の正の値に設定されていない場合は、データベースの始動時 (つまり、db2start コマンドが発行されるとき) より前に、すべてのキーワードが正しく構成されていることを確かめる。データベース・サーバーが稼働しているときにトレース設定を変更すると、予測不能な結果が生じることがあります。たとえば、サーバーが稼働しているときに TracePathName が変更されると、次にストアード・プロシージャーが実行されるときに、一部のトレース・ファイルは新しいパスに書き込まれ、他のファイルは元のパスに書き込まれることがあります。整合性を保証するため、 Trace または TracePIDList 以外のトレース・キーワードが変更されたときは、サーバーを再始動してください。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 217 ページの『CLI および JDBC トレース・ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetEnvAttr 関数 (CLI) - 環境属性の設定』
- 「*コマンド・リファレンス*」の『db2trc - トレース・コマンド』
- 「*コマンド・リファレンス*」の『GET CLI CONFIGURATION コマンド』
- 「*コマンド・リファレンス*」の『UPDATE CLI CONFIGURATION コマンド』
- 「*管理ガイド: パフォーマンス*」の『その他の変数』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

CLI および JDBC トレース・ファイル

DB2® CLI および DB2 JDBC ドライバーにアクセスするアプリケーションは、DB2 CLI および DB2 JDBC トレース機能を利用できます。これらのユーティリティーは、 DB2 CLI または DB2 JDBC ドライバーによりなされたすべての機能呼び出しを、問題判別で利用するログ・ファイルに記録します。このトピックでは、トレース機能により生成されたこれらのログ・ファイルにアクセスする方法と解釈する方法について説明します。

- 『CLI および JDBC トレース・ファイルの場所』
- 218 ページの『CLI トレース・ファイルの解釈』
- 223 ページの『JDBC トレース・ファイルの解釈』

CLI および JDBC トレース・ファイルの場所:

TraceFileName キーワードが `db2cli.ini` ファイルで使用されて完全修飾ファイル名が指定されている場合、 DB2 CLI トレース・ログ・ファイルは指定された場所に作成されます。 DB2 CLI トレース・ログ・ファイル名として相対ファイル名が指定される場合は、そのファイルの場所は、オペレーティング・システムがアプリケーションの現行パスとして認識する場所に依存します。

注: アプリケーションを実行するユーザーに特定のパスにトレース・ログ・ファイルを書き込む十分な権限がない場合、ファイルは生成されませんが、警告にもエラーにもなりません。

TracePathName および JDBCTracePathName キーワードのうち少なくともどちらかが `db2cli.ini` ファイルで使用されて完全修飾ディレクトリーが指定されている場合、 DB2 CLI および DB2 JDBC トレース・ログ・ファイルは指定された場所に作成されます。 2 つのトレース・ディレクトリーの少なくとも 1 つが相対ディレクトリーで指定される場合は、その場所は、オペレーティング・システムがアプリケーションの現行パスとして認識する場所に基づいて決定されます。

注: アプリケーションを実行するユーザーに指定されているパスにトレース・ファイルを書き込む十分な権限がない場合、ファイルは生成されませんが、警告にもエラーにもなりません。指定されたトレース・パスが存在しない場合、そのパスは作成されません。

TracePathName および JDBCTracePathName キーワードが設定されているときには、 DB2 CLI および DB2 JDBC トレース機能は自動的にアプリケーションのプロセス ID とスレッド・シーケンス番号を使用してトレース・ログ・ファイルに名前を付けます。たとえば、スレッドが 3 つあるアプリケーションの DB2 CLI トレースは次のような DB2 CLI トレース・ログ・ファイルを生成します。`100390.0`、`100390.1`、 `100390.2`。

同様に、スレッドが 2 つある Java™ アプリケーションの DB2 JDBC トレースは、次のような JDBC トレース・ログ・ファイルを生成します。`7960main.trc`、`7960Thread-1.trc`。

注: トレース・ディレクトリーに古いトレース・ログ・ファイルと新しいトレース・ログ・ファイルの両方がある場合、ファイルのタイム・スタンプ情報を使用して最新のトレース・ファイルを見つけることができます。

DB2 CLI または DB2 JDBC トレース出力ファイルが作成されていないように思える場合、次のことを行ってください。

- トレース構成キーワードが `db2cli.ini` ファイルに正しくセットされていることを確認する。コマンド行プロセッサーから `db2 GET CLI CFG FOR SECTION COMMON` コマンドを発行すれば、簡単にこのことを実行できます。
- `db2cli.ini` ファイルを更新した後、アプリケーションが確実に再始動するようにする。特に、DB2 CLI および DB2 JDBC トレース機能はアプリケーションの始動時に初期化されます。いったん初期化されると、DB2 JDBC トレース機能は再構成できません。 DB2 CLI トレース機能は、ランタイムでも再構成できますが、アプリケーションの始動前に TraceRefreshInterval キーワードが適切に指定されている場合に限ります。

  注: Trace および TracePIDList DB2 CLI キーワードだけはランタイムに再構成できます。*他の DB2 CLI キーワード (TraceRefreshInterval を含む) を変更しても、アプリケーションを再始動しなければ反映されません。*
- TraceRefreshInterval キーワードをアプリケーションの始動前に指定し、 Trace キーワードが 0 に初期設定されている場合、 DB2 CLI トレース機能が Trace キーワードの値を再度読むために十分な時間を取るようにしてください。
- TracePathName および JDBCTracePathName キーワードのうち少なくとも 1 つが使用され、トレース・ディレクトリーが指定されている場合、アプリケーションの開始前にそれらのディレクトリーが存在していることを確かめてください。
- アプリケーションに指定されたトレース・ログ・ファイルまたはトレース・ディレクトリーに対する書き込みアクセス権限があることを確かめてください。
- DB2CLIINIPATH 環境変数をチェックしてください。セットされている場合、DB2 CLI および DB2 JDBC トレース機能は `db2cli.ini` ファイルがこの変数で指定されている場所にあるものと想定します。
- アプリケーションが DB2 CLI ドライバーとのインターフェースに ODBC を使用している場合、 `SQLConnect()`、`SQLDriverConnect()`、または `SQLBrowseConnect()` 関数のうちのいずれかが正常に呼び出されていることを確認してください。データベース接続が正常に行われるまで、DB2 CLI トレース・ログ・ファイルには何も書き込まれません。

CLI トレース・ファイルの解釈:

DB2 CLI トレースの先頭には常に、トレースを生成したアプリケーションのプロセス ID、トレースの開始時刻、ローカル DB2 ビルド・レベルや DB2 CLI ドライバーのバージョンを示すヘッダーがあります。たとえば、

```
1   [ Process: 1227, Thread: 1024 ]
2   [ Date, Time:           01-27-2002 13:46:07.535211 ]
3   [ Product:              QDB2/LINUX 7.1.0 ]
4   [ Level Identifier:     02010105 ]
5   [ CLI Driver Version:   07.01.0000 ]
6   [ Informational Tokens: "DB2 v7.1.0","n000510","" ]
```

注: このセクションで使用するトレースの例では、トレースの左側に行番号を追加しています。これらの行番号は説明の助けとして追加したもので、実際の DB2 CLI トレースにはありません。

トレース・ヘッダーのすぐ次には、通常、環境と接続ハンドルの割り振りと初期化に関連したいくつかのトレース項目があります。たとえば、

```
7   SQLAllocEnv( phEnv=&bffff684 )
8       ---> Time elapsed - +9.200000E-004 seconds

9   SQLAllocEnv( phEnv=0:1 )
10      <--- SQL_SUCCESS   Time elapsed - +7.500000E-004 seconds

11  SQLAllocConnect( hEnv=0:1, phDbc=&bffff680 )
12      ---> Time elapsed - +2.334000E-003 seconds

13  SQLAllocConnect( phDbc=0:1 )
14      <--- SQL_SUCCESS   Time elapsed - +5.280000E-004 seconds

15  SQLSetConnectOption( hDbc=0:1, fOption=SQL_ATTR_AUTOCOMMIT, vParam=0 )
16      ---> Time elapsed - +2.301000E-003 seconds

17  SQLSetConnectOption( )
18      <--- SQL_SUCCESS   Time elapsed - +3.150000E-004 seconds

19  SQLConnect( hDbc=0:1, szDSN="SAMPLE", cbDSN=-3, szUID="", cbUID=-3,
                            szAuthStr="", cbAuthStr=-3 )
20      ---> Time elapsed - +7.000000E-005 seconds
21  ( DBMS NAME="DB2/LINUX", Version="07.01.0000", Fixpack="0x22010105" )

22  SQLConnect( )
23      <--- SQL_SUCCESS   Time elapsed - +5.209880E-001 seconds
24  ( DSN=""SAMPLE"" )

25  ( UID=" " )

26  ( PWD="*" )
```

上記のトレース例に、それぞれの DB2 CLI 関数呼び出しごとに 2 つの項目がある (たとえば、 19-21 行と 22-26 行に SQLConnect() 関数呼び出しに対する項目があります) ことに注意してください。 DB2 CLI トレースでは常にこうなります。最初の項目は関数呼び出しに渡された入力パラメーターを示し、 2 番目の項目は関数の出力パラメーター値とアプリケーションへの戻りコードを示します。

上記のトレース例では SQLAllocEnv() 関数が正常に環境ハンドルを割り振ったこと (phEnv=0:1) が 9 行目に示されています。ハンドルはその後、13 行目で正常にデータベース接続ハンドルを割り振った (phDbc=0:1) SQLAllocConnect() 関数に渡されています。次に、15 行目で SQLSetConnectOption() 関数が phDbc=0:1 接続の SQL_ATTR_AUTOCOMMIT 属性を SQL_AUTOCOMMIT_OFF (vParam=0) にセッ

トするために使用されています。最後に、19 行目で SQLConnect() が呼び出され、ターゲット・データベース (SAMPLE) に接続しています。

21 行目の SQLConnect() 関数の入力トレース項目に含められているのは、ターゲット・データベース・サーバーのビルドおよびフィックスパックのレベルです。このトレース項目に表れている他の情報には、入力接続ストリング・キーワードおよびクライアントとサーバーのコード・ページが含まれます。たとえば、次の情報も SQLConnect() トレース項目にあったとします。

```
( Application Codepage=819, Database  Codepage=819,
  Char Send/Recv Codepage=819, Graphic Send/Recv Codepage=819,
  Application Char Codepage=819, Application Graphic Codepage=819 )
```

これは、アプリケーションとデータベース・サーバーが同じコード・ページ (819) を使用していることを意味しています。

SQLConnect() 関数の戻りトレース項目には、重要な接続情報 (上記のトレース例では 24-26 行目) も含まれています。戻り項目に表示される可能性がある追加情報には、接続に適用されるすべての PATCH1 または PATCH2 キーワード値が含まれます。たとえば、PATCH2=27,28 が db2cli.ini ファイルの COMMON セクションに指定されている場合、次の行も SQLConnect() 戻り項目に表示されます。

```
( PATCH2="27,28" )
```

環境と接続に関連したトレース項目の次は、ステートメント関連のトレース項目です。たとえば、

```
27  SQLAllocStmt( hDbc=0:1, phStmt=&bffff684 )
28      ---> Time elapsed - +1.868000E-003 seconds

29  SQLAllocStmt( phStmt=1:1 )
30      <--- SQL_SUCCESS   Time elapsed - +6.890000E-004 seconds

31  SQLExecDirect( hStmt=1:1, pszSqlStr="CREATE TABLE GREETING (MSG
                                VARCHAR(10))", cbSqlStr=-3 )
32      ---> Time elapsed - +2.863000E-003 seconds
33  ( StmtOut="CREATE TABLE GREETING (MSG VARCHAR(10))" )

34  SQLExecDirect( )
35      <--- SQL_SUCCESS   Time elapsed - +2.387800E-002 seconds
```

上記のトレース例では、29 行目で、データベース接続ハンドル (phDbc=0:1) が使用されてステートメント・ハンドル (phStmt=1:1) が割り振られています。それから 31 行目で、準備されていない SQL ステートメントがそのステートメント・ハンドルで実行されています。 TraceComm=1 キーワードが db2cli.ini ファイルでセットされている場合、 SQLExecDirect() 関数呼び出しのトレース項目に次のような追加のクライアント/サーバー通信情報が見られます。

```
SQLExecDirect( hStmt=1:1, pszSqlStr="CREATE TABLE GREETING (MSG
                           VARCHAR(10))", cbSqlStr=-3 )
    ---> Time elapsed - +2.876000E-003 seconds
( StmtOut="CREATE TABLE GREETING (MSG VARCHAR(10))" )

    sqlccsend( ulBytes - 232 )
    sqlccsend( Handle - 1084869448 )
    sqlccsend( ) - rc - 0, time elapsed - +1.150000E-004
    sqlccrecv( )
    sqlccrecv( ulBytes - 163 ) - rc - 0, time elapsed - +2.243800E-002
```

```
SQLExecDirect( )
    <--- SQL_SUCCESS   Time elapsed - +2.384900E-002 seconds
```

このトレース項目にある追加の sqlccsend() および sqlccrecv() 関数呼び出し情報に注意してください。 sqlccsend() 呼び出し情報では、クライアントからサーバーに送られたデータ量、伝送にかかった時間、およびその伝送が成功した (0 = SQL_SUCCESS) ことが示されています。 sqlccrecv() 呼び出し情報には、クライアントがサーバーからの応答を待った時間と応答に含まれていたデータ量が示されています。

しばしば、DB2 CLI トレースには複数のステートメント・ハンドルがあります。ステートメント・ハンドル ID のクローズに注意すると、トレースに表示されている他のすべてのステートメント・ハンドルに関係なく、あるステートメント・ハンドルの実行パスを容易に追えます。

DB2 CLI トレースに表示されるステートメント実行パスは、通常は上記の例よりももっと複雑です。たとえば、

```
36  SQLAllocStmt( hDbc=0:1, phStmt=&bffff684 )
37      ---> Time elapsed - +1.532000E-003 seconds

38  SQLAllocStmt( phStmt=1:2 )
39      <--- SQL_SUCCESS   Time elapsed - +6.820000E-004 seconds

40  SQLPrepare( hStmt=1:2, pszSqlStr="INSERT INTO GREETING VALUES ( ? )",
                cbSqlStr=-3 )
41      ---> Time elapsed - +2.733000E-003 seconds
42  ( StmtOut="INSERT INTO GREETING VALUES ( ? )" )

43  SQLPrepare( )
44      <--- SQL_SUCCESS   Time elapsed - +9.150000E-004 seconds

45  SQLBindParameter( hStmt=1:2, iPar=1, fParamType=SQL_PARAM_INPUT,
                      fCType=SQL_C_CHAR, fSQLType=SQL_CHAR, cbColDef=14,
                      ibScale=0, rgbValue=&080eca70, cbValueMax=15,
                      pcbValue=&080eca4c )
46      ---> Time elapsed - +4.091000E-003 seconds

47  SQLBindParameter( )
48      <--- SQL_SUCCESS   Time elapsed - +6.780000E-004 seconds

49  SQLExecute( hStmt=1:2 )
50      ---> Time elapsed - +1.337000E-003 seconds
51  ( iPar=1, fCType=SQL_C_CHAR, rgbValue="Hello World!!!", pcbValue=14,
      piIndicatorPtr=14 )

52  SQLExecute( )
53      <--- SQL_ERROR   Time elapsed - +5.951000E-003 seconds
```

上記のトレース例では、38 行目で、データベース接続ハンドル (phDbc=0:1) が使用されて 2 番目のステートメント・ハンドル (phStmt=1:2) が割り振られています。 40 行目で、 1 つのパラメーター・マーカーのある SQL ステートメントがそのステートメント・ハンドルで準備されています。次に 45 行目で、適切な SQL タイプ (SQL_CHAR) の入力パラメーター (iPar=1) がそのパラメーター・マーカーにバインドされています。最後に、49 行目でステートメントが実行されています。入力データパラメーター (rgbValue="Hello World!!!", pcbValue=14) の内容と長さの両方が、トレースの 51 行目に表示されていることに注意してください。

SQLExecute() 関数が 52 行目で失敗しています。アプリケーションが、SQLError() のような診断 DB2 CLI 関数を呼び出して失敗の原因を診断する場合、その原因はトレースに表示されます。たとえば、

```
54  SQLError( hEnv=0:1, hDbc=0:1, hStmt=1:2, pszSqlState=&bffff680,
              pfNativeError=&bfffee78, pszErrorMsg=&bffff280,
              cbErrorMsgMax=1024, pcbErrorMsg=&bfffee76 )
55      ---> Time elapsed - +1.512000E-003 seconds

56  SQLError( pszSqlState="22001", pfNativeError=-302, pszErrorMsg="[IBM][CLI
       Driver][DB2/LINUX] SQL0302N  The value of a host variable in the EXECUTE
       or OPEN statement is too large for its corresponding use.
       SQLSTATE=22001", pcbErrorMsg=157 )
57      <--- SQL_SUCCESS   Time elapsed - +8.060000E-004 seconds
```

56 行目で戻されているエラー・メッセージには、生成された DB2 のネイティブ・エラー・コード (SQL0302N)、そのコード (SQLSTATE=22001) に対応する sqlstate およびエラーの要旨が含まれています。この例では、エラーの原因は明らかです。 31 行目で VARCHAR(10) として定義されている列に 49 行目でアプリケーションが 14 文字のストリングを挿入しようとしています。

アプリケーションが SQLError() のような診断機能を呼び出して DB2 CLI 関数の警告やエラー・コードに応答しなかったとしても、警告またはエラー・メッセージは DB2 CLI トレースに書き込まれます。しかし、そのメッセージの位置は、実際にエラーが発生したところの近くにない可能性があります。さらに、エラーまたは警告メッセージがアプリケーションにより検索されなかったことがトレースに示されます。たとえば、検索されなかった場合、次のように、上記の例のエラー・メッセージはもっと後の関連のないと思われる DB2 CLI 関数呼び出しになるまで出力されないかもしれません。

```
SQLDisconnect( hDbc=0:1 )
    ---> Time elapsed - +1.501000E-003 seconds
    sqlccsend( ulBytes - 72 )
    sqlccsend( Handle - 1084869448 )
    sqlccsend( ) - rc - 0, time elapsed - +1.080000E-004
    sqlccrecv( )
    sqlccrecv( ulBytes - 27 ) - rc - 0, time elapsed - +1.717950E-001
( Unretrieved error message="SQL0302N  The value of a host variable in the
  EXECUTE or OPEN statement is too large for its corresponding use.
  SQLSTATE=22001" )

SQLDisconnect( )
    <--- SQL_SUCCESS   Time elapsed - +1.734130E-001 seconds
```

DB2 CLI トレースの最後の部分は、トレースの前の部分で割り振ったデータベース接続や環境ハンドルの解放を示します。たとえば、

```
58  SQLTransact( hEnv=0:1, hDbc=0:1, fType=SQL_ROLLBACK )
59      ---> Time elapsed - +6.085000E-003 seconds
60  ( ROLLBACK=0 )

61  SQLTransact( )
        <--- SQL_SUCCESS   Time elapsed - +2.220750E-001 seconds

62  SQLDisconnect( hDbc=0:1 )
63      ---> Time elapsed - +1.511000E-003 seconds

64  SQLDisconnect( )
65      <--- SQL_SUCCESS   Time elapsed - +1.531340E-001 seconds

66  SQLFreeConnect( hDbc=0:1 )
```

```
67      ---> Time elapsed - +2.389000E-003 seconds

68  SQLFreeConnect( )
69      <--- SQL_SUCCESS   Time elapsed - +3.140000E-004 seconds

70  SQLFreeEnv( hEnv=0:1 )
71      ---> Time elapsed - +1.129000E-003 seconds

72  SQLFreeEnv( )
73      <--- SQL_SUCCESS   Time elapsed - +2.870000E-004 seconds
```

JDBC トレース・ファイルの解釈:

DB2 JDBC トレースの先頭には、常に、重要な環境変数設定値、JDK または JRE レベル、 DB2 JDBC ドライバー・レベル、 DB2 ビルド・レベルなどの重要なシステム情報をリストするヘッダーがあります。たとえば、

```
1   ========================================================
2   |    Trace beginning on 2002-1-28 7:21:0.19
3   ========================================================

4   System Properties:
5   ------------------
6   user.language = en
7   java.home = c:¥Program Files¥SQLLIB¥java¥jdk¥bin¥..
8   java.vendor.url.bug =
9   awt.toolkit = sun.awt.windows.WToolkit
10  file.encoding.pkg = sun.io
11  java.version = 1.1.8
12  file.separator = ¥
13  line.separator =
14  user.region = US
15  file.encoding = Cp1252
16  java.compiler = ibmjitc
17  java.vendor = IBM® Corporation
18  user.timezone = EST
19  user.name = db2user
20  os.arch = x86
21  java.fullversion = JDK 1.1.8 IBM build n118p-19991124 (JIT ibmjitc
                       V3.5-IBMJDK1.1-19991124)
22  os.name = Windows® NT
23  java.vendor.url = http://www.ibm.com/
24  user.dir = c:¥Program Files¥SQLLIB¥samples¥java
25  java.class.path =
         .:C:¥Program Files¥SQLLIB¥lib;C:¥Program Files¥SQLLIB¥java;
         C:¥Program Files¥SQLLIB¥java¥jdk¥bin¥
26  java.class.version = 45.3
27  os.version = 5.0
28  path.separator = ;
29  user.home = C:¥home¥db2user
30  ----------------------------------------
```

注: このセクションで使用するトレースの例では、トレースの左側に行番号を追加しています。これらの行番号は説明の助けとして追加したもので、実際の DB2 JDBC トレースにはありません。

トレース・ヘッダーのすぐ次には、通常、 JDBC 環境の初期化とデータベース接続の確立に関連したいくつかのトレース項目があります。たとえば、

```
31  jdbc.app.DB2Driver -> DB2Driver() (2002-1-28 7:21:0.29)
32  | Loaded db2jdbc from java.library.path
33  jdbc.app.DB2Driver <- DB2Driver() [Time Elapsed = 0.01]

34  DB2Driver - connect(jdbc:db2:sample)
```

```
35  jdbc.app.DB2ConnectionTrace -> connect( sample, info, db2driver, 0, false )
                 (2002-1-28 7:21:0.59)
36  | 10: connectionHandle = 1
37  jdbc.app.DB2ConnectionTrace <- connect() [Time Elapsed = 0.16]

38  jdbc.app.DB2ConnectionTrace -> DB2Connection (2002-1-28 7:21:0.219)
39  | source = sample
40  | Connection handle = 1
41  jdbc.app.DB2ConnectionTrace <- DB2Connection
```

上記のトレース例では、31 行目で、DB2 JDBC ドライバーのロード要求がなされています。この要求が正常に戻ったことが、33 行目で報告されています。

DB2 JDBC トレース機能は、特定の Java クラスを使用してトレース情報をキャプチャーします。上記のトレース例では、それらのトレース・クラスの 1 つである DB2ConnectionTrace が、 35-37 行目と 38-41 行目の 2 つのトレース項目を生成しています。

35 行目には、 connect() メソッドの呼び出しとそのメソッド呼び出しの入力パラメーターが示されています。 37 行目は connect() メソッド呼び出しが正常に戻ったことを示し、 36 行目にはその呼び出しの出力パラメーター (Connection handle = 1) が示されています。

JDBC トレースでは、接続関連の項目、ステートメント関連の項目が続きます。たとえば、

```
42  jdbc.app.DB2ConnectionTrace -> createStatement() (2002-1-28 7:21:0.219)
43  | Connection handle = 1
44  | jdbc.app.DB2StatementTrace -> DB2Statement( con, 1003, 1007 )
          (2002-1-28 7:21:0.229)
45  | jdbc.app.DB2StatementTrace <- DB2Statement() [Time Elapsed = 0.0]
46  | jdbc.app.DB2StatementTrace -> DB2Statement (2002-1-28 7:21:0.229)
47  | | Statement handle = 1:1
48  | jdbc.app.DB2StatementTrace <- DB2Statement
49  jdbc.app.DB2ConnectionTrace <- createStatement - Time Elapsed = 0.01

50  jdbc.app.DB2StatementTrace -> executeQuery(SELECT * FROM EMPLOYEE WHERE
                                  empno = 000010) (2002-1-28 7:21:0.269)
51  | Statement handle = 1:1
52  | jdbc.app.DB2StatementTrace -> execute2( SELECT * FROM EMPLOYEE WHERE
                                    empno = 000010 ) (2002-1-28 7:21:0.269)
52  | | jdbc.DB2Exception -> DB2Exception() (2002-1-28 7:21:0.729)
53  | | | 10: SQLError = [IBM][CLI Driver][DB2/NT] SQL0401N  The data types of
              the operands for the operation "=" are not compatible.
              SQLSTATE=42818
54  | | |     SQLState = 42818
55  | | |     SQLNativeCode = -401
56  | | |     LineNumber = 0
57  | | |     SQLerrmc = =
58  | | jdbc.DB2Exception <- DB2Exception() [Time Elapsed = 0.0]
59  | jdbc.app.DB2StatementTrace <- executeQuery - Time Elapsed = 0.0
```

42 行目と 43 行目では、 JDBC createStatement() メソッドが接続ハンドル 1 で呼び出されたことを DB2ConnectionTrace クラスが報告しています。もう 1 つの DB2 JDBC トレース機能クラス DB2StatementTrace による報告に伴い、そのメソッド内では内部メソッド DB2Statement() が呼び出されています。この社内メソッド呼び出しは、トレース項目内では「ネスト」されているように表示されることに注意してください。 47-49 行目には、メソッドが正常に戻り、ステートメント・ハンドル 1:1 が割り振られたことが示されています。

50 行目では、SQL 照会メソッド呼び出しがステートメント 1:1 で行われていますが、その呼び出しは 52 行目で例外を生成しています。 53 行目で報告されているエラー・メッセージには、生成された DB2 のネイティブ・エラー・コード (SQL0401N)、そのコード (SQLSTATE=42818) に対応する sqlstate およびエラーの要旨が含まれています。この例では、EMPLOYEE.EMPNO 列は CHAR(6) として定義されているので、照会では整数値でない値が想定されます。

関連概念:

- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『その他の変数』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 362 ページの『TraceComm CLI/ODBC 構成キーワード』
- 364 ページの『TraceFileName CLI/ODBC 構成キーワード』
- 367 ページの『TracePathName CLI/ODBC 構成キーワード』
- 369 ページの『TracePIDList CLI/ODBC 構成キーワード』
- 370 ページの『TraceRefreshInterval CLI/ODBC 構成キーワード』

第 21 章 CLI のバインド・ファイルおよびパッケージ名

DB2 CLI のバインド・ファイルおよびパッケージ名

データベースの作成または移行時に、 DB2 CLI パッケージは、自動的にデータベースにバインドされます。ただし、FixPak がクライアントまたはサーバーのいずれかに適用される場合、またはユーザーが意図的にパッケージをドロップした場合には、以下のコマンドを発行することによって db2cli.lst を再バインドする必要があります。

UNIX

```
db2 bind <BNDPATH>/@db2cli.lst blocking all grant public
```

Windows

```
db2 bind "%DB2PATH%¥bnd¥@db2cli.lst" blocking all grant public
```

db2cli.lst ファイルには、 DB2 CLI が DB2 バージョン 8 サーバーに接続するのに必要なバインド・ファイルの名前 (db2clipk.bnd および db2clist.bnd) が含まれています。

ホストおよび iSeries サーバーの場合は、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。

DB2 バージョン 8 CLI パッケージ (db2clist.bnd または db2cli.lst など) をワークステーションやホスト・サーバーにバインドするときに生成される警告が、この場合にも生成されることが予期されます。それは、DB2 は総称バインド・ファイルを使用しますが、 DB2 バージョン 8 CLI パッケージのバインド・ファイル・パッケージには特定のプラットフォームに適用されるセクションが含まれているからです。そのため、サーバーへのバインド中に、そのサーバーに適用されないプラットフォーム固有のセクションを検出すると、 DB2 は警告を生成することがあります。

次に示す警告の例は、バージョン 8 CLI パッケージ (db2clist.bnd または db2cli.lst など) をワークステーション・サーバーにバインドするときに起き得る警告で、無視することができます。

```
LINE    MESSAGES FOR db2clist.bnd
------  ---------------------------------------------------------------------
  235   SQL0440N  No authorized routine named "POSSTR" of type
                  "FUNCTION" having compatible arguments was found.
                  SQLSTATE=42884
```

表 20. 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 Universal Database で必要?	ホスト・サーバーで必要?	説明
db2clipk.bnd	SYSSHxyy	はい	はい	動的プレースホルダー - スモール・パッケージ WITH HOLD
	SYSSNxyy	はい	はい	動的プレースホルダー - スモール・パッケージ NOT WITH HOLD
	SYSLHxyy	はい	はい	動的プレースホルダー - ラージ・パッケージ WITH HOLD
	SYSLNxyy	はい	はい	動的プレースホルダー - ラージ・パッケージ NOT WITH HOLD
db2clist.bnd	SYSSTAT	はい	はい	共通静的 CLI 関数
db2schema.bnd	SQLL9Eyy	はい	いいえ	カタログ関数サポート
db2cliws.bnd	SQLL65zz	サーバー・バージョン 2 から 7 まで	いいえ	DB2 for Intel/UNIX カタログ関数サポート
db2cliv2.bnd	SQLL95zz	サーバー・バージョン 2 から 7 まで	いいえ	共通静的 CLI 関数

表 20. 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 Universal Database で必要?	ホスト・サーバーで必要?	説明
注: • 'S' はスモール・パッケージ、'L' はラージ・パッケージ • 'H' は WITH HOLD、'N' は NOT WITH HOLD • 'x' は分離レベル (0=NC、1=UR、2=CS、3=RS、4=RR) • 'yy' はパッケージ反復 00 から FF まで • 'zz' は各プラットフォームのユニークな値 たとえば、動的パッケージの場合、 • SYSSN100: スモール・パッケージ (65 セクション)、カーソル宣言はすべて非保留カーソルが対象。分離レベル UR でバインドされます。これは、このパッケージの最初の反復です。 • SYSLH401: ラージ・パッケージ (385 セクション)、カーソル宣言はすべて保留カーソルが対象。分離レベル RS でバインドされます。これは、このパッケージの 2 番目の反復です。 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。 BIND オプション SQLERROR(CONTINUE) を使用することにより、すべてのプラットフォーム上で同一のパッケージをバインドでき、そこでサポートされていないステートメントに関するエラーが無視されるようにしてください。				

db2schema.bnd バインド・ファイル:

db2schema.bnd バインド・ファイルは、 DB2 Universal Database for Linux、UNIX、および Windows、バージョン 8 サーバーにおいてデータベースの作成または移行時に自動的にバインドされ、それらのタイプのサーバーにのみ存在します。このバインド・ファイルはサーバー側に存在します。それを手動でバインドすることが必要になるのは、パッケージがユーザーによって意図的にドロップされた場合か、またはデータベースの作成または移行時に SQL1088W (+1088) 警告を受け取った場合だけです。

パッケージ名の形式は NULLID.SQLL9Exx という形式です。 xx は英数字 2 文字の組み合わせです (たとえば NULLID.SQLL9E0L)。必要となるのは、このパッケージの最新バージョンだけです。

パッケージが欠落している場合、それをサーバー上でローカルに再バインドする必要があります。このパッケージをリモート・サーバーに対してバインドしないようにしてください (たとえば、ホスト・データベースに対して)。バインド・ファイルは、インスタンスのホーム・ディレクトリーの sqllib/bnd ディレクトリーにあり、次のコマンドによって再バインドします。

```
bind db2schema.bnd blocking all grant public
```

データベースを作成または移行した後に SQL1088W の警告が出た場合、db2schema.bnd パッケージが欠落しているなら、 APPLHEAPSZ データベース構成パラメーターを 128 以上にしてから再バインドしてください。バインド時にエラーが出ないようにしてください。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『パッケージ』

関連タスク:

- 233 ページの『CLI 環境のセットアップ』

関連資料:

- 「コマンド・リファレンス」の『BIND コマンド』

第 3 部 CLI 環境とアプリケーションの構築

第 22 章 CLI 環境のセットアップ

CLI アプリケーションを実行する前に、CLI 環境をセットアップする必要があります。この章では、UNIX および Windows プラットフォームで、 CLI または ODBC 環境をセットアップする方法について説明します。

CLI 環境のセットアップ

DB2 CLI アプリケーションのランタイム・サポートは、すべての DB2 クライアントに含まれています。 DB2 CLI アプリケーションを作成して実行するためのサポートは、 DB2 Application Development (DB2 AD) Client に含まれています。このセクションでは、 DB2 CLI ランタイム・サポートで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。

前提条件:

CLI 環境をセットアップする前に、アプリケーション開発環境をセットアップしておきます。

手順:

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. DB2 CLI/ODBC ドライバーが DB2 クライアント・インストールの際にインストールされたことを確認します。
2. データベースがリモート・クライアントからアクセスされる場合、 DB2 データベースおよびノードをカタログ化します。

   Windows プラットフォームでは、「CLI/ODBC 設定 (CLI/ODBC Settings)」GUI を使用して、 DB2 データベースをカタログ化することができます。
3. オプション: DB2 CLI/ODBC バインド・ファイルを、次のコマンドでデータベースに明示的にバインドします。

   ```
   db2 bind ~/sqllib/bnd/@db2cli.lst blocking all sqlerror continue ¥
              messages cli.msg grant public
   ```

   Windows プラットフォームでは、「CLI/ODBC 設定 (CLI/ODBC Settings)」GUI を使用して、 DB2 CLI/ODBC バインド・ファイルをデータベースにバインドすることができます。
4. オプション: `db2cli.ini` ファイルを編集して、 DB2 CLI/ODBC 構成キーワードを変更します。このファイルは、 Windows では `sqllib` ディレクトリーにあり、 UNIX プラットフォームでは `sqllib/cfg` ディレクトリーにあります。

Windows プラットフォームでは、「CLI/ODBC 設定 (CLI/ODBC Settings)」GUI を使用して、 DB2 CLI/ODBC 構成キーワードを設定することができます。

上記のステップを完了したら、Windows CLI 環境の設定に進むか、 UNIX で ODBC アプリケーションを実行しているのであれば、UNIX ODBC 環境の設定に進みます。

関連概念:

- 21 ページの『CLI での初期化と終了の概説』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 234 ページの『UNIX ODBC 環境のセットアップ』
- 241 ページの『Windows CLI 環境のセットアップ』

関連資料:

- 「コマンド・リファレンス」の『BIND コマンド』
- 「コマンド・リファレンス」の『CATALOG DATABASE コマンド』

UNIX ODBC 環境のセットアップ

このトピックでは、ODBC アプリケーション用に、 DB2 への UNIX クライアント・アクセスをセットアップする方法を説明します。 (ご使用のアプリケーションが DB2 CLI アプリケーションである場合、『前提条件』の節の作業を実行すると、CLI 環境のセットアップは完了します。)

前提条件:

UNIX ODBC 環境をセットアップする前に、CLI 環境をセットアップしておきます。

手順:

DB2 データベースにアクセスする必要のある UNIX 上の ODBC アプリケーションでは、下記のステップに従います。

1. ODBC Driver Manager 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します。 DB2 は ODBC Driver Manager をインストールしないため、 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 Driver Manager 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。
2. エンド・ユーザーのデータ・ソース構成である `.odbc.ini` をセットアップします。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始まることに注意してください。ほとんどのプラットフォームでは、必要なファイルは通常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。

ASCII エディターを使用して、適切なデータ・ソース構成情報を反映するようファイルを更新します。 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 (例で参照されているのは、 SAMPLE データベース・データ・ソースの構成です)。

- [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、

  AIX では、たとえば次のようになります。

  ```
  [SAMPLE]
  Driver=/u/thisuser/sqllib/lib/libdb2.a
  Description=Sample DB2 ODBC Database
  ```

  Solaris オペレーティング環境では、たとえば以下のようになります。

  ```
  [SAMPLE]
  Driver=/u/thisuser/sqllib/lib/libdb2.so
  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。

64 ビット開発環境が導入されたことにより、特定のパラメーターのサイズの解釈方法に関して、ベンダー間でかなりの不整合がみられます。たとえば、64 ビットの Microsoft ODBC Driver Manager では SQLHANDLE と SQLLEN がいずれも長さ 64 ビットとして処理されますが、 Data Direct Connect およびオープン・ソースの ODBC Driver Manager では、 SQLHANDLE は 64 ビット、そして SQLLEN は 32 ビットとして処理されます。したがって開発者は、どのバージョンの DB2 ドライバーが必要かということに特別に注意を払う必要があります。下記の情報に従って、データ・ソース・スタンザの中で適切な DB2 ドライバーを指定してください。

表 21. CLI および ODBC アプリケーションの DB2 ドライバー

アプリケーションのタイプ	指定する DB2 ドライバー
32 ビット CLI	libdb2.*
32 ビット ODBC	libdb2.*
32 ビット Data Direct Connect for ODBC	db2_36.*
64 ビット CLI	libdb2.*
64 ビット・オープン・ソース ODBC	libdb2o.*
64 ビット Data Direct Connect for ODBC	db2_36.*

注: 指定する DB2 ドライバーのファイル拡張子は、オペレーティング・システムによって違います。拡張子は下記のとおりです。

– .a - AIX

– .so - Linux、Solaris オペレーティング環境
– .sl - HPUX

3. アプリケーション実行環境が、 LIBPATH (AIX の場合) または LD_LIBRARY_PATH (UNIX の場合) 環境変数に `libodbc.a` (AIX の場合) または `libodbc.so` (UNIX の場合) を含めることにより、 ODBC Driver Manager への参照を持っていることを確認します。
4. ODBCINI 環境変数を .ini ファイルの完全修飾パス名に設定することにより、`.odbc.ini` ファイルをシステム全体で使用できるようにします。一部の ODBC Driver Manager は、集中制御を可能にするこの機能をサポートしています。以下の例で、ODBCINI の設定方法を示します。

   C シェルには、

   ```
   setenv ODBCINI /opt/odbc/system_odbc.ini
   ```

   Bourne または Korn シェルには、

   ```
   ODBCINI=/opt/odbc/system_odbc.ini;export ODBCINI
   ```

5. いったん `.odbc.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。

関連概念:

- 11 ページの『DB2 CLI と Microsoft ODBC の比較』
- 21 ページの『CLI での初期化と終了の概説』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 233 ページの『CLI 環境のセットアップ』
- 245 ページの『UNIX での CLI アプリケーションの作成』
- 249 ページの『UNIX での CLI ルーチンの作成』

unixODBC Driver Manager のセットアップ

ODBC Driver Manager は、UNIX プラットフォーム上では、オペレーティング・システムの一部としては提供されていません。そのため、UNIX システム上で ODBC を使用するためには、市販またはオープン・ソースの ODBC Driver Manager が必要です。 unixODBC Driver Manager は、オープン・ソースの ODBC Driver Manager で、サポートされているすべての DB2 UNIX プラットフォーム上の DB2 ODBC アプリケーション用に、 DB2 UDB バージョン 8.1 および DB2 Connect バージョン 8.1 でサポートされています。このトピックでは、unixODBC Driver Manager をセットアップして DB2 と協働させる方法を説明します。詳細については、unixODBC の配布パッケージに同梱されている README ファイルや、unixODBC の Web サイト (http://www.unixodbc.com) を参照してください。

サポート・ステートメント:

unixODBC Driver Manager と DB2 UDB ODBC ドライバーを正しくインストールおよび構成したにもかかわらず、これらの組み合わせに問題が発生した場合は、

DB2 サービス (http://www.ibm.com/software/data/db2/udb/winos2unix/support) に、問題診断の援助を依頼することができます。問題の原因が unixODBC Driver Manager にある場合には、以下のことを行うことができます。

- Easysoft (unixODBC の商用スポンサー) からの技術サポートのサービス契約を購入します (http://www.easysoft.com)。
- http://www.unixodbc.com のオープン・ソース・サポート・チャネルのいずれかに参加します。

手順:

DB2 CLI や ODBC アプリケーションで使用できるように unixODBC Driver Manager をセットアップするには、次のようにします。

1. http://www.unixodbc.com から、unixODBC のソース・コードをダウンロードします。 DB2 UDB バージョン 8.1 では、バージョン 2.2.3 の unixODBC Driver Manager がサポートされています。
2. ソース・ファイルを untar します。

   ```
   gzip -d unixODBC-2.2.3.tar.gz
   tar xf unixODBC-2.2.3.tar
   ```

3. AIX の場合にのみ、C コンパイラーを構成してスレッドを使用可能にします。

   ```
   export CC=xlc_r
   export CCC=xlC_r
   ```

   ドライバー・マネージャーの 64 ビット・バージョンを xlc_r コンパイラーを使用してコンパイルするには、環境変数 OBJECT_MODE および CFLAGS を次のように設定します。

   ```
   export OBJECT_MODE=64
   export CFLAGS=-q64
   ```

4. ホーム・ディレクトリーか、デフォルトの `/usr/local` の下にドライバー・マネージャーをインストールします。
   - (ホーム・ディレクトリーの場合) ソース・ファイルを untar したディレクトリーから、次のコマンドを発行します。

     ```
     ./configure --prefix=$HOME --enable-gui=no --enable-drivers=no
     ```

   - (`/usr/local` をルートにした場合) 次のコマンドを発行します。

     ```
     ./configure --enable-gui=no --enable-drivers=no
     ```

5. 必要なら、次のコマンドを実行してすべての構成オプションを確認します。

   ```
   ./configure --help
   ```

6. ドライバー・マネージャーをビルドおよびインストールします。

   ```
   make
   make install
   ```

   ライブラリーは `[prefix]/lib` ディレクトリーにコピーされ、実行可能ファイルは `[prefix]/bin` ディレクトリーにコピーされます。
7. AIX の場合のみ: DB2 用の ODBC ドライバーから共用ライブラリーを抽出します。これにより、`shr.o` (32 ビット・プラットフォームの場合) および `shr_64.o` (64 ビット・プラットフォームの場合) が生成されます。
   - 32 ビットの場合、

     ```
     cd /u/db2inst1/sqllib/lib
     ar -x libdb2.a
     ```

- 64 ビットの場合、

```
cd /u/db2inst1/sqllib/lib
ar -x -X 64 libdb2o.a
```

AIX 上でこのステップが必要なのは、 unixODBC Driver Manager がドライバーを動的にロードするためです。混乱を避けるため、次のようにして、結果として生成されたファイルの名前を変更し、 INI ファイルが正しいライブラリーを示すようにしてください。

- 32 ビットの場合、

```
mv shr.o db2.o
```

- 64 ビットの場合、

```
mv shr_64.o db2_64.o
```

8. AIX においてのみ、ドライバー・マネージャーを動的にロードする場合には、`libodbc.a`、`libodbcinst.a`、および `libodbccr.a` を抽出します。この作業はオプションです。

```
ar -x libodbc.a
ar -x libodbcinst.a
ar -x libodbccr.a
```

これにより、`[prefix]/lib/so` ディレクトリーの中に、それぞれ `libodbc.so.1`、`libodbcinst.so.1`、および `libodbccr.so.1` が生成されます。

9. アプリケーションをビルドし、 compile および link コマンドに `-L[prefix]/lib -lodbc` オプションを含めることによって、アプリケーションが unixODBC Driver Manager にリンクするようにしてください。
10. 少なくともユーザー INI ファイル (`odbc.ini`) またはシステム INI ファイル (`odbcinst.ini`) のパスを指定し、 ODBCHOME 環境変数をシステム INI ファイルが作成されたディレクトリーに設定してください。

重要: ユーザー INI ファイルやシステム INI ファイルのパスを指定するときは、絶対パスを使用してください。相対パスや環境変数は使用しないでください。

関連タスク:

- 234 ページの『UNIX ODBC 環境のセットアップ』

関連資料:

- 252 ページの『AIX CLI アプリケーションのコンパイルおよびリンク・オプション』
- 259 ページの『HP-UX CLI アプリケーションのコンパイルおよびリンク・オプション』
- 265 ページの『Linux CLI アプリケーションのコンパイルおよびリンク・オプション』
- 268 ページの『Solaris CLI アプリケーションのコンパイルおよびリンク・オプション』
- 239 ページの『unixODBC Driver Manager のビルド・スクリプトおよび構成の例』

unixODBC Driver Manager のビルド・スクリプトおよび構成の例

unixODBC Driver Manager は、 UNIX プラットフォーム上で使用するためのオープン・ソース ODBC Driver Manager です。 DB2 UDB バージョン 8 は、サポートされる DB2 UDB UNIX プラットフォーム上の ODBC アプリケーション用にこのドライバー・マネージャーをサポートします。このトピックでは、DB2 とともに unixODBC Driver Manager を使うときに使用できる、可能なビルド・スクリプトおよび構成のいくつかの例を示します。

サポート・ステートメント:

unixODBC Driver Manager と DB2 UDB ODBC ドライバーを正しくインストールおよび構成したにもかかわらず、これらの組み合わせに問題が発生した場合は、DB2 サービス (http://www.ibm.com/software/data/db2/udb/winos2unix/support) に、問題診断の援助を依頼することができます。問題の原因が unixODBC Driver Manager にある場合には、以下のことを行うことができます。

- Easysoft (unixODBC の商用スポンサー) からの技術サポートのサービス契約を購入します (http://www.easysoft.com)。
- http://www.unixodbc.com のオープン・ソース・サポート・チャネルのいずれかに参加します。

ビルド・スクリプトの例:

以下に示すのは、 unixODBC Driver Manager を使用する環境をセットアップするビルド・スクリプトの例です。

AIX:

```
#! /bin/sh

echo "Unzipping and extracting"
gzip -d unixODBC-2.2.3.tar.gz
tar xf unixODBC-2.2.3.tar

cd unixODBC-2.2.3

#Comment this out if not AIX
export CC=xlc_r
export CCC=xlC_r

echo "Configuring, compiling and installing"
configure --prefix=$HOME --enable-gui=no --enable-drivers=no
make
make install

echo "Setting ini env vars."
export ODBCHOME=~/etc
export ODBCINI=~/odbc.ini

#Comment this out if not AIX
echo "Extracting unixODBC libraries"
cd ~/lib
ar -x libodbc.a
ar -x libodbcinst.a
ar -x libodbccr.a

echo "¥n***Still need to set up your ini files"
```

UNIX (AIX 以外):

```
#! /bin/sh

echo "Unzipping and extracting"
gzip -d unixODBC-2.2.3.tar.gz
tar xf unixODBC-2.2.3.tar

cd unixODBC-2.2.3

echo "Configuring, compiling and installing"
configure --prefix=$HOME --enable-gui=no --enable-drivers=no
make
make install

echo "Setting ini env vars."
export ODBCHOME=~/etc
export ODBCINI=~/odbc.ini

echo "¥n***Still need to set up your ini files"
```

INI ファイル構成の例:

以下に示すのは、unixODBC Driver Manager を使用するユーザーおよびシステム INI ファイルの例です。

ユーザー INI ファイル (odbc.ini):

```
[DEFAULT]
Driver = DB2

[SAMPLE]
DESCRIPTION = Connection to DB2
DRIVER = DB2
```

システム INI ファイル (odbcinst.ini):

```
[DEFAULT]
Description = Default Driver
Driver = /u/db2inst1/sqllib/lib/db2.o
fileusage=1
dontdlclose=1

[DB2]
Description = DB2 Driver
Driver = /u/db2inst1/sqllib/lib/db2.o
fileusage=1
dontdlclose=1

[ODBC]
Trace = yes
Tracefile = /u/user1trc.log
```

このシステム INI ファイルは、トレース・ログ・ファイルを `trc.log` に設定して、 ODBC トレースを使用可能にします。

注: ドライバー・マネージャーをクローズするとき (`SQLDisconnect()` の際など) に問題が発生する場合には、上記の例に示されているように、 `odbcinst.ini` ファイル内に値 `dontdlclose=1` を設定してください。

関連概念:

- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連タスク:

- 234 ページの『UNIX ODBC 環境のセットアップ』
- 236 ページの『unixODBC Driver Manager のセットアップ』

関連資料:

- 252 ページの『AIX CLI アプリケーションのコンパイルおよびリンク・オプション』
- 259 ページの『HP-UX CLI アプリケーションのコンパイルおよびリンク・オプション』
- 265 ページの『Linux CLI アプリケーションのコンパイルおよびリンク・オプション』
- 268 ページの『Solaris CLI アプリケーションのコンパイルおよびリンク・オプション』

Windows CLI 環境のセットアップ

この作業では、 CLI または ODBC を使用して DB2 への Windows クライアント・アクセスを実行する方法を説明します。

前提条件:

Windows CLI 環境をセットアップする前に、CLI 環境をセットアップしておきます。

制限:

Windows 64 ビット・プラットフォームで構成アシスタントを使用する場合、ODBC データ・ソースは、 64 ビット・アプリケーション用にだけ 構成できます。 32 ビット・アプリケーション用の ODBC データ・ソースは、 Windows 64 ビット・オペレーティング・システムに付属する、 Microsoft 32 ビット「ODBC データ ソース アドミニストレータ」(32 ビットの odbcad32.exe) を使用して構成する必要があります。

手順:

DB2 CLI および ODBC アプリケーションが、 Windows 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. Microsoft ODBC Driver Manager 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 オペレーティング・システムでは、インストール時に ODBC コンポーネント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 Driver Manager 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 オペレーティング・システム上で DB2 インストールを再実行し、ODBC コンポーネントを選択します。

注: 最新バージョンの Microsoft ODBC Driver Manager は、 Microsoft Data Access Components (MDAC) の一部として組み込まれていて、 http://www.microsoft.com/data/ からダウンロードして使用できます。

2. DB2 データベースをデータ・ソースとして ODBC Driver Manager に登録します。 Windows 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。
   - 構成アシスタントを使用します。
     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)」プッシュ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. ラジオ・ボタンを使用して、データ・ソースをユーザー、システム、またはファイルのいずれかのデータ・ソースとして追加することができます。
   - Microsoft 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. 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」プッシュ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。
   - `CATALOG` コマンドを使用して、 DB2 データベースをデータ・ソースとして ODBC Driver Manager に登録します。

     ```
     CATALOG [ user | system ] ODBC DATA SOURCE
     ```

     管理者は、このコマンドを使用して、コマンド行プロセッサー・スクリプトを作成し、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。
3. オプション: 構成アシスタントを使用し、次のようにして DB2 CLI/ODBC ドライバーを構成します。
   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)」プッシュ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)」プッシュ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)」プッシュ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。

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。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 21 ページの『CLI での初期化と終了の概説』

関連タスク:

- 23 ページの『CLI アプリケーションの初期設定』
- 233 ページの『CLI 環境のセットアップ』
- 272 ページの『Windows での CLI アプリケーションの作成』
- 276 ページの『Windows での CLI ルーチンの作成』

第 23 章 CLI アプリケーションのビルド

UNIX

以下のセクションでは、サポートされている UNIX オペレーティング・システムにおいて CLI アプリケーションおよびルーチンを構築する方法について説明します。また、各オペレーティング・システムごとのサンプル・ビルド・スクリプトを示し、それらのビルド・スクリプトで使用されているコンパイル・オプションおよびリンク・オプションについて説明します。

UNIX での CLI アプリケーションの作成

DB2 には、CLI プログラムをコンパイルしてリンクするための、ビルド・スクリプトが備えられています。これは、このファイルで作成できるサンプル・プログラムと共に、 `sqllib/samples/cli` ディレクトリーにあります。

スクリプト・ファイル `bldapp` には、 DB2 CLI アプリケーションを作成するためのコマンドが入っています。これは、パラメーターを 4 つまでとりますが、スクリプト・ファイルの中では、変数 `$1`、`$2`、`$3`、および `$4` によって表されます。

パラメーター `$1` には、ソース・ファイルの名前を指定します。必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない CLI アプリケーションに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、オプションとして 3 つのパラメーターが用

意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

プログラムに組み込み SQL が含まれている場合 (拡張子が .sqc の場合) は、embprep スクリプトが呼び出されてそのプログラムをプリコンパイルし、 .c という拡張子のプログラム・ファイルを生成します。

手順:

以下の例では、CLI アプリケーションを作成して実行する方法が示されています。

ソース・ファイル tbinfo.c からサンプル・プログラム tbinfo を作成するには、次のように入力します。

```
bldapp tbinfo
```

結果として、実行可能ファイル tbinfo が作成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
tbinfo
```

組み込み SQL アプリケーションの構築と実行

ソース・ファイル dbusemx.sqc から組み込み SQL アプリケーション dbusemx を作成する場合、次の 3 つの方法があります。

1. 同じインスタンス上のサンプル・データベースに接続している場合には、次のように入力します。

   ```
   bldapp dbusemx
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   bldapp dbusemx *database*

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   bldapp dbusemx *database userid password*

結果として、実行可能ファイル dbusemx が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   dbusemx
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   dbusemx *database*

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   dbusemx *database userid password*

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『アプリケーション開発環境のセットアップ』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『UNIX アプリケーション開発環境のセットアップ』
- 249 ページの『UNIX での CLI ルーチンの作成』

関連資料:

- 252 ページの『AIX CLI アプリケーションのコンパイルおよびリンク・オプション』
- 259 ページの『HP-UX CLI アプリケーションのコンパイルおよびリンク・オプション』
- 265 ページの『Linux CLI アプリケーションのコンパイルおよびリンク・オプション』
- 268 ページの『Solaris CLI アプリケーションのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldapp -- Builds AIX CLI applications』
- 『dbusemx.sqc -- How to execute embedded SQL statements in CLI』
- 『tbinfo.c -- How to get information about tables from the system catalog tables』

UNIX での CLI 複数接続アプリケーションの作成

DB2 には、CLI プログラムをコンパイルしてリンクするための、ビルド・スクリプトが備えられています。これは、このファイルで作成できるサンプル・プログラムと共に、 `sqllib/samples/cli` ディレクトリーにあります。

バッチ・ファイル `bldmc` には、 2 つのデータベースを必要とする DB2 複数接続プログラムを作成するためのコマンドが入っています。コンパイルとリンクのオプションは、`bldapp` で使用されるものと同じです。

最初のパラメーター `$1` には、ソース・ファイルの名前を指定します。 2 番目のパラメーター `$2` には、接続先の最初のデータベースの名前を指定します。 3 番目のパラメーター `$3` には、接続先の 2 番目のデータベースの名前を指定します。それらはすべて必要パラメーターです。

注: makefile には、データベース名のデフォルト値として ″sample″ と ″sample2″ (それぞれ `$2` および `$3`) がハードコーディングされているため、 makefile を使用する場合、それらのデフォルトを使用するのであれば、指定する必要があるのはプログラム名だけです (`$1` パラメーター)。 `bldmc` スクリプトを使用する場合は、 3 つのパラメーターをすべて指定する必要があります。

オプショナル・パラメーターは、ローカル接続の場合は不要ですが、リモート・クライアントからサーバーに接続する場合は必要になります。オプショナル・パラメーターのうち、`$4` と `$5` には、最初のデータベースのためのユーザー ID とパスワードをそれぞれ指定します。また、`$6` と `$7` には、 2 番目のデータベースのためのユーザー ID とパスワードをそれぞれ指定します。

手順:

複数接続のサンプル・プログラム dbmconx には、2 つのデータベースが必要です。sample データベースがまだ作成されていないなら、コマンド行で db2sampl を入力することによってそれを作成できます。 2 番目のデータベース sample2 は、以下のいずれかのコマンドによって作成できます。

データベースをローカルに作成する場合、

```
db2 create db sample2
```

データベースをリモートに作成する場合、

```
db2 attach to <node_name>
db2 create db sample2
db2 detach
db2 catalog db sample2 as sample2 at node <node_name>
```

<node_name> は、データベースの存在するノードです。

複数接続では、TCP/IP Listener が実行されていることも必要になります。そのためには、次のことを実行してください。

1. 環境変数 DB2COMM を TCP/IP に設定します。それには、次のようにします。

   ```
   db2set DB2COMM=TCPIP
   ```

2. サービス・ファイルの中で指定されている TCP/IP サービス名を使用して、データベース・マネージャー構成ファイルを更新します。

   ```
   db2 update dbm cfg using SVCENAME <TCP/IP service name>
   ```

   サービス・ファイルには、各インスタンスごとに 1 つの TCP/IP サービス名が含まれています。それがわからない場合、またはサービス・ファイルを読むためのファイル・アクセス許可がない場合は、システム管理者にお問い合わせください。

3. 以上の変更内容を有効にするため、データベース・マネージャーを停止してから再開します。

   ```
   db2stop
   db2start
   ```

dbmconx プログラムは、以下の 5 個のファイルで構成されています。

dbmconx.c
: 2 つのデータベースに接続するためのメイン・ソース・ファイル。

dbmconx1.sqc
: 最初のデータベースにバインドされたパッケージを作成するためのソース・ファイル。

dbmconx1.h
: dbmconx.sqc に組み込まれている dbmconx1.sqc のためのヘッダー・ファイル。これは、最初のデータベースにバインドされた表を作成したりドロップしたりするための SQL ステートメントにアクセスするために必要です。

dbmconx2.sqc
: 2 番目のデータベースにバインドされたパッケージを作成するためのソース・ファイル。

dbmconx2.h

dbmconx.sqc に組み込まれている dbmconx2.sqc のためのヘッダー・ファイル。これは、2 番目のデータベースにバインドされた表を作成したりドロップしたりするための SQL ステートメントにアクセスするために必要です。

複数接続のサンプル・プログラム dbmconx を作成するには、次のように入力します。

```
bldmc dbmconx sample sample2
```

結果として、実行可能ファイル dbmconx が作成されます。

その実行可能ファイルを実行するには、その実行可能ファイルの名前を入力します。

```
dbmconx
```

プログラムにより、2 つのデータベースに対する 2 フェーズ・コミットのデモが実行されます。

関連概念:

- 145 ページの『CLI アプリケーションでのマルチサイト更新 (2 フェーズ・コミット)』

関連資料:

- 252 ページの『AIX CLI アプリケーションのコンパイルおよびリンク・オプション』
- 259 ページの『HP-UX CLI アプリケーションのコンパイルおよびリンク・オプション』
- 265 ページの『Linux CLI アプリケーションのコンパイルおよびリンク・オプション』
- 268 ページの『Solaris CLI アプリケーションのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldmc -- Builds AIX CLI multi-connection applications』
- 『bldmc -- Builds HP-UX CLI multi-connection applications』
- 『bldmc -- Builds Linux CLI multi-connection applications』
- 『bldmc -- Builds Solaris CLI multi-connection applications』
- 『dbmconx.c -- How to use multiple databases with embedded SQL.』
- 『dbmconx1.h -- Functions used in dbmconx.c』
- 『dbmconx1.sqc -- This file contains functions used in dbmconx.c』
- 『dbmconx2.h -- Functions used in dbmconx.c』
- 『dbmconx2.sqc -- This file contains functions used in dbmconx.c』

UNIX での CLI ルーチンの作成

DB2 には、CLI プログラムをコンパイルしてリンクするための、ビルド・スクリプトが備えられています。これは、このファイルで作成できるサンプル・プログラムと共に、 sqllib/samples/cli ディレクトリーにあります。

スクリプト・ファイル bldrtn には、 DB2 CLI ルーチン (ストアード・プロシージャーおよびユーザー定義関数) を作成するためのコマンドがあります。 bldrtn は、サーバー上で共有ライブラリーを作成します。これは、ソース・ファイル名のパラメーターを取りますが、スクリプト・ファイルの中で、変数 $1 によって表されます。

手順:

ソース・ファイル spserver.c からサンプル・プログラム spserver を構築するには、次のようにします。

1. 次のビルド・スクリプト名およびプログラム名を入力します。

   ```
   bldrtn spserver
   ```

   スクリプト・ファイルは、共有ライブラリーを sqllib/function ディレクトリーにコピーします。

2. 次に、サーバー上で次のように spcat スクリプトを実行し、ルーチンをカタログします。

   ```
   spcat
   ```

   このスクリプトは、サンプル・データベースに接続し、 spdrop.db2 を呼び出すことによって以前にカタログされている場合には、そのルーチンをアンカタログし、spcreate.db2 を呼び出してカタログ化し、最後にデータベースから切断します。 spdrop.db2 および spcreate.db2 スクリプトを個別に呼び出すことも可能です。

3. その後、これが共用ライブラリーの初回ビルドでないなら、データベースを一度停止してから再始動し、新しい共有ライブラリーが認識されるようにします。必要であれば、共有ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共有ライブラリー spserver を作成したなら、 CLI クライアント・アプリケーション spclient を構築することができます。これは、共有ライブラリー内のルーチンを呼び出すアプリケーションです。

クライアント・アプリケーションは、スクリプト・ファイル bldapp を使用することにより、他の CLI クライアント・アプリケーションのように構築することができます。

共有ライブラリーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

説明

データベース
: 接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
: 有効なパスワードです。

クライアント・アプリケーションは共有ライブラリー spserver にアクセスし、ルーチンをサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『UNIX アプリケーション開発環境のセットアップ』
- 245 ページの『UNIX での CLI アプリケーションの作成』

関連資料:

- 255 ページの『AIX CLI ルーチンのコンパイルおよびリンク・オプション』
- 262 ページの『HP-UX CLI ルーチンのコンパイルおよびリンク・オプション』
- 266 ページの『Linux CLI ルーチンのコンパイルおよびリンク・オプション』
- 270 ページの『Solaris CLI ルーチンのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldrtn -- Builds AIX CLI routines (stored procedures and UDFs)』
- 『spclient.c -- Call various stored procedures』
- 『spserver.c -- Definition of various types of stored procedures』
- 『spcat -- To catalog stored procedures on UNIX (C)』
- 『spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc (C)』
- 『spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc (C)』

AIX

AIX アプリケーションのビルド・スクリプト

AIX 上で CLI アプリケーションを構築するための bldapp スクリプトを以下に示します。

```
#! /bin/sh
# SCRIPT: bldapp
# Builds AIX CLI applications
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Set lib32 for 32-bit programs, lib for 64-bit,
# and set extra compile flag for 64-bit programs.
bitwidth=`LANG=C db2level | awk '/bits/{print $5}'`
if [ $bitwidth = "¥"32¥"" ]; then
  LIB=lib32
  EXTRA_CFLAG=
else
  LIB=lib
  EXTRA_CFLAG=-q64
fi

# If an embedded SQL program, precompile and bind it.
if [ -f $1".sqc" ]
then
```

```
  ./embprep $1 $2 $3 $4
fi

# Compile the error-checking utility.
xlc $EXTRA_CFLAG -I$DB2PATH/include -c utilcli.c

# Compile the program.
xlc $EXTRA_CFLAG -I$DB2PATH/include -c $1.c

# Link the program.
xlc $EXTRA_CFLAG -o $1 $1.o utilcli.o -L$DB2PATH/$LIB -ldb2
```

AIX CLI アプリケーションのコンパイルおよびリンク・オプション

AIX IBM C コンパイラーを使用して CLI アプリケーションを作成するために、DB2 ではこれらのコンパイルおよびリンク・オプションを使用することをお勧めします。これらは、sqllib/samples/cli/bldapp ビルド・スクリプトで例示されます。

bldapp のコンパイルおよびリンク・オプション
コンパイル・オプション
xlc IBM C コンパイラー。
$EXTRA_CFLAG 64 ビット環境では値が "-q64"、それ以外の場合は値が含まれていません。
-I$DB2PATH/include DB2 組み込みファイルのロケーションを指定します。例: $HOME/sqllib/include
-c コンパイルのみを実行し、リンクは実行しません。このスクリプトでは、コンパイルとリンクは別個のステップです。
リンク・オプション:
xlc コンパイラーをリンカーのフロントエンドとして使用します。
$EXTRA_CFLAG 64 ビット環境では値が "-q64"、それ以外の場合は値が含まれていません。
-o $1 実行可能プログラムを指定します。
$1.o オブジェクト・ファイルを指定します。
utilcli.o エラー・チェック用のユーティリティー・オブジェクト・ファイルを組み込みます。
-L$DB2PATH/$LIB DB2 ランタイム共有ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/$LIB と指定します。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。
-ldb2 DB2 ライブラリーとリンクします。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

関連タスク:

- 245 ページの『UNIX での CLI アプリケーションの作成』
- 253 ページの『構成ファイルを使用した CLI アプリケーションの作成』

関連資料:

- 255 ページの『AIX CLI ルーチンのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldapp -- Builds AIX CLI applications』

AIX での CLI アプリケーションと構成ファイル

以下のセクションでは、構成ファイルを使用して CLI アプリケーションを構築する方法について説明します。 CLI アプリケーションをこの方法で作成できるのは、AIX 上で VisualAge C++ コンパイラーを使用する場合だけです。

構成ファイルを使用した CLI アプリケーションの作成: DB2 CLI プログラムは、`sqllib/samples/cli` にある構成ファイル `cli.icc` を使用して構築することができます。

手順:

構成ファイルを使用して、ソース・ファイル `tbinfo.c` から DB2 CLI サンプル・プログラム `tbinfo` を構築するには、以下のようにします。

1. CLI 環境変数を設定します。

   ```
   export CLI=tbinfo
   ```

2. `cli.icc` ファイルを使用して異なるプログラムを作成することによって生成された `cli.ics` ファイルが作業ディレクトリーにある場合は、次のコマンドで `cli.ics` ファイルを削除してください。

   ```
   rm cli.ics
   ```

   既存の `cli.ics` ファイルが、再構築するその同じプログラム用に生成されているのであれば、削除する必要はありません。

3. サンプル・プログラムを以下のように入力してコンパイルします。

   ```
   vacbld cli.icc
   ```

   注: `vacbld` コマンドは、VisualAge C++ で提供されます。

結果として、実行可能ファイル `tbinfo` が作成されます。このプログラムを実行するには、次の実行可能ファイル名を入力します。

```
tbinfo
```

組み込み SQL アプリケーションの構築と実行

構成ファイルは、`embprep` ファイルでプログラムをプリコンパイルした後に使用します。この `embprep` ファイルは、ソース・ファイルをプリコンパイルし、プログラムをデータベースにバインドします。プリコンパイルされたファイルをコンパイルするには、 `cli.icc` 構成ファイルを使用します。

ソース・ファイル `dbusemx.sqc` から組み込み SQL アプリケーション `dbusemx` をプリコンパイルする方法には、次の 3 つがあります。

1. 同じインスタンス上のサンプル・データベースに接続している場合には、次のように入力します。

   ```
   embprep dbusemx
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   embprep dbusemx database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   embprep dbusemx database userid password
   ```

結果として、プリコンパイルされた C ファイル dbusemx.c が作成されます。

プリコンパイルした後、この C ファイルは、次のようにして cli.icc ファイルでコンパイルすることができます。

1. 次のように入力して、CLI 環境変数をプログラム名に設定します。

   ```
   export CLI=dbusemx
   ```

2. cli.icc または cliapi.icc ファイルを使用して異なるプログラムを作成することによって生成された cli.ics ファイルが作業ディレクトリーにある場合は、次のコマンドで cli.ics ファイルを削除してください。

   ```
   rm cli.ics
   ```

   既存の cli.ics ファイルが、再構築するその同じプログラム用に生成されているのであれば、削除する必要はありません。

3. サンプル・プログラムを以下のように入力してコンパイルします。

   ```
   vacbld cli.icc
   ```

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   dbusemx
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   dbusemx database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   dbusemx database userid password
   ```

関連タスク:

- 257 ページの『構成ファイルを使用した CLI ストアード・プロシージャーの作成』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『構成ファイルによる C++ 組み込み SQL アプリケーションの構築』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『構成ファイルによる C++ DB2 API アプリケーションの構築』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『構成ファイルによる C++ ストアード・プロシージャーの構築』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『構成ファイルによる C++ ユーザー定義関数の構築』

- 「アプリケーション開発ガイド アプリケーションの構築および実行」の『構成ファイルによる VisualAge C++ プログラムの構築』

関連サンプル:

- 『dbusemx.sqc -- How to execute embedded SQL statements in CLI』
- 『tbinfo.c -- How to get information about tables from the system catalog tables』

AIX ルーチンのビルド・スクリプト

AIX 上で CLI ルーチンを構築するための `bldrtn` スクリプトを以下に示します。

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds AIX CLI routines (stored procedures and UDFs)
# Usage: bldrtn <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Set lib32 for 32-bit programs, lib for 64-bit,
# and set extra compile flag for 64-bit programs.
bitwidth=`LANG=C db2level | awk '/bits/{print $5}'`
if [ $bitwidth = "¥"32¥"" ]; then
  LIB=lib32
  EXTRA_CFLAG=
else
  LIB=lib
  EXTRA_CFLAG=-q64
fi

# Compile the error-checking utility.
xlc_r $EXTRA_CFLAG -I$DB2PATH/include -c utilcli.c

# Compile the program.
xlc_r $EXTRA_CFLAG -I$DB2PATH/include -c $1.c

# Link the program.
xlc_r $EXTRA_CFLAG -qmkshrobj -o $1 $1.o utilcli.o -L$DB2PATH/$LIB ¥
  -ldb2 -bE:$1.exp

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

AIX CLI ルーチンのコンパイルおよびリンク・オプション

AIX IBM C コンパイラーを使用して CLI ルーチン (ストアード・プロシージャーおよびユーザー定義関数) を作成するために、 DB2 ではこれらのコンパイルおよびリンク・オプションを使用することをお勧めします。これらは、`sqllib/samples/cli/bldrtn` ビルド・スクリプトで例示されます。

bldrtn のコンパイルおよびリンク・オプション
コンパイル・オプション
xlc_r マルチスレッド・バージョンの IBM C コンパイラーを使用します。これは、ルーチンが他のルーチンと同じプロセスで実行される場合 (THREADSAFE)、またはエンジンそれ自体で実行される場合 (NOT FENCED) に必要になります。
$EXTRA_CFLAG 64 ビット環境では値が "-q64"、それ以外の場合は値が含まれていません。
-I$DB2PATH/include DB2 組み込みファイルのロケーションを指定します。例: `$HOME/sqllib/include`
-c コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。
リンク・オプション:
xlc_r マルチスレッド・バージョンのコンパイラーをリンカーのフロントエンドとして使用します。
$EXTRA_CFLAG 64 ビット環境では値が "-q64"、それ以外の場合は値が含まれていません。
-qmkshrobj 共有ライブラリーを作成します。
-o $1 実行可能プログラムを指定します。
$1.o オブジェクト・ファイルを指定します。
utilcli.o エラー・チェック用のユーティリティー・オブジェクト・ファイルを組み込みます。
-L$DB2PATH/$LIB DB2 ランタイム共有ライブラリーのロケーションを指定します。たとえば、`$HOME/sqllib/$LIB` と指定します。 `-L` オプションを指定しないと、コンパイラーは次のパスを想定します。 `/usr/lib:/lib`。
-ldb2 DB2 ライブラリーとリンクします。
-bE:$.exp エクスポート・ファイルを指定します。エクスポート・ファイルには、ルーチンのリストが含まれています。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

関連タスク:

- 249 ページの『UNIX での CLI ルーチンの作成』
- 257 ページの『構成ファイルを使用した CLI ストアード・プロシージャーの作成』

関連資料:

- 252 ページの『AIX CLI アプリケーションのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldrtn -- Builds AIX CLI routines (stored procedures and UDFs)』

AIX での CLI ルーチンと構成ファイル

以下のセクションでは、構成ファイルを使用して CLI ルーチンを構築する方法について説明します。 CLI ルーチンをこの方法で作成できるのは、AIX 上で VisualAge C++ コンパイラーを使用する場合だけです。

構成ファイルを使用した CLI ストアード・プロシージャーの作成: DB2 CLI ストアード・プロシージャーは、 `sqllib/samples/cli` にある構成ファイル `clis.icc` を使用して構築することができます。

手順:

構成ファイルを使用して、ソース・ファイル `spserver.c` から DB2 CLI ストアード・プロシージャー `spserver` を作成するには、以下のようにします。

1. 次のように入力して、CLIS 環境変数をプログラム名に設定します。

   ```
   export CLIS=spserver
   ```

2. `clis.icc` ファイルを使用して異なるプログラムを作成することによって生成された `clis.ics` ファイルが作業ディレクトリーにある場合は、次のコマンドで `clis.ics` ファイルを削除してください。

   ```
   rm clis.ics
   ```

   既存の `clis.ics` ファイルが、再構築するその同じプログラム用に生成されているのであれば、削除する必要はありません。

3. サンプル・プログラムを以下のように入力してコンパイルします。

   ```
   vacbld clis.icc
   ```

   注: `vacbld` コマンドは、VisualAge C++ で提供されます。

ストアード・プロシージャーは、サーバー上の `sqllib/function` というパスにコピーされます。

次に、サーバー上で `spcreate.db2` スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースがあるインスタンスのユーザー ID とパスワードを使用して、データベースに接続します。

```
db2 connect to sample userid password
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを一度停止してから再始動し、新しい共有ライブラリーが認識されるようにします。必要であれば、共有ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

ストアード・プロシージャー `spserver` を作成したなら、そのストアード・プロシージャーを呼び出す CLI クライアント・アプリケーション `spclient` を構築できます。 `spclient` は、構成ファイル `cli.icc` を使用して構築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ためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

`spclient` *database userid password*

説明

データベース
接続先のデータベースの名前です。名前は、`sample` またはそのリモート別名、あるいはその他の名前にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共有ライブラリー `spserver` にアクセスし、多くのストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

関連タスク:

- 249 ページの『UNIX での CLI ルーチンの作成』

関連サンプル:

- 『spclient.c -- Call various stored procedures』
- 『spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc (C)』
- 『spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc (C)』

HP-UX

HP-UX アプリケーションのビルド・スクリプト

HP-UX 上で CLI アプリケーションを構築するための `bldapp` スクリプトを以下に示します。

```
#! /bin/sh
# SCRIPT: bldapp
# Builds HP-UX CLI applications
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Determine the HP platform and set correct compile/link options
hpplat=`uname -m`
bitwidth=`LANG=C db2level | awk '/bits/{print $5}'`
if [ $hpplat = "ia64" ]; then
    if [ $bitwidth = "¥"64¥"" ]; then
      EXTRA_CFLAG="+DD64"
      LIB="lib"
    else
      EXTRA_CFLAG="+DD32"
      LIB="lib32"
    fi
else
    if [ $bitwidth = "¥"64¥"" ]; then
```

```
      EXTRA_CFLAG="+DA2.0W"
      LIB="lib"
    else
      EXTRA_CFLAG=
      LIB="lib32"
    fi
fi

# The runtime path is recommended for all applications.
# If you need to use SHLIB_PATH or LD_LIBRARY_PATH, unset
# the RUNTIME variable by commenting out the following line.
RUNTIME=true

if [ "$RUNTIME" != "" ]
then
  EXTRA_LFLAG="-Wl,+b$DB2PATH/$LIB"
else
  EXTRA_LFLAG=""
fi

# If an embedded SQL program, precompile and bind it.
if [ -f $1".sqc" ]
then
  ./embprep $1 $2 $3 $4
fi

# Compile the error-checking utility.
cc $EXTRA_CFLAG -Ae -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc $EXTRA_CFLAG -Ae -I$DB2PATH/include -c $1.c

# Link the program.
cc $EXTRA_CFLAG -o $1 $1.o utilcli.o $EXTRA_LFLAG -L$DB2PATH/$LIB -ldb2
```

HP-UX CLI アプリケーションのコンパイルおよびリンク・オプション

HP-UX C コンパイラーを使用して CLI アプリケーションを構築する場合は、 DB2 ではこの種のコンパイルおよびリンク・オプションが推奨されています。これらは、sqllib/samples/cli/bldapp ビルド・スクリプトで例示されます。

bldapp のコンパイルおよびリンク・オプション
コンパイル・オプション
cc C コンパイラーを使用します。
$EXTRA_CFLAG HP-UX プラットフォームが IA64 の場合、 64 ビット・サポートが有効ならこのフラグの値は **+DD64** であり、 32 ビット・サポートが有効ならその値は **+DD32** です。 HP-UX プラットフォームが PA-RISC の場合、 64 ビット・サポートが有効なら、その値は **+DA2.0W** です。 PA-RISC プラットフォームの 32 ビット・サポートの場合、このフラグには値が含まれません。 **+DD64** IA64 の HP-UX で 64 ビット・コードを生成する場合に使用する必要があります。 **+DD32** IA64 の HP-UX で 32 ビット・コードを生成する場合に使用する必要があります。 **+DA2.0W** PA-RISC の HP-UX で 64 ビット・コードを生成する場合に使用する必要があります。
-Ae HP ANSI 拡張モードを使用可能にします。
-I$DB2PATH/include DB2 組み込みファイルのロケーションを指定します。例: `$HOME/sqllib/include`
-c コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。

bldapp のコンパイルおよびリンク・オプション
リンク・オプション:
cc コンパイラーをリンカーのフロントエンドとして使用します。
$EXTRA_CFLAG HP-UX プラットフォームが IA64 の場合、 64 ビット・サポートが有効ならこのフラグの値は **+DD64** であり、 32 ビット・サポートが有効ならその値は **+DD32** です。 HP-UX プラットフォームが PA-RISC の場合、 64 ビット・サポートが有効なら、その値は **+DA2.0W** です。 PA-RISC プラットフォームの 32 ビット・サポートの場合、このフラグには値が含まれません。 **+DD64** IA64 の HP-UX で 64 ビット・コードを生成する場合に使用する必要があります。 **+DD32** IA64 の HP-UX で 32 ビット・コードを生成する場合に使用する必要があります。 **+DA2.0W** PA-RISC の HP-UX で 64 ビット・コードを生成する場合に使用する必要があります。
-o $1 実行可能プログラムを指定します。
$1.o オブジェクト・ファイルを指定します。
utilcli.o エラー・チェック用のユーティリティー・オブジェクト・ファイルを組み込みます。
$EXTRA_LFLAG ランタイムのパスを指定します。設定する場合、32 ビットならその値は `-Wl,+b$HOME/sqllib/lib32` であり、 64 ビットなら `-Wl,+b$HOME/sqllib/lib` です。設定しない場合、値はありません。
-L$DB2PATH/$LIB DB2 ランタイム共有ライブラリーのロケーションを指定します。 32 ビットの場合は `$HOME/sqllib/lib32`、 64 ビットの場合は `$HOME/sqllib/lib` です。
-ldb2 データベース・マネージャー・ライブラリーとリンクします。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

関連タスク:

- 245 ページの『UNIX での CLI アプリケーションの作成』

関連資料:

- 262 ページの『HP-UX CLI ルーチンのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldapp -- Builds HP-UX C applications (C)』

HP-UX ルーチンのビルド・スクリプト

HP-UX 上で CLI ルーチンを構築するための `bldrtn` スクリプトを以下に示します。

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds HP-UX CLI routines (stored procedures and UDFs)
# Usage: bldrtn <prog_name>
```

```
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Determine the HP platform and set correct compile/link options
hpplat=`uname -m`
bitwidth=`LANG=C db2level | awk '/bits/{print $5}'`
if [ $hpplat = "ia64" ]; then
    if [ $bitwidth = "¥"64¥"" ]; then
      EXTRA_CFLAG="+DD64"
      LIB="lib"
    else
      EXTRA_CFLAG="+DD32"
      LIB="lib32"
    fi
else
    if [ $bitwidth = "¥"64¥"" ]; then
      EXTRA_CFLAG="+DA2.0W"
      LIB="lib"
    else
      EXTRA_CFLAG=
      LIB="lib32"
    fi
fi

# The runtime path is recommended for all applications.
# If you need to use SHLIB_PATH or LD_LIBRARY_PATH, unset
# the RUNTIME variable by commenting out the following line.
RUNTIME=true

if [ "$RUNTIME" != "" ]
then
  EXTRA_LFLAG="+b$DB2PATH/$LIB"
else
  EXTRA_LFLAG=""
fi

# Compile the error-checking utility.
cc $EXTRA_CFLAG +u1 +z -Ae -I$DB2PATH/include ¥
  -D_POSIX_C_SOURCE=199506L -c utilcli.c

# Compile the program.
cc $EXTRA_CFLAG +u1 +z -Ae -I$DB2PATH/include ¥
  -D_POSIX_C_SOURCE=199506L -c $1.c

# Link the program.
ld -b -o $1 $1.o utilcli.o $EXTRA_LFLAG -L$DB2PATH/$LIB ¥
  -ldb2 -lpthread

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

HP-UX CLI ルーチンのコンパイルおよびリンク・オプション

HP-UX C コンパイラーを使用して CLI ルーチンを構築する場合は、 DB2 ではこの種のコンパイルおよびリンク・オプションが推奨されています。これらは、 `sqllib/samples/cli/bldrtn` ビルド・スクリプトで例示されます。

bldrtn のコンパイルおよびリンク・オプション
コンパイル・オプション **cc** C コンパイラー。 **$EXTRA_CFLAG** HP-UX プラットフォームが IA64 の場合、 64 ビット・サポートが有効ならこのフラグの値は **+DD64** であり、 32 ビット・サポートが有効ならその値は **+DD32** です。 HP-UX プラットフォームが PA-RISC の場合、 64 ビット・サポートが有効なら、その値は **+DA2.0W** です。 PA-RISC プラットフォームの 32 ビット・サポートの場合、このフラグには値が含まれません。 **+DD64** IA64 の HP-UX で 64 ビット・コードを生成する場合に使用する必要があります。 **+DD32** IA64 の HP-UX で 32 ビット・コードを生成する場合に使用する必要があります。 **+DA2.0W** PA-RISC の HP-UX で 64 ビット・コードを生成する場合に使用する必要があります。 **+u1** 位置合わせしないデータ・アクセスを認めます。アプリケーションが位置合わせしないデータを使用する場合にのみ使用します。 **+z** 位置に依存しないコードを生成します。 **-Ae** HP ANSI 拡張モードを使用可能にします。 **-I$DB2PATH/include** DB2 組み込みファイルのロケーションを指定します。例: `$HOME/sqllib/include` **-D_POSIX_C_SOURCE=199506L** _REENTRANT が定義されていることを確認する POSIX スレッド・ライブラリー・オプション。これは、ルーチンが他のルーチンと同じプロセスで実行する (THREADSAFE) 場合、またはエンジン自体の中で実行する (NOT FENCED) 場合に必要になります。 **-c** コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。
リンク・オプション: **ld** リンクにリンカーを使用します。 **-b** 通常の実行可能ファイルではなく、共有ライブラリーを作成します。 **-o $1** 実行可能ファイルを指定します。 **$1.o** オブジェクト・ファイルを指定します。 **utilcli.o** エラー・チェック・ユーティリティー・オブジェクト・ファイル中にリンクします。 **$EXTRA_LFLAG** ランタイムのパスを指定します。設定する場合、32 ビットならその値は `+b$HOME/sqllib/lib32` であり、 64 ビットなら `+b$HOME/sqllib/lib` です。設定しない場合、値はありません。 **-L$DB2PATH/$LIB** DB2 ランタイム共有ライブラリーのロケーションを指定します。 32 ビットの場合は `$HOME/sqllib/lib32`、 64 ビットの場合は `$HOME/sqllib/lib` です。 **-ldb2** DB2 ライブラリーとリンクします。 **-lpthread** POSIX スレッド・ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

関連タスク:

- 249 ページの『UNIX での CLI ルーチンの作成』

関連資料:

- 259 ページの『HP-UX CLI アプリケーションのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldrtn -- Builds HP-UX C routines (stored procedures and UDFs) (C)』

Linux

Linux アプリケーションのビルド・スクリプト

Linux 上で CLI アプリケーションを構築するための bldapp スクリプトを以下に示します。

```
# SCRIPT: bldapp
# Builds Linux CLI applications
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Determine if we are running with 32-bit, and
# if we are running with 32-bit on Linux AMD64
LIB="lib"
EXTRA_C_FLAGS=""
HARDWAREPLAT=`uname -m`
bitwidth=`LANG=C db2level | awk '/bits/{print $5}'`
if [ $bitwidth = "¥"32¥"" ]; then
   LIB="lib32"
   if [ "$HARDWAREPLAT" = "x86_64" ]; then
       EXTRA_C_FLAGS="-m32"
   fi
fi

# The runtime path is recommended for all applications.
# If you need to use LD_LIBRARY_PATH, unset the RUNTIME
# variable by commenting out the following line.
RUNTIME=true

if [ "$RUNTIME" != "" ]
then
EXTRA_LFLAG="-Wl,-rpath,$DB2PATH/$LIB"
else
  EXTRA_LFLAG=""
fi

# If an embedded SQL program, precompile and bind it.
if [ -f $1".sqc" ]
then
  ./embprep $1 $2 $3 $4
fi

# Compile the error-checking utility.
gcc $EXTRA_C_FLAGS -I$DB2PATH/include -c utilcli.c

# Compile the program.
gcc $EXTRA_C_FLAGS -I$DB2PATH/include -c $1.c
```

```
# Link the program.
gcc $EXTRA_C_FLAGS -o $1 $1.o utilcli.o $EXTRA_LFLAG -L$DB2PATH/$LIB -ldb2
```

Linux CLI アプリケーションのコンパイルおよびリンク・オプション

GNU/Linux gcc コンパイラーを使用して CLI アプリケーションを構築する場合は、 DB2 ではこの種のコンパイルおよびリンク・オプションが推奨されています。これらは、`sqllib/samples/cli/bldapp` ビルド・スクリプトで例示されます。

bldapp のコンパイルおよびリンク・オプション
コンパイル・オプション
gcc C コンパイラー。
$EXTRA_C_FLAGS Linux AMD64 上の 32 ビットの場合は `-m32`、それ以外の場合は値が含まれません。
-I$DB2PATH/include DB2 組み込みファイルのロケーションを指定します。例: `$HOME/sqllib/include`
-c コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。
リンク・オプション:
gcc コンパイラーをリンカーのフロントエンドとして使用します。
$EXTRA_C_FLAGS Linux AMD64 上の 32 ビットの場合は `-m32`、それ以外の場合は値が含まれません。
-o $1 実行可能ファイルを指定します。
$1.o プログラム・オブジェクト・ファイルを組み込みます。
utilcli.o エラー・チェック用のユーティリティー・オブジェクト・ファイルを組み込みます。
$EXTRA_LFLAG 「RUNTIME=true」がコメントになっていない場合、その値は 32 ビットなら ″-Wl,-rpath,$DB2PATH/lib32″、また 64 ビットならその値は ″-Wl,-rpath,$DB2PATH/lib″ です。コメントになっている場合、値はありません。
-L$DB2PATH/$LIB リンク時の DB2 静的ライブラリーおよび共有ライブラリーのロケーションを示します。たとえば、32 ビットの場合は `$HOME/sqllib/lib32`、 64 ビットの場合は `$HOME/sqllib/lib` のように指定します。 `-L` オプションを指定しない場合、`/usr/lib:/lib` であると見なされます。
-ldb2 DB2 ライブラリーとリンクします。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

関連タスク:

- 245 ページの『UNIX での CLI アプリケーションの作成』

関連資料:

- 266 ページの『Linux CLI ルーチンのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldapp -- Builds Linux C applications (C)』

Linux ルーチンのビルド・スクリプト

Linux 上で CLI ルーチンを構築するための `bldrtn` スクリプトを以下に示します。

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds Linux CLI routines (stored procedures or UDFs)
# Usage: bldrtn <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Determine if we are running with 32-bit, and
# if we are running with 32-bit on Linux AMD64
LIB="lib"
EXTRA_C_FLAGS=""
HARDWAREPLAT=`uname -m`
bitwidth=`LANG=C db2level | awk '/bits/{print $5}'`
if [ $bitwidth = "¥"32¥"" ]; then
   LIB="lib32"
   if [ "$HARDWAREPLAT" = "x86_64" ]; then
       EXTRA_C_FLAGS="-m32"
   fi
fi

# Set the runtime path.
EXTRA_LFLAG="-Wl,-rpath,$DB2PATH/$LIB"

# Compile the error-checking utility.
gcc $EXTRA_C_FLAGS -fpic -I$DB2PATH/include -c utilcli.c -D_REENTRANT

# Compile the program.
gcc $EXTRA_C_FLAGS -fpic -I$DB2PATH/include -c $1.c -D_REENTRANT

# Link the program.
gcc $EXTRA_C_FLAGS -o $1 $1.o utilcli.o -shared $EXTRA_LFLAG ¥
  -L$DB2PATH/$LIB -ldb2 -lpthread

# Copy the shared library to the function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Linux CLI ルーチンのコンパイルおよびリンク・オプション

GNU/Linux gcc コンパイラーを使用して CLI ルーチンを構築する場合は、 DB2 ではこの種のコンパイルおよびリンク・オプションが推奨されています。これらは、`sqllib/samples/cli/bldrtn` ビルド・スクリプトで例示されます。

bldrtn のコンパイルおよびリンク・オプション
コンパイル・オプション **gcc** C コンパイラー。 **$EXTRA_C_FLAGS** Linux AMD64 上の 32 ビットの場合は -m32、それ以外の場合は値が含まれません。 **-fpic** 位置独立コードを使用できます。 **-I$DB2PATH/include** DB2 組み込みファイルのロケーションを指定します。例: $HOME/sqllib/include **-c** コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。 **-D_REENTRANT** _REENTRANT を定義します。これは、構築中のルーチンが他のルーチンと同じプロセス中で実行する (THREADSAFE) か、またはエンジン自体の中で実行する (NOT FENCED) 場合に必要になります。
リンク・オプション: **gcc** コンパイラーをリンカーのフロントエンドとして使用します。 **$EXTRA_C_FLAGS** Linux AMD64 上の 32 ビットの場合は -m32、それ以外の場合は値が含まれません。 **-o $1** 実行可能ファイルを指定します。 **$1.o** プログラム・オブジェクト・ファイルを組み込みます。 **utilcli.o** エラー・チェック用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-shared** 共有ライブラリーを生成します。 **$EXTRA_LFLAG** ランタイムの DB2 共有ライブラリーのロケーションを示します。 32 ビットの場合、その値は ″-Wl,-rpath,$DB2PATH/lib32″ です。 64 ビットの場合、その値は ″-Wl,-rpath,$DB2PATH/lib″ です。 **-L$DB2PATH/$LIB** リンク時の DB2 静的ライブラリーおよび共有ライブラリーのロケーションを示します。たとえば、32 ビットの場合は $HOME/sqllib/lib32、 64 ビットの場合は $HOME/sqllib/lib のように指定します。 -L オプションを指定しない場合、/usr/lib:/lib であると見なされます。 **-ldb2** DB2 ライブラリーとリンクします。 **-lpthread** POSIX スレッド・ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

関連タスク:

- 249 ページの『UNIX での CLI ルーチンの作成』

関連資料:

- 265 ページの『Linux CLI アプリケーションのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldrtn -- Builds Linux C routines (stored procedures or UDFs) (C)』

Solaris

Solaris アプリケーションのビルド・スクリプト

Solaris 上で CLI アプリケーションを構築するための bldapp スクリプトを以下に示します。

```
#! /bin/sh
# SCRIPT: bldapp
# Builds Solaris CLI applications
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Set compile and link flags for 32-bit and 64-bit programs.
bitwidth=`LANG=C db2level | awk '/bits/{print $5}'`
if [ $bitwidth = "¥"64¥"" ];
then
  CFLAG_ARCH=v9
  LIB=lib
else
  CFLAG_ARCH=v8plusa
  LIB=lib32
fi

# Set the runtime path.
# LD_LIBRARY_PATH will be followed instead of the runtime path unless
# you unset LD_LIBRARY_PATH first to allow the runtime path to be used.
EXTRA_LFLAG="-R$DB2PATH/$LIB"

# If an embedded SQL program, precompile and bind it.
if [ -f $1".sqc" ]
then
  ./embprep $1 $2 $3 $4
fi

# Compile the error-checking utility.
cc -xarch=$CFLAG_ARCH -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc -xarch=$CFLAG_ARCH -I$DB2PATH/include  -c $1.c

# Link the program.
cc -xarch=$CFLAG_ARCH -mt -o $1 $1.o utilcli.o ¥
  -L$DB2PATH/$LIB $EXTRA_LFLAG -ldb2
```

Solaris CLI アプリケーションのコンパイルおよびリンク・オプション

Solaris C コンパイラーを使用して CLI アプリケーションを構築する場合、 DB2 ではこれらのコンパイルおよびリンク・オプションが推奨されています。 これらは、sqllib/samples/cli/bldapp ビルド・スクリプトで例示されます。

bldapp のコンパイルおよびリンク・オプション
コンパイル・オプション **cc** C コンパイラーを使用します。 **-xarch=$CFLAG_ARCH** このオプションは、`libdb2.so` とのリンク時にコンパイラーが有効な実行可能プログラムを確実に生成するようにします。 $CFLAG_ARCH の値は、32 ビットの場合は「v8plusa」、64 ビットの場合は「v9」に設定されます。 **-I$DB2PATH/include** DB2 組み込みファイルのロケーションを指定します。例: `$HOME/sqllib/include` **-c** コンパイルのみを実行し、リンクは実行しません。このスクリプトでは、コンパイルとリンクは別個のステップです。
リンク・オプション: **cc** コンパイラーをリンカーのフロントエンドとして使用します。 **-xarch=$CFLAG_ARCH** このオプションは、`libdb2.so` とのリンク時にコンパイラーが有効な実行可能プログラムを確実に生成するようにします。 $CFLAG_ARCH の値は、32 ビットの場合は「v8plusa」、64 ビットの場合は「v9」に設定されます。 **-mt** マルチスレッド・サポートにリンクし、fopen の呼び出し時に問題が起きないようにします。 **注:** POSIX スレッドを使用する際には、DB2 アプリケーションはスレッド化されているかどうかにかかわらず、 `-lpthread` とリンクする必要もあります。 **-o $1** 実行可能プログラムを指定します。 **$1.o** プログラム・オブジェクト・ファイルを組み込みます。 **utilcli.o** エラー・チェック用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/$LIB** リンク時の DB2 静的ライブラリーおよび共有ライブラリーのロケーションを示します。たとえば、32 ビットの場合は `$HOME/sqllib/lib32`、 64 ビットの場合は `$HOME/sqllib/lib` のように指定します。 **$EXTRA_LFLAG** ランタイムの DB2 共有ライブラリーのロケーションを示します。 32 ビットの場合その値は ″-R$DB2PATH/lib32″、 64 ビットの場合の値は ″-R$DB2PATH/lib″ です。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

関連タスク:

- 245 ページの『UNIX での CLI アプリケーションの作成』

関連資料:

- 270 ページの『Solaris CLI ルーチンのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldapp -- Builds Solaris C applications (C)』

Solaris ルーチンのビルド・スクリプト

Solaris 上で CLI ルーチンを構築するための bldrtn スクリプトを以下に示します。

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds Solaris CLI routines (stored procedures or UDFs)
# Usage: bldrtn <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Set compile and link flags for 32-bit and 64-bit programs.
bitwidth=`LANG=C db2level | awk '/bits/{print $5}'`
if [ $bitwidth = "¥"64¥"" ];
then
  CFLAG_ARCH=v9
  LIB=lib
else
  CFLAG_ARCH=v8plusa
  LIB=lib32
fi

# Set the runtime path.
# LD_LIBRARY_PATH will be followed instead of the runtime path unless
# you unset LD_LIBRARY_PATH first to allow the runtime path to be used.
EXTRA_LFLAG="-R$DB2PATH/$LIB"

# Compile the error-checking utility.
cc -xarch=$CFLAG_ARCH -mt -DUSE_UI_THREADS -Kpic ¥
  -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc -xarch=$CFLAG_ARCH -mt -DUSE_UI_THREADS -Kpic ¥
  -I$DB2PATH/include -c $1.c

# Link the program.
cc -xarch=$CFLAG_ARCH -mt -G -o $1 $1.o utilcli.o ¥
  -L$DB2PATH/$LIB $EXTRA_LFLAG -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Solaris CLI ルーチンのコンパイルおよびリンク・オプション

Solaris C コンパイラーを使用して CLI ルーチンを構築する場合、 DB2 ではこれらのコンパイルおよびリンク・オプションが推奨されています。 これらは、sqllib/samples/cli/bldrtn ビルド・スクリプトで例示されます。

bldrtn のコンパイルおよびリンク・オプション
コンパイル・オプション
cc C コンパイラー。
-xarch=$CFLAG_ARCH このオプションは、`libdb2.so` とのリンク時にコンパイラーが有効な実行可能プログラムを確実に生成するようにします。 $CFLAG_ARCH の値は、32 ビットの場合は「v8plusa」、 64 ビットの場合は「v9」に設定されます。
-mt マルチスレッド・サポートを使用できるようにします。これは、構築中のルーチンが他のルーチンと同じプロセス中で実行する (THREADSAFE) か、またはエンジン自体の中で実行する (NOT FENCED) 場合に必要になります。
-DUSE_UI_THREADS Sun 社の「UNIX International」スレッド API を使用できるようにします。
-Kpic 共有ライブラリー用の位置独立コードを生成します。
-I$DB2PATH/include DB2 組み込みファイルのロケーションを指定します。例: `$HOME/sqllib/include`
-c コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。
リンク・オプション:
cc コンパイラーをリンカーのフロントエンドとして使用します。
-xarch=$CFLAG_ARCH このオプションは、`libdb2.so` とのリンク時にコンパイラーが有効な実行可能プログラムを確実に生成するようにします。 $CFLAG_ARCH の値は、32 ビットの場合は「v8plusa」、64 ビットの場合は「v9」に設定されます。
-mt マルチスレッド・サポートを使用できるようにします。これは、構築中のルーチンが他のルーチンと同じプロセス中で実行する (THREADSAFE) か、またはエンジン自体の中で実行する (NOT FENCED) 場合に必要になります。
-G 共有ライブラリーを生成します。
-o $1 実行可能ファイルを指定します。
$1.o プログラム・オブジェクト・ファイルを組み込みます。
utilcli.o エラー・チェック用のユーティリティー・オブジェクト・ファイルを組み込みます。
-L$DB2PATH/$LIB リンク時の DB2 静的ライブラリーおよび共有ライブラリーのロケーションを示します。たとえば、32 ビットの場合は `$HOME/sqllib/lib32`、 64 ビットの場合は `$HOME/sqllib/lib` のように指定します。
$EXTRA_LFLAG ランタイムの DB2 共有ライブラリーのロケーションを示します。 32 ビットの場合その値は ″-R$DB2PATH/lib32″、 64 ビットの場合の値は ″-R$DB2PATH/lib″ です。
-ldb2 DB2 ライブラリーとリンクします。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

関連タスク:

- 249 ページの『UNIX での CLI ルーチンの作成』

関連資料:

- 268 ページの『Solaris CLI アプリケーションのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldrtn -- Builds Solaris C routines (stored procedures or UDFs) (C)』

Windows

以下のセクションでは、サポートされている Windows オペレーティング・システムにおいて CLI アプリケーションおよびルーチンを構築する方法について説明します。また、DB2 プログラムを構築するためのサンプル・バッチ・ファイルを示し、それらのバッチ・ファイルで使用されているコンパイル・オプションおよびリンク・オプションについて説明します。

Windows での CLI アプリケーションの作成

DB2 には、CLI プログラムをコンパイルしてリンクするための、バッチ・ファイルが備えられています。これは、このファイルで作成できるサンプル・プログラムと共に、 `sqllib¥samples¥cli` ディレクトリーにあります。

バッチ・ファイル `bldapp.bat` には、 DB2 CLI プログラムを作成するためのコマンドが入っています。これは、パラメーターを 4 つまでとりますが、バッチ・ファイルの中では、変数 `%1`、`%2`、`%3`、および `%4` によって表されます。

このパラメーター `%1` には、ソース・ファイルの名前を指定します。必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない CLI プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは `%2` で、接続するデータベースの名前を指定します。 3 番目のパラメーターは `%3` で、データベースのユーザー ID を指定します。そしてもう 1 つが `%4` で、データベースのパスワードを指定します。

プログラムに組み込み SQL (`.sqc` または `.sqx` 拡張子が付いている) が含まれている場合、 `embprep.bat` バッチ・ファイルは、`.c` または `.cxx` 拡張子を持つプログラム・ファイルを生成して、プログラムをプリコンパイルするために呼び出されます。

手順:

以下の例では、CLI アプリケーションを作成して実行する方法が示されています。

ソース・ファイル `tbinfo.c` からサンプル・プログラム `tbinfo` を作成するには、次のように入力します。

```
bldapp tbinfo
```

結果として、実行可能ファイル tbinfo が作成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
tbinfo
```

組み込み SQL アプリケーションの構築と実行

ソース・ファイル dbusemx.sqc から組み込み SQL アプリケーション dbusemx を作成する場合、次の 3 つの方法があります。

1. 同じインスタンス上のサンプル・データベースに接続している場合には、次のように入力します。

   ```
   bldapp dbusemx
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp dbusemx database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp dbusemx database userid password
   ```

結果として、実行可能ファイル dbusemx が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   dbusemx
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   dbusemx database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   dbusemx database userid password
   ```

関連タスク:

- 233 ページの『CLI 環境のセットアップ』
- 241 ページの『Windows CLI 環境のセットアップ』
- 276 ページの『Windows での CLI ルーチンの作成』

関連資料:

- 278 ページの『Windows CLI アプリケーションのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldapp.bat -- Builds C applications on Windows』
- 『embprep.bat -- Prep and binds a C/C++ or Micro Focus COBOL embedded SQL program on Windows』
- 『dbusemx.sqc -- How to execute embedded SQL statements in CLI』
- 『tbinfo.c -- How to get information about tables from the system catalog tables』

Windows での CLI 複数接続アプリケーションの作成

DB2 には、CLI プログラムをコンパイルしてリンクするための、バッチ・ファイルが備えられています。これは、このファイルで作成できるサンプル・プログラムと共に、 sqllib¥samples¥cli ディレクトリーにあります。

バッチ・ファイル bldmc.bat には、 2 つのデータベースを必要とする DB2 複数接続プログラムを作成するためのコマンドが入っています。コンパイルとリンクのオプションは、bldapp.bat で使用されるものと同じです。

最初のパラメーター %1 には、ソース・ファイルの名前を指定します。 2 番目のパラメーター %2 には、接続先の最初のデータベースの名前を指定します。 3 番目のパラメーター %3 には、接続先の 2 番目のデータベースの名前を指定します。それらはすべて必要パラメーターです。

注: makefile には、データベース名のデフォルト値として ″sample″ と ″sample2″ (それぞれ %2 および %3) がハードコーディングされているため、 makefile を使用する場合、それらのデフォルトを使用するのであれば、指定する必要があるのはプログラム名だけです (%1 パラメーター)。 bldmc.bat ファイルを使用する場合は、 3 つのパラメーターをすべて指定する必要があります。

オプショナル・パラメーターは、ローカル接続の場合は不要ですが、リモート・クライアントからサーバーに接続する場合は必要になります。オプショナル・パラメーターのうち、%4 と %5 には、最初のデータベースのためのユーザー ID とパスワードをそれぞれ指定します。また、%6 と %7 には、 2 番目のデータベースのためのユーザー ID とパスワードをそれぞれ指定します。

手順:

複数接続のサンプル・プログラム dbmconx には、2 つのデータベースが必要です。sample データベースがまだ作成されていないなら、コマンド行で db2sampl を入力することによってそれを作成できます。 2 番目のデータベース sample2 は、以下のいずれかのコマンドによって作成できます。

データベースをローカルに作成する場合、

```
db2 create db sample2
```

データベースをリモートに作成する場合、

```
db2 attach to <node_name>
db2 create db sample2
db2 detach
db2 catalog db sample2 as sample2 at node <node_name>
```

<node_name> は、データベースの存在するノードです。

複数接続では、TCP/IP Listener が実行されていることも必要になります。そのためには、次のことを実行してください。

1. 環境変数 DB2COMM を TCP/IP に設定します。それには、次のようにします。

   ```
   db2set DB2COMM=TCPIP
   ```

2. サービス・ファイルの中で指定されている TCP/IP サービス名を使用して、データベース・マネージャー構成ファイルを更新します。

```
db2 update dbm cfg using SVCENAME <TCP/IP service name>
```

サービス・ファイルには、各インスタンスごとに 1 つの TCP/IP サービス名が含まれています。それがわからない場合、またはサービス・ファイルを読むためのファイル・アクセス許可がない場合は、システム管理者にお問い合わせください。

3. 以上の変更内容を有効にするため、データベース・マネージャーを停止してから再開します。

```
db2stop
db2start
```

dbmconx プログラムは、以下の 5 個のファイルで構成されています。

dbmconx.c
: 2 つのデータベースに接続するためのメイン・ソース・ファイル。

dbmconx1.sqc
: 最初のデータベースにバインドされたパッケージを作成するためのソース・ファイル。

dbmconx1.h
: dbmconx.sqc に組み込まれている dbmconx1.sqc のためのヘッダー・ファイル。これは、最初のデータベースにバインドされた表を作成したりドロップしたりするための SQL ステートメントにアクセスするために必要です。

dbmconx2.sqc
: 2 番目のデータベースにバインドされたパッケージを作成するためのソース・ファイル。

dbmconx2.h
: dbmconx.sqc に組み込まれている dbmconx2.sqc のためのヘッダー・ファイル。これは、2 番目のデータベースにバインドされた表を作成したりドロップしたりするための SQL ステートメントにアクセスするために必要です。

複数接続のサンプル・プログラム dbmconx を作成するには、次のように入力します。

```
bldmc dbmconx sample sample2
```

結果として、実行可能ファイル dbmconx が作成されます。

その実行可能ファイルを実行するには、その実行可能ファイルの名前を入力します。

```
dbmconx
```

プログラムにより、2 つのデータベースに対する 2 フェーズ・コミットのデモが実行されます。

関連概念:

- 145 ページの『CLI アプリケーションでのマルチサイト更新 (2 フェーズ・コミット)』

関連資料:

- 278 ページの『Windows CLI アプリケーションのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldmc.bat -- Builds Windows CLI multi-connection applications』
- 『dbmconx.c -- How to use multiple databases with embedded SQL.』
- 『dbmconx1.h -- Functions used in dbmconx.c』
- 『dbmconx1.sqc -- This file contains functions used in dbmconx.c』
- 『dbmconx2.h -- Functions used in dbmconx.c』
- 『dbmconx2.sqc -- This file contains functions used in dbmconx.c』

Windows での CLI ルーチンの作成

DB2 には、CLI プログラムをコンパイルしてリンクするための、バッチ・ファイルが備えられています。これは、このファイルで作成できるサンプル・プログラムと共に、 `sqllib¥samples¥cli` ディレクトリーにあります。

バッチ・ファイル `bldrtn.bat` には、 CLI ルーチン (ストアード・プロシージャーおよびユーザー定義関数) を作成するためのコマンドがあります。 `bldrtn.bat` は、サーバー上に DLL を作成します。これは、バッチ・ファイルの中で変数 `%1` で表される 1 つのパラメーターを取ります。これはソース・ファイルの名前を指定するものです。バッチ・ファイルでは、ソース・ファイル名を DLL 名に使用します。

手順:

ソース・ファイル `spserver.c` から `spserver` DLL を構築するには、次のようにします。

1. 次のバッチ・ファイル名およびプログラム名を入力します。

   ```
   bldrtn spserver
   ```

   このバッチ・ファイルは、CLI サンプル・プログラムと同じディレクトリーに入っている、モジュール定義ファイル `spserver.def` を使用して DLL を作成します。その後、このバッチ・ファイルは、 DLL の `spserver.dll` をサーバー上の `sqllib¥function` というパスにコピーします。
2. 次に、サーバー上で次のように `spcat` スクリプトを実行し、ルーチンをカタログします。

   ```
   spcat
   ```

   このスクリプトは、サンプル・データベースに接続し、 `spdrop.db2` を呼び出すことによって以前にカタログされている場合には、そのルーチンをアンカタログし、`spcreate.db2` を呼び出してカタログ化し、最後にデータベースから切断します。 `spdrop.db2` および `spcreate.db2` スクリプトを個別に呼び出すことも可能です。
3. その後、これが共用ライブラリーの初回ビルドでないなら、データベースを一度停止してから再始動し、新しい共有ライブラリーが認識されるようにします。必要であれば、共有ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

DLL spserver を作成したなら、その中のルーチンを呼び出す CLI クライアント・アプリケーション spclient を構築できます。

spclient は、スクリプト・ファイル bldapp を使用して構築することができます。

ルーチンを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

説明

データベース
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは DLL spserver にアクセスし、ルーチンをサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

関連タスク:

- 233 ページの『CLI 環境のセットアップ』
- 241 ページの『Windows CLI 環境のセットアップ』
- 272 ページの『Windows での CLI アプリケーションの作成』

関連資料:

- 280 ページの『Windows CLI ルーチンのコンパイルおよびリンク・オプション』

関連サンプル:

- 『bldapp.bat -- Builds C applications on Windows』
- 『bldrtn.bat -- Builds C routines (stored procedures and UDFs) on Windows』
- 『spcat -- To catalog stored procedures on UNIX (C)』
- 『spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc (C)』
- 『spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc (C)』
- 『spclient.c -- Call various stored procedures』
- 『spserver.c -- Definition of various types of stored procedures』

Windows アプリケーションのためのバッチ・ファイル

Windows 上で CLI アプリケーションを構築するための bldapp.bat バッチ・ファイルを以下に示します。

```
@echo off
rem BATCH FILE: bldapp.bat
rem Builds Windows CLI applications
rem Usage: bldapp prog_name [ db_name [ userid password ]]
```

```
rem Default compiler is set to Microsoft Visual C++
rem To use a different compiler, comment out 'set BLDCOMP=cl'
rem and uncomment 'set BLDCOMP=icl' or 'set BLDCOMP=ecl'
rem Microsoft C/C++ Compiler
set BLDCOMP=cl

rem Intel C++ Compiler for 32-bit applications
rem set BLDCOMP=icl

rem Intel C++ Compiler for Itanium 64-bit applications
rem set BLDCOMP=ecl

if exist "%1.sqc" call embprep %1 %2 %3 %4
if exist "%1.sqx" call embprep %1 %2 %3 %4

rem Compile the error-checking utility.
%BLDCOMP% -Zi -Od -c -W1 -DWIN32 utilcli.c

rem Compile the program.
if exist "%1.sqx" goto cpp
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 %1.c
goto link_step
:cpp
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 %1.cxx

rem Link the program.
:link_step
link -debug -out:%1.exe %1.obj utilcli.obj db2api.lib
@echo on
```

Windows CLI アプリケーションのコンパイルおよびリンク・オプション

Microsoft Visual C++ コンパイラーを使用して CLI アプリケーションを構築する場合は、 DB2 ではこの種のコンパイルおよびリンク・オプションが推奨されています。これらのオプションは、 `sqllib¥samples¥cli¥bldapp.bat` バッチ・ファイル中に例示されています。

bldapp のコンパイルおよびリンク・オプション	
コンパイル・オプション	
%BLDCOMP%	コンパイラーの変数。デフォルトは `cl` で、これは Microsoft Visual C++ コンパイラーを示します。 `icl` に設定することもでき、この場合は Intel C++ Compiler for 32-bit applications を示します。また `ecl` に設定することもでき、この場合は Intel C++ Compiler for Itanium 64-bit applications を示します。
-Zi	デバッグ情報を使用可能にします。
-Od	最適化なし。最適化をオフにしてデバッガーを使用する方が簡単です。
-c	コンパイルのみを実行し、リンクは実行しません。
-W2	警告レベルを設定します。
-DWIN32	Windows オペレーティング・システムに必要なコンパイラー・オプション。

bldapp のコンパイルおよびリンク・オプション
リンク・オプション:
link 32 ビットのリンカーを使用します。
-debug デバッグ情報を組み込みます。
-out:%1.exe 実行可能ファイルを指定します。
%1.obj オブジェクト・ファイルを組み込みます。
utilcli.obj エラー・チェック用のユーティリティー・オブジェクト・ファイルを組み込みます。
db2api.lib DB2 API ライブラリーとリンクします。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

関連タスク:

- 272 ページの『Windows での CLI アプリケーションの作成』
- 276 ページの『Windows での CLI ルーチンの作成』

関連サンプル:

- 『bldapp.bat -- Builds C applications on Windows』

Windows ルーチンのためのバッチ・ファイル

Windows 上で CLI ルーチンを構築するための `bldrtn.bat` バッチ・ファイルを以下に示します。

```
@echo off
rem BATCH FILE: bldrtn.bat
rem Builds Windows CLI routines (stored procedures and UDFs)
rem using the Microsoft Visual C++ compiler
rem Usage: bldrtn prog_name

rem Default compiler is set to Microsoft Visual C++
rem To use a different compiler, comment out 'set BLDCOMP=cl'
rem and uncomment 'set BLDCOMP=icl' or 'set BLDCOMP=ecl'
rem Microsoft C/C++ Compiler
set BLDCOMP=cl

rem Intel C++ Compiler for 32-bit applications
rem set BLDCOMP=icl

rem Intel C++ Compiler for Itanium 64-bit applications
rem set BLDCOMP=ecl

if "%1" == "" goto error

rem Compile the program.
if exist "%1.cxx" goto cpp
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 -MD %1.c  utilcli.c
goto link_step
:cpp
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 -MD %1.cxx  utilcli.c

rem Link the program.
:link_step
```

```
link -debug -dll -out:%1.dll %1.obj utilcli.obj db2api.lib -def:%1.def

rem Copy the stored procedure DLL to the 'function' directory
copy %1.dll "%DB2PATH%¥function"

goto exit
:error
echo Usage: bldrtn prog_name
:exit
@echo on
```

Windows CLI ルーチンのコンパイルおよびリンク・オプション

Microsoft Visual C++ コンパイラーを使用して CLI ルーチンを構築する場合は、DB2 ではこの種のコンパイルおよびリンク・オプションが推奨されています。これらのオプションは、 `sqllib¥samples¥cli¥bldrtn.bat` バッチ・ファイル中に例示されています。

bldrtn のコンパイルおよびリンク・オプション	
コンパイル・オプション	
%BLDCOMP%	コンパイラーの変数。デフォルトは `cl` で、これは Microsoft Visual C++ コンパイラーを示します。 `icl` に設定することもでき、この場合は Intel C++ Compiler for 32-bit applications を示します。また `ecl` に設定することもでき、この場合は Intel C++ Compiler for Itanium 64-bit applications を示します。
-Zi	デバッグ情報を使用可能にします。
-Od	最適化なし。最適化をオフにしてデバッガーを使用する方が簡単です。
-c	コンパイルのみを実行し、リンクは実行しません。このバッチ・ファイルでは、コンパイルとリンクは別個のステップです。
-W2	警告レベルを設定します。
-DWIN32	Windows オペレーティング・システムに必要なコンパイラー・オプション。
-MD	MSVCRT.LIB を使用してマルチスレッド DLL を作成します。
リンク・オプション:	
link	32 ビットのリンカーを使用します。
-debug	デバッグ情報を組み込みます。
-out:%1.dll	.DLL ファイルを作成します。
%1.obj	オブジェクト・ファイルを組み込みます。
utilcli.obj	エラー・チェック用のユーティリティー・オブジェクト・ファイルを組み込みます。
db2api.lib	DB2 API ライブラリーとリンクします。
-def:%1.def	モジュール定義ファイルを使用します。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。	

関連タスク:

- 272 ページの『Windows での CLI アプリケーションの作成』
- 276 ページの『Windows での CLI ルーチンの作成』

関連サンプル:

- 『bldrtn.bat -- Builds C routines (stored procedures and UDFs) on Windows』

第 24 章 CLI サンプル・プログラム

CLI サンプル・プログラム

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。

- Windows® オペレーティング・システム: `%DB2PATH%¥sqllib¥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®: `$HOME/sqllib/samples/cli` ($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。

関連タスク:

- 233 ページの『CLI 環境のセットアップ』
- 245 ページの『UNIX での CLI アプリケーションの作成』
- 249 ページの『UNIX での CLI ルーチンの作成』
- 272 ページの『Windows での CLI アプリケーションの作成』
- 276 ページの『Windows での CLI ルーチンの作成』

関連資料:

- 283 ページの『CLI のサンプル』

CLI のサンプル

表 22. サンプル CLI プログラム・ファイル

サンプル・プログラムの名前	プログラムの説明
チュートリアル・サンプル - データベースの基本操作の例を示すプログラム。	
`tut_mod.c`	表データの修正方法。
`tut_read.c`	表の読み取り方法。
`tut_use.c`	データベースの使用法。
インストール・イメージ・レベル - DB2 と 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) の入手および設定方法。
クライアント・レベル - DB2 のクライアント・レベルを扱うサンプル。	
`cli_info.c`	クライアント・レベル情報の入手および設定方法。
`clihandl.c`	ハンドルの割り当ておよび解放方法。
`clisqlca.c`	SQLCA データの処理方法。
インスタンス・レベル - DB2 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・レベル情報の入手および設定方法。
データベース・レベル - DB2 内のデータベース・オブジェクトを扱うサンプル。	

表 22. サンプル CLI プログラム・ファイル (続き)

サンプル・プログラムの名前	プログラムの説明
dbcongui.c	グラフィカル・ユーザー・インターフェース (GUI) を使用したデータベースへの接続方法。
dbconn.c	データベースからの接続および切断方法。
dbinfo.c	データベース・レベルの情報の入手および設定方法。
dbmcon.c	複数のデータベースからの接続および切断方法。
dbmconx.c	組み込み SQL による複数のデータベースからの接続および切断方法。
dbmconx1.h	dbmconx1.sqc のヘッダー・ファイル。
dbmconx1.sqc	dbmconx プログラムの組み込み SQL ファイル。
dbmconx2.h	dbmconx2.sqc のヘッダー・ファイル。
dbmconx2.sqc	dbmconx プログラムの組み込み SQL ファイル。
dbnative.c	ODBC エスケープ文節を含むステートメントを、データ・ソース特有の形式に変換する方法。
dbuse.c	データベース・オブジェクトの使用法。
dbusemx.sqc	組み込み SQL によるデータベース・オブジェクトの使用法。
表レベル - DB2 内の表オブジェクトを扱うサンプル。	
tbconstr.c	表の制約の処理方法。
tbcreate.c	表の作成、変更、およびドロップ方法。
tbinfo.c	表レベルの情報の入手および設定方法。
tbload.c	CLI LOAD ユーティリティーを使用してデータを挿入する方法です。
tbmod.c	表内の情報の修正方法。
tbread.c	表内の情報の読み取り方法。
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の入手方法。
dtlob.c	LOB データの読み取りおよび書き込み方法。
dtudt.c	ユーザー定義特殊タイプの作成、使用、およびドロップ方法。
ストアード・プロシージャー・レベル - ストアード・プロシージャーを示すサンプル。	
spcat	spserver プログラムのストアード・プロシージャー・カタログ・スクリプト。このスクリプトは、spdrop.db2 と spcreate.db2 を呼び出します。
spcreate.db2	CREATE PROCEDURE ステートメントを発行するための CLP スクリプト。
spdrop.db2	カタログからストアード・プロシージャーをドロップするための CLP スクリプト。
spclient.c	spserver.c 内で宣言されるサーバー関数を呼び出すために使用されるクライアント・プログラム。
spserver.c	サーバー上で構築および実行されるストアード・プロシージャー関数。
spclires.c	複数の結果セットの SQLMoreResults と SQLNextResults の相違を示すクライアント・アプリケーション。
spcall.c	任意のストアード・プロシージャーを呼び出すためのクライアント・プログラム。
UDF レベル - ユーザー定義関数を示すサンプル。	
udfcli.c	udfsrv.c 内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 によって呼び出されるユーザー定義関数 ScalarUDF。
共通ユーティリティー・ファイル	

表 22. サンプル CLI プログラム・ファイル (続き)

サンプル・プログラムの名前	プログラムの説明
`utilcli.c`	CLI サンプルで使用されるユーティリティー関数。
`utilcli.h`	CLI サンプルで使用されるユーティリティー関数用のヘッダー・ファイル。

関連概念:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『サンプル・ファイル』

第 4 部 CLI/ODBC の構成キーワード

第 25 章 CLI/ODBC の構成キーワード

CLI/ODBC の構成キーワードを使用すると、DB2 CLI ドライバーの動作をカスタマイズできます。この章では、db2cli.ini 初期設定ファイルによってこれらのキーワードを設定する方法について説明し、使用可能な構成キーワードを示します。

db2cli.ini 初期設定ファイル

db2cli.ini 初期化ファイルには、 DB2 CLI とこの製品を使うアプリケーションの動作を構成する場合に使用できる、さまざまなキーワードと値が入っています。キーワードは、データベース*別名* と関連しており、そのデータベースにアクセスするすべての DB2 CLI および ODBC アプリケーションに影響を与えます。

デフォルトでは、DB2® CLI/ODBC 構成キーワード・ファイルは、 Window 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。 Windows® プラットフォーム上で ODBC Driver Manager を使用してユーザー・データ・ソースを構成するには、ユーザーのホーム (プロファイル) ディレクトリー中に db2cli.ini を作成できます。

環境変数 *DB2CLIINIPATH* を使って、デフォルト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® GRAPHIC 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。
- アプリケーションによって指定されるデフォルト接続オプションをオーバーライドする。たとえば、アプリケーションが SQL_ATTR_ANSI_APP 接続属性を設定することによって CLI ドライバーに対して Unicode サポートを要求している場合でも、 `db2cli.ini` ファイルの中で DisableUnicode=1 が設定されていると、CLI ドライバーはそのアプリケーションに Unicode サポートを提供しません。

  注: `db2cli.ini` ファイルの中で設定されている CLI/ODBC 構成キーワードが、`SQLDriverConnect()` 接続ストリングに含まれるキーワードと矛盾する場合、`SQLDriverConnect()` キーワードが優先されます。

`db2cli.ini` 初期設定ファイルは、 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。

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。それには以下のキーワードが含まれます。

- DisableMultiThread
- JDBCTrace
- JDBCTraceFlush
- JDBCTracePathName
- Trace
- TraceComm
- TraceFileName
- TraceFlush
- TraceLocks
- TracePathName
- TracePIDList
- TracePIDTID
- TraceRefreshInterval
- TraceStmtOnly
- TraceTime
- TraceTimeStamp

他のすべてのキーワードはデータベース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。

注: 構成キーワードは COMMON セクション中で有効になりますが、すべてのデータベース接続に適用されます。

`db2cli.ini` ファイルの COMMON セクションは、次の語で始まります。

```
[COMMON]
```

共通キーワードを設定する前に、クライアントからのすべての 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 つだけをトラブルシューティング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・ソース名 (DSN) の名前で始まります。

[*data source name***]**

これを**セクション・ヘッダー** と呼びます。

パラメーターを設定するには、キーワードとその関連キーワード値を次の形式で指定します。

KeywordName =*keywordValue*

- 各データベースのすべてのキーワードとその関連値は、そのデータベースのセクション・ヘッダーの下になければなりません。
- データベース固有のセクションに DBAlias キーワードが含まれていない場合は、接続が確立される際にはデータ・ソース名がデータベース別名として使用されます。各セクションのキーワード設定値は、該当するデータベース別名だけに適用されます。
- キーワードは大文字小文字の区別はありません。しかし、その値が文字ベースのものであれば値にその区別がある場合もあります。
- .INI ファイルにデータベースがない場合、これらのキーワードのデフォルトが有効になっています。
- 新しい行の先頭位置にセミコロンを入れると、注釈行になります。
- ブランク行は許可されています。
- 1 つのキーワードに重複項目があると、最初の項目が使用されます (警告は与えられません)。

2 つのデータベース別名セクションがある .INI サンプル・ファイルを次に示します。

```
; This is a comment line.
[MYDB22]
AutoCommit=0
TableType="'TABLE','SYSTEM TABLE'"

; This is another comment line.
[MYDB2MVS]
CurrentSQLID=SAAID
TableType="'TABLE'"
SchemaList="'USER1',CURRENT SQLID,'USER2'"
```

db2cli.ini ファイルは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構成アシスタントを使用するか、または UPDATE CLI CONFIGURATION コマンド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。

関連資料:

- 「コマンド・リファレンス」の『UPDATE CLI CONFIGURATION コマンド』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 319 ページの『DBAlias CLI/ODBC 構成キーワード』
- 361 ページの『Trace CLI/ODBC 構成キーワード』

CLI/ODBC 構成キーワード (カテゴリー別)

CLI/ODBC 構成キーワードは、以下のカテゴリーに分けられます。

- 292 ページの『互換性の構成キーワード』
- 292 ページの『データ・ソースの構成キーワード』
- 292 ページの『データ・タイプの構成キーワード』

- 『エンタープライズの構成キーワード』
- 293 ページの『環境の構成キーワード』
- 293 ページの『ファイルの DSN 構成キーワード』
- 293 ページの『最適化の構成キーワード』
- 294 ページの『サービスの構成キーワード』
- 294 ページの『静的 SQL の構成キーワード』
- 295 ページの『トランザクションの構成キーワード』

互換性の構成キーワード:

オプションの**互換性**設定は、DB2 の動作を定義するのに使用されます。これらのオプションを設定して、他のアプリケーションと DB2 との互換性を確保できます。

- 314 ページの『CursorTypes CLI/ODBC 構成キーワード』
- 321 ページの『DeferredPrepare CLI/ODBC 構成キーワード』
- 323 ページの『DescribeParam CLI/ODBC 構成キーワード』
- 323 ページの『DisableKeysetCursor CLI/ODBC 構成キーワード』
- 324 ページの『DisableMultiThread CLI/ODBC 構成キーワード』
- 324 ページの『DisableUnicode CLI/ODBC 構成キーワード』

データ・ソースの構成キーワード:

一般キーワード。

- 319 ページの『DBAlias CLI/ODBC 構成キーワード』
- 348 ページの『PWD CLI/ODBC 構成キーワード』
- 374 ページの『UID CLI/ODBC 構成キーワード』

データ・タイプの構成キーワード:

オプションの**データ・タイプ**設定は、さまざまなデータ・タイプを DB2 でどのように報告および処理するかを定義するために使用されます。

- 298 ページの『BitData CLI/ODBC 構成キーワード』
- 316 ページの『DateTimeStringFormat CLI/ODBC 構成キーワード』
- 325 ページの『FloatPrecRadix CLI/ODBC 構成キーワード』
- 328 ページの『Graphic CLI/ODBC 構成キーワード』
- 334 ページの『LOBMaxColumnSize CLI/ODBC 構成キーワード』
- 335 ページの『LongDataCompat CLI/ODBC 構成キーワード』
- 336 ページの『MapDateCDefault CLI/ODBC 構成キーワード』
- 337 ページの『MapDateDescribe CLI/ODBC 構成キーワード』
- 338 ページの『MapGraphicDescribe CLI/ODBC 構成キーワード』
- 339 ページの『MapTimeCDefault CLI/ODBC 構成キーワード』
- 340 ページの『MapTimeDescribe CLI/ODBC 構成キーワード』
- 341 ページの『MapTimestampCDefault CLI/ODBC 構成キーワード』
- 341 ページの『MapTimestampDescribe CLI/ODBC 構成キーワード』
- 343 ページの『OleDbReturnCharAsWChar CLI/ODBC 構成キーワード』

エンタープライズの構成キーワード:

オプションの**エンタープライズ**設定は、大規模なデータベースへの接続の効率を最大限まで上げるために使用されます。

- 306 ページの『CLISchema CLI/ODBC 構成キーワード』
- 307 ページの『ConnectNode CLI/ODBC 構成キーワード』
- 310 ページの『CurrentPackagePath CLI/ODBC 構成キーワード』
- 310 ページの『CurrentPackageSet CLI/ODBC 構成キーワード』
- 311 ページの『CurrentRefreshAge CLI/ODBC 構成キーワード』
- 312 ページの『CurrentSchema CLI/ODBC 構成キーワード』
- 312 ページの『CurrentSQLID CLI/ODBC 構成キーワード』
- 320 ページの『DBName CLI/ODBC 構成キーワード』
- 326 ページの『GranteeList CLI/ODBC 構成キーワード』
- 327 ページの『GrantorList CLI/ODBC 構成キーワード』
- 351 ページの『ReportPublicPrivileges CLI/ODBC 構成キーワード』
- 352 ページの『SchemaList CLI/ODBC 構成キーワード』
- 359 ページの『TableType CLI/ODBC 構成キーワード』

環境の構成キーワード:

オプションの**環境**設定は、サーバーおよびクライアント・マシン上のさまざまなファイルのロケーションのような、環境固有の設定を定義するために使用されます。

- 308 ページの『CurrentFunctionPath CLI/ODBC 構成キーワード』
- 321 ページの『DefaultProcLibrary CLI/ODBC 構成キーワード』
- 349 ページの『QueryTimeoutInterval CLI/ODBC 構成キーワード』
- 360 ページの『TempDir CLI/ODBC 構成キーワード』

ファイルの DSN 構成キーワード:

オプションの**ファイル DSN** 設定は、ファイルの DSN 接続の TCP/IP 設定を設定するために使用されます。

- 315 ページの『Database CLI/ODBC 構成キーワード』
- 329 ページの『Hostname CLI/ODBC 構成キーワード』
- 346 ページの『Port CLI/ODBC 構成キーワード』
- 348 ページの『Protocol CLI/ODBC 構成キーワード』
- 353 ページの『ServiceName CLI/ODBC 構成キーワード』

最適化の構成キーワード:

オプションの**最適化**設定は、 CLI/ODBC ドライバーとサーバーとの間のネットワークの流れを高速化したり、その量を削減したりするために使用されます。

- 299 ページの『BlockForNRows CLI/ODBC 構成キーワード』
- 300 ページの『BlockLobs CLI/ODBC 構成キーワード』
- 303 ページの『ClientBuffersUnboundLOBS CLI/ODBC 構成キーワード』
- 309 ページの『CurrentMaintainedTableTypesForOpt CLI/ODBC 構成キーワード』
- 317 ページの『DB2Degree CLI/ODBC 構成キーワード』
- 317 ページの『DB2Explain CLI/ODBC 構成キーワード』
- 319 ページの『DB2Optimization CLI/ODBC 構成キーワード』

- 322 ページの『DescribeInputOnPrepare CLI/ODBC 構成キーワード』
- 331 ページの『KeepDynamic CLI/ODBC 構成キーワード』
- 332 ページの『KeepStatement CLI/ODBC 構成キーワード』
- 332 ページの『LOBCacheSize CLI/ODBC 構成キーワード』
- 333 ページの『LOBFileThreshold CLI/ODBC 構成キーワード』
- 334 ページの『LockTimeout CLI/ODBC 構成キーワード』
- 344 ページの『OptimizeForNRows CLI/ODBC 構成キーワード』
- 354 ページの『SkipTrace CLI/ODBC 構成キーワード』
- 358 ページの『StreamPutData CLI/ODBC 構成キーワード』
- 375 ページの『Underscore CLI/ODBC 構成キーワード』

サービスの構成キーワード:

オプションの**サービス**設定は、 CLI/ODBC 接続での問題のトラブルシューティングに役立てるために使用されます。プログラマーは、一部のオプションを、 CLI プログラムがどのようにサーバーへの呼び出しに変換されるのかをより深く理解するために使用することもできます。

- 295 ページの『AppendAPIName CLI/ODBC 構成キーワード』
- 329 ページの『IgnoreWarnings CLI/ODBC 構成キーワード』
- 330 ページの『IgnoreWarnList CLI/ODBC 構成キーワード』
- 332 ページの『LoadXAInterceptor CLI/ODBC 構成キーワード』
- 345 ページの『Patch1 CLI/ODBC 構成キーワード』
- 345 ページの『Patch2 CLI/ODBC 構成キーワード』
- 350 ページの『ReportRetryErrorsAsWarnings CLI/ODBC 構成キーワード』
- 351 ページの『RetryOnError CLI/ODBC 構成キーワード』
- 347 ページの『ProgramName CLI/ODBC 構成キーワード』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 362 ページの『TraceComm CLI/ODBC 構成キーワード』
- 363 ページの『TraceErrImmediate CLI/ODBC 構成キーワード』
- 364 ページの『TraceFileName CLI/ODBC 構成キーワード』
- 365 ページの『TraceFlush CLI/ODBC 構成キーワード』
- 366 ページの『TraceFlushOnError CLI/ODBC 構成キーワード』
- 367 ページの『TraceLocks CLI/ODBC 構成キーワード』
- 367 ページの『TracePathName CLI/ODBC 構成キーワード』
- 369 ページの『TracePIDList CLI/ODBC 構成キーワード』
- 369 ページの『TracePIDTID CLI/ODBC 構成キーワード』
- 370 ページの『TraceRefreshInterval CLI/ODBC 構成キーワード』
- 371 ページの『TraceStmtOnly CLI/ODBC 構成キーワード』
- 372 ページの『TraceTime CLI/ODBC 構成キーワード』
- 373 ページの『TraceTimestamp CLI/ODBC 構成キーワード』
- 377 ページの『WarningList CLI/ODBC 構成キーワード』

静的 SQL の構成キーワード:

オプションの**静的 SQL** 設定は、 CLI/ODBC アプリケーションで静的 SQL ステートメントを実行するときに使用されます。

- 355 ページの『StaticCapFile CLI/ODBC 構成キーワード』
- 356 ページの『StaticLogFile CLI/ODBC 構成キーワード』
- 357 ページの『StaticMode CLI/ODBC 構成キーワード』
- 357 ページの『StaticPackage CLI/ODBC 構成キーワード』

トランザクションの構成キーワード:

オプションの**トランザクション**設定は、アプリケーションで使用される SQL ステートメントをコントロールおよび高速化するために使用されます。

- 296 ページの『ArrayInputChain CLI/ODBC 構成キーワード』
- 297 ページの『AsyncEnable CLI/ODBC 構成キーワード』
- 297 ページの『AutoCommit CLI/ODBC 構成キーワード』
- 301 ページの『ClientAcctStr CLI/ODBC 構成キーワード』
- 302 ページの『ClientApplName CLI/ODBC 構成キーワード』
- 304 ページの『ClientUserID CLI/ODBC 構成キーワード』
- 305 ページの『ClientWrkStnName CLI/ODBC 構成キーワード』
- 308 ページの『ConnectType CLI/ODBC 構成キーワード』
- 313 ページの『CursorHold CLI/ODBC 構成キーワード』
- 343 ページの『Mode CLI/ODBC 構成キーワード』
- 354 ページの『SQLOverrideFileName CLI/ODBC 構成キーワード』
- 359 ページの『SyncPoint CLI/ODBC 構成キーワード』
- 373 ページの『TxnIsolation CLI/ODBC 構成キーワード』
- 376 ページの『UseOldStpCall CLI/ODBC 構成キーワード』

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 3 ページの『CLI の紹介』

AppendAPIName CLI/ODBC 構成キーワード

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・テキストに付加します。

db2cli.ini キーワード構文:

AppendAPIName = **0** | **1**

デフォルト設定:

DB2 CLI 関数名を表示しません。

使用上の注意:

エラーを生成した DB2 CLI 関数 (API) 名は、 `SQLGetDiagRec()` または `SQLError()` を使用して検索されたエラー・メッセージに付加されます。関数名は中括弧 { } で囲まれます。

たとえば、以下のようになります。

```
[IBM][CLI Driver]" CLIxxxx: < text >
SQLSTATE=XXXXX {SQLGetData}"
```

0 = DB2 CLI 関数名を付加しません (デフォルト)
1 = DB2 CLI 関数名を付加します

このキーワードはデバッグにのみ役立ちます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLError 関数 (CLI) - エラー情報の取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

ArrayInputChain CLI/ODBC 構成キーワード

キーワードの説明:

配列入力を使用可能にします。通常の配列入力の場合とは異なり、事前にサイズやメモリー割り振り要件を指定する必要はありません。

db2cli.ini キーワード構文:

ArrayInputChain = **-1** | **0** | **<positive integer>**

デフォルト設定:

通常の入力配列が使用可能になります。この場合、対応する `SQLExecute()` 呼び出しが実行される前に、配列とそのサイズを指定する必要があります。

使用上の注意:

デフォルトでは、配列入力 (値の配列が入力パラメーターにバインドされている) の場合、対応する `SQLExecute()` 関数が呼び出される前に、配列とそのサイズが指定されていなければなりません。しかし、アプリケーションにおいて配列のサイズが事前にはわからない場合や、配列のサイズが大きいため、アプリケーションが使用可能なメモリーのプールから割り振ることができない場合があります。そのような場合、アプリケーションでは ArrayInputChain=-1 を設定し、SQL_ATTR_CHAINING_BEGIN および SQL_ATTR_CHAINING_END のステートメント属性を使用することによって、チェーニングを有効にすることができます。その場合、通常の配列入力の場合とは異なり、事前にサイズやメモリー要件を指定することなく配列入力を利用できます。

チェーニングを使用可能にするには、

1. keyword ArrayInputChain = -1 を設定します。
2. 入力パラメーターを準備し、SQL ステートメントにバインドします。
3. `SQLSetStmtAttr()` により SQL_ATTR_CHAINING_BEGIN ステートメント属性を設定します。

4. バインドされたパラメーターを入力データにより更新し、 `SQLExecute()` を呼び出します。
5. 入力配列に含まれる行の数だけステップ 4 を繰り返します。
6. ステップ 4 で配列の最後の行が処理された後、 `SQLSetStmtAttr()` を使用して SQL_ATTR_CHAINING_END ステートメント属性を設定します。

これらのステップを完了した結果は、通常の配列入力を使用した場合と同じになります。

ArrayInputChain=0 (デフォルト値) を設定すると、この配列入力機能がオフになります。 ArrayInputChain には、入力配列に対して使用する配列サイズを設定するための任意の正整数を設定することもできます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 70 ページの『CLI 配列入力チェーニングによるネットワーク・フローの削減』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

AsyncEnable CLI/ODBC 構成キーワード

注: このキーワードは DB2 バージョン 8 ではサポートされず、旧バージョンへの互換性がある場合にのみ使用できます。このキーワードについては、 DB2 の旧バージョンの資料 (http://www.ibm.com/software/data/db2/library) を参照してください。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

AutoCommit CLI/ODBC 構成キーワード

キーワードの説明:

デフォルトでアプリケーションがステートメントごとにコミットするかどうかを指定します。

db2cli.ini キーワード構文:

AutoCommit = **1** | **0**

デフォルト設定:

各ステートメントは、単一の完全なトランザクションとして扱われます。

同等の接続属性:

SQL_ATTR_AUTOCOMMIT

使用上の注意:

ODBC との整合性を保つために、DB2 CLI では AutoCommit のデフォルトはオンになっています。つまり、各ステートメントは単一の完全なトランザクションとして扱われます。このキーワードでは別のデフォルトを提供することができますが、アプリケーションが、SQL_ATTR_AUTOCOMMIT の値を指定しない場合にのみ使用してください。

1 = SQL_ATTR_AUTOCOMMIT_ON (デフォルト)
0 = SQL_ATTR_AUTOCOMMIT_OFF

注: ほとんどの ODBC アプリケーションでは、 AutoCommit のデフォルトはオンであるということを前提にしています。アプリケーションはこのデフォルトの下で正しく機能する場合があるので、ランタイムの間にこのデフォルトをオーバーライドする場合は、十分に注意する必要があります。

このキーワードにより、分散作業単位 (DUOW) 環境で自動コミットを使用可能にするかどうかを指定することもできます。接続が整合分散作業単位の一部であり、AutoCommit が設定されないと、デフォルトは適用されません。自動コミット処理から生じる暗黙的コミットは抑制されます。 AutoCommit が 1 に設定され、接続が整合分散作業単位の一部である場合は、暗黙的コミットが処理されます。これが重大な性能低下につながり、DUOW システム外で他の予期しない結果が生じる可能性があります。ただし、これが使用可能でなければ、一部のアプリケーションはまったく機能しない可能性があります。

アプリケーションのトランザクション処理に関して、特にサード・パーティーによって作成されたアプリケーションについては、これを DUOW 環境に適用する前に、完全に理解する必要があります。

関連概念:

- 145 ページの『CLI アプリケーションでのマルチサイト更新 (2 フェーズ・コミット)』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

BitData CLI/ODBC 構成キーワード

キーワードの説明:

バイナリー・データ・タイプをバイナリー・データ・タイプとして報告するか、文字データ・タイプとして報告するかを指定します。

db2cli.ini キーワード構文:

BitData = **1** | **0**

デフォルト設定:

FOR BIT DATA および BLOB データ・タイプをバイナリー・データ・タイプとして報告します。

使用上の注意:

このオプションを使用すると、 ODBC バイナリー・データ・タイプ (SQL_BINARY、SQL_VARBINARY、 SQL_LONGVARBINARY、および SQL_BLOB) をバイナリー・タイプ・データとして報告するかどうかを指定できます。 FOR BIT DATA 属性を持つ CHAR、VARCHAR、および LONG VARCHAR 列を定義することによって、 IBM DBMS はバイナリー・データ・タイプの列をサポートします。 DB2 Universal Database は BLOB を介してバイナリー・データもサポートします (その場合、このデータは CLOB データ・タイプにマップされます)。

FOR BIT DATA または BLOB として定義されたすべての列に文字データのみが含まれ、アプリケーションがバイナリー・データ列を表示できないことが確実な場合にのみ、 BitData = 0 を設定してください。

1 = FOR BIT DATA および BLOB データ・タイプをバイナリー・データ・タイプとして報告します (デフォルト)。

0 = FOR BIT DATA および BLOB データ・タイプを文字データ・タイプとして報告します。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

BlockForNRows CLI/ODBC 構成キーワード

キーワードの説明:

1 回のフェッチで戻されるデータ行の数を指定します。

db2cli.ini キーワード構文:

BlockForNRows = <正整数>

デフォルト設定:

サーバーは、1 回のフェッチ要求に対して、 1 つの照会ブロックに入るだけの数の行を戻します。

使用上の注意:

BlockForNRows キーワードは、 1 回のフェッチ要求でクライアントに戻されるデータ行の数を制御します。 BlockForNRows が指定されていない場合 (デフォルトの設定の場合)、 1 個の照会ブロックに入るだけの数の非 LOB データ行がサーバーから戻されます。結果セットに LOB データが含まれている場合の BlockForNRows の

動作は、 BlockLobs CLI/ODBC 構成キーワード、および LOB ブロッキングがサーバーでサポートされているかどうかによって異なります。

BlockForNRows が指定されていない場合、 BlockLobs が 1 に設定されていて LOB ブロッキングがサーバーでサポートされているなら、 1 つの照会ブロックに完全に入るだけの数の行に関連するすべての LOB データが、 1 回のフェッチ要求で戻されます。ここで LOB データが行に関連するものとして記述されているのは、結果セットの LOB データ自体は行に含まれていないからです。行に含まれるのは、実際の LOB データへの参照です。

BlockForNRows が正の整数 n に設定されている場合には、 1 回のフェッチ要求に対して n 個のデータ行が戻されます。結果セットに LOB データが含まれていて、LOB ブロッキングがサーバーでサポートされている場合、 n 個のデータ行に対応する LOB データが 1 回のフェッチ要求に対して戻されます。結果セットに LOB データが含まれているが、サーバーで LOB ブロッキングがサーバーされていない場合には、 1 回のフェッチ要求に対して戻されるのは LOB データを含む 1 行だけになります。

関連概念:

- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『ラージ・オブジェクトの使用法』
- 109 ページの『CLI アプリケーションでのラージ・オブジェクトの使用』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 300 ページの『BlockLobs CLI/ODBC 構成キーワード』

BlockLobs CLI/ODBC 構成キーワード

キーワードの説明:

LOB ブロッキングをサポートするサーバーに対して、 LOB ブロッキング・フェッチを有効にします。

db2cli.ini キーワード構文:

BlockLobs = **0** | **1**

デフォルト設定:

LOB ブロッキング・フェッチは無効です。

同等のステートメント属性:

SQL_ATTR_BLOCK_LOBS

使用上の注意:

BlockLobs が 1 に設定されている場合、 LOB ブロッキングがサーバーでサポートされているなら、 1 つの照会ブロックに完全に入るだけの数の行に関連するすべての LOB データが、 1 回のフェッチ要求で戻されます。ここで LOB データが行に関連するものとして記述されているのは、結果セットの LOB データ自体は行に含まれていないからです。行に含まれるのは、実際の LOB データへの参照です。し

たがって、LOB ブロッキング・フェッチを使用する場合、サーバーで LOB ブロッキングがサポートされているなら、結果セットのうち照会ブロック内に完全に入るだけの数の行が、サーバーから戻される LOB データに関連付けられることになります (LOB データは行に直接格納されるわけではないため、各行の内容は非 LOB データです)。

サーバーで LOB ブロッキングがサーバーされていない場合、結果セットに LOB データが含まれているなら、 BlockLobs の設定値とは関係なく、 1 回のフェッチ要求に対して戻されるのは LOB データを含む 1 行だけになります。

関連概念:

- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『ラージ・オブジェクトの使用法』
- 109 ページの『CLI アプリケーションでのラージ・オブジェクトの使用』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 299 ページの『BlockForNRows CLI/ODBC 構成キーワード』

ClientAcctStr CLI/ODBC 構成キーワード

キーワードの説明:

ホスト・データベースに送られたクライアント・アカウンティング・ストリングを設定します。

db2cli.ini キーワード構文:

ClientAcctStr = *accounting string*

デフォルト設定:

なし

次の場合にのみ適用可能:

DB2 Connect を使用してホスト・データベースに接続されているとき

同等の接続属性:

SQL_ATTR_INFO_ACCTSTR

使用上の注意:

このオプションによって、 CLI アプリケーションは DB2 Connect を介してホスト・データベースに送られたクライアント・アカウンティング・ストリングを設定できます。デフォルトでアカウンティング・ストリングを提供しないアプリケーションは、その情報を提供するためにこのキーワードを利用できます。

以下の条件に注意してください。

- 値の設定中、サーバーによっては、指定した長さ全体を処理せず、値を切り捨てる場合があります。

- DB2 for z/OS および OS/390 サーバーがサポートするのは、最大 200 文字までの長さです。
- ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 302 ページの『ClientApplName CLI/ODBC 構成キーワード』
- 304 ページの『ClientUserID CLI/ODBC 構成キーワード』
- 305 ページの『ClientWrkStnName CLI/ODBC 構成キーワード』

ClientApplName CLI/ODBC 構成キーワード

キーワードの説明:

ホスト・データベースに送られたクライアント・アプリケーション名を設定します。

db2cli.ini キーワード構文:

ClientApplName = *application name*

デフォルト設定:

なし

次の場合にのみ適用可能:

DB2 Connect を使用してホスト・データベースに接続されているとき

同等の接続属性:

SQL_ATTR_INFO_APPLNAME

使用上の注意:

このオプションによって、 CLI アプリケーションは DB2 Connect を介してホスト・データベースに送られたクライアント・アプリケーション名を設定できます。デフォルトでアプリケーション名を提供しないアプリケーションは、その情報を提供するためにこのキーワードを利用できます。

以下の条件に注意してください。

- 値の設定中、サーバーによっては、指定した長さ全体を処理せず、値を切り捨てる場合があります。
- DB2 for z/OS および OS/390 サーバーがサポートするのは、最大 32 文字までの長さです。
- ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 301 ページの『ClientAcctStr CLI/ODBC 構成キーワード』
- 304 ページの『ClientUserID CLI/ODBC 構成キーワード』
- 305 ページの『ClientWrkStnName CLI/ODBC 構成キーワード』

ClientBuffersUnboundLOBS CLI/ODBC 構成キーワード

キーワードの説明:

アプリケーション・パラメーターにバインドされていない LOB 列に対して、 LOB ロケーターの代わりに LOB データを取り出します。

db2cli.ini キーワード構文:

ClientBuffersUnboundLOBS = **0** | **1**

デフォルト設定:

アプリケーション・パラメーターにバインドされていない LOB 列に対して、実際の LOB データの代わりに LOB ロケーターが検索されます。

使用上の注意:

デフォルトでは、アプリケーション・パラメーターにバインドされていない LOB 列が結果セットに含まれている場合、 DB2 CLI は LOB データそのものではなく対応する LOB ロケーターを取り出します。その場合、アプリケーションは LOB データを取り出すのに `SQLGetLength()`、 `SQLGetPosition()`、および `SQLGetSubString()` CLI 関数を使用しなければなりません。アプリケーションが定期的に LOB データの検索を要求する場合は、このデフォルトの 2 段階処理は不要であり、効率は低下する可能性があります。この場合、ClientBuffersUnboundLOBS = 1 を設定して、強制的に DB2 CLI が LOB ロケーターではなく LOB データを取り出すようにします。

関連概念:

- 111 ページの『CLI アプリケーションでの LOB ロケーター』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetLength 関数 (CLI) - ストリング値の長さの取り出し』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetPosition 関数 (CLI) - ストリングの開始位置を戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetSubString 関数 (CLI) - ストリング値の部分的な取り出し』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

ClientUserID CLI/ODBC 構成キーワード

キーワードの説明:

ホスト・データベースに送られたクライアント・ユーザー ID を設定します。

db2cli.ini キーワード構文:

ClientUserID = *userid*

デフォルト設定:

なし

次の場合にのみ適用可能:

DB2 Connect を使用してホスト・データベースに接続されているとき

同等の接続属性:

SQL_ATTR_INFO_USERID

使用上の注意:

このオプションによって、 CLI アプリケーションは DB2 Connect を介してホスト・データベースに送られたクライアント・ユーザー ID (アカウンティング・ユーザー ID) を設定できます。デフォルトでユーザー ID を提供しないアプリケーションは、その情報を提供するためにこのキーワードを利用できます。

以下の条件に注意してください。

- 値の設定中、サーバーによっては、指定した長さ全体を処理せず、値を切り捨てる場合があります。
- DB2 for z/OS および OS/390 サーバーがサポートするのは、最大 16 文字までの長さです。
- このユーザー ID を認証ユーザー ID と混同しないでください。このユーザー ID は識別目的でのみ使用され、許可のために使われることはありません。
- ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 301 ページの『ClientAcctStr CLI/ODBC 構成キーワード』
- 302 ページの『ClientApplName CLI/ODBC 構成キーワード』
- 305 ページの『ClientWrkStnName CLI/ODBC 構成キーワード』

ClientWrkStnName CLI/ODBC 構成キーワード

キーワードの説明:

ホスト・データベースに送られたクライアント・ワークステーション名を設定します。

db2cli.ini キーワード構文:

ClientWrkStnName = *workstation name*

デフォルト設定:

なし

次の場合にのみ適用可能:

DB2 Connect を使用してホスト・データベースに接続されているとき

同等の接続属性:

SQL_ATTR_INFO_WRKSTNNAME

使用上の注意:

このオプションによって、 CLI アプリケーションは DB2 Connect を介してホスト・データベースに送られたクライアント・ワークステーション名を設定できます。デフォルトでワークステーション名を提供しないアプリケーションは、その情報を提供するためにこのキーワードを利用できます。

以下の条件に注意してください。

- 値の設定中、サーバーによっては、指定した長さ全体を処理せず、値を切り捨てる場合があります。
- DB2 for z/OS および OS/390 サーバーがサポートするのは、最大 18 文字までの長さです。
- ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 301 ページの『ClientAcctStr CLI/ODBC 構成キーワード』
- 302 ページの『ClientApplName CLI/ODBC 構成キーワード』
- 304 ページの『ClientUserID CLI/ODBC 構成キーワード』

CLIPkg CLI/ODBC 構成キーワード

キーワードの説明:

生成するラージ・パッケージの数を指定します。

db2cli.ini キーワード構文:

CLIPkg = **3** | **4** | ... | **30**

デフォルト設定:

3 つの大きなパッケージが生成されます。

使用上の注意:

このキーワードは、CLI/ODBC アプリケーションでの SQL セクションの数を増やすのに使用されます。これが使用される場合には、管理者は CLIPkg BIND オプションを使用して、必要なバインド・ファイルを明示的にバインドする必要があります。 クライアント・アプリケーションの場合、クライアント上の `db2cli.ini` ファイルは、この CLIPkg の値で更新する必要があります。 CLI/JDBC ストアード・プロシージャーの場合、 (UNIX または Intel プラットフォームでの DB2 UDB バージョン 6.1 以降の) サーバー上の `db2cli.ini` ファイルは、CLIPkg の同じ値で更新する必要があります。

値として 3 以上 30 以下の整数が指定されていない場合には、デフォルトが使用されます。このとき、エラーまたは警告は出されません。

この設定は、ラージ・パッケージ (384 個のセクションを含む) にのみ適用されます。スモール・パッケージ (64 個のセクションを含む) の数は 3 個であり、変更できません。

パッケージはデータベースでスペースをとるため、増やすセクションの数は、ご使用のアプリケーションを実行できるだけの数にとどめるようお勧めします。

関連概念:

- 19 ページの『CLI でのハンドル』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「コマンド・リファレンス」の『BIND コマンド』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

CLISchema CLI/ODBC 構成キーワード

キーワードの説明:

使用する DB2 ODBC カタログ・ビューを設定します。

db2cli.ini キーワード構文:

CLISchema = *ODBC catalog view*

デフォルト設定:

なし - ODBC カタログ・ビューを使用しません

同等の接続属性:

SQL_ATTR_CLISCHEMA

使用上の注意:

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

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

システム管理者は、複数の DB2 ODBC カタログ・ビューを作成し、そのおのおのに個々のユーザー・グループが必要とする行だけを取り込むことができます。これにより、各エンド・ユーザーは使用する DB2 ODBC カタログ・ビューを (このキーワードを設定することによって) 選択できます。

CLISchema 設定の使用は、ODBC アプリケーションに対して完全に透過的です。このオプションはすべての ODBC アプリケーションとともに使用できます。

CLISchema はデータ・アクセス効率を向上させます。 SYSSCHEMA とともに使用されるユーザー定義表は DB2 カタログ表のミラー・イメージで、 CLI/ODBC ドライバーは CLI/ODBC アプリケーションが必要とする情報を作成するために複数の表の行を結合する必要があります。 CLISchema を使用すると、カタログ表のロック競合も少なくなります。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

ConnectNode CLI/ODBC 構成キーワード

キーワードの説明:

接続が行われるデータベース・パーティション・サーバーを指定してください。

db2cli.ini キーワード構文:

ConnectNode = **0 から 999 までの整数値** |
SQL_CONN_CATALOG_NODE

デフォルト設定:

マシンでポート 0 に定義されているデータベース・パーティション・サーバーが使用されます。

次の場合にのみ適用可能:

パーティション・データベース環境への接続時

同等の接続属性:

SQL_ATTR_CONNECT_NODE

使用上の注意:

これは、接続するターゲット・データベース・パーティション・サーバーを指定するために使用されます。このキーワード (または属性設定) は、環境変数 DB2NODE の値をオーバーライドします。以下のように設定できます。

- 0 から 999 までの整数
- SQL_CONN_CATALOG_NODE

この変数が設定されていない場合、ターゲットにはデフォルトとしてマシンのポート 0 に定義されたデータベース・パーティション・サーバーが使用されます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連タスク:

- 「*管理ガイド: インプリメンテーション*」の『Windows 上の環境変数の設定』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

ConnectType CLI/ODBC 構成キーワード

注: このキーワードは DB2 バージョン 8 ではサポートされず、旧バージョンへの互換性がある場合にのみ使用できます。このキーワードについては、 DB2 の旧バージョンの資料 (http://www.ibm.com/software/data/db2/library) を参照してください。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

CurrentFunctionPath CLI/ODBC 構成キーワード

キーワードの説明:

動的 SQL ステートメントで関数参照およびデータ・タイプ参照の解決に使用されるスキーマを指定します。

db2cli.ini キーワード構文:

CurrentFunctionPath = *current_function_path*

デフォルト設定:

以下の記述を参照してください。

使用上の注意:

このキーワードは、動的 SQL で使用される関数参照およびデータ・タイプ参照の解決に使用されるパスを定義します。これには、1 つ以上のスキーマ名のリストが含まれます。スキーマ名は二重引用符で囲まれ、コンマで区切られます。

デフォルト値は "SYSIBM","SYSFUN",X です。 X は、二重引用符で区切られた、USER 特殊レジスターの値です。スキーマ SYSIBM を指定する必要はありません。関数パスに組み込まれていない場合、暗黙的にこのスキーマが最初のスキーマであると見なされます。

このキーワードは、現在のユーザーのスキーマ以外のスキーマ名で定義された可能性のある非修飾関数およびストアード・プロシージャー参照を解決する処理の一部として使用されます。スキーマ名の順序によって、関数名およびプロシージャー名が解決される順序が決定されます。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『スキーマ』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『USER 特殊レジスター』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

CurrentMaintainedTableTypesForOpt CLI/ODBC 構成キーワード

キーワードの説明:

CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターの値を設定します。

db2cli.ini キーワード構文:

CurrentMaintainedTableTypesForOpt = **ALL** | **FEDERATED_TOOL** | **NONE** | **SYSTEM** | **USER** | **<リスト>**

デフォルト設定:

照会の最適化において、システムの管理するリフレッシュ据え置きマテリアライズ照会表が考慮されます。

使用上の注意:

このキーワードは、 CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターのデフォルト値を定義します。この特殊レジスターの値は、照会の最適化において考慮される表の種類に影響します。サポートされている ALL、FEDERATED_TOOL、NONE、SYSTEM、または USER の設定値については、 SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION SQL ステートメントの説明を参照してください。 *<リスト>* オプションは、サポートされている複数の設定値の組み合わせを表しています。しかし、ALL と NONE については、他の値と組み合わせて指定することはできません。また、同じ値を複数回指定することもできません。リストの中では、値と値の間をコンマで区切ってください。たとえば、

```
CurrentMaintainedTableTypesForOpt = SYSTEM,USER
```

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION ステートメント』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 「*管理ガイド: パフォーマンス*」の『dft_mttb_types - 最適化用デフォルト保守表タイプ構成パラメーター』

CurrentPackagePath CLI/ODBC 構成キーワード

キーワードの説明:

すべての接続の後に、'SET CURRENT PACKAGE PATH = *schema1, schema2, ...*' を出します。

db2cli.ini キーワード構文:

CurrentPackagePath = *schema1, schema2, ...*

デフォルト設定:

文節は付加されません。

同等の接続属性:

SQL_ATTR_CURRENT_PACKAGE_PATH

使用上の注意:

このオプションを設定すると、データベースに対するすべての接続の後にコマンド "SET CURRENT PACKAGE PATH = *schema1, schema2, ...*" が発行されます。この設定値は、他のスキーマからのパッケージが存在するときに検索されるスキーマ名 (コレクション ID) のリストを指定します。

このキーワードは、 CLI アプリケーションではなく ODBC 静的処理アプリケーションで使用することに最適です。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

CurrentPackageSet CLI/ODBC 構成キーワード

キーワードの説明:

すべての接続の後に 'SET CURRENT PACKAGESET スキーマ' を発行します。

db2cli.ini キーワード構文:

CurrentPackageSet = スキーマ名

デフォルト設定:

文節は付加されません。

同等の接続属性:

SQL_ATTR_CURRENT_PACKAGE_SET

使用上の注意:

このオプションは、データベースへのすべての接続の後にコマンド 「SET CURRENT PACKAGESET スキーマ」 を発行します。デフォルトでは、この文節は付加されません。

このステートメントは、後続の SQL ステートメントのために使用するパッケージの選択に使用されるスキーマ名 (コレクション ID) を設定します。

CLI/ODBC アプリケーションは、動的 SQL ステートメントを発行します。このオプションを使用すると、これらのステートメントの実行に使用される特権を制御することができます。

- CLI/ODBC アプリケーションから SQL ステートメントを実行するときに使用するスキーマを選択します。
- スキーマ内のオブジェクトに必要な特権があることを確認してから、それに従って再バインドします。
- CurrentPackageSet オプションをこのスキーマに設定します。

CLI/ODBC アプリケーションからの SQL ステートメントは、指定されたスキーマの下で実行され、そこで定義された特権を使用します。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*SQL リファレンス 第 2 巻*の『SET CURRENT PACKAGESET ステートメント』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

CurrentRefreshAge CLI/ODBC 構成キーワード

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** | **positive integer**

デフォルト設定:

REFRESH IMMEDIATE で定義されたマテリアライズ照会表だけを使って照会処理を最適化できます。

使用上の注意:

このキーワードを設定すると、CURRENT REFRESH AGE 特殊レジスターの値が設定されます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『SET CURRENT REFRESH AGE ステートメント』
- 「*SQL リファレンス 第 1 巻*」の『CURRENT REFRESH AGE 特殊レジスター』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

CurrentSchema CLI/ODBC 構成キーワード

キーワードの説明:

接続成功時に SET CURRENT SCHEMA ステートメントで使用されるスキーマを指定します。

db2cli.ini キーワード構文:

CurrentSchema = *スキーマ名*

デフォルト設定:

ステートメントは発行されません。

使用上の注意:

このオプションが設定されている場合、接続成功時には SET CURRENT SCHEMA ステートメントが DBMS に送信されます。これにより、エンド・ユーザーまたはアプリケーションは、 SQL オブジェクトをスキーマ名で限定せずに指定することができます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『SET SCHEMA ステートメント』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

CurrentSQLID CLI/ODBC 構成キーワード

キーワードの説明:

接続成功時に DBMS に送信される SET CURRENT SQLID ステートメントで使用される ID を指定します。

db2cli.ini キーワード構文:

CurrentSQLID = *current_sqlid*

デフォルト設定:

ステートメントは発行されません。

次の場合にのみ適用可能:

SET CURRENT SQLID がサポートされている DB2 DBMS に接続する。

使用上の注意:

このオプションが設定されている場合、接続成功時には SET CURRENT SQLID ステートメントが DBMS に送信されます。これにより、エンド・ユーザーおよびアプリケーションは、 SQL オブジェクトをスキーマ名で限定せずに指定することができます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『SET SCHEMA ステートメント』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

CursorHold CLI/ODBC 構成キーワード

キーワードの説明:

オープン・カーソル上でのトランザクション完了の効果。

db2cli.ini キーワード構文:

CursorHold = **1** | **0**

デフォルト設定:

選択 -- カーソルは破棄されません。

同等のステートメント属性:

SQL_ATTR_CURSOR_HOLD

使用上の注意:

このオプションは、オープン・カーソル上でのトランザクション完了の効果を制御します。

1 = SQL_CURSOR_HOLD_ON、カーソルはトランザクションがコミットされても破棄されません (デフォルト)。

0 = SQL_CURSOR_HOLD_OFF、カーソルはトランザクションがコミットされると破棄されます。

注: カーソルはトランザクションがロールバックされるたびにクローズされます。

このオプションは、 SQL_CURSOR_COMMIT_BEHAVIOR または SQL_CURSOR_ROLLBACK_BEHAVIOR と一緒に呼び出されると、 `SQLGetInfo()` によって戻される結果に影響します。 保留カーソルがサポートされていない DB2 Server for VSE & VM への接続の場合、 CursorHold の値は無視されます。

このオプションを使用して、パフォーマンスを調整することができます。アプリケーションが以下にあてはまることが確実である場合は、SQL_CURSOR_HOLD_OFF (0) に設定できます。

1. `SQLGetInfo()` によって戻される SQL_CURSOR_COMMIT_BEHAVIOR または SQL_CURSOR_ROLLBACK_BEHAVIOR 情報に依存した動作を行わない。
2. あるトランザクションから次のトランザクションまでカーソルを保持する必要がない。

トランザクションの終了後にリソースを保守する必要がなくなるため、 CursorHold が使用不可の状態で DBMS はより効果的に稼働します。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetInfo 関数 (CLI) - 一般情報の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

CursorTypes CLI/ODBC 構成キーワード

キーワードの説明:

可能なカーソル・タイプを指定します。

db2cli.ini キーワード構文:

CursorTypes = **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7**

デフォルト設定:

サーバーでサポートされているなら、順方向、静的、キー・セット主導、および動的の各カーソルがサポートされます。

使用上の注意:

CursorTypes キーワードは、アプリケーションでどのタイプのカーソルを開くことができるかを指定するためのビット・マスクです。

- 0x0 - 順方向専用 (常に可能)
- 0x1 - 静的
- 0x2 - キー・セット主導
- 0x4 - 動的

たとえば、

- アプリケーションで動的スクロール可能カーソルを開くことができないようにするには、 CursorTypes に 3 を設定します。
- アプリケーションがスクロール不可カーソルしか開くことができないようにするには、 CursorTypes を 0 に設定します。

このキーワードは、以下の DB2 CLI 関数の呼び出しにのみ影響します。

- `SQLBulkOperations()`
- `SQLExecDirect()`
- `SQLExecute()`

- SQLFetchScroll()
- SQLPrepare()
- SQLSetPos()

関連概念:

- 73 ページの『CLI アプリケーションのカーソル』
- 76 ページの『CLI アプリケーションのカーソルに関する考慮事項』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLExecute 関数 (CLI) - ステートメントの実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLPrepare 関数 (CLI) - ステートメントの準備』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetPos 関数 (CLI) - 行セット (Rowset) 内のカーソル位置の設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBulkOperations 関数 (CLI) - 行のセットの追加、更新、削除、または取り出し』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

Database CLI/ODBC 構成キーワード

キーワードの説明:

ファイル DSN 使用時に接続するサーバーのデータベース。

db2cli.ini キーワード構文:

Database = データベース名

デフォルト設定:

なし

次の場合にのみ適用可能:

プロトコルが TCPIP に設定されている。

使用上の注意:

ファイル DSN を使用するときは、このオプションを使用して、接続するサーバーのデータベースを指定する必要があります。この値はクライアントで指定されるデータベース別名とは関係ありません。これは、サーバーのデータベース名自体に設定される必要があります。

この設定は、Protocol オプションが TCPIP に設定されているときにのみ考慮されます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 329 ページの『Hostname CLI/ODBC 構成キーワード』
- 348 ページの『Protocol CLI/ODBC 構成キーワード』
- 353 ページの『ServiceName CLI/ODBC 構成キーワード』

DateTimeStringFormat CLI/ODBC 構成キーワード

キーワードの説明:

日付または時刻を文字列に挿入する際に使用する形式を指定します。

db2cli.ini キーワード構文:

DateTimeStringFormat = **JIS** | **ISO** | **EUR** | **USA**

デフォルト設定:

日付または時刻のデータを文字列に挿入する際、JIS 形式が使用されます。

使用上の注意:

DateTimeStringFormat キーワードは、日付または時刻のデータを文字列に挿入する際の形式を指定します。その設定値は、 SQL_C_DATE、SQL_C_TIME、または SQL_C_TIMESTAMP のデータを、下記のタイプの列に挿入する方法に影響します。

- SQL_CHAR
- SQL_VARCHAR
- SQL_LONGVARCHAR
- SQL_CLOB

設定値には 4 種類あり、それらは次のとおりです。

形式	日付	時刻	タイム・スタンプ
JIS	yyyy-mm-dd	hh:mm:ss	yyyy-mm-dd hh:mm:ss.ffffff
ISO	yyyy-mm-dd	hh.mm.ss	yyyy-mm-dd-hh.mm.ss.ffffff
EUR	dd.mm.yyyy	hh.mm.ss	yyyy-mm-dd hh:mm:ss.ffffff*
	mm/dd/yyyy	hh:mm AM または PM	yyyy-mm-dd hh:mm:ss.ffffff*
* EUR または USA が指定された場合、タイム・スタンプはデフォルトの形式になります。 DB2 UDB バージョン 8 の場合、デフォルトの形式は JIS です。			

重要: このキーワードは、日付または時刻の列から取り出されて文字ストリングに入れられるデータの形式には影響しません。たとえば、SQL_TIMESTAMP 列からデータを取り出して SQL_C_CHAR ストリングに入れる場合、このキーワードの設定値は影響しません。

関連概念:

- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 「*SQL リファレンス 第 1 巻*」の『日付/時刻の値』

DB2Degree CLI/ODBC 構成キーワード

キーワードの説明:

SQL ステートメントの実行について並列処理の度合いを設定します。

db2cli.ini キーワード構文:

DB2Degree = **0** | **integer value from 1 to 32767** | **ANY**

デフォルト設定:

SET CURRENT DEGREE ステートメントは発行されません。

次の場合にのみ適用可能:

クラスター・データベース・システムに接続する。

使用上の注意:

指定された値が 0 (デフォルト) 以外のものである場合、 DB2 CLI は接続成功後に次の SQL ステートメントを発行します。

```
SET CURRENT DEGREE 値
```

これにより、SQL ステートメントの実行について並列処理の度合いが指定されます。 ANY を指定するとデータベース・マネージャーによって並列処理の度合いが決定されます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『SET CURRENT DEGREE ステートメント』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

DB2Explain CLI/ODBC 構成キーワード

キーワードの説明:

Explain スナップショットと Explain 表情報のどちらか、またはその両方がサーバーによって生成されるかどうかを決定します。

db2cli.ini キーワード構文:

DB2Explain = **0** | **1** | **2** | **3**

デフォルト設定:

Explain スナップショットも Explain 表情報もサーバーによって生成されません。

同等の接続属性:

SQL_ATTR_DB2EXPLAIN

使用上の注意:

このキーワードは、Explain スナップショットと Explain 表情報のどちらか、またはその両方がサーバーによって生成されるかどうかを決定します。

- 0 = 両方をオフにします (デフォルト)

  'SET CURRENT EXPLAIN SNAPSHOT=NO' および 'SET CURRENT EXPLAIN MODE=NO' ステートメントがサーバーに送信され、 Explain スナップショットおよび Explain 表情報キャプチャー機能の両方を使用不可にします。
- 1 = Explain スナップショット機能のみをオンにします

  'SET CURRENT EXPLAIN SNAPSHOT=YES' および 'SET CURRENT EXPLAIN MODE=NO' ステートメントがサーバーに送信され、 Explain スナップショット機能を使用可能にして、 Explain 表情報キャプチャー機能を使用不可にします。
- 2 = EXPLAIN 表情報キャプチャー機能のみをオンにします

  'SET CURRENT EXPLAIN MODE=YES' および 'SET CURRENT EXPLAIN SNAPSHOT=NO' がサーバーに送信され、 Explain 表情報キャプチャー機能を使用可能にして、 Explain スナップショット機能を使用不可にします。
- 3 = 両方をオンにします

  'SET CURRENT EXPLAIN MODE=YES' および 'SET CURRENT EXPLAIN SNAPSHOT=YES' がサーバーに送信され、 Explain スナップショットおよび Explain 表情報キャプチャー機能の両方を使用可能にします。

EXPLAIN 情報は EXPLAIN 表に挿入されます。 EXPLAIN 情報を生成するには、EXPLAIN 表が作成されていなければなりません。現在の許可 ID に、EXPLAIN 表の INSERT 特権が必要です。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『CLI と ODBC 関数のサマリー』
- 「*SQL リファレンス 第 2 巻*」の『EXPLAIN ステートメント』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』

DB2Optimization CLI/ODBC 構成キーワード

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 から 9 までの整数値*

デフォルト設定:

SET CURRENT QUERY OPTIMIZATION ステートメントは発行されません。

使用上の注意:

このオプションが設定されると、DB2 CLI は接続成功後に次の SQL ステートメントを発行します。

```
SET CURRENT QUERY OPTIMIZATION 正数
```

これにより、オプティマイザーが SQL 照会を操作する照会最適化レベルが指定されます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『SET CURRENT QUERY OPTIMIZATION ステートメント』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

DBAlias CLI/ODBC 構成キーワード

キーワードの説明:

データ・ソース名のデータベース別名を 8 文字以上で指定します。

db2cli.ini キーワード構文:

DBAlias = *dbalias*

デフォルト設定:

ODBC データ・ソース名として DB2 データベース別名を使用します。

使用上の注意:

このキーワードは、単一バイトで 8 文字を超えるデータ・ソース名を許可します。データ・ソース名 (DSN) は、大括弧で囲まれた名前です。これは、`db2cli.ini` のセクション・ヘッダーを示します。一般に、このセクション・ヘッダーは最大 8 バイトの長さのデータベース別名です。これよりも長く意味のある名前を持つデータ・ソースを参照したいユーザーは、その長い名前をセクション・ヘッダーに入れ、このキーワード値を CATALOG コマンドで使用されるデータベース別名に設定することができます。以下はその例です。

```
; The much longer name maps to an 8 single byte character dbalias
[MyMeaningfulName]
DBAlias=DB2DBT10
```

エンド・ユーザーは、実際のデータベース別名が DB2DBT10 でも、接続時にデータ・ソースの名前として [MyMeaningfulName] を指定することができます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「コマンド・リファレンス」の『CATALOG DATABASE コマンド』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

DBName CLI/ODBC 構成キーワード

キーワードの説明:

アプリケーションが z/OS または OS/390 表情報の照会に要する時間を削減するために、データベース名を指定します。

db2cli.ini キーワード構文:

DBName = *dbname*

デフォルト設定:

DBNAME 列でフィルターをかけません。

次の場合にのみ適用可能:

DB2 for z/OS および OS/390 に接続する。

使用上の注意:

このオプションは、DB2 for z/OS および OS/390 に接続しているとき、およびアプリケーションによって (基本) 表カタログ情報が要求された場合にのみ使用されます。 z/OS または OS/390 サブシステムに存在する表の数が多い場合、アプリケーションが表情報の照会に要する時間を削減し、アプリケーションによってリストされる表の数を減らすために、 *dbname* を指定できます。

このオプションが設定されると、 IN DATABASE *dbname* ステートメントは CREATE TABLE などのさまざまなステートメントに付加されます。

この値は z/OS または OS/390 システム・カタログ表の DBNAME 列にマップします。値が指定されていない場合、あるいはビュー、同義語、システム表、または別名が TableType を介して指定されている場合も、表情報のみが制限されます。ビュー、別名、および同義語は、DBName で制限されません。これを SchemaList および TableType とともに使用して、情報が戻される表の数をさらに制限することができます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 352 ページの『SchemaList CLI/ODBC 構成キーワード』
- 359 ページの『TableType CLI/ODBC 構成キーワード』

DefaultProcLibrary CLI/ODBC 構成キーワード

注: このキーワードは DB2 バージョン 8 ではサポートされず、旧バージョンへの互換性がある場合にのみ使用できます。このキーワードについては、 DB2 の旧バージョンの資料 (http://www.ibm.com/software/data/db2/library) を参照してください。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

DeferredPrepare CLI/ODBC 構成キーワード

キーワードの説明:

PREPARE 要求を、対応する実行要求と結合することにより、ネットワークの流れを最小化します。

db2cli.ini キーワード構文:

DeferredPrepare = **0** | **1**

デフォルト設定:

準備要求は実行要求が送信されるまで実行されません。

同等のステートメント属性:

SQL_ATTR_DEFERRED_PREPARE

使用上の注意:

対応する実行要求が発行されるまで、PREPARE 要求の送信を据え置きます。その後、ネットワーク・フローを最小化しパフォーマンスを改善するため、 2 つの要求が 2 つではなく 1 つのコマンド/応答のフローに結合されます。

- 0 = SQL_DEFERRED_PREPARE_OFF。 PREPARE 要求は、発行された時点で実行されます。
- 1 = SQL_DEFERRED_PREPARE_ON (デフォルト)。対応する実行要求が発行されるまで、PREPARE 要求の実行は据え置かれます。

  ターゲット DBMS が据え置き準備をサポートしていない場合、クライアントはその接続のための据え置き準備を使用不可にします。

注: 据え置き準備を使用可能にすると、通常は SQLCA の PREPARE ステートメントの SQLERRD(3) と SQLERRD(4) に戻される行およびコスト見積もりが、ゼロになる可能性があります。これは、これらの値を使用して SQL ステートメントを続行するかどうかを決定するユーザーにとっては重要です。

関連概念:

- 30 ページの『CLI アプリケーションでの据え置き準備』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連タスク:

- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『PREPARE ステートメント』
- 「*SQL リファレンス 第 1 巻*」の『SQLCA (SQL 連絡域)』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

DescribeInputOnPrepare CLI/ODBC 構成キーワード

キーワードの説明:

SQL ステートメントを準備するときに、記述情報の要求を使用可能または使用不可にする。

db2cli.ini キーワード構文:

DescribeInputOnPrepare = **0** | **1**

デフォルト設定:

SQL ステートメントの準備中に、記述情報を要求しません。

使用上の注意:

デフォルトでは、SQL ステートメントを準備する際に、 DB2 CLI は入力パラメーター記述情報を要求しません。アプリケーションがパラメーターをステートメントに正常にバインドした場合には、この記述情報は不要になるため、記述情報を要求しないとパフォーマンスが改善されます。しかし、パラメーターが正常にバインドされなかった場合には、ステートメントの実行が失敗し、 CLI エラー・リカバリー再試行論理によって入力パラメーター記述情報が要求されることになります。結果として、記述情報が準備の際に要求された場合と比べると、サーバー要求が増え、パフォーマンスは低下します。 DescribeInputOnPrepare を 1 に設定すると、入力記述情報は準備の際に要求されるようになります。このように設定すると、バインド・エラーからリカバリーを CLI 再試行論理にかなり依存しているアプリケーションのパフォーマンスが改善できる場合があります。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連タスク:

- 28 ページの『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 33 ページの『CLI アプリケーションでのパラメーター・マーカーのバインディング』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

DescribeParam CLI/ODBC 構成キーワード

キーワードの説明:

SQLDescribeParam() がサポートされているかどうかを示します。

db2cli.ini キーワード構文:

DescribeParam = **0** | **1**

デフォルト設定:

SQLDescribeParam() 関数は使用可能になっています。

使用上の注意:

このキーワードは、SQLDescribeParam() 関数を使用可能または使用不可にします。

1 (デフォルト) に設定すると、SQLDescribeParam() は使用可能となり、SQLGetFunctions() はサポートされているものとして SQLDescribeParam() を戻します。

0 に設定すると、SQLDescribeParam() は使用不可となります。SQLDescribeParam() が呼び出されると、CLI0150E "Driver not capable" が戻されます。 SQLGetFunctions() はサポートされていないものとして SQLDescribeParam() を戻します。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeParam 関数 (CLI) - パラメーター・マーカーの記述を戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetFunctions 関数 (CLI) - 関数の取得』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 「*メッセージ・リファレンス 第 1 巻*」の『CLI メッセージ』

DisableKeysetCursor CLI/ODBC 構成キーワード

キーワードの説明:

キー・セット主導スクロール可能カーソルを使用不可にします。

db2cli.ini キーワード構文:

DisableKeysetCursor = **0** | **1**

デフォルト設定:

キー・セット主導スクロール可能カーソルは要求時に戻されます。

使用上の注意:

1 に設定した場合、このキーワードは、アプリケーションがキー・セット主導スクロール可能カーソルを要求した場合でも、 CLI ドライバーがアプリケーションに対し静的カーソルを戻すように強制します。デフォルトの設定値 (0) により、キー・

セット主導カーソルは、アプリケーションの要求時に戻されます。このキーワードは、スクロール可能カーソルがサポートされるまで、動作のリストアに使用できます。

関連概念:

- 73 ページの『CLI アプリケーションのカーソル』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

DisableMultiThread CLI/ODBC 構成キーワード

キーワードの説明:

マルチスレッド化を使用不可にします。

db2cli.ini キーワード構文:

DisableMultiThread = **0** | **1**

デフォルト設定:

マルチスレッド化は使用可能です。

使用上の注意:

CLI/ODBC ドライバーは複数の並行スレッドをサポートできます。

このオプションは、マルチスレッド・サポートを使用可能または使用不可にするために使用されます。

- 0 = マルチスレッド化を使用可能にします (デフォルト)。
- 1 = マルチスレッド化を使用不可にします。

マルチスレッド化が使用不可の場合、すべてのスレッドのすべての呼び出しが処理レベルでシリアライズされます。動作をシリアライズする必要のあるマルチスレッド化アプリケーションのこの設定を使用してください。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 139 ページの『マルチスレッド CLI アプリケーション』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

DisableUnicode CLI/ODBC 構成キーワード

キーワードの説明:

基礎 Unicode サポートを使用不可にします。

db2cli.ini キーワード構文:

DisableUnicode = **<設定値なし>** | **0** | **1**

デフォルト設定:

Unicode サポートは使用可能です。

使用上の注意:

Unicode サポートが使用可能の場合で、 Unicode アプリケーションにより呼び出された場合は、 CLI は可能なクライアント・コード・ページで最良のものを使用してデータベースへの接続を試行し、コード・ページの変換により不要なデータの損失がないことを確認します。これにより、コード・ページの変更時に接続時間が増加するか、またはこのサポートが追加されるまでは発生しなかったクライアントのコード・ページの変換が発生する場合があります。

アプリケーションが Unicode の場合 (SQL_ATTR_ANSI_APP 接続属性が SQL_AA_FALSE に設定されているか、 SQLConnectW() を使用して接続された場合)、 DisableUnicode キーワードによって 3 つの異なる接続動作を設定できます。

- DisableUnicode が db2cli.ini ファイルで設定されていない場合: ターゲット・データベースが Unicode をサポートしていなければ、 DB2 CLI は Unicode コード・ページ (1208 および 1200) で接続します。ターゲット・データベースが Unicode をサポートしていれば、 DB2 CLI はアプリケーションのコード・ページで接続します。
- DisableUnicode=0 が設定されている場合: DB2 CLI は、ターゲット・データベースが Unicode をサポートしているかどうかに関係なく、常に Unicode で接続します。
- DisableUnicode=1 が設定されている場合: DB2 CLI は、ターゲット・データベースが Unicode をサポートしているかどうかに関係なく、常にアプリケーションのコード・ページで接続します。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 159 ページの『Unicode CLI アプリケーション』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

FloatPrecRadix CLI/ODBC 構成キーワード

キーワードの説明:

浮動小数点タイプの NUM_PREC_RADIX 値を 2 または 10 に強制的に指定します。

db2cli.ini キーワード構文:

FloatPrecRadix = **2** | **10**

デフォルト設定:

浮動小数点タイプが 10 ではなく 2 を基数としている場合に、浮動小数点タイプについて NUM_PREC_RADIX を 2 と報告します。

使用上の注意:

NUM_PREC_RADIX 値はデータ・タイプの基数を表します。浮動小数点数などのバイナリー数は 2 を基数とし、整数は 10 を基数とします。アプリケーションは最大桁数を表すために、 COLUMN_SIZE フィールドにあるすべての値を予期しますが、その場合に想定される NUM_PREC_RADIX 値は 10 です。ただし、浮動小数点数タイプの場合、 NUM_PREC_RADIX は 2 であり、このとき COLUMN_SIZE はデータ・タイプの表記に最大桁数ではなくビット数を報告します。

FloatPrecRadix は、浮動小数点データ・タイプに対しては NUM_PREC_RADIX が 10 として報告されるように強制することができ、この場合 COLUMN_SIZE は最大桁数を報告します。

FloatPrecRadix キーワードは `SQLColumns()`、 `SQLGetDescField()` (SQL_DESC_NUM_PREC_RADIX フィールドの場合)、 `SQLGetTypeInfo()`、 `SQLProcedureColumns()`、および `SQLSpecialColumns()` に影響します。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLColumns 関数 (CLI) - 表の列の情報の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetTypeInfo 関数 (CLI) - データ・タイプ情報の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLProcedureColumns 関数 (CLI) - プロシージャーの入力/出力パラメーター情報の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSpecialColumns 関数 (CLI) - 特殊な (行 ID) 列の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『記述子 FieldIdentifier 引き数の値 (CLI)』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

GranteeList CLI/ODBC 構成キーワード

キーワードの説明:

アプリケーションが表または列特権のリストを取得するときに戻される情報量を削減します。

db2cli.ini キーワード構文:

GranteeList = " '*userID1*', '*userID2*',... '*userIDn*' "

デフォルト設定:

結果にフィルターをかけません。

使用上の注意:

このオプションは、アプリケーションがデータベース内の表または表内の列の特権のリストを取得するときに戻される情報量を削減するために使用されます。指定された許可 ID のリストがフィルターとして使用されます。これらの *ID* に対して 付与された特権を持つ表または列のみが戻されます。

このオプションは、特権を付与された 1 つ以上の許可 ID を単一引用符で囲み、コンマで区切ったリストとして設定してください。ストリング全体が二重引用符で囲まれている必要もあります。たとえば、以下のようにします。

```
GranteeList=" 'USER1', 'USER2', 'USER8' "
```

上の例では、アプリケーションが特定の表について特権のリストを取得する場合に、 *USER1*、*USER2*、または *USER8* に対して 付与された特権を持つ列のみが戻されます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 327 ページの『GrantorList CLI/ODBC 構成キーワード』

GrantorList CLI/ODBC 構成キーワード

キーワードの説明:

アプリケーションが表または列特権のリストを取得するときに戻される情報量を削減します。

db2cli.ini キーワード構文:

GrantorList = ″ ’*userID1*’, ’*userID2*’,... ’*userIDn*’ ″

デフォルト設定:

結果にフィルターをかけません。

使用上の注意:

このオプションは、アプリケーションがデータベース内の表または表内の列の特権のリストを取得するときに戻される情報量を削減するために使用されます。指定された許可 ID のリストがフィルターとして使用されます。これらの *ID* によって 付与された特権を持つ表または列のみが戻されます。

このオプションは、特権を付与された 1 つ以上の許可 ID を単一引用符で囲み、コンマで区切ったリストとして設定してください。ストリング全体が二重引用符で囲まれている必要もあります。たとえば、以下のようにします。

```
GrantorList=" 'USER1', 'USER2', 'USER8' "
```

上の例では、アプリケーションが特定の表について特権のリストを取得する場合に、 *USER1*、*USER2*、または *USER8* によって 付与された特権を持つ列のみが戻されます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 326 ページの『GranteeList CLI/ODBC 構成キーワード』

Graphic CLI/ODBC 構成キーワード

キーワードの説明:

DB2 CLI が、サポートされる SQL データ・タイプとして SQL_GRAPHIC (2 バイト文字) を戻すかどうか、および GRAPHIC 列の長さを報告する際にどんな単位を使用するかを指定します。

db2cli.ini キーワード構文:

Graphic = **0** | **1** | **2** | **3**

デフォルト設定:

SQL_GRAPHIC GRAPHIC データは、サポートされている SQL データ・タイプとしては戻されず、 GRAPHIC 列の長さは、その列の DBCS 文字の数の最大値になります。

使用上の注意:

Graphic キーワードは、 `SQLGetTypeInfo()` が呼び出された場合にサポートされる SQL データ・タイプとして SQL_GRAPHIC (2 バイト文字) データ・タイプが報告されるかどうか、また、出力引き数として、または結果セットの一部として、長さまたは精度を戻すすべての DB2 CLI 関数において GRAPHIC 列の長さを報告するために使用される単位を指定します。

Graphic キーワードは、次のように設定します。

- 0 - SQL_GRAPHIC は、サポートされている SQL データ・タイプとしては戻されず、 GRAPHIC 列の長さとして報告される値はその列の DBCS 文字の数の最大値になります。
- 1 - SQL_GRAPHIC は、サポートされている SQL データ・タイプとして戻され、 GRAPHIC 列の長さとして報告される値はその列の DBCS 文字の数の最大値になります。
- 2 - SQL_GRAPHIC は、サポートされている SQL データ・タイプとしては戻されず、 GRAPHIC 列の長さとして報告される値はその列のバイト数の最大値になります。
- 3 - SQL_GRAPHIC は、サポートされている SQL データ・タイプとしては戻され、 GRAPHIC 列の長さとして報告される値はその列のバイト数の最大値になります。

関連概念:

- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetTypeInfo 関数 (CLI) - データ・タイプ情報の取得』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

Hostname CLI/ODBC 構成キーワード

キーワードの説明:

ファイル DSN 接続で、または DSN なし接続で使用されるサーバー・システムのホスト名または IP アドレス。

db2cli.ini キーワード構文:

Hostname = *ホスト名* | *IP アドレス*

デフォルト設定:

なし

次の場合にのみ適用可能:

プロトコルが TCPIP に設定されている。

使用上の注意:

このオプションは、このクライアント・マシンと DB2 が実行されているサーバーとの TCP/IP 接続に必要な属性を指定するために、 ServiceName オプションと共に使用します。これらの 2 つの値は、Protocol オプションが TCPIP に設定されているときにのみ使用されます。

サーバー・システムのホスト名または IP アドレスのどちらかを指定します。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 348 ページの『Protocol CLI/ODBC 構成キーワード』
- 353 ページの『ServiceName CLI/ODBC 構成キーワード』

IgnoreWarnings CLI/ODBC 構成キーワード

キーワードの説明:

データベース・マネージャーの警告を無視します。

db2cli.ini キーワード構文:

IgnoreWarnings = **0** | **1**

デフォルト設定:

警告は通常どおりに戻されます。

使用上の注意:

まれなことではありますが、アプリケーションが警告メッセージを正しく処理しない場合があります。このキーワードを使用することにより、データベース・マネージャーからの警告のうち、アプリケーションに渡さないものを指定できます。可能な設定値は次のとおりです。

- 0 - 警告は通常の方法で報告されます (デフォルト)。
- 1 - データベース・マネージャーの警告は無視され、 SQL_SUCCESS が戻されます。一方、DB2 CLI/ODBC ドライバーからの警告は戻されます。通常の操作では、その多くが必要です。

このキーワードはそれだけでも使用可能ですが、 WarningList CLI/ODBC 構成キーワードと共に使用することもできます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 330 ページの『IgnoreWarnList CLI/ODBC 構成キーワード』
- 377 ページの『WarningList CLI/ODBC 構成キーワード』

IgnoreWarnList CLI/ODBC 構成キーワード

キーワードの説明:

指定された sqlstate を無視します。

db2cli.ini キーワード構文:

IgnoreWarnList = 『'sqlstate1', 'sqlstate2', ...』

デフォルト設定:

警告は通常どおりに戻されます

使用上の注意:

まれに、アプリケーションが処理できない警告メッセージがあるが、すべての警告メッセージは無視しないという場合があります。このキーワードは、アプリケーションに渡さない警告を指示するために使用できます。データベース・マネージャーのすべての警告を無視する場合には、 IgnoreWarnings キーワードを使用してください。

sqlstate は、IgnoreWarnList と WarningList の両方に組み込まれている場合にはすべて無視されます。

それぞれの sqlstate は、大文字で指定し、単一引用符で囲み、コンマで区切る必要があります。ストリング全体が二重引用符で囲まれている必要もあります。たとえば、

```
IgnoreWarnList="'01000', '01004','01504'"
```

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 329 ページの『IgnoreWarnings CLI/ODBC 構成キーワード』
- 377 ページの『WarningList CLI/ODBC 構成キーワード』

KeepDynamic CLI/ODBC 構成キーワード

キーワードの説明:

DB2 CLI アプリケーションが KEEPDYNAMIC 機能を利用できるかどうかを指定します。

db2cli.ini キーワード構文:

KeepDynamic = **0** | **1**

デフォルト設定:

DB2 CLI アプリケーションは KEEPDYNAMIC 機能を利用できません。

同等の接続属性:

SQL_ATTR_KEEP_DYNAMIC

使用上の注意:

KeepDynamic CLI/ODBC 構成キーワードは、 DB2 UDB for z/OS および OS/390 サーバー上での CLI パッケージのバインド方法に応じて設定する必要があります。KeepDynamic は、次のように設定します。

- 0 - サーバー上の CLI パッケージが KEEPDYNAMIC NO オプションを指定してバインドされている場合
- 1 - サーバー上の CLI パッケージが KEEPDYNAMIC YES オプションを指定してバインドされている場合

KeepDynamic を使用する場合、CurrentPackageSet CLI/ODBC キーワードも設定することをお勧めします。それらのキーワードを併用する方法については、KEEPDYNAMIC サポートを有効にすることに関する資料を参照してください。

関連概念:

- 61 ページの『CLI アプリケーション用のプログラミングのヒントと提案』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 310 ページの『CurrentPackageSet CLI/ODBC 構成キーワード』

KeepStatement CLI/ODBC 構成キーワード

注: このキーワードは DB2 バージョン 8 ではサポートされず、旧バージョンへの互換性がある場合にのみ使用できます。このキーワードについては、 DB2 の旧バージョンの資料 (http://www.ibm.com/software/data/db2/library) を参照してください。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

LoadXAInterceptor CLI/ODBC 構成キーワード

キーワードの説明:

XA Interceptor をデバッグのためにロードします。

db2cli.ini キーワード構文:

LoadXAInterceptor = **0** | **1**

デフォルト設定:

XA Interceptor はロードされません。

使用上の注意:

このキーワードは、XA Interceptor をデバッグ目的で MTS にロードします。

関連概念:

- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『DB2 Universal Database における、C/C++ アプリケーション用の MTS サポートの使用可能化』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

LOBCacheSize CLI/ODBC 構成キーワード

キーワードの説明:

LOB の最大キャッシュ・サイズを指定します。

db2cli.ini キーワード構文:

LOBCacheSize = *正整数*

デフォルト設定:

LOB はキャッシュに入れられません。

使用上の注意:

このオプションは、DB2 CLI がメモリーのバッファーに入れる LOB の最大定義済みサイズを示します。 LOB の定義済みサイズが LOBCacheSize の設定値を超過する場合、LOB はキャッシュに入れられません。たとえば、CLOB 列が 100MB で作成された表が現在 20MB のデータを保持していて、 LOBCacheSize は 50MB に設定されているとします。この場合、LOB のサイズ (20MB) 自体は LOBCacheSize で設定されたサイズよりも小さいにもかかわらず、定義されている CLOB サイズ (100MB) が LOBCacheSize によって設定されてい最大キャッシュ・サイズ (50MB) より大きいため、 CLOB 列はキャッシュに入れられません。

このキーワードを設定すると、アンバインドされた LOB データを検索するときに LOB ロケーターを使用することを回避できます。たとえば、アプリケーションが `SQLFetch()` を呼び出す前に列をバインドしないで、その後に `SQLGetData()` を呼び出して LOB を取り出す場合、 LOBCacheSize が取り出される LOB 全体を含むことのできる十分な大きさに設定されていれば、 LOB は LOB ロケーターではなく LOB キャッシュから検索されます。この場合、LOB ロケーターの代わりに LOB キャッシュを使用することによって、パフォーマンスが改善されます。

関連概念:

- 109 ページの『CLI アプリケーションでのラージ・オブジェクトの使用』
- 111 ページの『CLI アプリケーションでの LOB ロケーター』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

LOBFileThreshold CLI/ODBC 構成キーワード

キーワードの説明:

`SQLPutData()` の使用時にバッファーに入れられる LOB データの最大バイト数を示します。

db2cli.ini キーワード構文:

LOBFileThreshold = *正整数*

デフォルト設定:

25 MB

使用上の注意:

このオプションは、`SQLPutData()` が呼び出されたときに、 DB2 CLI がメモリーのバッファーに入れる LOB データの最大バイト数を指定します。指定したキャッシュ・サイズが超過された場合、 LOB データがサーバーに送られる前に保持される一時ファイルがディスク上に作成されます。

関連概念:

- 109 ページの『CLI アプリケーションでのラージ・オブジェクトの使用』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『SQLPutData 関数 (CLI) - パラメーターのデータ値の引き渡し』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

LOBMaxColumnSize CLI/ODBC 構成キーワード

キーワードの説明:

LOB データ・タイプのデフォルトの COLUMN_SIZE をオーバーライドします。

db2cli.ini キーワード構文:

LOBMaxColumnSize = *ゼロよりも大きい整数*

デフォルト設定:

2 ギガバイト (DBCLOB については 1G)

次の場合にのみ適用可能:

LongDataCompat オプションが使用されている。

使用上の注意:

SQL_CLOB、SQL_BLOB、および SQL_DBCLOB SQL データ・タイプの COLUMN_SIZE 列について `SQLGetTypeInfo()` から戻される 2 ギガバイト (DBCLOB については 1G) の値をオーバーライドします。 LOB 列を含む後続の CREATE TABLE ステートメントは、ここに設定する列サイズ値をデフォルトの代わりに使用します。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 335 ページの『LongDataCompat CLI/ODBC 構成キーワード』

LockTimeout CLI/ODBC 構成キーワード

キーワードの説明:

LOCKTIMEOUT 構成パラメーターのデフォルト値を設定します。

db2cli.ini キーワード構文:

LockTimeout = **-1** | **0** | **正整数 ≤ 32767**

デフォルト設定:

タイムアウトはオフ (-1) であり、アプリケーションはロックが付与されるかデッドロックが発生するまでロックを待機します。

使用上の注意:

LockTimeout キーワードは、 DB2 CLI アプリケーションがロックの取得を待機する秒数を指定します。このキーワードが 0 に設定されている場合、ロックを待機することはありません。 -1 に設定されている場合、アプリケーションは、ロックが付与されるかデッドロックが発生するまで無期限に待機します。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『locktimeout - 「ロック・タイムアウト」構成パラメーター』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

LongDataCompat CLI/ODBC 構成キーワード

キーワードの説明:

LOB を長データ・タイプまたはラージ・オブジェクト・タイプとして報告します。

db2cli.ini キーワード構文:

LongDataCompat = **0** | **1**

デフォルト設定:

LOB データ・タイプをラージ・オブジェクト・タイプとして参照します。

同等の接続属性:

SQL_ATTR_LONGDATA_COMPAT

使用上の注意:

このオプションは、アプリケーションがラージ・オブジェクト (LOB) 列を持つデータベースで作業を行うときに予期するデータ・タイプを DB2 CLI に示します。

表 23. LOB データに対応するラージ・オブジェクトと長データ・タイプ

データベース・データ・タイプ	ラージ・オブジェクト (0 - デフォルト)	長データ・タイプ (1)
CLOB	SQL_CLOB	SQL_LONGVARCHAR
BLOB	SQL_BLOB	SQL_LONGVARBINARY
DBCLOB	SQL_DBCLOB	SQL_LONGVARGRAPHIC*

* LongDataCompat と共に MapGraphicDescribe キーワードが設定されている場合、 DBCLOB 列は、MapGraphicDescribe が 1 なら SQL_LONGVARCHAR、 MapGraphicDescribe が 2 なら SQL_WLONGVARCHAR の SQL タイプを戻します。

このオプションは、ラージ・オブジェクト・データ・タイプを扱うことができない ODBC アプリケーションを実行するときに役立ちます。

データに宣言されるデフォルト・サイズを小さくするために、 DB2 CLI/ODBC オプション LOBMAXCOLUMNSIZE をこのオプションとともに使用することができます。

関連概念:

- 109 ページの『CLI アプリケーションでのラージ・オブジェクトの使用』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 334 ページの『LOBMaxColumnSize CLI/ODBC 構成キーワード』
- 338 ページの『MapGraphicDescribe CLI/ODBC 構成キーワード』

MapDateCDefault CLI/ODBC 構成キーワード

キーワードの説明:

DATE 列およびパラメーター・マーカーのデフォルトの C タイプを指定します。

db2cli.ini キーワード構文:

MapDateCDefault = **0** | **1** | **2**

デフォルト設定:

DATE データのデフォルトの C タイプ表記は SQL_C_TYPE_DATE です。

使用上の注意:

MapDateCDefault は、 DATE 列およびパラメーター・マーカーに対して SQL_C_DEFAULT が指定されている場合に使用される C タイプを制御します。このキーワードは、主として Microsoft Access など、 datetime 値のデフォルトの C タイプが SQL_C_CHAR であることを前提としている Microsoft アプリケーションで、使用する必要があります。 MapDateCDefault は、次のように設定します。

- 0 - デフォルトの SQL_C_TYPE_DATE C タイプ表記の場合: 年、月、日のそれぞれに対する数値メンバーを含む構造体。
- 1 - SQL_C_CHAR C タイプ表記の場合: ″2004-01-01″
- 2 - SQL_C_WCHAR C タイプ表記の場合: ″2004-01-01″ (UTF-16)

このキーワードは、 `SQLBindParameter()`、`SQLBindCol()`、`SQLGetData()` など、SQL_C_DEFAULT を C タイプとして指定する CLI 関数の動作に影響を与えます。

関連概念:

- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『CLI と ODBC 関数のサマリー』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 339 ページの『MapTimeCDefault CLI/ODBC 構成キーワード』
- 341 ページの『MapTimestampCDefault CLI/ODBC 構成キーワード』

MapDateDescribe CLI/ODBC 構成キーワード

キーワードの説明:

DATE 列およびパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを指定します。

db2cli.ini キーワード構文:

MapDateDescribe = **0** | **1** | **2**

デフォルト設定:

DATE データのデフォルト SQL データ・タイプが戻されます (ODBC 2.0 の場合は SQL_DATE、 ODBC 3.0 の場合は SQL_TYPE_DATE)。

使用上の注意:

DATE 列およびパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを指定するには、 MapDateDescribe を次のように設定します。

- 0 - デフォルトの SQL データ・タイプを戻す場合 (ODBC 2.0 の場合は SQL_DATE、 ODBC 3.0 の場合は SQL_TYPE_DATE)
- 1 - SQL_CHAR SQL データ・タイプを戻す場合
- 2 - SQL_WCHAR SQL データ・タイプを戻す場合

MapDateDescribe の設定値は、次の DB2 CLI 関数にのみ影響します。

- `SQLDescribeCol()`
- `SQLDescribeParam()`
- `SQLGetDescField()`
- `SQLGetDescRec()`

このキーワードは、DB2 CLI カタログ関数には影響しません。

関連概念:

- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeParam 関数 (CLI) - パラメーター・マーカーの記述を戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 340 ページの『MapTimeDescribe CLI/ODBC 構成キーワード』
- 341 ページの『MapTimestampDescribe CLI/ODBC 構成キーワード』
- 338 ページの『MapGraphicDescribe CLI/ODBC 構成キーワード』

MapGraphicDescribe CLI/ODBC 構成キーワード

キーワードの説明:

GRAPHIC、VARGRAPHIC、および LONGVARGRAPHIC 列およびパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを指定します。

db2cli.ini キーワード構文:

MapGraphicDescribe = **0** | **1** | **2**

デフォルト設定:

デフォルトの SQL データ・タイプが戻されます。つまり、GRAPHIC 列では SQL_GRAPHIC、VARGRAPHIC 列では SQL_VARGRAPHIC、 LONG VARGRAPHIC 列では SQL_LONGVARGRAPHIC です。

使用上の注意:

GRAPHIC ベースの列およびパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを指定するには、 MapGraphicDescribe を次のように設定します。

- 0 - デフォルトの SQL データ・タイプを戻す場合
- 1 - CHAR ベースの SQL データ・タイプを戻す場合 (GRAPHIC 列では SQL_CHAR、 VARGRAPHIC 列では SQL_VARCHAR、および LONG VARGRAPHIC 列では SQL_LONGVARCHAR)
- 2 - WCHAR ベースの SQL データ・タイプを戻す場合 (GRAPHIC 列では SQL_WCHAR、 VARGRAPHIC 列では SQL_WVARCHAR、および LONG VARGRAPHIC 列では SQL_WLONGVARCHAR)

MapGraphicDescribe の設定値は、次の DB2 CLI 関数にのみ影響します。

- `SQLDescribeCol()`
- `SQLDescribeParam()`
- `SQLGetDescField()`
- `SQLGetDescRec()`

このキーワードは、DB2 CLI カタログ関数には影響しません。

関連概念:

- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeParam 関数 (CLI) - パラメーター・マーカーの記述を戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 340 ページの『MapTimeDescribe CLI/ODBC 構成キーワード』
- 341 ページの『MapTimestampDescribe CLI/ODBC 構成キーワード』
- 337 ページの『MapDateDescribe CLI/ODBC 構成キーワード』

MapTimeCDefault CLI/ODBC 構成キーワード

キーワードの説明:

TIME 列およびパラメーター・マーカーのデフォルトの C タイプを指定します。

db2cli.ini キーワード構文:

MapTimeCDefault = **0** | **1** | **2**

デフォルト設定:

TIME データのデフォルトの C タイプ表記は SQL_C_TYPE_TIME です。

使用上の注意:

MapTimeCDefault は、 TIME 列およびパラメーター・マーカーに対して SQL_C_DEFAULT が指定されている場合に使用される C タイプを制御します。このキーワードは、主として Microsoft Access など、 datetime 値のデフォルトの C タイプが SQL_C_CHAR であることを前提としている Microsoft アプリケーションで、使用する必要があります。 MapTimeCDefault は、次のように設定します。

- 0 - デフォルトの SQL_C_TYPE_TIME C タイプ表記の場合: 時、分、秒のそれぞれに対する数値メンバーを含む構造体。
- 1 - SQL_C_CHAR C タイプ表記の場合: ″12:34:56″
- 2 - SQL_C_WCHAR C タイプ表記の場合: ″12:34:56″ (UTF-16)

このキーワードは、 `SQLBindParameter()`、`SQLBindCol()`、`SQLGetData()` など、SQL_C_DEFAULT を C タイプとして指定する CLI 関数の動作に影響を与えます。

注: MapTimeCDefault は Patch2=24 を置き換えます。 MapTimeCDefault と Patch2=24 の両方が設定されている場合には、 MapTimeCDefault の値が優先されます。

関連概念:

- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『CLI と ODBC 関数のサマリー』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 345 ページの『Patch2 CLI/ODBC 構成キーワード』
- 336 ページの『MapDateCDefault CLI/ODBC 構成キーワード』
- 341 ページの『MapTimestampCDefault CLI/ODBC 構成キーワード』

MapTimeDescribe CLI/ODBC 構成キーワード

キーワードの説明:

TIME 列およびパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを指定します。

db2cli.ini キーワード構文:

MapTimeDescribe = **0** | **1** | **2**

デフォルト設定:

TIME データのデフォルト SQL データ・タイプが戻されます (ODBC 2.0 の場合は SQL_TIME、 ODBC 3.0 の場合は SQL_TYPE_TIME)。

使用上の注意:

TIME 列およびパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを指定するには、 MapTimeDescribe を次のように設定します。

- 0 - デフォルトの SQL データ・タイプを戻す場合 (ODBC 2.0 の場合は SQL_TIME、 ODBC 3.0 の場合は SQL_TYPE_TIME)
- 1 - SQL_CHAR SQL データ・タイプを戻す場合
- 2 - SQL_WCHAR SQL データ・タイプを戻す場合

MapTimeStampDescribe の設定値は、次の DB2 CLI 関数にのみ影響します。

- `SQLDescribeCol()`
- `SQLDescribeParam()`
- `SQLGetDescField()`
- `SQLGetDescRec()`

このキーワードは、DB2 CLI カタログ関数には影響しません。

関連概念:

- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeParam 関数 (CLI) - パラメーター・マーカーの記述を戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 341 ページの『MapTimestampDescribe CLI/ODBC 構成キーワード』
- 337 ページの『MapDateDescribe CLI/ODBC 構成キーワード』
- 338 ページの『MapGraphicDescribe CLI/ODBC 構成キーワード』

MapTimestampCDefault CLI/ODBC 構成キーワード

キーワードの説明:

TIMESTAMP 列およびパラメーター・マーカーのデフォルトの C タイプを指定します。

db2cli.ini キーワード構文:

MapTimestampCDefault = **0** | **1** | **2**

デフォルト設定:

TIMESTAMP データのデフォルトの C タイプ表記は SQL_C_TYPE_TIMESTAMP です。

使用上の注意:

MapTimestampCDefault は、 TIMESTAMP 列およびパラメーター・マーカーに対して SQL_C_DEFAULT が指定されている場合に使用される C タイプを制御します。このキーワードは、主として Microsoft Access など、 datetime 値のデフォルトの C タイプが SQL_C_CHAR であることを前提としている Microsoft アプリケーションで、使用する必要があります。 MapTimestampCDefault は、次のように設定します。

- 0 - デフォルトの SQL_C_TYPE_TIMESTAMP C タイプ表記の場合: 年、月、日、時、分、秒、および秒の小数部分のそれぞれに対する数値メンバーを含む構造体。
- 1 - SQL_C_CHAR C タイプ表記の場合: ″2004-01-01 12:34:56.123456″
- 2 - SQL_C_WCHAR C タイプ表記の場合: ″2004-01-01 12:34:56.123456″ (UTF-16)

このキーワードは、 `SQLBindParameter()`、`SQLBindCol()`、`SQLGetData()` など、SQL_C_DEFAULT を C タイプとして指定する CLI 関数の動作に影響を与えます。

関連概念:

- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『CLI と ODBC 関数のサマリー』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 339 ページの『MapTimeCDefault CLI/ODBC 構成キーワード』
- 336 ページの『MapDateCDefault CLI/ODBC 構成キーワード』

MapTimestampDescribe CLI/ODBC 構成キーワード

キーワードの説明:

TIMESTAMP 列およびパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを指定します。

db2cli.ini キーワード構文:

MapTimestampDescribe = **0** | **1** | **2**

デフォルト設定:

TIMESTAMP データのデフォルト SQL データ・タイプが戻されます (ODBC 2.0 の場合は SQL_TIMESTAMP、 ODBC 3.0 の場合は SQL_TYPE_TIMESTAMP)。

使用上の注意:

TIMESTAMP 列およびパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを指定するには、 MapTimestampDescribe を次のように設定します。

- 0 - デフォルトの SQL データ・タイプを戻す場合 (ODBC 2.0 の場合は SQL_TIMESTAMP、 ODBC 3.0 の場合は SQL_TYPE_TIMESTAMP)
- 1 - SQL_CHAR SQL データ・タイプを戻す場合
- 2 - SQL_WCHAR SQL データ・タイプを戻す場合

MapTimeStampDescribe の設定値は、次の DB2 CLI 関数にのみ影響します。

- `SQLDescribeCol()`
- `SQLDescribeParam()`
- `SQLGetDescField()`
- `SQLGetDescRec()`

このキーワードは、DB2 CLI カタログ関数には影響しません。

関連概念:

- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLDescribeParam 関数 (CLI) - パラメーター・マーカーの記述を戻す』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLGetDescRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 340 ページの『MapTimeDescribe CLI/ODBC 構成キーワード』
- 337 ページの『MapDateDescribe CLI/ODBC 構成キーワード』
- 338 ページの『MapGraphicDescribe CLI/ODBC 構成キーワード』

Mode CLI/ODBC 構成キーワード

キーワードの説明:

デフォルトの接続モード。

db2cli.ini キーワード構文:

Mode = **SHARE** | **EXCLUSIVE**

デフォルト設定:

SHARE

次の場合には適用不可:

ホストまたは iSeries サーバーに接続している。

使用上の注意:

CONNECT モードを SHARE または EXCLUSIVE のいずれかに設定します。モードがアプリケーションによって接続時に設定される場合、この値は無視されます。デフォルトは SHARE です。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 1) ステートメント』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

OleDbReturnCharAsWChar CLI/ODBC 構成キーワード

キーワードの説明:

IBM DB2 OLE DB Provider が CHAR、VARCHAR、LONG VARCHAR、および CLOB データを記述する方法を指定します。

db2cli.ini キーワード構文:

OleDbReturnCharAsWChar = **0** | **1**

デフォルト設定:

IBM DB2 OLE DB Provider は、CHAR、VARCHAR、LONG VARCHAR、および CLOB データを DBTYPE_WSTR と記述します。

使用上の注意:

IBM DB2 OLE DB Provider はデフォルトで、CHAR、VARCHAR、LONG VARCHAR、および CLOB データを DB2 UDB バージョン 8.1.2 の DBTYPE_WSTR として記述します。 CLI/ODBC 構成キーワード OleDbReturnCharAsWChar を使用すると、このデフォルトを変更して、これらの文字データ・タイプを DBTYPE_STR として報告できるようになります。

可能な設定値は次のとおりです。

- 0 - CHAR、VARCHAR、LONG VARCHAR、および CLOB データは DBTYPE_STR として記述され、 ISequentialStream 内のデータのコード・ページはクライアントのローカル・コード・ページになります。
- 1 - CHAR、VARCHAR、LONG VARCHAR、および CLOB データは DBTYPE_WSTR として報告され、 ISequentialStream 内のデータのコード・ページは UCS-2 になります。

関連概念:

- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『IBM OLE DB Provider for DB2 の目的』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『DB2 と OLE DB の間のデータ・タイプ・マッピング』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

OptimizeForNRows CLI/ODBC 構成キーワード

キーワードの説明:

'OPTIMIZE FOR n ROWS' 文節をすべての SELECT ステートメントに付加します。

db2cli.ini キーワード構文:

OptimizeForNRows = *整数*

デフォルト設定:

文節は付加されません。

同等のステートメント属性:

SQL_ATTR_OPTIMIZE_FOR_NROWS

使用上の注意:

このオプションは、"OPTIMIZE FOR n ROWS" 文節をすべての SELECT ステートメントに付加します。 n は 0 よりも大きな整数です。 0 (デフォルト) に設定された場合、この文節は付加されません。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『SELECT ステートメント』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

Patch1 CLI/ODBC 構成キーワード

キーワードの説明:

認識されている CLI/ODBC アプリケーション問題について、予備手段を使用します。

db2cli.ini キーワード構文:

Patch1 = **{ 0 | 1 | 2 | 4 | 8 | 16 | ... }**

デフォルト設定:

予備手段を使用しません。

使用上の注意:

このキーワードは、 ODBC アプリケーションの認識されている問題について予備手段を指定するために使用されます。値は、予備手段なしを指定するか、1 つまたは複数の予備手段について指定できます。ここに指定されたパッチ値は、一緒に設定できる Patch2 値とともに使用されます。

「DB2 CLI/ODBC 設定」ノートブックを使用して、使用する 1 つ以上のパッチを選択できます。 `db2cli.ini` ファイル自体に値を設定して複数のパッチ値を使用する場合は、単純に値を足してキーワード値を作成します。たとえば、パッチ 1、4、および 8 を使用する場合は、Patch1=13 と指定してください。

- 0 = 予備手段なし (デフォルト)

Patch1 値の現行のリストについては、以下の DB2 アプリケーション開発 Web サイトを参照してください。

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

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 345 ページの『Patch2 CLI/ODBC 構成キーワード』

Patch2 CLI/ODBC 構成キーワード

キーワードの説明:

認識されている CLI/ODBC アプリケーション問題について、予備手段を使用します。

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, ..."*

デフォルト設定:

予備手段を使用しません

使用上の注意:

このキーワードは、CLI/ODBC アプリケーションの認識されている問題について予備手段を指定するために使用されます。値は、予備手段なしを指定するか、1 つまたは複数の予備手段について指定できます。ここに指定されたパッチ値は、一緒に設定できる Patch1 値とともに使用される場合があります。

複数のパッチを指定するときには、コンマで区切られたストリングとして値を指定します (値が加算されて合計が使用される Patch1 オプションとは異なります)。

- 0 = 予備手段なし (デフォルト)

Patch2 値 3、4、および 8 を設定するには、次を指定してください。

```
Patch2="3, 4, 8"
```

Patch2 値の現行のリストについては、以下の DB2 アプリケーション開発 Web サイトを参照してください。

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

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 345 ページの『Patch1 CLI/ODBC 構成キーワード』

Port CLI/ODBC 構成キーワード

キーワードの説明:

ファイル DSN とともにまたは接続頻度の低い DSN で使用される、サーバー・システムのサービス名およびポート番号。

db2cli.ini キーワード構文:

Port = *サービス名* | *ポート番号*

デフォルト設定:

なし

次の場合にのみ適用可能:

プロトコルが TCPIP に設定されている。

使用上の注意:

このオプションは、 DB2 を実行しているサーバーへのこのクライアント・マシンからの TCP/IP 接続に必要な属性を指定するために、 Hostname オプションとともに使用します。これらの 2 つの値は、Protocol オプションが TCPIP に設定されているときにのみ使用されます。

サーバー・システムのサービス名またはポート番号のどちらかを指定します。サービス名は、使用時にクライアント・マシンでの検索に使用可能である必要があります。

関連概念:

• 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

• 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
• 329 ページの『Hostname CLI/ODBC 構成キーワード』
• 348 ページの『Protocol CLI/ODBC 構成キーワード』
• 353 ページの『ServiceName CLI/ODBC 構成キーワード』

ProgramName CLI/ODBC 構成キーワード

キーワードの説明:

モニター時にサーバーでアプリケーションを識別するために使用されるクライアント・アプリケーションのデフォルト名を、ユーザー定義の名前に変更します。

db2cli.ini キーワード構文:

ProgramName = **<ストリング>** | **PID**

デフォルト設定:

ユーザー定義名は使用されません。デフォルトでは、アプリケーションは DB2 の割り当てる名前によって識別されます。

同等の接続属性:

SQL_ATTR_INFO_PROGRAMNAME

使用上の注意:

CLI アプリケーションをモニターする場合、 DB2 が割り当てるデフォルトの識別子ではなく、アプリケーションをユーザー定義のストリングによって識別することが役立つ場合があります。 ProgramName を使用することによりユーザーは、20 バイト以下のストリング、またはストリング ″PID″ (引用符は含まない) のいずれかとして識別子を指定できます。

CLI アプリケーションに対して ProgramName が ″PID″ に設定されている場合、アプリケーションの名前は、接頭部 ″CLI″ の後、アプリケーションのプロセス ID、および現行のアクティブ接続ハンドルで構成されます。つまり、CLI<pid>:<connectionHandle#> という形式です。 ″PID″ の設定値は、同じデータベースとの数多くの接続を使用して複数のアプリケーションを実行するアプリケーション・サーバーをモニターする場合に便利です。

(他の種類のアプリケーションに対して ProgramName キーワードが ″PID″ に設定されている場合、接頭部 ″CLI″ はそれぞれのアプリケーションの種類に対応する値になります。すなわち、 JDBC アプリケーションの場合は ″JDBC″、 OLE DB アプリケーションの場合は ″OLEDB″、および .NET アプリケーションの場合は ″ADONET″。)

関連概念:

• 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻」の『接続属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

Protocol CLI/ODBC 構成キーワード

キーワードの説明:

ファイル DSN に対して、または接続頻度の低い DSN で使用される通信プロトコル。

db2cli.ini キーワード構文:

Protocol = **TCPIP**

デフォルト設定:

なし

使用上の注意:

TCP/IP はファイル DSN を使用するときにサポートされる唯一のプロトコルです。オプションをストリング TCPIP (スラッシュなし) に設定してください。

このオプションを設定するときには、以下のオプションも設定しなければなりません。

- Database
- ServiceName
- Hostname

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 315 ページの『Database CLI/ODBC 構成キーワード』
- 329 ページの『Hostname CLI/ODBC 構成キーワード』
- 353 ページの『ServiceName CLI/ODBC 構成キーワード』

PWD CLI/ODBC 構成キーワード

キーワードの説明:

デフォルト・パスワードを定義します。

db2cli.ini キーワード構文:

PWD = *パスワード*

デフォルト設定:

なし

使用上の注意:

このパスワード 値は、パスワードが接続時にアプリケーションによって提供されない場合に使用されます。

これは、プレーン・テキストとして db2cli.ini ファイルに保管されるため、保護されません。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 374 ページの『UID CLI/ODBC 構成キーワード』

QueryTimeoutInterval CLI/ODBC 構成キーワード

キーワードの説明:

照会タイムアウトのチェック間の遅延 (秒単位)。

db2cli.ini キーワード構文:

QueryTimeoutInterval = **0** | **5** | **正の整数**

デフォルト設定:

5 秒

使用上の注意:

アプリケーションは `SQLSetStmtAttr()` 関数を使用することにより、SQL_ATTR_QUERY_TIMEOUT ステートメント属性を設定できます。この属性により、実行の取り消しが試行されてアプリケーションに戻るまでの、 SQL ステートメントの実行完了待ちの秒数が示されます。

QueryTimeoutInterval 構成キーワードは、照会が完了したかどうかのチェックの間の CLI ドライバーの待ち時間を指示するために使用されます。

たとえば、SQL_ATTR_QUERY_TIMEOUT を 25 秒 (25 秒待った後にタイムアウト) に設定して、 QueryTimeoutInterval を 10 秒 (照会を 10 秒ごとにチェック) に設定するとします。照会は 30 秒後 (25 秒の制限の後の最初のチェック) にならないとタイムアウトになりません。

注: DB2 CLI は、実行中の照会ごとの状況を定期的に照会するスレッドを開始して、照会タイムアウトをインプリメントします。 QueryTimeoutInterval 値は、有効期限切れの照会のチェック間に、照会タイムアウト・スレッドが待機する時間の長さを指定します。これは、実行中の照会に対する非同期操作であるために、 SQL_ATTR_QUERY_TIMEOUT + QueryTimeoutInterval 秒までに、指定された照会がタイムアウトにならない可能性もあります。上記の例で、最良の場合のタイムアウトは 26 秒であり、悪い場合のタイムアウトは 35 秒です。

SQL_ATTR_QUERY_TIMEOUT が小さすぎる値に設定されていて、照会がタイムアウトになってはならない場合があるかもしれません。アプリケーションが変更できない場合 (サード・パーティーの ODBC アプリケーション)、 QueryTimeoutInterval

は 0 に設定可能で、 CLI ドライバーは SQL_ATTR_QUERY_TIMEOUT 設定を無視するので、アプリケーションに戻るまでに SQL ステートメントが実行を完了するのを待機します。

注: QueryTimeoutInterval が 0 に設定されている場合、アプリケーションが SQL_ATTR_QUERY_TIMEOUT を設定しようとすると、 SQLSTATE 01S02 (オプション値が変更された) になります。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

一方、QueryTimeoutInterval を SQL_ATTR_QUERY_TIMEOUT の設定よりも大きい値に設定し、指定したインターバルでタイムアウトが発生しないようにすることができます。たとえば、アプリケーションが SQL_ATTR_QUERY_TIMEOUT 値を 15 秒に設定している場合でも、サーバーは、照会の実行に少なくとも 30 秒を必要とし、 QueryTimeoutInterval は値 30 に設定可能で、この照会が 15 秒後にタイムアウトにならないようにします。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

ReportRetryErrorsAsWarnings CLI/ODBC 構成キーワード

キーワードの説明:

DB2 CLI エラー・リカバリー中に警告として発覚したエラーを戻します。

db2cli.ini キーワード構文:

ReportRetryErrorsAsWarnings = **0** | **1**

次の場合にのみ適用可能:

RetryOnError キーワードが 1 に設定されている場合。

デフォルト設定:

DB2 CLI エラー・リカバリー中に発覚したエラーをアプリケーションに戻しません。

使用上の注意:

デフォルトでは、DB2 CLI 再試行ロジックは、致命的ではないエラーから正常にリカバリーできる場合に、 SQL_SUCCESS を戻してそのエラーをアプリケーションから隠します。このようにしてアプリケーション・バインディング・エラーを隠すことができるため、デバッグの場合は、ReportRetryErrorsAsWarnings を 1 に設定する

ことができます。この設定はエラー・リカバリーをオンに維持しますが、警告として発覚したエラーを強制的に DB2 CLI がすべてアプリケーションに戻すようにします。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 56 ページの『CLI 関数戻りコード』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 351 ページの『RetryOnError CLI/ODBC 構成キーワード』

ReportPublicPrivileges CLI/ODBC 構成キーワード

キーワードの説明:

`SQLColumnPrivileges()` および `SQLTablePrivileges()` 結果で PUBLIC 特権について報告します。

db2cli.ini キーワード構文:

ReportPublicPrivileges = **0** | **1**

デフォルト設定:

PUBLIC 特権は報告されません。

使用上の注意:

このキーワードは、`SQLColumnPrivileges()` および `SQLTablePrivileges()` 結果で、 PUBLIC グループに割り当てられた特権を PUBLIC がユーザーであったかのように報告するかどうかを指定します。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLColumnPrivileges 関数 (CLI) - 表の列に関連した特権の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLTablePrivileges 関数 (CLI) - 表に関連する特権の取得』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

RetryOnError CLI/ODBC 構成キーワード

キーワードの説明:

DB2 CLI ドライバーのエラー・リカバリー動作をオン/オフします。

db2cli.ini キーワード構文:

RetryOnError = **0** | **1**

デフォルト設定:

DB2 CLI ドライバーが致命的ではないエラーのエラー・リカバリーを試行することを許可します。

使用上の注意:

デフォルトでは、DB2 CLI は、アプリケーション・パラメーターの不正なバインドなどの致命的ではないエラーからのリカバリーを、失敗した SQL ステートメントに関する追加情報を検索してからそのステートメントを再び実行することによって行います。検索される追加情報には、データベース・カタログ表からの入力パラメーター情報が含まれます。 DB2 CLI がエラーから正常にリカバリーできる場合は、デフォルトでは、アプリケーションへのエラー報告はされません。 CLI/ODBC 構成キーワード ReportRetryErrorsAsWarnings を使用することにより、エラー・リカバリー警告がアプリケーションに戻されるようにするかどうかを設定できます。

重要: いったん DB2 CLI が正常にエラー・リカバリーを完了すると、アプリケーションが通常とは異なる動作をする場合がありますが、それは DB2 CLI が元の `SQLBindParameter()` 関数呼び出しで提供された情報ではなく、それに続けて実行した SQL ステートメントに対するリカバリー中に収集されたカタログ情報を使用するためです。このような動作が望ましくない場合は、RetryOnError を 0 に設定し、強制的に DB2 CLI がリカバリーを試行しないようにします。しかし、ステートメント・パラメーターを正しくバインドするように、アプリケーションを変更する必要があります。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連タスク:

- 33 ページの『CLI アプリケーションでのパラメーター・マーカーのバインディング』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 350 ページの『ReportRetryErrorsAsWarnings CLI/ODBC 構成キーワード』

SchemaList CLI/ODBC 構成キーワード

キーワードの説明:

表情報の照会に使用されるスキーマを制限します。

db2cli.ini キーワード構文:

SchemaList = " '*schema1*', '*schema2*',... '*schemaN*' "

デフォルト設定:

なし

使用上の注意:

SchemaList は、DBMS 内のすべての表をリストするアプリケーションに、より制限されたデフォルトを提供して、そのアプリケーションのパフォーマンスを向上させるために使用されます。

データベースに定義されている表の数が多い場合に、スキーマ・リストを指定して、アプリケーションが表情報を照会する時間を削減し、アプリケーションでリストされる表の数を削減することができます。各スキーマ名は、単一引用符で囲んでコンマで区切る必要があり、大文字小文字が区別されます。ストリング全体が二重引用符で囲まれている必要もあります。以下に例を示します。

```
SchemaList="'USER1','USER2','USER3'"
```

DB2 for z/OS 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、

```
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 とともに使用して、情報が戻される表の数をさらに制限することができます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 320 ページの『DBName CLI/ODBC 構成キーワード』
- 359 ページの『TableType CLI/ODBC 構成キーワード』

ServiceName CLI/ODBC 構成キーワード

キーワードの説明:

ファイル DSN 接続または DSN なしの接続で使用される、サーバー・システムのサービス名およびポート番号。

db2cli.ini キーワード構文:

ServiceName = *サービス名* | *ポート番号*

デフォルト設定:

なし

次の場合にのみ適用可能:

プロトコルが TCPIP に設定されている。

使用上の注意:

このオプションは、 DB2 を実行しているサーバーへのこのクライアント・マシンからの TCP/IP 接続に必要な属性を指定するために、 Hostname オプションとともに使用します。これらの 2 つの値は、PROTOCOL オプションが TCPIP に設定されているときにのみ使用されます。

サーバー・システムのサービス名またはポート番号のどちらかを指定します。サービス名は、使用時にクライアント・マシンでの検索に使用可能である必要があります。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 329 ページの『Hostname CLI/ODBC 構成キーワード』
- 348 ページの『Protocol CLI/ODBC 構成キーワード』

SkipTrace CLI/ODBC 構成キーワード

キーワードの説明:

CLI トレース情報を DB2 トレースから除外します。

db2cli.ini キーワード構文:

SkipTrace = **0** | **1**

デフォルト設定:

トレース関数をスキップしません。

使用上の注意:

このキーワードにより、DB2 トレース関数が CLI アプリケーションをバイパスし、パフォーマンスが向上します。したがって、DB2 トレース機能 db2trc がオンになっていて、このキーワードが 1 に設定されると、CLI アプリケーションの実行の情報はトレースに含まれません。

トレース情報を必要としない UNIX プラットフォームの実稼働環境では、SkipTrace をオンにすることをお勧めします。ただし、テスト環境ではトレース出力が役に立つこともあるので、詳細な実行情報を得たい場合は、このキーワードをオフにする (またはデフォルト設定のままにする) ことができます。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「コマンド・リファレンス」の『db2trc - トレース・コマンド』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

SQLOverrideFileName CLI/ODBC 構成キーワード

キーワードの説明:

特定の SQL ステートメントに関する CLI ステートメント属性の設定値のリストを含む、オーバーライド・ファイルの場所を指定します。

db2cli.ini キーワード構文:

SQLOverrideFileName = **<絶対または相対パス名>**

デフォルト設定:

オーバーライド・ファイルは使用されません。

使用上の注意:

SQLOverrideFileName キーワードは、 DB2 CLI ドライバーの読むオーバーライド・ファイルの場所を指定します。オーバーライド・ファイルには、特定の SQL ステートメントに適用される CLI ステートメント属性の値が含まれています。サポートされているステートメント属性は、どれでも指定できます。以下に、 2 つの SQL ステートメントに固有の属性設定値を含むオーバーライド・ファイルの例を示します。

```
[Common]
Stmts=2

[1]
StmtIn=SELECT * FROM Employee
StmtAttr=SQL_ATTR_BLOCK_FOR_NROWS=50;SQL_ATTR_OPTIMIZE_FOR_NROWS=1;

[2]
StmtIn=SELECT * FROM Sales
StmtAttr=SQL_ATTR_MAX_ROWS=25;
```

オーバーライド・ファイルの ″[Common]″ セクションの ″Stmts″ で指定されている数値は、そのオーバーライド・ファイルに含まれる SQL ステートメントの数と同じです。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

StaticCapFile CLI/ODBC 構成キーワード

キーワードの説明:

キャプチャー・ファイル名を指定して、ファイルが保管されるパスを任意で指定します。

db2cli.ini キーワード構文:

StaticCapFile = < **完全ファイル名** >

デフォルト設定:

なし - キャプチャー・ファイルを指定してください。

次の場合にのみ適用可能:

StaticMode が Capture または Match に設定されている

使用上の注意:

このキーワードは、キャプチャー・ファイル名を指定して、ファイルが保管されるディレクトリーを任意で指定するために使用されます。

関連概念:

- 208 ページの『CLI/ODBC/JDBC 静的プロファイル作成のためのキャプチャー・ファイル』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連タスク:

- 205 ページの『CLI/ODBC/JDBC 静的プロファイル作成による静的 SQL の作成』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 356 ページの『StaticLogFile CLI/ODBC 構成キーワード』
- 357 ページの『StaticMode CLI/ODBC 構成キーワード』
- 357 ページの『StaticPackage CLI/ODBC 構成キーワード』

StaticLogFile CLI/ODBC 構成キーワード

キーワードの説明:

静的プロファイル・ログ・ファイル名を指定して、ファイルが保管されるディレクトリーを任意で指定します。

db2cli.ini キーワード構文:

StaticLogFile = < **完全ファイル名** >

デフォルト設定:

静的プロファイル・ログは作成されません。ファイル名をパス名なしで指定する場合は、現行パスが使用されます。

次の場合にのみ適用可能:

StaticMode が Capture または Match に設定されている

使用上の注意:

このキーワードは、静的プロファイル・ログ・ファイル名を指定して、ファイルが保管されるディレクトリーを任意で指定するために使用されます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連タスク:

- 205 ページの『CLI/ODBC/JDBC 静的プロファイル作成による静的 SQL の作成』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 355 ページの『StaticCapFile CLI/ODBC 構成キーワード』
- 357 ページの『StaticMode CLI/ODBC 構成キーワード』

• 357 ページの『StaticPackage CLI/ODBC 構成キーワード』

StaticMode CLI/ODBC 構成キーワード

キーワードの説明:

この DSN のために CLI/ODBC アプリケーションが SQL をキャプチャーするか静的 SQL パッケージを使用するかを選択します。

db2cli.ini キーワード構文:

StaticMode = **DISABLED** | **CAPTURE** | **MATCH**

デフォルト設定:

使用不可 - SQL ステートメントのキャプチャーは行われず、静的 SQL パッケージは使用されません。

使用上の注意:

このオプションを使用すると、 CLI/ODBC アプリケーションがこの DSN について発行した SQL が処理される方法を以下のように指定することができます。

- DISABLED = 静的モードは使用不可です。特定の処理を行いません。 CLI/ODBC ステートメントは、変更のない動的 SQL として実行されます。これはデフォルトです。
- CAPTURE = キャプチャー・モード。 CLI/ODBC ステートメントを動的 SQL として実行します。 SQL ステートメントが正常に終了した場合、SQL ステートメントは、後で DB2CAP コマンドによってバインドされるためにファイル (キャプチャー・ファイルとして知られる) にキャプチャーされます。
- MATCH = 一致モード。 CLI/ODBC ステートメントを、StaticPackage に指定されたキャプチャー・ファイルに一致ステートメントが見つかった場合に、静的 SQL ステートメントとして実行します。キャプチャー・ファイルは、最初に DB2CAP コマンドでバインドされる必要があります。

関連概念:

• 289 ページの『db2cli.ini 初期設定ファイル』

関連タスク:

• 205 ページの『CLI/ODBC/JDBC 静的プロファイル作成による静的 SQL の作成』

関連資料:

• 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
• 355 ページの『StaticCapFile CLI/ODBC 構成キーワード』
• 356 ページの『StaticLogFile CLI/ODBC 構成キーワード』
• 357 ページの『StaticPackage CLI/ODBC 構成キーワード』

StaticPackage CLI/ODBC 構成キーワード

キーワードの説明:

静的プロファイル機能で使用されるパッケージを指定します。

db2cli.ini キーワード構文:

StaticPackage = *collection_id.package_name*

デフォルト設定:

なし - パッケージ名を指定してください。

次の場合にのみ適用可能:

STATICMODE が CAPTURE に設定されている

使用上の注意:

このキーワードは、アプリケーションが一致モードで動作する時に使用されるパッケージを指定するために使用されます。キャプチャー・ファイルを作成するには、最初にキャプチャー・モードを使用する必要があります。

示されたパッケージ名の最初の 7 文字のみが使用されます。 1 バイトの接尾部が、それぞれの分離レベルを表すために以下のように追加されます。

- 0 - 非コミット読み取り (UR)
- 1 - カーソル固定 (CS)
- 2 - 読み取り固定 (RS)
- 3 - 反復可能読み取り (RR)
- 4 - コミットなし (NC)

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『分離レベル』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連タスク:

- 205 ページの『CLI/ODBC/JDBC 静的プロファイル作成による静的 SQL の作成』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 355 ページの『StaticCapFile CLI/ODBC 構成キーワード』
- 356 ページの『StaticLogFile CLI/ODBC 構成キーワード』
- 357 ページの『StaticMode CLI/ODBC 構成キーワード』

StreamPutData CLI/ODBC 構成キーワード

キーワードの説明:

1 つのステートメント・ハンドルで `SQLPutData()` 関数を通して渡されるデータのパフォーマンスを向上させます。そのために、内部の接続レベル通信バッファーにデータを直接書き込みます。

db2cli.ini キーワード構文:

StreamPutData = **0** | **1**

デフォルト設定:

接続レベル・バッファーにデータを直接書き込むのではなく、デフォルトのステートメント・レベル・バッファーに書き込みます。

使用上の注意:

デフォルトでは DB2 CLI は、 `SQLPutData()` 関数呼び出しを介して渡されたデータを内部のステートメント・レベル・バッファーに書き込みます。その後の `SQLParamData()` 呼び出しによって、そのバッファーの内容が内部の接続レベル通信バッファーに書き込まれて、サーバーに送られます。特定の接続上のターゲット・データベースに所定のポイント・イン・タイムにデータを挿入するのに 1 つのステートメント・ハンドルだけを使用する場合、 StreamPutData=1 と設定すれば、パフォーマンスを向上することができます。この場合、DB2 CLI は挿入データを接続レベル・バッファーに直接書き込むことになります。ただし、特定の接続上のターゲット・データベースに対して複数のステートメントが並行してデータを挿入する場合に StreamPutData=1 と設定すると、パフォーマンスが低下する可能性があり、そのために不測のアプリケーション・エラーが生じることがあります。それは、共用の接続レベル通信バッファー内のステートメントは、シリアライゼーションしやすくなるからです。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLParamData 関数 (CLI) - データ値が必要な次のパラメーターの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLPutData 関数 (CLI) - パラメーターのデータ値の引き渡し』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

SyncPoint CLI/ODBC 構成キーワード

注: このキーワードは DB2 バージョン 8 ではサポートされず、旧バージョンへの互換性がある場合にのみ使用できます。このキーワードについては、 DB2 の旧バージョンの資料 (http://www.ibm.com/software/data/db2/library) を参照してください。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

TableType CLI/ODBC 構成キーワード

キーワードの説明:

表情報の照会時に戻される TABLETYPES のデフォルト・リストを定義します。

db2cli.ini キーワード構文:

TableType = " '*TABLE*' | ,'*ALIAS*' | ,'*VIEW*' | , '*INOPERATIVE VIEW*' | , '*SYSTEM TABLE*' | ,'*SYNONYM*' "

デフォルト設定:

TABLETYPES のデフォルト・リストは定義されません。

使用上の注意:

データベースに定義されている表の数が多い場合に、表タイプ・ストリングを指定して、アプリケーションが表情報を照会する時間を削減し、アプリケーションでリストされる表の数を削減することができます。

値の数の制限はありません。それぞれのタイプを大文字で指定し、単一引用符で囲み、コンマで区切ってください。ストリング全体が二重引用符で囲まれている必要もあります。以下に例を示します。

```
TableType="'TABLE','VIEW'"
```

このオプションを DBNAME および SCHEMALIST とともに使用して、情報が戻される表の数をさらに制限することができます。

TableType は、データベース内の表、ビュー、別名、および同義語を検索する DB2 CLI 関数にデフォルトを提供するために使用します。アプリケーションが関数呼び出しで表タイプを指定せず、このキーワードが使用されなかった場合は、すべての表タイプの情報が戻されます。アプリケーションが関数呼び出しで表タイプ の値を提供した場合は、引き数値がこのキーワード値をオーバーライドします。

TableType に TABLE 以外の値が組み込まれている場合は、 DBName キーワード設定で特定の DB2 for z/OS データベースに情報を制限することはできません。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 320 ページの『DBName CLI/ODBC 構成キーワード』
- 352 ページの『SchemaList CLI/ODBC 構成キーワード』

TempDir CLI/ODBC 構成キーワード

キーワードの説明:

一時ファイルに使用されるディレクトリーを定義します。

db2cli.ini キーワード構文:

TempDir = < *絶対パス名* >

デフォルト設定:

TEMP または TMP 環境変数によって指定されているシステム一時ディレクトリーを使用します。

使用上の注意:

ラージ・オブジェクト (CLOBS、BLOBS など) を操作するときは、データ変換が起きた場合、またはデータがサーバーに少しずつ送られる場合に、多くの場合、情報

を保管するためにクライアント・マシンに一時ファイルが作成されます。このオプションを使用すると、このような一時ファイルのロケーションを指定することができます。何も指定しなかった場合は、システム一時ディレクトリーが使用されます。

キーワードは db2cli.ini ファイルのデータ・ソース特有のセクションに置かれます。構文は次のとおりです。

- TempDir= F:¥DB2TEMP

指定されたパスはすでに存在している必要があり、アプリケーションを実行しているユーザーはそのパスへのファイルに書き込みできる権限を持っている必要があります。 DB2 CLI ドライバーが一時ファイルを作成しようとして、パス名が無効の場合、または指定のディレクトリーに一時ファイルを作成できない場合には、HY507 の SQLSTATE が戻されます。

関連概念:

- 109 ページの『CLI アプリケーションでのラージ・オブジェクトの使用』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

Trace CLI/ODBC 構成キーワード

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**

デフォルト設定:

トレース情報はキャプチャーされません。

同等の接続属性:

SQL_ATTR_TRACE

使用上の注意:

このオプションが (1) のときには、CLI/ODBC トレース・レコードが、TraceFileName 構成パラメーターによって指示されたファイルに付加されるか、または TracePathName 構成パラメーターによって指示されたサブディレクトリー内のファイルに付加されます。 Trace は、TraceFileName または TracePathName のいずれも設定されていない場合、有効ではありません。

TraceRefreshInterval キーワードは、 Trace キーワードが db2cli.ini ファイルから読み取られるインターバルを秒単位で設定します。これにより、n 秒内に動的に CLI/ODBC トレースをオフにすることができます。

たとえば、それぞれのトレース入力後にディスクに書き込む 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共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 364 ページの『TraceFileName CLI/ODBC 構成キーワード』
- 365 ページの『TraceFlush CLI/ODBC 構成キーワード』
- 367 ページの『TracePathName CLI/ODBC 構成キーワード』
- 370 ページの『TraceRefreshInterval CLI/ODBC 構成キーワード』

TraceComm CLI/ODBC 構成キーワード

キーワードの説明:

それぞれのネットワーク要求に関する情報をトレース・ファイルに組み込みます。

db2cli.ini キーワード構文:

TraceComm = **0** | **1**

デフォルト設定:

0 - ネットワーク要求情報はキャプチャーされません。

次の場合にのみ適用可能:

CLI/ODBC Trace オプションがオンになっている。

使用上の注意:

TraceComm が (1) に設定されていると、それぞれのネットワーク要求に関する下記の情報がトレース・ファイルに含められます。

- クライアントで完全に処理された DB2 CLI 関数と、サーバーとの通信に関係した DB2 CLI 関数
- サーバーとの各通信で送受信されたバイト数
- クライアントとサーバーの間でのデータの通信に費やされた時間

このオプションは、Trace CLI/ODBC オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 364 ページの『TraceFileName CLI/ODBC 構成キーワード』
- 365 ページの『TraceFlush CLI/ODBC 構成キーワード』
- 367 ページの『TracePathName CLI/ODBC 構成キーワード』

TraceErrImmediate CLI/ODBC 構成キーワード

キーワードの説明:

診断レコードが生成された場合に、それらを CLI/ODBC トレースに書き込みます。

db2cli.ini キーワード構文:

TraceErrImmediate = **0** | **1**

デフォルト設定:

診断レコードがトレース・ファイルに書き込まれるのは、`SQLGetDiagField()` または `SQLGetDiagRec()` が呼び出された場合だけです。あるいは、取り出されていない診断レコードのあるハンドルについては、 "Unretrieved Error Message" (取り出されていないエラー・メッセージ) がトレース・ファイルに書き込まれます。

次の場合にのみ適用可能:

CLI/ODBC Trace オプションがオンになっている。

使用上の注意:

TraceErrImmediate=1 と設定し、診断レコードが生成された時点でそれを CLI/ODBC トレース・ファイルに書き込むなら、アプリケーションの実行中にエラーが発生したタイミングを調べることができます。 `SQLGetDiagField()` と `SQLGetDiagRec()` を使用して診断情報を取り出さないアプリケーションの場合、これは特に便利です。というのは、ハンドルに対して診断レコードが生成された場合、そのハンドルに対して次の関数が呼び出される前にそれらが取り出されたりトレース・ファイルに書き込んだりされなければ、それらの診断レコードは失われてしまうからです。

TraceErrImmediate=0 (デフォルトの設定値) の場合、診断レコードがトレース・ファイルに書き込まれるのは、アプリケーションが `SQLGetDiagField()` または `SQLGetDiagRec()` を呼び出して診断情報を取り出す場合だけです。このキーワードが 0 に設定されている場合、アプリケーションが関数呼び出しによって診断情報を取り出さないのであれば、ハンドルに対して次に関数が呼び出された時点で診断レコードが存在する場合に、 "Unretrieved Error Message" の項目がトレース・ファイルに書き込まれます。

このオプションは、Trace CLI/ODBC オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 366 ページの『TraceFlushOnError CLI/ODBC 構成キーワード』

TraceFileName CLI/ODBC 構成キーワード

キーワードの説明:

すべての DB2 CLI/ODBC トレース情報の書き込み先のファイル。

db2cli.ini キーワード構文:

TraceFileName = < **完全修飾ファイル名** >

デフォルト設定:

なし

次の場合にのみ適用可能:

Trace オプションがオンになっている。

同等の接続属性:

SQL_ATTR_TRACEFILE

使用上の注意:

指定されたファイルが存在しない場合は、ファイルが作成されるか、または新しいトレース情報がファイルの終わりに付加されます。ただし、目的のパスのファイルが存在している必要があります。

指定されたファイル名が無効の場合、またはファイルの作成または書き込みが不可能な場合、トレースは行われず、エラー・メッセージも戻されません。

このオプションは、Trace オプションがオンになっているときにのみ使用します。このオプションは、CLI/ODBC 構成ユーティリティーで設定すると自動的に実行されます。

このオプションを設定した場合、TracePathName オプションは無視されます。

DB2 CLI トレースはデバッグ目的でのみ使用する必要があります。オンにしたまま期間が経過すると、CLI/ODBC ドライバーの実行がスローダウンし、トレース情報が非常に大きくなる可能性があります。

TraceFileName キーワード・オプションは、マルチプロセスまたはマルチスレッド・アプリケーションでは使用しないでください。その理由は、すべてのスレッドまたはプロセスに関するトレース出力が同じログ・ファイルに書き込まれ、各スレッドまたはプロセスに関する出力を解読することが難しくなるためです。さらに、共有

トレース・ファイルへのアクセスを制御するために使用されるセマフォーにより、マルチスレッド・アプリケーションの動作が変更される可能性があります。デフォルトの DB2 CLI トレース出力ログ・ファイル名はありません。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 365 ページの『TraceFlush CLI/ODBC 構成キーワード』
- 367 ページの『TracePathName CLI/ODBC 構成キーワード』

TraceFlush CLI/ODBC 構成キーワード

キーワードの説明:

n CLI/ODBC トレース入力後にディスクへの書き込みを強制します。

db2cli.ini キーワード構文:

TraceFlush = **0** | **正の整数**

デフォルト設定:

入力ごとに書き込みを行いません。

次の場合にのみ適用可能:

CLI/ODBC Trace オプションがオンになっている。

使用上の注意:

TraceFlush は、トレース情報が CLI トレース・ファイルに書き込まれる頻度を指定します。デフォルトでは、TraceFlush は 0 に設定されており、各 DB2 CLI トレース・ファイルは、トレース対象アプリケーションまたはスレッドが正常終了するまで開かれたままになっています。アプリケーションが異常終了すると、トレース・ログ・ファイルに書き込まれていない一部のトレース情報が失われる可能性があります。

このキーワードを正の整数に設定して、 DB2 CLI ドライバーが指定されたトレース入力回数の後に適切なトレース・ファイルをクローズおよび再オープンするように強制します。 TraceFlush キーワードの値が小さいほど、アプリケーションのパフォーマンスへの DB2 CLI トレースの影響は大きくなります。 TraceFlush=1 と設定すると、パフォーマンスへの影響が最大になりますが、アプリケーションが次のステートメントに移る前に各項目が確実にディスクに書き込まれます。

このオプションは、Trace CLI/ODBC オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 364 ページの『TraceFileName CLI/ODBC 構成キーワード』
- 367 ページの『TracePathName CLI/ODBC 構成キーワード』

TraceFlushOnError CLI/ODBC 構成キーワード

キーワードの説明:

エラーが発生したなら、すべての CLI/ODBC トレース項目をディスクに書き込みます。

db2cli.ini キーワード構文:

TraceFlushOnError = **0** | **1**

デフォルト設定:

エラー発生時に、ただちに CLI/ODBC トレース項目を書き込むことはしません。

次の場合にのみ適用可能:

CLI/ODBC Trace オプションがオンになっている。

使用上の注意:

TraceFlushOnError=1 と設定すると、 DB2 CLI ドライバーは、エラーが検出されるたびにトレース・ファイルをクローズしてから再オープンします。
TraceFlushOnError がデフォルト値 (0) のままである場合、トレース・ファイルがクローズされるのは、アプリケーションが正常に終了した時点、または TraceFlush キーワードによって指定された時間間隔に達した場合だけです。 TraceFlushOnError=0 の場合にアプリケーション・プロセスが異常終了すると、貴重なトレース情報が失われてしまう可能性があります。 TraceFlushOnError=1 と設定するとパフォーマンスに影響を与えることがありますが、エラーに関連するトレース項目は確実にディスクに書き込まれるようになります。

このオプションは、Trace CLI/ODBC オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 365 ページの『TraceFlush CLI/ODBC 構成キーワード』
- 363 ページの『TraceErrImmediate CLI/ODBC 構成キーワード』

TraceLocks CLI/ODBC 構成キーワード

キーワードの説明:

ロック・タイムアウトのみを CLI/ODBC トレースにトレースします。

db2cli.ini キーワード構文:

TraceLocks = **0** | **1**

デフォルト設定:

トレース情報はロック・タイムアウトのみに限定されません。

次の場合にのみ適用可能:

Trace オプションがオンになっている。

使用上の注意:

TraceLocks が 1 に設定されていると、ロック・タイムアウトはトレース・ファイルに記録されます。

このオプションは、CLI/ODBC TRACE オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 364 ページの『TraceFileName CLI/ODBC 構成キーワード』
- 365 ページの『TraceFlush CLI/ODBC 構成キーワード』
- 367 ページの『TracePathName CLI/ODBC 構成キーワード』

TracePathName CLI/ODBC 構成キーワード

キーワードの説明:

個々の DB2 CLI/ODBC トレース・ファイルの保管に使用されるサブディレクトリー。

db2cli.ini キーワード構文:

TracePathName = < **完全修飾サブディレクトリー名** >

デフォルト設定:

なし

次の場合にのみ適用可能:

Trace オプションがオンになっている。

次の場合には適用不可:

TraceFileName オプションがオンになっている。

使用上の注意:

同じ DLL または共有ライブラリーを使用する各スレッドまたは処理は、指定のディレクトリーに別々の DB2 CLI/ODBC トレース・ファイルを作成します。トレース・ファイルの名前は、アプリケーション・プロセス ID とスレッド・シーケンス番号を連結したものにより、自動的に付けられます。

指定されたサブディレクトリーが無効の場合、またはサブディレクトリーの書き込みが不可能な場合、トレースは行われず、エラー・メッセージも戻されません。

このオプションは、Trace オプションがオンになっているときにのみ使用します。このオプションは、CLI/ODBC 構成ユーティリティーで設定すると自動的に実行されます。

このオプションは、DB2 CLI/ODBC オプション TraceFileName が使用された場合には無視されます。

DB2 CLI トレースはデバッグ目的でのみ使用する必要があります。オンにしたまま期間が経過すると、CLI/ODBC ドライバーの実行がスローダウンし、トレース情報が非常に大きくなる可能性があります。

TraceFileName と TracePathName の両方が指定されている場合は、 TraceFileName キーワードが優先され、TracePathName は無視されます。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 364 ページの『TraceFileName CLI/ODBC 構成キーワード』
- 365 ページの『TraceFlush CLI/ODBC 構成キーワード』

TracePIDList CLI/ODBC 構成キーワード

キーワードの説明:

CLI/ODBC トレースが使用可能になる処理 ID を制限します。

db2cli.ini キーワード構文:

TracePIDList = **<値の指定なし>** | **<コンマで区切られた処理 ID のリスト>**

デフォルト設定:

CLI/ODBC トレースが実行されている場合、どのプロセス ID でもトレースされます。

使用上の注意:

このキーワードに値が指定されていない場合、すべての処理 ID がトレースされます。すべての処理 ID をトレースしない場合は、 CLI/ODBC トレースの実行時にトレースする処理 ID のリストをコンマ区切りで指定してください。

このキーワードを最も有効にするには、ご使用のアプリケーションを初期化する前に、 TraceRefreshInterval キーワードを任意の値に設定してください。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 370 ページの『TraceRefreshInterval CLI/ODBC 構成キーワード』
- 369 ページの『TracePIDTID CLI/ODBC 構成キーワード』

TracePIDTID CLI/ODBC 構成キーワード

キーワードの説明:

各項目がトレースされるごとにプロセス ID およびスレッド ID をキャプチャーします。

db2cli.ini キーワード構文:

TracePIDTID = **0** | **1**

デフォルト設定:

トレース入力のプロセス ID およびスレッド ID はキャプチャーされません。

次の場合にのみ適用可能:

Trace オプションがオンになっている。

使用上の注意:

TracePIDTID が 1 に設定されると、キャプチャーされた項目ごとにプロセス ID およびスレッド ID がトレース・ファイルに記録されます。 Trace キーワードが使用可能で、複数のアプリケーションが実行中の場合に効果があります。これは、Trace により、すべての実行アプリケーションのトレース情報が単一のファイルに書き込まれるためです。 TracePIDTID を使用可能にすることにより、処理とスレッドによって記録された情報を見分けることができます。

このオプションは、CLI/ODBC Trace オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 364 ページの『TraceFileName CLI/ODBC 構成キーワード』
- 365 ページの『TraceFlush CLI/ODBC 構成キーワード』
- 367 ページの『TracePathName CLI/ODBC 構成キーワード』
- 369 ページの『TracePIDList CLI/ODBC 構成キーワード』

TraceRefreshInterval CLI/ODBC 構成キーワード

キーワードの説明:

Trace キーワードと TracePIDList キーワードが db2cli.ini ファイルの Common セクションから読み取られるインターバル (秒単位) を設定します。

db2cli.ini キーワード構文:

TraceRefreshInterval = **0** | **正の整数**

デフォルト設定:

Trace および TracePIDList キーワードは、アプリケーションの初期化時に db2cli.ini ファイルから読み取られるのみです。

使用上の注意:

アプリケーションの初期化前にこのキーワードを設定すると、 n 秒以内に CLI/ODBC トレースを動的にオフにすることができます。

注: アプリケーションの実行中に TraceRefreshInterval を設定しても、無効です。このキーワードを有効にするには、アプリケーションの初期化前に設定されている必要があります。

このキーワードが設定されている場合、 Trace および TracePIDList キーワードのみが db2cli.ini ファイルから更新されます。他の CLI/ODBC 構成キーワードは再読み取りされません。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 369 ページの『TracePIDList CLI/ODBC 構成キーワード』

TraceStmtOnly CLI/ODBC 構成キーワード

キーワードの説明:

動的 SQL ステートメントのみを CLI/ODBC トレースにトレースします。

db2cli.ini キーワード構文:

TraceStmtOnly = **0** | **1**

デフォルト設定:

トレース情報は動的 SQL ステートメントのみに限定されません。

次の場合にのみ適用可能:

Trace オプションがオンになっている。

使用上の注意:

TraceStmtOnly が 1 に設定されると、動的 SQL ステートメントのみがトレース・ファイルに記録されます。

このオプションは、CLI/ODBC Trace オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 364 ページの『TraceFileName CLI/ODBC 構成キーワード』
- 365 ページの『TraceFlush CLI/ODBC 構成キーワード』

• 367 ページの『TracePathName CLI/ODBC 構成キーワード』

TraceTime CLI/ODBC 構成キーワード

キーワードの説明:

経過時間カウンターをトレース・ファイルにキャプチャーします。

db2cli.ini キーワード構文:

TraceTime = **1** | **0**

デフォルト設定:

経過時間カウンターがトレース・ファイルに組み込まれます。

次の場合にのみ適用可能:

Trace オプションがオンになっている。

使用上の注意:

TraceTime が 1 に設定されると、経過時間カウンターがトレース・ファイルにキャプチャーされます。以下に例を示します。

```
SQLPrepare( hStmt=1:1, pszSqlStr="SELECT * FROM ORG", cbSqlStr=-3 )
  ---> Time elapsed - +6.785751E+000 seconds ( StmtOut="SELECT * FROM ORG" )
SQLPrepare( )
  <--- SQL_SUCCESS Time elapsed - +2.527400E-002 seconds
```

パフォーマンスを向上させたり、トレース・ファイルをより小さくしたりするためには、 TraceTime を 0 に設定して、これをオフにします。たとえば、

```
SQLPrepare( hStmt=1:1, pszSqlStr="SELECT * FROM ORG", cbSqlStr=-3 )
( StmtOut="SELECT * FROM ORG" )
SQLPrepare( )
  <--- SQL_SUCCESS
```

このオプションは、CLI/ODBC Trace オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 364 ページの『TraceFileName CLI/ODBC 構成キーワード』
- 365 ページの『TraceFlush CLI/ODBC 構成キーワード』
- 367 ページの『TracePathName CLI/ODBC 構成キーワード』
- 373 ページの『TraceTimestamp CLI/ODBC 構成キーワード』

TraceTimestamp CLI/ODBC 構成キーワード

キーワードの説明:

異なるタイプのタイム・スタンプ情報を CLI/ODBC トレースにキャプチャーします。

db2cli.ini キーワード構文:

TraceTimestamp = **0** | **1** | **2** | **3**

デフォルト設定:

タイム・スタンプ情報はトレース・ファイルに書き込まれません。

次の場合にのみ適用可能:

Trace オプションがオンになっている。

使用上の注意:

TraceTimeStamp をデフォルトの 0 以外の値に設定すると、現在のタイム・スタンプまたは絶対実行時が、トレース情報の各行の先頭に (DB2 CLI トレース・ファイルに書き込まれる時点で) 追加されます。以下の設定値は、トレース・ファイルにキャプチャーされるタイム・スタンプ情報のタイプを示しています。

- 0 = タイム・スタンプ情報なし
- 1 = プロセッサー時刻と ISO タイム・スタンプ (絶対実行時 (秒およびミリ秒) の後にタイム・スタンプ)
- 2 = プロセッサー時刻 (絶対実行時 (秒およびミリ秒))
- 3 = ISO タイム・スタンプ

このオプションは、CLI/ODBC Trace オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、 DB2 へのすべての接続に適用されます。)

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』
- 211 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 361 ページの『Trace CLI/ODBC 構成キーワード』
- 364 ページの『TraceFileName CLI/ODBC 構成キーワード』
- 365 ページの『TraceFlush CLI/ODBC 構成キーワード』
- 367 ページの『TracePathName CLI/ODBC 構成キーワード』
- 372 ページの『TraceTime CLI/ODBC 構成キーワード』

TxnIsolation CLI/ODBC 構成キーワード

キーワードの説明:

デフォルトの分離レベルを設定します。

db2cli.ini キーワード構文:

TxnIsolation = **1** | **2** | **4** | **8** | **32**

デフォルト設定:

コミット読み取り (カーソル固定)

次の場合にのみ適用可能:

デフォルトの分離レベルが使用されています。このキーワードは、アプリケーションが明確に分離レベルを設定している場合は無効になります。

同等のステートメント属性:

SQL_ATTR_TXN_ISOLATION

使用上の注意:

分離レベルを以下に設定します。

- 1 = SQL_TXN_READ_UNCOMMITTED - 読み取り非コミット (非コミット読み取り)
- 2 = SQL_TXN_READ_COMMITTED (デフォルト) - コミット読み取り (カーソル固定)
- 4 = SQL_TXN_REPEATABLE_READ - 反復可能読み取り (読み取り固定)
- 8 = SQL_TXN_SERIALIZABLE - シリアライズ可能 (反復可能読み取り)
- 32 = SQL_TXN_NOCOMMIT - (コミットなし、DB2 Universal Database for AS/400 専用; これは自動コミットと同様です)

括弧の中の用語は、SQL92 分離レベルに相当する IBM の用語です。コミットなしは SQL92 分離レベルではなく、 DB2 Universal Database for AS/400 でのみサポートされることに注意してください。

このキーワードは、デフォルトの分離レベルが使用されている場合にのみ適用できます。アプリケーションが明確に接続またはステートメント・ハンドルの分離レベルを設定している場合は、このキーワードはそのハンドルで無効になります。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『分離レベル』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『ステートメント属性 (CLI) のリスト』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

UID CLI/ODBC 構成キーワード

キーワードの説明:

デフォルトのユーザー ID を定義します。

db2cli.ini キーワード構文:

UID = *userid*

デフォルト設定:

なし

使用上の注意:

指定された *userid* 値は、接続時にユーザー ID がアプリケーションによって提供されない場合に使用されます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 348 ページの『PWD CLI/ODBC 構成キーワード』

Underscore CLI/ODBC 構成キーワード

キーワードの説明:

下線文字 (_) をワイルドカードとして処理するかどうかを指定します。

db2cli.ini キーワード構文:

Underscore = **0** | **1**

デフォルト設定:

下線文字は、任意の 1 文字または 0 文字と一致します。

使用上の注意:

このキーワードは、下線文字 (_) がワイルドカードとして認識されるか、それとも下線文字としてのみ認識されるかを指定します。可能な設定値は次のとおりです。

- 0 - 下線文字は下線文字としてのみ処理されます。
- 1 - 下線文字は任意の 1 文字または 0 文字と一致するワイルドカードとして処理されます。

名前に下線文字が含まれるデータベース・オブジェクトがある場合、 Underscore を 0 に設定すると、パフォーマンスが向上することがあります。

このキーワードが適用されるのは、引き数として検索パターンを受け入れる下記のカタログ関数だけです。

- `SQLColumnPrivileges()`
- `SQLColumns()`
- `SQLProcedureColumns()`
- `SQLProcedures()`
- `SQLTablePrivileges()`
- `SQLTables()`

カタログ関数は特定の引き数についてのみ検索パターンを受け入れる場合があることに注意してください。詳細については、特定の関数の資料を参照してください。

関連概念:

- 186 ページの『CLI アプリケーションのカタログ関数の入力引き数』
- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLColumnPrivileges 関数 (CLI) - 表の列に関連した特権の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLColumns 関数 (CLI) - 表の列の情報の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*の『SQLProcedureColumns 関数 (CLI) - プロシージャーの入力/出力パラメーター情報の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLProcedures 関数 (CLI) - プロシージャー名のリストの取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLTablePrivileges 関数 (CLI) - 表に関連する特権の取得』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLTables 関数 (CLI) - 表の情報の取得』
- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

UseOldStpCall CLI/ODBC 構成キーワード

キーワードの説明:

カタログ式プロシージャーが呼び出される方法をコントロールします。

db2cli.ini キーワード構文:

UseOldStpCall = **0** | **1**

デフォルト設定:

GRANT EXECUTE がプロシージャーに関して付与される必要のある新しい CALL 方式を使用して、そのプロシージャーを呼び出します。

使用上の注意:

バージョン 8 より前のバージョンでは、プロシージャーの呼び出し側で、プロシージャーから呼び出されたパッケージに関して、 EXECUTE 特権を持っている必要がありました。今回から、呼び出し側はプロシージャーについての EXECUTE 特権を持っている必要があり、プロシージャーの定義者のみが、必要とされるどのパッケージに関しても EXECUTE 特権を持っている必要があります。

このキーワードは、プロシージャーを呼び出すのにどの方式を使用するかをコントロールします。 UseOldStpCall をオンに設定すると、プリコンパイラーが CALL ステートメントのプロシージャーの解決に失敗した場合、使用すべきでない sqleproc() API を使用してプロシージャーが呼び出されます。このキーワードをオフにすると、GRANT EXECUTE がプロシージャーで付与される必要のある、そのプロシージャーが呼び出されます。

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連タスク:

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

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』

WarningList CLI/ODBC 構成キーワード

キーワードの説明:

警告にグレードを下げるエラーを指定します。

db2cli.ini キーワード構文:

WarningList = ″ ’**xxxxx**’, ’**yyyyy**’, ...″

デフォルト設定:

SQLSTATE のグレードを下げません。

使用上の注意:

エラーとして戻される多くの SQLSTATE のグレードを警告に下げることができます。それぞれを大文字で指定し、単一引用符で囲み、コンマで区切ります。ストリング全体が二重引用符で囲まれている必要もあります。たとえば、

```
WarningList=" '01S02', 'HY090' "
```

関連概念:

- 289 ページの『db2cli.ini 初期設定ファイル』

関連資料:

- 291 ページの『CLI/ODBC 構成キーワード (カテゴリー別)』
- 330 ページの『IgnoreWarnList CLI/ODBC 構成キーワード』

第 5 部 データ変換

第 26 章 データ変換

この章では、 SQL から C へのデータ・タイプの変換と C から SQL へのデータ・タイプの変換の詳細に加えて、 CLI でサポートされるデータ変換について説明します。

CLI でサポートされているデータ変換

次の表には、DB2 CLI でサポートされているデータ・タイプ変換が示されています。

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

D 変換はサポートされており、SQL データ・タイプのデフォルト変換です。

X すべての IBM DBMS が変換をサポートします。

ブランク
いずれの IBM DBMS も変換をサポートしません。

例として、 SQLCHAR (または 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、SQLINTEGER を SQL_C_DBCHAR に変換することはできません。

データ・タイプの形式についての詳細は、データ・タイプ属性 (精度、スケール、長さ、および表示) の表を参照してください。

表 24. サポートされるデータ変換

SQL データ・タイプ	SQL_CHAR	SQL_C_BINARY	SQL_C_WCHAR	SQL_C_DBCHAR	SQL_C_SHORT	SQL_C_USHORT	SQL_C_LONG	SQL_C_ULONG	SQL_C_SBIGINT	SQL_C_UBIGINT	SQL_C_TINYINT	SQL_C_UTINYINT	SQL_C_BIT	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_NUMERIC	SQL_C_TYPE_DATE	SQL_C_TYPE_TIME	SQL_C_TYPE_TIMESTAMP	SQL_C_DECIMAL_IBM	SQL_C_DATALINK	SQL_C_CLOB_LOCATOR	SQL_C_BLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR
SQL_CHAR	D	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X					

表 24. 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_BINARY	SQL_C_WCHAR	SQL_C_DBCHAR	SQL_C_SHORT	SQL_C_USHORT	SQL_C_LONG	SQL_C_ULONG	SQL_C_SBIGINT	SQL_C_UBIGINT	SQL_C_TINYINT	SQL_C_UTINYINT	SQL_C_BIT	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_NUMERIC	SQL_C_TYPE_DATE	SQL_C_TYPE_TIME	SQL_C_TYPE_TIMESTAMP	SQL_C_DECIMAL_IBM	SQL_C_DATALINK	SQL_C_CLOB_LOCATOR	SQL_C_BLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR
SQL_VARCHAR	D	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X					
SQL_LONG VARCHAR	D	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X					
SQL_BINARY	X	D	X																					
SQL_VARBINARY	X	D	X																					
SQL_LONG VARBINARY	X	D	X																					
SQL_GRAPHIC	X	X	X	D																				
SQL_VARGRAPHIC	X	X	X	D																				
SQL_LONG VARGRAPHIC	X	X	X	D																				
SQL_CLOB	D	X	X																			X		
SQL_BLOB	X	D	X																				X	
SQL_DBCLOB	X	X	X	D																				X
SQL_CLOB_LOCATOR																						D		
SQL_BLOB_LOCATOR																							D	
SQL_DBCLOB_LOCATOR																								D
SQL_NUMERIC	D	X	X		X	X	X	X	X	X	X	X	X	X	X	X				X				
SQL_DECIMAL	D	X	X		X	X	X	X	X	X	X	X	X	X	X	X				X				
SQL_INTEGER	X	X	X		X	X	D	X	X	X	X	X	X	X	X	X								
SQL_SMALLINT	X	X	X		D	X	X	X	X	X	X	X	X	X	X	X								
SQL_FLOAT	X	X	X		X	X	X	X	X	X	X	X	X	X	D	X								
SQL_DOUBLE	X	X	X		X	X	X	X	X	X	X	X	X	X	D	X								

表 24. 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_BINARY	SQL_C_WCHAR	SQL_C_DBCHAR	SQL_C_SHORT	SQL_C_USHORT	SQL_C_LONG	SQL_C_ULONG	SQL_C_SBIGINT	SQL_C_UBIGINT	SQL_C_TINYINT	SQL_C_UTINYINT	SQL_C_BIT	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_NUMERIC	SQL_C_TYPE_DATE	SQL_C_TYPE_TIME	SQL_C_TYPE_TIMESTAMP	SQL_C_DECIMAL_IBM	SQL_C_DATALINK	SQL_C_CLOB_LOCATOR	SQL_C_BLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR
SQL_REAL	X	X	X		X	X	X	X	X	X	X	X	X	D	X	X								
SQL_BIGINT	X	X	X		X	X	X	X	D	X	X	X	X	X	X	X								
SQL_TINYINT	X	X	X		X	X	X	X	X	X	D	X	X	X	X	X								
SQL_BIT	X	X	X										D											
SQL_TYPE_DATE	X		X														D		X					
SQL_TYPE_TIME	X		X															D	X					
SQL_TYPE_TIMESTAMP	X		X														X	X	D					
SQL_DATALINK	D		X																		X			

関連概念:

- 46 ページの『CLI アプリケーションにおけるデータ・タイプとデータ変換』

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 383 ページの『CLI での SQL から C へのデータ変換例』
- 390 ページの『CLI での C から SQL へのデータ変換例』

CLI での SQL から C へのデータ変換例

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 NULL 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 NULL 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 25. 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	データ値は有効な日付 [a]	データ	6 [b]	00000
	データ値は有効な日付ではない [a]	影響なし	6 [b]	22007
SQL_C_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

注:

a *cbValueMax* の値はこの変換では無視されます。ドライバーは、 *rgbValue* のサイズは C データ・タイプのサイズであると想定します。

b これは、対応する C データ・タイプのサイズです。

c SQL_C_NUMERIC は Windows プラットフォーム上でのみサポートされます。

SQLSTATE **00**000 は `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。

GRAPHIC SQL データから C データへの変換:

GRAPHIC SQL データ・タイプは、以下のとおりです。

SQL_GRAPHIC
SQL_VARGRAPHIC
SQL_LONGVARGRAPHIC

SQL_DBCLOB

表 26. GRAPHIC SQL データから C データへの変換

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。

数値 SQL データから C データへの変換:

SQL 数値データ・タイプは、以下のとおりです。

SQL_DECIMAL
SQL_NUMERIC
SQL_SMALLINT
SQL_INTEGER
SQL_BIGINT
SQL_REAL
SQL_FLOAT
SQL_DOUBLE

表 27. 数値 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
	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

表 27. 数値 SQL データから C データへの変換 (続き)

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_SHORT SQL_C_LONG SQL_C_FLOAT SQL_C_DOUBLE SQL_C_TINYINT SQL_C_BIT SQL_C_UBIGINT SQL_C_SBIGINT SQL_C_NUMERIC [b]	切り捨てられずに変換されたデータ [a]	データ	C データ・タイプのサイズ	00000
	変換され切り捨てられたデータ、ただし有効桁は失われていない [a]	切り捨てデータ	C データ・タイプのサイズ	01004
	データの変換で、有効桁が失われる [a]	影響なし	C データ・タイプのサイズ	22003

注:

a *cbValueMax* の値はこの変換では無視されます。ドライバーは、 *rgbValue* のサイズは C データ・タイプのサイズであると想定します。

b SQL_C_NUMERIC は Windows プラットフォーム上でのみサポートされます。

SQLSTATE **00**000 は `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。

バイナリー SQL データから C データへの変換:

バイナリー SQL データ・タイプは次のとおりです。

SQL_BINARY
SQL_VARBINARY
SQL_LONGVARBINARY
SQL_BLOB

表 28. バイナリー 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	N/A
	(データ長) >= cbValueMax	切り捨てデータ	データの長さ	01004
SQL_C_BINARY	データ長 <= cbValueMax	データ	データの長さ	N/A
	データ長 > cbValueMax	切り捨てデータ	データの長さ	01004

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 29. 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

表 29. 日付 SQL データから C データへの変換 (続き)

fCType	Test	rgbValue	pcbValue	SQLSTATE
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 30. 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 31. 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、タイム・スタンプ SQL データ・タイプの精度には関係なく「yyyy-mm-dd hh:mm:ss.ffffff」という形式になります。アプリケーションに ISO フォーマットが必要な場合、CLI/ODBC 構成キーワード PATCH2=33 に設定します。

SQL から C へのデータ変換例:

表 32. SQL から C へのデータ変換例

SQL データ・タイプ	SQL データ値	C データ・タイプ	cbValue max	rgbValue	SQL STATE
SQL_CHAR	abcdef	SQL_C_CHAR	7	abcdef¥0 [a]	00000
SQL_CHAR	abcdef	SQL_C_CHAR	6	abcde¥0 [a]	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	8	1234.56¥0 [a]	00000
SQL_DECIMAL	1234.56	SQL_C_CHAR	5	1234¥0 [a]	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	4	---	22003
SQL_DECIMAL	1234.56	SQL_C_FLOAT	無視されます	1234.56	00000
SQL_DECIMAL	1234.56	SQL_C_SHORT	無視されます	1234	01004
SQL_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

表 32. 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_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 NULL 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。

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 381 ページの『CLI でサポートされているデータ変換』
- 390 ページの『CLI での C から SQL へのデータ変換例』
- 345 ページの『Patch2 CLI/ODBC 構成キーワード』

CLI での C から SQL へのデータ変換例

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。

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 NULL 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 NULL 終了バイトを除く)。これは、データ・ソースに送られる前にデータが切り捨てられる場合にも当てはまります。

列の長さ
データがデフォルト C データ・タイプへ転送されるときにアプリケーションに戻されるバイトの最大数。文字データの場合、長さには NULL 終了バイトは含まれません。

表示サイズ
データを文字書式で表示するために必要な最大バイト数。

有効桁 負符号 (必要な場合) および小数点の左の桁。

文字 C データから SQL データへの変換:

文字 C データ・タイプは次のとおりです。
SQL_C_CHAR

表 33. 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	N/A
	データ長 > 列長	22001
SQL_DECIMAL SQL_NUMERIC SQL_SMALLINT SQL_INTEGER SQL_BIGINT SQL_REAL SQL_FLOAT SQL_DOUBLE	切り捨てられずに変換されたデータ	N/A
	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	N/A
	(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	N/A
	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	N/A
	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	N/A
	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	N/A
	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。

数値 C データから SQL データへの変換:

数値 C データ・タイプは次のとおりです。

SQL_C_SHORT
SQL_C_LONG
SQL_C_FLOAT
SQL_C_DOUBLE
SQL_C_TINYINT
SQL_C_SBIGINT
SQL_C_BIT

表 34. 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	N/A
SQL_SMALLINT SQL_INTEGER SQL_BIGINT SQL_REAL SQL_FLOAT SQL_DOUBLE	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
SQL_CHAR SQL_VARCHAR	切り捨てられずに変換されたデータ	N/A
	データ変換の結果、有効桁が消失する	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。

バイナリー C データから SQL データへの変換:

バイナリー C データ・タイプは次のとおりです。

SQL_C_BINARY

表 35. バイナリー 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	N/A
	データ長 > 列長	22001
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_BLOB	データ長 <= 列長	N/A
	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・タイプは以下のとおりです。

SQL_C_DBCHAR

表 36. DBCHAR 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 × 2	N/A
	データ長 > 列長 × 2	22001
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_BLOB	データ長 <= 列長 × 2	N/A
	データ長 > 列長 × 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 37. 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	N/A
	列長 < 10	22003
SQL_DATE	データ値は有効な日付	N/A
	データ値は有効な日付ではない	22007
SQL_TIMESTAMP [a]	データ値は有効な日付	N/A
	データ値は有効な日付ではない	22007

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 の時刻コンポーネントはゼロに設定されます。

時刻 C データから SQL データへの変換:

時刻 C データ・タイプは次のとおりです。
SQL_C_TIME

表 38. 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	N/A
	列長 < 8	22003
SQL_TIME	データ値は有効な時刻	N/A
	データ値は有効な時刻ではない	22007
SQL_TIMESTAMP [a]	データ値は有効な時刻	N/A
	データ値は有効な時刻ではない	22007

注: SQLSTATE **00**000 は `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 の日付コンポーネントは、アプリケーションが実行しているマシンのシステム日付に設定されます。

タイム・スタンプ C データから SQL データへの変換:

タイム・スタンプ C データ・タイプは次のとおりです。
SQL_C_TIMESTAMP

表 39. タイム・スタンプ C データから SQL データへの変換

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	列長 >= 表示サイズ	N/A
	19 <= 列長 < 表示サイズ [a]	22001
	列長 < 19	22003
SQL_DATE	時刻フィールドがゼロ	N/A
	時刻フィールドがゼロ以外	22008
	データ値に有効な日付が含まれていない [b]	22007
SQL_TIME	小数秒フィールドがゼロ	N/A
	小数秒フィールドがゼロ以外	22008
	データ値に有効な時刻が含まれていない	22007
SQL_TIMESTAMP	データ値は有効なタイム・スタンプ	N/A
	データ値は有効なタイム・スタンプではない	22007

注:

a タイム・スタンプの小数秒は切り捨てられます。

b timestamp_struct は、時間、分、秒および小数部を 0 にリセットする必要があります。そうでない場合は、SQLSTATE 22007 が返されます。

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 40. C から SQL へのデータ変換例

C データ・タイプ	C データ値	SQL データ・タイプ	列の長さ	SQL データ値	SQL STATE
SQL_C_CHAR	abcdef¥0	SQL_CHAR	6	abcdef	N/A
SQL_C_CHAR	abcdef¥0	SQL_CHAR	5	abcde	22001
SQL_C_CHAR	1234.56¥0	SQL_DECIMAL	6	1234.56	N/A
SQL_C_CHAR	1234.56¥0	SQL_DECIMAL	5	1234.5	22001
SQL_C_CHAR	1234.56¥0	SQL_DECIMAL	3	---	22003
SQL_C_CHAR	4.46.32	SQL_TIME	6	4.46.32	N/A
SQL_C_CHAR	4-46-32	SQL_TIME	6	該当せず	22007
SQL_C_DOUBLE	123.45	SQL_CHAR	22	1.23450000 000000e+02	N/A
SQL_C_FLOAT	1234.56	SQL_FLOAT	該当せず	1234.56	N/A
SQL_C_FLOAT	1234.56	SQL_INTEGER	該当せず	1234	22001
SQL_C_ TIMESTAMP	1992-12-31 23:45:55. 123456	SQL_DATE	6	1992-12-31	01004

表 40. C から SQL へのデータ変換例 (続き)

C データ・タイプ	C データ値	SQL データ・タイプ	列の長さ	SQL データ値	SQL STATE

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。

関連資料:

- 48 ページの『CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ』
- 50 ページの『CLI アプリケーション用の C データ・タイプ』
- 381 ページの『CLI でサポートされているデータ変換』
- 383 ページの『CLI での SQL から C へのデータ変換例』

第 6 部 付録

付録 A. DB2 Universal Database の技術情報の概要

DB2 ドキュメンテーションおよびヘルプ

DB2 の技術情報は、以下のツールや手段によって利用できます。

- DB2 インフォメーション・センター
  - トピック
  - DB2 ツールのヘルプ
  - サンプル・プログラム
  - チュートリアル
- ダウンロード可能な PDF ファイル、CD 上の PDF ファイル、および印刷資料
  - ガイド
  - リファレンス・マニュアル
- コマンド行ヘルプ
  - コマンド・ヘルプ
  - メッセージ・ヘルプ
  - SQL 状態ヘルプ
- インストールされているソース・コード
  - サンプル・プログラム

ibm.com において、オンラインでも付加的な DB2 Universal Database の技術情報を利用できます。その中には、技術情報、技術白書、およびレッドブック (Redbooks) が含まれています。 DB2 Information Management Library サイト (www.ibm.com/software/data/db2/udb/support.html) をご覧ください。

DB2 ドキュメンテーションの更新

IBM では、利用可能な DB2 インフォメーション・センターに対するドキュメンテーションのフィックスパックや、その他のドキュメンテーション更新情報を定期的に作成する場合があります。 http://publib.boulder.ibm.com/infocenter/db2help/ から DB2 インフォメーション・センターを利用する場合は、常にほとんど最新の情報を参照できます。 DB2 インフォメーション・センターをローカルにインストールした場合、更新された情報を表示するためには、更新情報をインストールする必要があります。ドキュメンテーション更新情報をインストールすると、新しい情報が利用可能になった場合に、 *DB2* インフォメーション・センター *CD* からインストールした情報が更新されます。

インフォメーション・センターは、 PDF やハードコピー資料に比べて更新が頻繁です。 DB2 の最新の技術情報を入手するには、ドキュメンテーション更新情報が利用可能になった時点でそれをインストールするか、または www.ibm.com サイトの DB2 インフォメーション・センターを利用してください。

関連概念:

- 283 ページの『CLI サンプル・プログラム』

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Java サンプル・プログラム』
- 400 ページの『DB2 インフォメーション・センター』

関連タスク:

- 420 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 411 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 422 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 422 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』
- 423 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』

関連資料:

- 413 ページの『DB2 PDF 資料および印刷された資料』

DB2 インフォメーション・センター

DB2® インフォメーション・センターを使用すると、 DB2 Universal Database™、 DB2 Connect™、 DB2 Information Integrator および DB2 Query Patroller™ などの DB2 ファミリー製品を最大限に活用するのに必要なすべての情報にアクセスできます。また、DB2 インフォメーション・センターは、 DB2 の主な機能とコンポーネントに関する情報を提供します (レプリケーション、データウェアハウジング、および DB2 の種々の Extender など)。

Mozilla 1.0 以上または Microsoft® Internet Explorer 5.5 以上で表示する場合、 DB2 インフォメーション・センターには以下の機能があります。以下のいくつかの機能では、 JavaScript™ のサポートを使用可能にする必要があります:

柔軟なインストール・オプション

以下の中から、 ご使用の環境に最も適したオプションを使って DB2 資料を表示できます。

- 最新の資料を常に自動的に利用できるようにするには、 IBM® の Web サイト (http://publib.boulder.ibm.com/infocenter/db2help/) にある DB2 インフォメーション・センターからすべての資料に直接アクセスします。
- 更新処理を最小化し、イントラネット内のネットワーク・トラフィックだけに制限するには、イントラネット上の 1 つのサーバーに DB2 資料をインストールします。
- 柔軟性を改善し、ネットワーク接続への依存を軽減するには、個々のコンピューターに DB2 資料をインストールします。

検索

「検索」テキスト・フィールドに検索語を入力することにより、 DB2 インフォメーション・センターのすべてのトピックを検索できます。複数の語句を引用符で囲めば、完全一致を検索できます。また、ワイルドカード演算子 (*、?) とブール演算子 (AND、 NOT、OR) を使用して検索を絞り込むことができます。

タスク指向の目次

単一の目次の中から、DB2 資料のトピックを見付けることができます。目

次は、主に実行するタスクの種類に従って編成されていますが、そのほかに製品概要、特定のゴール (目的) の情報、参照情報、索引、および用語集も含まれます。

- 製品概要では、DB2 ファミリーで使用可能な製品間の関係、そうした各製品で提供される機能、および各製品の最新リリース情報について説明されています。
- インストール、管理および開発などのゴール・カテゴリーには、タスクを迅速に完了し、そのための背景情報をよく理解できるようにするトピックが含まれています。
- 「参照」トピックでは、その対象に関する詳細な情報 (ステートメントとコマンドの構文、メッセージ・ヘルプ、構成パラメーターなど) が説明されています。

現在のトピックを目次に表示する

現在のトピックが目次のどの部分に該当するかを表示するには、目次フレーム内の**「リフレッシュ/現在のトピックの表示 (Refresh/Show Current Topic)」**ボタンをクリックするか、コンテンツ・フレーム内の**「目次に表示 (Show in Table of Contents)」**ボタンをクリックします。幾つかのファイルで関連トピックへの複数のリンクをたどった場合、または検索結果からトピックにアクセスした場合には、この機能が役立ちます。

索引 索引から、すべての資料にアクセスすることができます。索引では、用語が 50 音順に編成されています。

用語集 用語集を見れば、DB2 資料で使われているさまざまな用語の定義を調べることができます。用語集では、用語が 50 音順に編成されています。

組み込まれているローカライズ情報

DB2 インフォメーション・センターは、ブラウザーで設定された言語でトピックを表示します。設定された言語のトピックが利用できない場合、DB2 インフォメーション・センターにはそのトピックの英語版が表示されます。

iSeries™ 技術情報については、 IBM eServer™ iSeries Information Center (www.ibm.com/eserver/iseries/infocenter/) を参照してください。

関連概念:

- 402 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 411 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 412 ページの『DB2 インフォメーション・センターのトピックを特定の言語で表示する方法』
- 410 ページの『DB2 インフォメーション・センターの呼び出し』
- 405 ページの『DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール (UNIX)』
- 407 ページの『DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール (Windows)』

DB2 インフォメーション・センターのインストール・シナリオ

さまざまな作業環境にある人々が、それぞれの環境に応じた方法で DB2 製品資料にアクセスする必要があります。それで、DB2 製品資料にアクセスする方法には、IBM Web サイトからアクセスする方法、イントラネット・サーバーからアクセスする方法、そしてコンピューター上にインストールしてアクセスする方法の 3 種類があります。それら 3 種類のいずれにおいても、資料は DB2 インフォメーション・センターに含まれています。それは、ブラウザーで表示できるトピック単位の情報として設計された Web です。 DB2 製品は、デフォルトで IBM Web サイトから DB2 インフォメーション・センターにアクセスするようになっています。しかし、DB2 インフォメーション・センターをイントラネット・サーバーからアクセスしたり、自分のコンピューターでアクセスしたりしたい場合には、製品メディア・パックに含まれている DB2 インフォメーション・センター CD を使用することによって、 DB2 インフォメーション・センターをインストールする必要があります。以下に示す 3 つのシナリオを参考にすることにより、自分にとって、または自分の作業環境においてどの方法で DB2 インフォメーション・センターにアクセスするのが最善かを決定し、インストールに関して考慮しなければならない問題は何かを判別してください。

シナリオ: IBM Web サイトから DB2 インフォメーション・センターにアクセスする場合:

コリン氏は、ある教育機関の IT 関係の顧問をしています。彼は、データベース・テクノロジーや SQL を専門としており、北米全体の企業を対象として、 DB2 Universal Database Express Edition を使用したセミナーを開催しています。コリン氏のセミナーのある部分では、教材として DB2 のドキュメンテーションを使用します。たとえば、SQL の講座においてコリン氏は、データベース照会の基本的な構文や上級者向けの構文を教える手段として、 SQL に関する DB2 ドキュメンテーションを使用します。

コリン氏がセミナーを開く企業のほとんどにおいて、インターネットへのアクセスが可能です。それでコリン氏は、自分のモバイル・コンピューターに DB2 Universal Database Express Edition の最新版をインストールする際に、 IBM Web サイトにある DB2 インフォメーション・センターにアクセスするように設定することにします。それによりコリン氏は、セミナーの中で、最新の DB2 ドキュメンテーションにオンラインでアクセスできます。

しかし、コリン氏が旅行中はインターネットにアクセスできません。これは少し問題です。特に、セミナーの準備のために DB2 ドキュメンテーションにアクセスすることが必要になった場合に困ります。そのような状況に対処するため、コリン氏は、自分のモバイル・コンピューターにも DB2 インフォメーション・センターのコピーをインストールすることにします。

DB2 ドキュメンテーションのコピーがいつでも手元にあるので、コリン氏は柔軟に対応することができるようになりました。 **db2set** コマンドを使用して自分のモバイル・コンピューターのレジストリー変数の設定を容易に変更することにより、状況に応じて、IBM Web サイトの DB2 インフォメーション・センターにアクセスしたり、自分のモバイル・コンピューター上にあるものを利用したりできます。

シナリオ: イントラネット・サーバー上の DB2 インフォメーション・センターにアクセスする場合:

エバ氏は、ある生命保険会社の主任データベース管理者です。管理者としての彼女の責任には、 DB2 Universal Database Enterprise Server Edition の最新バージョンを会社の UNIX データベース・サーバーにインストールおよび構成することが含まれます。彼女の会社では、最近従業員に対して、セキュリティー上の理由により業務中はインターネットへのアクセスができなくなることが通知されました。彼女の会社はネットワーク環境にあるため、エバ氏は DB2 インフォメーション・センターのコピーをイントラネット・サーバーにインストールすることにします。そのようにすれば、その会社のデータウェアハウスを定常的に使用する従業員の全員 (営業員、営業部長、および経営分析担当者) が DB2 インフォメーション・センターにアクセスできます。

DB2 インフォメーション・センターをイントラネット・サーバーにインストールする際にエバ氏は、 DB2 セットアップ・ウィザードから、 DB2 インフォメーション・センターがネットワーク上の他のコンピューターからの通信を受信するためのポートを指定するよう求められます。そこで彼女は、 DB2 インフォメーション・センターをインストールするイントラネット・サーバーのサービス名とポート番号を指定します。

次にエバ氏は、自分のデータベース・チームに対して、応答ファイルをしようして全従業員のコンピューターに DB2 Universal Database の最新バージョンをインストールするよう指示します。それにより各コンピューターは、イントラネット・サーバーのホスト名とポート番号を使用して DB2 インフォメーション・センターにアクセスするよう構成されます。

ところが、エバ氏のチームのデータベース管理者の一人であるミゲルが指示を聞き間違えて、イントラネット・サーバー上の DB2 インフォメーション・センターにアクセスするよう DB2 Universal Database を構成するのでなく、何人かの従業員のコンピューターに DB2 インフォメーション・センターのコピーをインストールしてしまいました。この状況を正すためエバ氏は、ミゲルに対して、 **db2set** コマンドを使用することにより、それらの各コンピューターの DB2 インフォメーション・センター・レジストリー変数 (ホスト名の変数は DB2_DOCHOST、ポート番号の変数は DB2_DOCPORT) を変更するように指示します。これで、ネットワーク上の該当するすべてのコンピューターは、 DB2 インフォメーション・センターにアクセスできるようになり、従業員は DB2 に関する質問の回答を DB2 ドキュメンテーションから調べることができるようになりました。

シナリオ: 自分のコンピューター上の DB2 インフォメーション・センターにアクセスする場合:

ツーチェン氏は、小さな町で工場を経営しています。そこにはインターネット・アクセスを提供する地元の ISP がありません。彼は、在庫管理、製品注文情報、銀行口座の情報、および経費を管理するために、DB2 Universal Database Personal Edition を購入しました。それまで一度も DB2 製品を使用したことがなかったので、ツーチェン氏は、その使い方を調べるため DB2 製品資料を必要としています。

標準のインストール・オプションを使用して DB2 Universal Database Personal Edition を自分のコンピューターにインストールした後、ツーチェン氏は DB2 ドキュメンテーションを開こうとします。しかしブラウザーに、開こうとしたページが見つからないというエラー・メッセージが表示されます。ツーチェン氏は、「*DB2 Universal Database Personal Edition 概説およびインストール*」を見て、コンピューター上の DB2 ドキュメンテーションにアクセスするには DB2 インフォメーション・センターをインストールする必要があることを知ります。そこで、メディア・パックから、 *DB2 インフォメーション・センター CD* を取り出し、それをインストールします。

これでツーチェン氏は、オペレーティング・システムのアプリケーション・ランチャーから DB2 インフォメーション・センターを利用できるようになり、 DB2 製品を利用して作業効率を改善する方法について調べることができるようになりました。

DB2 ドキュメンテーションへのアクセス方法のサマリー:

各作業環境に応じて、 DB2 インフォメーション・センターに含まれる DB2 製品資料にアクセスするためにどのオプションが最善かを、次の表にまとめます。

インターネット・アクセス	イントラネット・アクセス	推奨
はい	はい	IBM Web サイトの DB2 インフォメーション・センターにアクセスするか、イントラネット・サーバーにインストールされている DB2 インフォメーション・センターにアクセスする。
はい	いいえ	IBM Web サイトの DB2 インフォメーション・センターにアクセスする。
いいえ	はい	イントラネット・サーバーにインストールされている DB2 インフォメーション・センターにアクセスする。
いいえ	いいえ	ローカル・コンピューター上の DB2 インフォメーション・センターにアクセスする。

関連概念:

- 400 ページの『DB2 インフォメーション・センター』

関連タスク:

- 411 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 405 ページの『DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール (UNIX)』
- 407 ページの『DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール (Windows)』

関連資料:

- 「コマンド・リファレンス」の『db2set - DB2 プロファイル・レジストリー・コマンド』

DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール (UNIX)

DB2 製品資料にアクセスするには、IBM Web サイトからアクセスする方法、イントラネット・サーバーからアクセスする方法、そしてコンピューター上にインストールしてアクセスする方法の 3 種類の方法があります。 DB2 製品は、デフォルトで IBM Web サイトから DB2 ドキュメンテーションにアクセスするようになっています。イントラネット・サーバーまたは自分のコンピューターから DB2 ドキュメンテーションにアクセスするには、 *DB2 インフォメーション・センター CD* からドキュメンテーションをインストールする必要があります。 DB2 セットアップ・ウィザードを使用すると、インストール・システムのさまざまな設定値を定義し、 UNIX オペレーティング・システムを使用するコンピューターに DB2 インフォメーション・センターをインストールすることができます。

前提条件:

ここでは、 UNIX コンピューターに DB2 インフォメーション・センターをインストールするためのハードウェア、オペレーティング・システム、ソフトウェア、および通信の要件のリストを示します。

- **ハードウェア要件**

  以下のうちいずれか 1 つのプロセッサーが必要です。
  - PowerPC (AIX)
  - HP 9000 (HP-UX)
  - Intel 32 ビット (Linux)
  - Solaris UltraSPARC コンピューター (Solaris オペレーティング環境)
- **オペレーティング・システム要件**

  以下のうちいずれか 1 つのオペレーティング・システムが必要です。
  - IBM AIX 5.1 (PowerPC)
  - HP-UX 11i (HP 9000)
  - Redhat Linux 8.0 (Intel 32 ビット)
  - SuSE Linux 8.1 (Intel 32 ビット)
  - Sun Solaris バージョン 8 (Solaris オペレーティング環境 UltraSPARC コンピューター)
- **ソフトウェア要件**
  - 以下のブラウザーがサポートされています。
    - Mozilla バージョン 1.0 以上
- DB2 セットアップ・ウィザードは、グラフィック・インストーラーです。ご使用のコンピューターで DB2 セットアップ・ウィザードを実行するには、グラフィカル・ユーザー・インターフェースを表示できる X Window System ソフトウェアが必要です。 DB2 セットアップ・ウィザードを実行する前に、ディスプレイを正しくエクスポートしたことを確認してください。たとえば、以下のコマンドをコマンド・プロンプトに入力します。

  ```
  export DISPLAY=9.26.163.144:0.
  ```

- **通信要件**
  - TCP/IP

手順:

DB2 セットアップ・ウィザードを使用して DB2 インフォメーション・センターをインストールするには、

1. システムにログオンします。
2. DB2 インフォメーション・センター CD をシステムに挿入し、マウントします。
3. 次のコマンドを入力することによって、 CD がマウントされているディレクトリーに移動します。

   ```
   cd /cd
   ```

   /cd は CD のマウント・ポイントです。
4. **./db2setup** コマンドを入力して、 DB2 セットアップ・ウィザードを開始します。
5. **「IBM DB2 セットアップ・ランチパッド」** が表示されます。 DB2 インフォメーション・センターのインストールに直接進むには、**「製品のインストール」**をクリックします。残りのステップについて説明しているオンライン・ヘルプを利用できます。オンライン・ヘルプを表示するには、**「ヘルプ」**をクリックします。**「キャンセル (Cancel)」**を押せば、いつでもインストールを終了できます。
6. **「インストールしたい製品を選択します」**ウィンドウで、**「次へ」**をクリックします。
7. **「DB2 インフォメーション・センターの DB2 セットアップ・ウィザードへようこそ」**ウィンドウで、**「次へ」**をクリックします。 DB2 セットアップ・ウィザードにより、プログラムのセットアップ・プロセスが案内されます。
8. インストールを続行するには、ご使用条件に同意する必要があります。**「ご使用条件」**ウィンドウで、**「使用条件の条項に同意します」**を選択し、**「次へ」**をクリックします。
9. **「インストール・アクションの選択」**ウィンドウで、 DB2 インフォメーション・センターのインストール先を選択します。後で応答ファイルを使用してこのコンピューターまたは他のコンピューターに DB2 インフォメーション・センターをインストールする場合は、**「設定を応答ファイルに保管する」**を選択します。**「次へ (Next)」**をクリックします。
10. **「インストールする言語の選択」**ウィンドウで、 DB2 インフォメーション・センターのインストール言語を選択します。**「次へ (Next)」**をクリックします。
11. **「DB2 インフォメーション・センター・ポートの指定」**で、 DB2 インフォメーション・センターが受け付ける通信について構成します。**「次へ」**をクリックして、インストールを続行します。
12. **「ファイルのコピーの開始」**ウィンドウで、それまでに選択したインストール・オプションを確認します。設定を確認または変更するには、**「戻る」**をクリックします。**「インストール」**をクリックすると、 DB2 インフォメーション・センターのファイルがコンピューターにコピーされます。

DB2 インフォメーション・センターは、応答ファイルを使用してインストールすることもできます。

インストール・ログ db2setup.his、db2setup.log、および db2setup.err は、デフォルトでは /tmp ディレクトリーに入っています。ログ・ファイルの位置は指定可能です。

db2setup.log ファイルには、エラーを含めてすべての DB2 製品インストール情報が入れられます。 db2setup.his ファイルには、そのコンピューター上のすべての DB2 製品インストールが記録されます。 DB2 は db2setup.log ファイルを db2setup.his ファイルに付加します。 db2setup.err ファイルには、例外やトラップ情報など、Java から戻されたエラー出力が入れられます。

インストールが完了すると、使用している UNIX オペレーティング・システムに応じて、以下のいずれかのディレクトリーに DB2 インフォメーション・センターがインストールされています。

- AIX: /usr/opt/db2_08_01
- HP-UX: /opt/IBM/db2/V8.1
- Linux: /opt/IBM/db2/V8.1
- Solaris オペレーティング環境 /opt/IBM/db2/V8.1

関連概念:

- 400 ページの『DB2 インフォメーション・センター』
- 402 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 「*インストールおよび構成 補足*」の『応答ファイルによる DB2 のインストール (UNIX)』
- 411 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 412 ページの『DB2 インフォメーション・センターのトピックを特定の言語で表示する方法』
- 410 ページの『DB2 インフォメーション・センターの呼び出し』
- 407 ページの『DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール (Windows)』

DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール (Windows)

DB2 製品資料にアクセスするには、IBM Web サイトからアクセスする方法、イントラネット・サーバーからアクセスする方法、そしてコンピューター上にインストールしてアクセスする方法の 3 種類の方法があります。 DB2 製品は、デフォルトで IBM Web サイトから DB2 ドキュメンテーションにアクセスするようになっています。イントラネット・サーバーまたは自分のコンピューターから DB2 ドキュメンテーションにアクセスするには、 *DB2 インフォメーション・センター CD* から DB2 ドキュメンテーションをインストールする必要があります。 DB2 セットアップ・ウィザードを使用すると、インストール・システムのさまざまな設定値を定

義し、 Windows オペレーティング・システムを使用するコンピューターに DB2 インフォメーション・センターをインストールすることができます。

前提条件:

ここでは、 Windows に DB2 インフォメーション・センターをインストールするためのハードウェア、オペレーティング・システム、ソフトウェア、および通信の要件のリストを示します。

- **ハードウェア要件**

  以下のプロセッサーが必要です。

  – 32 ビット・コンピューター: Pentium または Pentium と互換の CPU。

- **オペレーティング・システム要件**

  以下のうちいずれか 1 つのオペレーティング・システムが必要です。

  – Windows 2000
  – Windows XP

- **ソフトウェア要件**
  – 次のブラウザーがサポートされています。
    - Mozilla 1.0 以上
    - Internet Explorer バージョン 5.5 または 6.0 (Windows XP の場合はバージョン 6.0)

- **通信要件**
  – TCP/IP

手順:

DB2 セットアップ・ウィザードを使用して DB2 インフォメーション・センターをインストールするには、

1. DB2 インフォメーション・センターのインストールのために定義したアカウントで、システムにログオンします。
2. CD をドライブに挿入します。自動実行機能を使用可能にしている場合には、それによって IBM DB2 セットアップ・ランチパッドが起動されます。
3. DB2 セットアップ・ウィザードによりシステム言語が判別され、その言語用のセットアップ・プログラムが起動されます。セットアップ・プログラムを英語以外の言語で実行したい場合や、セットアップ・プログラムが自動始動に失敗する場合には、 DB2 セットアップ・ウィザードを手動で開始することができます。

   次のようにして、DB2 セットアップ・ウィザードを手動で開始します。

   a. **「スタート」**をクリックし、**「ファイル名を指定して実行」**を選択します。
   b. **「開く」**フィールドで、次のコマンドを入力します。

      ```
      x:¥setup language
      ```

      x: は CD のドライブ、 *language* はセットアップ・プログラムが実行されている言語です。

   c. **「OK」**をクリックします。

4. **「IBM DB2 セットアップ・ランチパッド」** が表示されます。 DB2 インフォメーション・センターのインストールに直接進むには、**「製品のインストール」**をクリックします。残りのステップについて説明しているオンライン・ヘルプを利用できます。オンライン・ヘルプを表示するには、**「ヘルプ」**をクリックします。**「キャンセル (Cancel)」**を押せば、いつでもインストールを終了できます。
5. **「インストールしたい製品を選択します」**ウィンドウで、**「次へ」**をクリックします。
6. **「DB2 インフォメーション・センターの DB2 セットアップ・ウィザードへようこそ」**ウィンドウで、**「次へ」**をクリックします。 DB2 セットアップ・ウィザードにより、プログラムのセットアップ・プロセスが案内されます。
7. インストールを続行するには、ご使用条件に同意する必要があります。**「ご使用条件」**ウィンドウで、**「使用条件の条項に同意します」**を選択し、**「次へ」**をクリックします。
8. **「インストール・アクションの選択」**ウィンドウで、 DB2 インフォメーション・センターのインストール先を選択します。後で応答ファイルを使用してこのコンピューターまたは他のコンピューターに DB2 インフォメーション・センターをインストールする場合は、**「設定を応答ファイルに保管する」**を選択します。**「次へ (Next)」**をクリックします。
9. **「インストールする言語の選択」**ウィンドウで、 DB2 インフォメーション・センターのインストール言語を選択します。**「次へ (Next)」**をクリックします。
10. **「DB2 インフォメーション・センター・ポートの指定」**で、 DB2 インフォメーション・センターが受け付ける通信について構成します。**「次へ」**をクリックして、インストールを続行します。
11. **「ファイルのコピーの開始」**ウィンドウで、それまでに選択したインストール・オプションを確認します。設定を確認または変更するには、**「戻る」**をクリックします。**「インストール」**をクリックすると、 DB2 インフォメーション・センターのファイルがコンピューターにコピーされます。

DB2 インフォメーション・センターは、応答ファイルを使用してインストールできます。また、**db2rspgn** コマンドを使用することによって、既存のインストール・システムに基づく応答ファイルを生成することもできます。

インストール時に検出されるエラーの詳細については、 `db2.log` と `db2wi.log` のファイルを参照してください。それらのファイルは、`'My Documents'¥DB2LOG¥` ディレクトリーに入っています。 `My Documents` ディレクトリーのロケーションは、ご使用のコンピューターの設定によって異なります。

db2wi.log ファイルには、最後の DB2 インストール情報が入れられます。 db2.log には、DB2 製品インストールの履歴が入れられます。

関連概念:

- 400 ページの『DB2 インフォメーション・センター』
- 402 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 「インストールおよび構成 補足」の『応答ファイルによる DB2 製品のインストール (Windows)』
- 411 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 412 ページの『DB2 インフォメーション・センターのトピックを特定の言語で表示する方法』
- 410 ページの『DB2 インフォメーション・センターの呼び出し』
- 405 ページの『DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール (UNIX)』

関連資料:

- 「コマンド・リファレンス」の『db2rspgn - 応答ファイル生成プログラム・コマンド』

DB2 インフォメーション・センターの呼び出し

DB2 インフォメーション・センターは、Linux、UNIX、および Windows オペレーティング・システム用の DB2 製品 (DB2 Universal Database、 DB2 Connect、DB2 Information Integrator、 DB2 Query Patroller など) を使用するために必要なすべての情報を提供します。

DB2 インフォメーション・センターは、以下の場所から呼び出すことができます。

- DB2 UDB クライアントまたはサーバーがインストールされているコンピューター
- DB2 インフォメーション・センターがインストールされているイントラネット・サーバーまたはローカル・コンピューター
- IBM の Web サイト

前提条件:

DB2 インフォメーション・センターを呼び出すための要件は、以下のとおりです。

- *オプショナル*: 希望する言語でトピックを表示するようブラウザーを構成する
- *オプショナル*: コンピューターまたはイントラネット・サーバーにインストール済みの DB2 インフォメーション・センターを使用するよう DB2 クライアントを構成する

手順:

DB2 UDB クライアントまたはサーバーがインストールされているコンピューターから DB2 インフォメーション・センターを呼び出すには、以下のようにします。

- (Windows オペレーティング・システムの)「スタート」メニューから: **「スタート」→「プログラム」→「IBM DB2」→「情報」→「インフォメーション・センター」**をクリックします。
- コマンド行プロンプトから:
  - Linux および UNIX オペレーティング・システムの場合、 **db2icdocs** コマンドを発行します。
  - Windows オペレーティング・システムの場合、 **db2icdocs.exe** コマンドを発行します。

イントラネット・サーバーまたはローカル・コンピューターにインストール済みの DB2 インフォメーション・センターを Web ブラウザーで開くには、以下のようにします。

- Web ページ http://<host-name>:<port-number>/ を開きます (<host-name> はホスト名、 <port-number> は DB2 インフォメーション・センターを利用可能なポート番号)。

IBM Web サイトにある DB2 インフォメーション・センターを Web ブラウザーで開くには、以下のようにします。

- Web ページ publib.boulder.ibm.com/infocenter/db2help/ を開きます。

関連概念:

- 400 ページの『DB2 インフォメーション・センター』

関連タスク:

- 420 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 411 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 422 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 422 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』
- 423 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』

コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール

http://publib.boulder.ibm.com/infocenter/db2help/ から利用できる DB2 インフォメーション・センターは、資料の新規追加または変更によって定期的に更新されます。さらに、更新された DB2 インフォメーション・センターをコンピューターまたはイントラネット・サーバーにダウンロードしてインストールできる場合もあります。DB2 インフォメーション・センターを更新しても、 DB2 クライアント製品またはサーバー製品は更新されません。

前提条件:

インターネットに接続されたコンピューターへのアクセスが必要です。

手順:

DB2 インフォメーション・センターの更新をコンピューターまたはイントラネット・サーバーにインストールするには、以下のようにします。

1. IBM の Web サイト (http://publib.boulder.ibm.com/infocenter/db2help/) にある DB2 インフォメーション・センターを開きます。
2. 「DB2 インフォメーション・センターにようこそ」ページの見出し「サービスおよびサポート」の「ダウンロード」セクションで、**「DB2 資料」**リンクをクリックします。
3. 最新のドキュメンテーション・イメージのレベルと、インストール済みのドキュメンテーション・レベルを比較して、 DB2 インフォメーション・センターを更

新する必要があるかどうかを確認します。「DB2 インフォメーション・センターにようこそ」ページに、インストール済みのドキュメンテーションのレベルがリストされます。

4. より新しいバージョンの DB2 インフォメーション・センターが存在する場合、ご使用のオペレーティング・システムに対応する最新の *DB2* インフォメーション・センター・イメージをダウンロードします。
5. 最新の *DB2* インフォメーション・センター・イメージをインストールするには、 Web ページの指示に従ってください。

関連概念:

- 402 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 410 ページの『DB2 インフォメーション・センターの呼び出し』
- 405 ページの『DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール (UNIX)』
- 407 ページの『DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール (Windows)』

DB2 インフォメーション・センターのトピックを特定の言語で表示する方法

DB2 インフォメーション・センターを表示した場合、ブラウザーの設定値で指定された言語でトピックを表示することが試行されます。希望する言語に翻訳されていないトピックがある場合、そのトピックは英語で表示されます。

手順:

Internet Explorer ブラウザーにおいて、希望する言語でトピックを表示するには、

1. Internet Explorer で、**「ツール」**→**「インターネット オプション」**→**「言語...」**ボタンをクリックします。「言語の優先順位」ウィンドウが表示されます。
2. 言語リストの中で、希望する言語が最初の項目として指定されていることを確認してください。
   - 新しい言語をリストに追加するには、**「追加...」**ボタンをクリックします。

     注: 言語を追加しても、その言語でトピックを表示するために必要なフォントがそのコンピューター上にあることが保証されるわけではありません。
   - ある言語をリストの先頭に移動するには、その言語を選択してから、その言語が言語リストの先頭になるまで**「上へ」**ボタンをクリックします。
3. DB2 インフォメーション・センターが希望する言語で表示されるようにするため、ページをリフレッシュします。

Mozilla ブラウザーの場合に、希望する言語でトピックを表示するには、

1. Mozilla で、**「編集 (Edit)」**→**「設定 (Preferences)」**→**「言語 (Languages)」**ボタンを選択します。「設定 (Preferences)」ウィンドウに「言語 (Languages)」パネルが表示されます。

2. 言語リストの中で、希望する言語が最初の項目として指定されていることを確認してください。
   - 新しい言語をリストに追加するには、**「追加... (Add...)」**ボタンをクリックします。
   - ある言語をリストの先頭に移動するには、その言語を選択してから、その言語が言語リストの先頭になるまで**「上へ」**ボタンをクリックします。
3. DB2 インフォメーション・センターが希望する言語で表示されるようにするため、ページをリフレッシュします。

関連概念:

- 400 ページの『DB2 インフォメーション・センター』

DB2 PDF 資料および印刷された資料

以下の表は、正式な資料名、資料番号、および PDF ファイル名を示しています。ハードコピー版の資料を注文するには、正式な資料名を知っておく必要があります。 PDF ファイルを印刷するには、PDF ファイル名を知っておく必要があります。

DB2 資料は、以下のカテゴリーに分類されています。

- DB2 中核情報
- 管理情報
- アプリケーション開発情報
- ビジネス・インテリジェンス情報
- DB2 Connect 情報
- 入門情報
- チュートリアル情報
- オプショナル・コンポーネント情報
- リリース・ノート

以下の表は、DB2 ライブラリー内の各資料について、その資料のハードコピー版を注文したり、PDF 版を印刷または表示したりするのに必要な情報を示しています。DB2 ライブラリー内の各資料に関する詳細な説明については、www.ibm.com/shop/publications/order にある IBM Publications Center にアクセスしてください。

DB2 の基本情報

こうした資料の情報は、すべての DB2 ユーザーに基本的なもので、プログラマーおよびデータベース管理者にとって役立つ情報であるとともに、DB2 Connect、DB2 Warehouse Manager、または他の DB2 製品を使用するユーザーにとっても役立つ内容です。

表 41. DB2 の基本情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database コマンド・リファレンス*」	SC88-9140	db2n0j81

表 41. DB2 の基本情報 (続き)

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database 用語集*」	資料番号なし	db2t0j81
「*IBM DB2 Universal Database メッセージ・リファレンス 第 1 巻*」	GC88-9152 (ハードコピーなし)	db2m1j81
「*IBM DB2 Universal Database メッセージ・リファレンス 第 2 巻*」	GC88-9153 (ハードコピーなし)	db2m2j81
「*IBM DB2 Universal Database 新機能*」	SC88-9158	db2q0j81

管理情報

これらの資料の情報は、DB2 データベース、データウェアハウス、およびフェデレーテッド・システムを効果的に設計し、インプリメントし、保守するために必要なトピックを扱っています。

表 42. 管理情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database 管理ガイド: プランニング*」	SC88-9135	db2d1j81
「*IBM DB2 Universal Database 管理ガイド: インプリメンテーション*」	SC88-9133	db2d2j81
「*IBM DB2 Universal Database 管理ガイド: パフォーマンス*」	SC88-9134	db2d3j81
「*IBM DB2 Universal Database 管理 API リファレンス*」	SC88-9136	db2b0j81
「*IBM DB2 Universal Database データ移動ユーティリティー ガイドおよびリファレンス*」	SC88-9142	db2dmj81
「*IBM DB2 Universal Database データ・リカバリーと高可用性 ガイドおよびリファレンス*」	SC88-9143	db2haj81
「*IBM DB2 Universal Database データウェアハウス・センター 管理ガイド*」	SC88-9165	db2ddj81
「*IBM DB2 Universal Database SQL リファレンス 第 1 巻*」	SC88-9155	db2s1j81
「*IBM DB2 Universal Database SQL リファレンス 第 2 巻*」	SC88-9156	db2s2j81
「*IBM DB2 Universal Database システム・モニター ガイドおよびリファレンス*」	SC88-9157	db2f0j81

アプリケーション開発情報

これらの資料の情報は、 DB2 Universal Database (DB2 UDB) のアプリケーション開発者またはプログラマーが特に関心を持つ内容です。サポートされるさまざまなプログラミング・インターフェース (組み込み SQL、ODBC、JDBC、SQLJ、CLI など) を使用して DB2 UDB にアクセスするのに必要な資料とともに、サポートされる言語およびコンパイラーについても紹介されています。また、DB2 インフォメーション・センターをご使用の場合には、サンプル・プログラムのソース・コードの HTML バージョンにアクセスすることもできます。

表 43. アプリケーション開発情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database アプリケーション開発ガイド アプリケーションの構築および実行*」	SC88-9137	db2axj81
「*IBM DB2 Universal Database アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」	SC88-9138	db2a1j81
「*IBM DB2 Universal Database アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」	SC88-9139	db2a2j81
「*IBM DB2 Universal Database コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」	SC88-9159	db2l1j81
「*IBM DB2 Universal Database コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」	SC88-9160	db2l2j81
「*IBM DB2 Universal Database データウェアハウス・センター アプリケーション統合ガイド*」	SC88-9166	db2adj81
「*IBM DB2 Universal Database XML Extender 管理およびプログラミングのガイド*」	SC88-9172	db2sxj81

ビジネス・インテリジェンス情報

これらの資料の情報は、さまざまなコンポーネントを使用して、 DB2 Universal Database のデータウェアハウジング機能および分析機能を拡張する方法を説明しています。

表 44. ビジネス・インテリジェンス情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Warehouse Manager Standard Edition インフォメーション・カタログ・センター 管理ガイド*」	SC88-9167	db2dij81
「*IBM DB2 Warehouse Manager Standard Edition インストール・ガイド*」	GC88-9164	db2idj81
「*IBM DB2 Warehouse Manager Standard Edition DB2 Warehouse Manager を使用時の ETI ソリューション・コンバージョン・プログラムの管理*」	SC88-9894	iwhe1mstx80

DB2 Connect 情報

このカテゴリーの情報は、 DB2 Connect Enterprise Edition または DB2 Connect Personal Edition を使用して、メインフレーム・サーバーおよびミッドレンジ・サーバー上のデータにアクセスする方法を説明しています。

表 45. DB2 Connect 情報

資料名	資料番号	PDF ファイル名
「*IBM コネクティビティー 補足*」	資料番号なし	db2h1j81
「*IBM DB2 Connect Enterprise Edition 概説およびインストール*」	GC88-9145	db2c6j81
「*IBM DB2 Connect Personal Edition 概説およびインストール*」	GC88-9146	db2c1j81
「*IBM DB2 Connect ユーザーズ・ガイド*」	SC88-9147	db2c0j81

入門情報

このカテゴリーの情報は、サーバー、クライアント、および他の DB2 製品をインストールして構成する場合に役立ちます。

表 46. 入門情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database DB2 クライアント機能 概説およびインストール*」	GC88-9144 (ハードコピーなし)	db2itj81
「*IBM DB2 Universal Database DB2 サーバー機能 概説およびインストール*」	GC88-9148	db2isj81

表 46. 入門情報 (続き)

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database DB2 Personal Edition 概説およびインストール*」	GC88-9150	db2i1j81
「*IBM DB2 Universal Database インストールおよび構成 補足*」	GC88-9149 (ハードコピーなし)	db2iyj81
「*IBM DB2 Universal Database DB2 Data Links Manager 概説およびインストール*」	GC88-9141	db2z6j81

チュートリアル情報

チュートリアル情報は、DB2 機能を紹介し、さまざまなタスクを実行する方法を示します。

表 47. チュートリアル情報

資料名	資料番号	PDF ファイル名
「*ビジネス・インテリジェンス・チュートリアル: データウェアハウス・センターの紹介*」	資料番号なし	db2tuj81
「*ビジネス・インテリジェンス・チュートリアル: データウェアハウジングの上級者向けガイド*」	資料番号なし	db2taj81
「*インフォメーション・カタログ・センター チュートリアル*」	資料番号なし	db2aij81
「*Video Central for e-business チュートリアル*」	資料番号なし	db2twj81
「*Visual Explain チュートリアル*」	資料番号なし	db2tvj81

オプショナル・コンポーネント情報

このカテゴリーの情報は、 DB2 のオプショナル・コンポーネントを使用する方法について説明しています。

表 48. オプショナル・コンポーネント情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Cube Views Guide and Reference*」	SC18–7298	db2aax81
「*IBM DB2 Query Patroller インストール、管理、使用法のガイド*」	GC88–9154	db2dwj81

表 48. オプショナル・コンポーネント情報 (続き)

資料名	資料番号	PDF ファイル名
「*IBM DB2 Spatial Extender and Geodetic Extender ユーザーズ・ガイドおよびリファレンス*」	SC88-9171	db2sbj81
「*IBM DB2 Universal Database Data Links Manager 管理ガイドおよびリファレンス*」	SC88-9169	db2z0x82
「*DB2 Net Search Extender 管理およびユーザーズ・ガイド*」 **注:** この資料の HTML 版は、HTML ドキュメンテーション CD からインストールされません。	SH88-8546	N/A

リリース・ノート

リリース・ノートは、ご使用の製品のリリースおよびフィックスパック・レベルに特有の追加情報を紹介します。また、リリース・ノートには、各リリース、アップデート、およびフィックスパックで組み込まれた資料上の更新の要約も含まれています。

表 49. リリース・ノート

資料名	資料番号	PDF ファイル名
「*DB2 リリース・ノート*」	「注」を参照。	「注」を参照。
「*DB2 インストール情報*」	製品 CD-ROM でのみ参照可能。	使用できません。

注: リリース・ノートは以下の形式で入手できます。

- XHTML およびテキスト形式 (製品 CD 内)
- PDF 形式 (PDF ドキュメンテーション CD 内)

さらに、リリース・ノートの中で、『*既知の問題と予備手段*』および『*リリース間の非互換性*』に関する部分は DB2 インフォメーション・センターにも表示されます。

UNIX ベースのプラットフォームでテキスト形式でリリース・ノートを確認するには、 `Release.Notes` ファイルを参照してください。このファイルは、`DB2DIR/Readme/%L` ディレクトリーに収録されています。 `%L` はロケール名を表しています。 `DB2DIR` は以下になります。

- AIX オペレーティング・システムの場合: `/usr/opt/db2_08_01`
- その他のすべての UNIX ベースのオペレーティング・システムの場合: `/opt/IBM/db2/V8.1`

関連概念:

- 399 ページの『DB2 ドキュメンテーションおよびヘルプ』

関連タスク:

- 419 ページの『PDF ファイルからの DB2 資料の印刷方法』
- 420 ページの『DB2 の印刷資料の注文方法』
- 420 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』

PDF ファイルからの DB2 資料の印刷方法

DB2 PDF ドキュメンテーション CD に収録されている DB2 資料を印刷することができます。 Adobe Acrobat Reader を使用すれば、資料全体または特定のページを印刷できます。

前提条件:

Adobe Acrobat Reader がインストールされていることを確認してください。 Adobe Acrobat Reader をインストールする必要がある場合、 Adobe Web サイト (www.adobe.com) から入手できます。

手順:

PDF ファイルから DB2 資料を印刷するには以下のようにします。

1. *DB2 PDF* ドキュメンテーション CD をドライブに挿入します。 UNIX オペレーティング・システムの場合、 DB2 PDF ドキュメンテーション CD をマウントします。 UNIX オペレーティング・システムで CD をマウントする方法については、「*概説およびインストール*」を参照してください。
2. index.htm を開きます。ブラウザー・ウィンドウにファイルが開きます。
3. 参照したい PDF のタイトルをクリックします。 Acrobat Reader で PDF が開きます。
4. **「ファイル」** → **「印刷」**を選択して、所要の資料の任意の部分を印刷します。

関連概念:

- 400 ページの『DB2 インフォメーション・センター』

関連タスク:

- 「*DB2 Universal Database サーバー機能 概説およびインストール*」の『CD-ROM のマウント (AIX)』
- 「*DB2 Universal Database サーバー機能 概説およびインストール*」の『HP-UX 上での CD-ROM のマウント』
- 「*DB2 Universal Database サーバー機能 概説およびインストール*」の『CD-ROM のマウント (Linux)』
- 420 ページの『DB2 の印刷資料の注文方法』
- 「*DB2 Universal Database サーバー機能 概説およびインストール*」の『CD-ROM のマウント (Solaris)』

関連資料:

- 413 ページの『DB2 PDF 資料および印刷された資料』

DB2 の印刷資料の注文方法

ハードコピー版の資料を望む場合には、以下のいずれかの方法で注文できます。

印刷資料の注文方法:

一部の国または地域では、印刷された資料を注文することもできます。お客様がお住まいの国または地域でこのサービスが利用可能かどうかを確認するには、お住まいの国または地域の IBM Publications Web サイトをご覧ください。資料のご注文が可能な場合、以下のようにすることができます。

- 正規の IBM 製品販売業者または営業担当員に連絡してください。お客様がお住まいの地域の IBM 担当員の情報については、お手数ですが IBM の Web サイト (www.ibm.com/planetwide) の IBM Worldwide Directory of Contacts で確認してください。
- IBM Publications Center (http://www.ibm.com/shop/publications/order) にアクセスしてください。なお、IBM Publications Center から資料を注文できない国もあります。

DB2 製品がご利用可能になった時点で、印刷された資料は *DB2 PDF* ドキュメンテーション *CD* にある PDF 形式の資料と同じものです。さらに、 *DB2* インフォメーション・センター *CD* に収録されている印刷された資料の内容もまた、これらと同じです。ただし、DB2 インフォメーション・センター CD には、 PDF 資料にない追加情報も含まれます (たとえば、SQL 管理作業や HTML サンプル)。 DB2 PDF ドキュメンテーション CD に収録されている資料の中には、ハードコピーとしてご注文できない資料もあります。

注: DB2 インフォメーション・センターは、 PDF またはハードコピー の資料よりも頻繁に更新されます。ドキュメンテーションの更新が入手可能になった時点でインストールするか、 DB2 インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2help/) を参照して最新の情報を入手してください。

関連タスク:

- 419 ページの『PDF ファイルからの DB2 資料の印刷方法』

関連資料:

- 413 ページの『DB2 PDF 資料および印刷された資料』

DB2 ツールからコンテキスト・ヘルプを呼び出す

コンテキスト・ヘルプは、特定のウィンドウ、ノートブック、ウィザード、またはアドバイザーに関連したタスクまたはコントロールの情報を提供します。コンテキスト・ヘルプは、グラフィカル・ユーザー・インターフェースのある DB2 管理ツールおよび開発ツールから利用できます。コンテキスト・ヘルプには、以下の 2 種類があります。

- それぞれのウィンドウまたはノートブックにある**「ヘルプ」**ボタンからアクセス可能なヘルプ

- infopop (ポップアップ情報ウィンドウ)。これは、マウス・カーソルを特定のフィールドまたはコントロール上に置いたとき、またはウィンドウ、ノートブック、ウィザード、アドバイザー内でフィールドまたはコントロールを選択して F1 を押すと表示されます。

「ヘルプ」ボタンを押すと、概説、前提条件、およびタスク情報が表示されます。infopop は、それぞれのフィールドおよびコントロールについて説明します。

手順:

コンテキスト・ヘルプを呼び出すには、以下のようにします。

- ウィンドウおよびノートブックのヘルプを表示するには、いずれかの DB2 ツールを開始して、任意のウィンドウまたはノートブックを開きます。ウィンドウまたはノートブックの右下隅にある**「ヘルプ」**ボタンをクリックして、コンテキスト・ヘルプを呼び出します。

  また、それぞれの DB2 ツール・センターの上部にある**「ヘルプ」**メニュー項目からコンテキスト・ヘルプにアクセスすることもできます。

  ウィザードおよびアドバイザーでは、最初のページの「タスクの概要」リンクをクリックすると、コンテキスト・ヘルプを表示できます。
- ウィンドウまたはノートブック上の各コントロールの infopop ヘルプを表示するには、コントロールをクリックしてから、**F1** を押します。コントロールの詳細情報を示すポップアップ情報が、黄色いウィンドウに表示されます。

  注: フィールドまたはコントロールにマウス・カーソルを置いておくだけで infopops が表示されるようにするには、「ツール設定」ノートブックの**「文書 (Documentation)」**ページの**「infopops の自動表示」**チェック・ボックスを選択します。

  infopop に似た別のコンテキスト・ヘルプに、診断ポップアップ情報があります。これにはデータ入力規則が示されます。診断ポップアップ情報は、無効または不十分なデータが入力されたとき、紫色のウィンドウに表示されます。診断ポップアップ情報は、以下に関して表示されます。

  – 必須フィールド。
  – 日付フィールドのように、正確なフォーマットを必要とするデータのフィールド。

関連タスク:

- 410 ページの『DB2 インフォメーション・センターの呼び出し』
- 422 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 422 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』
- 423 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』

- 『DB2 UDB ヘルプの使用法: Common GUI help』
- 『DB2 コンテキスト・ヘルプと資料へのアクセスを設定する: Common GUI help』

コマンド行プロセッサーからメッセージ・ヘルプを呼び出す

メッセージ・ヘルプは、メッセージが出された原因と、エラーへの応答として実行すべきアクションを説明します。

手順:

メッセージ・ヘルプを呼び出すには、コマンド行プロセッサーを開いて以下のように入力します。

```
? XXXnnnnn
```

ここで、*XXXnnnnn* は有効なメッセージ ID を表します。

たとえば、`? SQL30081` と入力すると、メッセージ SQL30081 に関するヘルプを表示します。

関連概念:

- 「*メッセージ・リファレンス 第 1 巻*」の『メッセージの概要』

関連資料:

- 「*コマンド・リファレンス*」の『db2 - コマンド行プロセッサーの呼び出しコマンド』

コマンド行プロセッサーからコマンド・ヘルプを呼び出す

コマンド・ヘルプは、コマンド行プロセッサーでのコマンドの構文を説明します。

手順:

コマンド・ヘルプを呼び出すには、コマンド行プロセッサーを開いて以下のように入力します。

```
? command
```

ここで *command* はキーワードまたはコマンド全体を表します。

たとえば、`? catalog` と入力すると、すべての CATALOG コマンドに関するヘルプが表示され、 `? catalog database` と入力すると、 CATALOG DATABASE コマンドのヘルプだけが表示されます。

関連タスク:

- 420 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 410 ページの『DB2 インフォメーション・センターの呼び出し』
- 422 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 423 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』

関連資料:

- 「*コマンド・リファレンス*」の『db2 - コマンド行プロセッサーの呼び出しコマンド』

コマンド行プロセッサーから SQL 状態ヘルプを呼び出す

DB2 Univerrsal Database は、 SQL ステートメントの結果の原因となったと考えられる条件の SQLSTATE 値を戻します。 SQLSTATE ヘルプは、SQL 状態および SQL 状態クラス・コードの意味を説明します。

手順:

SQL 状態ヘルプを呼び出すには、コマンド行プロセッサーを開いて以下のように入力します。

`? sqlstate` または `? class code`

ここで、*sqlstate* は有効な 5 桁の SQL 状態を、*class code* は SQL 状態の最初の 2 桁を表します。

たとえば、`? 08003` を指定すると SQL 状態 08003 のヘルプが表示され、 `? 08` を指定するとクラス・コード 08 のヘルプが表示されます。

関連タスク:

- 410 ページの『DB2 インフォメーション・センターの呼び出し』
- 422 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 422 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』

DB2 チュートリアル

DB2® チュートリアルは、 DB2 Universal Database のさまざまな機能について学習するのを支援します。このチュートリアルでは、アプリケーションの開発、SQL 照会のパフォーマンス調整、データウェアハウスの処理、メタデータの管理、および DB2 を使用した Web サービスの開発の各分野で、段階的なレッスンが用意されています。

はじめに:

インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2help/) から、このチュートリアルの XHTML 版を表示できます。

チュートリアルの中で、サンプル・データまたはサンプル・コードを使用する場合があります。個々のタスクの前提条件については、それぞれのチュートリアルを参照してください。

DB2 Universal Database チュートリアル:

以下に示すチュートリアルのタイトルをクリックすると、そのチュートリアルを表示できます。

ビジネス・インテリジェンス・チュートリアル: データウェアハウス・センターの紹介
データウェアハウス・センターを使用して簡単なデータウェアハウジング・タスクを実行します。

ビジネス・インテリジェンス・チュートリアル: データウェアハウジングの上級者向けガイド
データウェアハウス・センターを使用して高度なデータウェアハウジング・タスクを実行します。

インフォメーション・カタログ・センター・チュートリアル
インフォメーション・カタログを作成および管理して、インフォメーション・カタログ・センターを使用してメタデータを配置し使用します。

Visual Explain チュートリアル
Visual Explain を使用して、パフォーマンスを向上させるために SQL ステートメントを分析し、最適化し、調整します。

DB2 トラブルシューティング情報

DB2® 製品を使用する際に役立つ、トラブルシューティングおよび問題判別に関する広範囲な情報を利用できます。

DB2 ドキュメンテーション
トラブルシューティング情報は、DB2 インフォメーション・センター、および DB2 ライブラリーに含まれる PDF 資料の中でご利用いただけます。DB2 インフォメーション・センターで、 (ブラウザー・ウィンドウの左側の) ナビゲーション・ツリーの「サポートおよびトラブルシューティング (Support and troubleshooting)」ブランチを参照すると、 DB2 トラブルシューティング・ドキュメンテーションの詳細なリストが見つかります。

DB2 Technical Support の Web サイト
現在問題が発生していて、考えられる原因とソリューションを検索したい場合は、 DB2 Technical Support の Web サイトを参照してください。Technical Support サイトには、最新の DB2 出版物、TechNotes、プログラム診断依頼書 (APAR)、フィックスパック、DB2 内部エラー・コードの最新リスト、その他のリソースが用意されています。この知識ベースを活用して、問題に対する有効なソリューションを探し出すことができます。

DB2 Technical Support の Web サイト
(http://www.ibm.com/software/data/db2/udb/winos2unix/support) にアクセスしてください。

DB2 Problem Determination Tutorial Series
DB2 製品で作業中に直面するかもしれない問題を素早く識別し、解決する方法に関する情報を見つけるには、 DB2 Problem Determination Tutorial Series の Web サイトを参照してください。あるチュートリアルでは、使用可能な DB2 問題判別機能およびツールを紹介し、それらをいつ使用すべきかを判断する助けを与えます。別のチュートリアルは、『データベース・エンジン問題判別 (Database Engine Problem Determination)』、『パフォーマンス問題判別 (Performance Problem Determination)』、『アプリケーション問題判別 (Application Problem Determination)』などの関連トピックを扱っています。

DB2 Technical Support
(http://www.ibm.com/software/data/support/pdm/db2tutorials.html) には、 DB2 問題判別チュートリアルがすべて揃っています。

関連概念:

- 400 ページの『DB2 インフォメーション・センター』
- 「*トラブルシューティング・ガイド*」の『Introduction to Problem Determination - DB2 テクニカル・サポートのチュートリアル』

アクセス支援

アクセス支援機能は、身体に障害のある (身体動作が制限されている、視力が弱いなど) ユーザーがソフトウェア製品を十分活用できるように支援します。 DB2® バージョン 8 製品に備わっている主なアクセス支援機能は、以下のとおりです。

- すべての DB2 機能は、マウスの代わりにキーボードを使ってナビゲーションできます。詳細については、『キーボードによる入力およびナビゲーション』を参照してください。
- DB2 ユーザー・インターフェースのフォント・サイズおよび色をカスタマイズすることができます。詳細については、 426 ページの『アクセスしやすい表示』を参照してください。
- DB2 製品は、Java™ Accessibility API を使用するアクセス支援アプリケーションをサポートします。詳細については、 426 ページの『支援テクノロジーとの互換性』を参照してください。
- DB2 資料は、アクセスしやすい形式で提供されています。詳細については、 426 ページの『アクセスしやすい資料』を参照してください。

キーボードによる入力およびナビゲーション

キーボード入力

キーボードだけを使用して DB2 ツールを操作できます。マウスを使って実行できる操作は、キーまたはキーの組み合わせによっても実行できます。標準のオペレーティング・システム・キー・ストロークを使用して、標準のオペレーティング・システム操作を実行できます。

キーまたはキーの組み合わせによって操作を実行する方法について、詳しくは キーボード・ショートカットおよびアクセラレーター: Common GUI help を参照してください。

キーボード・ナビゲーション

キーまたはキーの組み合わせを使用して、 DB2 ツールのユーザー・インターフェースをナビゲートできます。

キーまたはキーの組み合わせによって DB2 ツールをナビゲートする方法の詳細については、キーボード・ショートカットおよびアクセラレーター: Common GUI help を参照してください。

キーボード・フォーカス

UNIX® オペレーティング・システムでは、アクティブ・ウィンドウの中で、キー・ストロークによって操作できる領域が強調表示されます。

アクセスしやすい表示

DB2 ツールには、視力の弱いユーザー、その他の視力障害をもつユーザーのためにアクセシビリティーを向上させる機能が備わっています。これらのアクセシビリティー拡張機能には、フォント・プロパティーのカスタマイズを可能にする機能も含まれています。

フォントの設定

「ツール設定」ノートブックを使用して、メニューおよびダイアログ・ウィンドウに使用されるテキストの色、サイズ、およびフォントを選択できます。

フォント設定に関する詳細情報は、メニューおよびテキストのフォントを変更する: Common GUI help を参照してください。

色に依存しない

本製品のすべての機能を使用するために、ユーザーは必ずしも色を識別する必要はありません。

支援テクノロジーとの互換性

DB2 ツールのインターフェースは、Java Accessibility API をサポートします。これによって、スクリーン・リーダーその他の支援テクノロジーを DB2 製品で利用できるようになります。

アクセスしやすい資料

DB2 形式は、ほとんどの Web ブラウザーで表示可能な XHTML 1.0 形式で提供されています。 XHTML により、ご使用のブラウザーに設定されている表示設定に従って資料を表示できます。さらに、スクリーン・リーダーや他の支援テクノロジーを使用することもできます。

シンタックス・ダイアグラムはドット 10 進形式で提供されます。この形式は、スクリーン・リーダーを使用してオンライン・ドキュメンテーションにアクセスする場合にのみ使用できます。

関連概念:

- 426 ページの『ドット 10 進シンタックス・ダイアグラム』

ドット 10 進シンタックス・ダイアグラム

スクリーン・リーダーを使用してインフォメーション・センターを利用するユーザーのために、シンタックス・ダイアグラムがドット 10 進形式で提供されます。

ドット 10 進形式では、各シンタックス・エレメントは別々の行に書き込まれます。複数のシンタックス・エレメントが常に同時に存在する (または常に同時に不在の) 場合、単一のコンパウンド・シンタックス・エレメントとみなせるので同一行に表示できます。

各行は、ドット 10 進数で開始します。たとえば、3 または 3.1 ないしは 3.1.1 です。こうした数を適切に聞き取るには、スクリーン・リーダーが句読点を読み取るように設定されていることを確認してください。同じドット 10 進数を持つすべて

のシンタックス・エレメント (たとえば、 3.1 という数値を持つすべてのシンタックス・エレメント) は、相互に排他的な代替エレメントです。 3.1 USERID および 3.1 SYSTEMID という行を聞き取る場合、シンタックスには両方ではなく USERID または SYSTEMID のどちらかが含まれることが分かります。

ドット 10 進レベルは、ネストのレベルを表示します。たとえば、ドット 10 進数 3 のシンタックス・エレメントの後に、一連のドット 10 進数 3.1 のシンタックス・エレメントが続きます。 3.1 の番号が付されたシンタックス・エレメントすべては、番号 3 の付されたシンタックス・エレメントに従属します。

シンタックス・エレメントに関する情報を追加するため、ドット 10 進数の次に特定のワードおよびシンボルが使用されます。 時折、こうしたワードおよびシンボルはエレメントの最初に表示される場合もあります。 簡単に識別するため、ワードやシンボルがシンタックス・エレメントの一部である場合には、円記号 (¥) 文字が先頭に付きます。 * シンボルはドット 10 進数の次に使用でき、シンタックス・エレメントが反復することを示します。 たとえば、ドット 10 進数 3 のシンタックス・エレメント *FILE は、3 ¥* FILE という形式になります。 3* FILE という形式は、シンタックス・エレメント FILE が反復されることを示します。 3* ¥* FILE という形式は、シンタックス・エレメント * FILE が反復されることを示します。

シンタックス・エレメントのストリングを分離するのに使用されるコンマなどの文字は、シンタックス内の分離する項目の直前に表示されます。 こうした文字は、それぞれの項目と同一行に表示するか、同じドット 10 進数を持つ関連する項目のある別の行に表示できます。 またその行には、シンタックス・エレメントに関する情報を提供する別のシンボルを表示することも可能です。 たとえば、複数の LASTRUN および DELETE シンタックス・エレメントを使用している場合には、5.1*、5.1 LASTRUN、および 5.1 DELETE という行は、エレメントをコンマで区切る必要があります。区切り文字が指定されないと、各シンタックス・エレメントを区切るのにブランクが使用されると想定されます。

シンタックス・エレメントの前に % シンボルが付く場合、他の箇所で定義されている参照であることを示します。 % シンボルの後のストリングは、リテラルではなくシンタックス・フラグメントの名前です。 たとえば、2.1 %OP1 という行は別のシンタックス・フラグメント OP1 を参照すべきことを意味します。

以下のワードおよびシンボルが、ドット 10 進数の次に使用されます。

- ? は、オプションのシンタックス・エレメントであることを表します。? シンボルが後に続くドット 10 進数は、対応するドット 10 進数のシンタックス・エレメント、および任意の従属のシンタックス・エレメントがオプションであることを示します。ドット 10 進数の付いたシンタックス・エレメントが 1 つしかない場合、 ? シンボルはそのシンタックス・エレメントと同じ行に表示されます (たとえば、5? NOTIFY)。ドット 10 進数の付いたシンタックス・エレメントが複数ある場合、 ? シンボルだけで行に表示され、その後にオプションのシンタックス・エレメントが続きます。たとえば、「5 ?、5 NOTIFY、および 5 UPDATE」という行を聞き取る場合、シンタックス・エレメント NOTIFY および UPDATE がオプションである、つまりそのいずれかを選択でき、どちらも選択しないこともできることが分かります。 ? シンボルは、線路型ダイアグラムのバイパス線に相当します。

- ! は、デフォルトのシンタックス・エレメントであることを表します。! シンボルおよびシンタックス・エレメントが後に続くドット 10 進数は、そのシンタックス・エレメントが、同じドット 10 進数を共用するシンタックス・エレメントすべてのデフォルト・オプションであることを示します。同じドット 10 進数を共用するシンタックス・エレメントのうち 1 つだけに、 ! シンボルを指定できます。たとえば、「2? FILE、2.1! (KEEP)、および 2.1 (DELETE)」という行を聞き取る場合、 FILE キーワードのデフォルト・オプションは (KEEP) になります。この例では、FILE キーワードを含めてもオプションを指定しない場合には、デフォルト・オプション KEEP が適用されます。デフォルト・オプションは、次に高位のドット 10 進数にも適用されます。 この例の場合、FILE キーワードが省略されると、デフォルトの FILE(KEEP) が使用されます。しかし、「2? FILE、2.1、2.1.1! (KEEP)、および 2.1.1 (DELETE)」という行を聞き取る場合、デフォルト・オプション KEEP は次に高位のドット 10 進数 2.1 (関連キーワードを持っていない) にのみ適用され、 2? FILE には適用されません。キーワード FILE が省略されると、どれも使用されません。
- * は、0 回以上反復できるシンタックス・エレメントを示します。 * シンボルが後に続くドット 10 進数は、このシンタックス・エレメントが 0 回以上使用できること、つまりオプションであり、なおかつ反復できることを表します。たとえば、5.1* データ域という行を聞き取る場合、 1 つまたは複数のデータ域を含めるか、またはデータ域を全く含めないことが可能です。「3*、3 HOST、および 3 STATE」という行を聞き取る場合、 HOST、STATE をどちらか一方または両方同時に含めるか、どちらも含めないことができます。

  注:

  1. ドット 10 進数の後にアスタリスク (*) が付き、ドット 10 進数の付いた項目が 1 つしかない場合には、同じ項目を複数回反復できます。
  2. ドット 10 進数の後にアスタリスクが付き、ドット 10 進数の付いた項目が複数ある場合、リストから複数の項目を使用できますが、各項目を複数回使用することはできません。 前述の例では、HOST STATE と書くことはできますが、HOST HOST とは書けません。
  3. * シンボルは、線路型シンタックス・ダイアグラムのループバック線に相当します。
- + は、1 回以上含める必要のあるシンタックス・エレメントであることを示します。 + シンボルが後に続くドット 10 進数は、このシンタックス・エレメントを 1 回以上含める必要があること、つまり少なくとも 1 回は含める必要があり、反復できることを表します。たとえば、「6.1+ データ域」という行を聞き取る場合、データ域を少なくとも 1 回は含めなければなりません。「2+、2 HOST、および 2 STATE」という行を聞き取る場合には、 HOST、STATE、またはその両方を含める必要があります。 * シンボルと同様に、+ シンボルは、ドット 10 進数の付いた項目が 1 つしかない場合に限り、その特定の項目のみを反復できます。 * シンボルと同様、 + シンボルは線路型シンタックス・ダイアグラムのループバック線に相当します。

関連概念:

- 425 ページの『アクセス支援』

関連タスク:

- 『目次』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『構文図の見方』

DB2 Universal Database 製品の共通基準認証

DB2 Universal Database は、 Common Criteria の評価検定レベル 4 (EAL4) で認証の評価を受けています。 Common Criteria の詳細については、以下の Common Criteria の Web サイトを参照してください。 http://niap.nist.gov/cc-scheme/

付録 B. 「DB2 コール・レベル・インターフェース ガイドおよびリファレンス」の特記事項

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、 (i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

本書には、X/Open Company Limited が著作権を持つテキストが含まれています。テキストは、許可を得て下記の資料から収録されています。

X/Open CAE Specification, March 1995,
Data Management: SQL Call Level Interface (CLI)
(ISBN: 1-85912-081-4, C451).

X/Open Preliminary Specification, March 1995,
Data Management: Structured Query Language (SQL), Version 2
(ISBN: 1-85912-093-8, P446).

本書には、Microsoft Corporation が著作権 (1992, 1993, 1994, 1997 年) を持つテキストが含まれています。テキストは、許可を得て下記の Microsoft の資料から収録されています。 *ODBC 2.0 Programmer's Reference and SDK Guide* ISBN 1-55615-658-8 および *ODBC 3.0 Software Development Kit and Programmer's Reference* ISBN 1-57231-516-4.

商標

以下は、IBM Corporation の商標です。

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational
  Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView

OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAgeVM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphereWIN-OS/2
z/OS
zSeries

以下は、それぞれ各社の商標または登録商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Action Media、LANDesk、MMX、 Pentium および ProShare は Intel Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

［マ行］

［ヤ行］

［ラ行］

［数字］

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

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、お客様サポートにご連絡ください。

製品情報

DB2 Universal Database 製品に関する情報は、
http://www.ibm.com/software/data/db2/udb から入手できます。

このサイトには、技術ライブラリー、資料の注文方法、製品のダウンロード、ニュースグループ、フィックスパック、ニュース、および Web リソースへのリンクに関する最新情報が掲載されています。

米国以外の国で IBM に連絡する方法については、 IBM Worldwide ページ (www.ibm.com/planetwide) にアクセスしてください。

Printed in Japan

SC88-9159-01

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12