IBM DB2 10.1
for Linux, UNIX, and Windows

コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻

SA88-4676-00
(英文原典：SC27-3866-00)

IBM DB2 10.1
for Linux, UNIX, and Windows

コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻

SA88-4676-00
(英文原典：SC27-3866-00)

ご注意

本書および本書で紹介する製品をご使用になる前に、 333 ページの『付録 B. 特記事項』に記載されている情報をお読みください。

IBM 資料は、オンラインでご注文いただくことも、ご自分の国または地域の IBM 担当員を通してお求めいただくこともできます。

- オンラインで資料を注文するには、IBM Publications Center (http://www.ibm.com/shop/publications/order) をご利用ください。
- ご自分の国または地域の IBM 担当員を見つけるには、IBM Directory of Worldwide Contacts (http://www.ibm.com/planetwide/) をお調べください。

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。

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： SC27-3866-00
IBM DB2 10.1
for Linux, UNIX, and Windows
Call Level Interface Guide and
Reference Volume 1

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2012.4

目次

本書について

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

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

第 1 章 DB2 コール・レベル・インターフェース と ODBC の紹介

DB2 コール・レベル・インターフェース (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数として渡します。

CLI インターフェースを使用して次に示す IBM® データ・サーバー・データベースにアクセスできます。

- DB2 バージョン 9 for Linux, UNIX, and Windows
- DB2 Universal Database™ バージョン 8 (およびそれ以降) for OS/390® and z/OS®
- DB2 for IBM i 5.4 およびそれ以降
- IBM Informix® バージョン 11.50 (DB2 バージョン 9.7 フィックスパック 1 以降)、バージョン 11.70 (DB2 バージョン 9.7 フィックスパック 3 以降)

CLI 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、これ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。アプリケーションは、さまざまなデータベースに応じて個別にコンパイルしなくても、それらのデータベースに対して実行することができます。アプリケーションは実行時にプロシージャー呼び出しを使用して、データベースへの接続、SQL ステートメントの発行、およびデータと状況情報の取得を行います。

CLI インターフェースは、組み込み SQL では使用できない多くのフィーチャーを提供しています。例えば、以下のようにします。

- CLI は、データベース・カタログを照会する 1 つの方法をサポートする、関数呼び出しを提供します。その方法は、DB2 ファミリーの中で一貫して使用されます。これにより、特定のデータベース・サーバーに合わせなければならないカタログ照会を作成する必要性が少なくなります。
- CLI は、カーソルを次のようにスクロールする機能を提供します。
  - 順方向、1 つ以上の行ずつ
  - 逆方向、1 つ以上の行ずつ
  - 順方向、先頭行から 1 つ以上の行ずつ
  - 逆方向、最後の行から 1 つ以上の行ずつ
  - カーソル内で以前に保管されたロケーションから。
- CLI を使用して作成されたアプリケーション・プログラムから呼び出されるストアード・プロシージャーは、それらのプログラムに結果セットを返すことができます。

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 フィーチャーを特に活用するのに役立ちます。

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。

CLI の背景情報

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、接続要求時に指定されたデータ・ソース (データベース名) に基づいて、ドライバー・マネージャーによってデータベース特定の ODBC ドライバーが実行時に動的にロードされる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実行時に仲介して、それが該当する DBMS 特定の ODBC ドライバーに確実に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。

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ます。これは、http://www.microsoft.com/downloads 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。 DB2 サーバーに接続する ODBC アプリケーションを開発する際は、コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻 および コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻 (DB2 固有

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。

CLI API を使用して書き込まれたアプリケーションは CLI ライブラリーに直接リンクします。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 フィーチャーは、ODBC と CLI の両方のアプリケーションで利用できます。

- 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)

CLI と 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、 CLI ドライバーとの相違点も説明します。

4 ページの図 1 では、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 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 CLI を示します。

Data Server Client は、すべての使用可能な IBM Data Server Client を指します。DB2 サーバーは、Linux、UNIX、および Windows 上のすべての DB2 サーバー製品を指します。

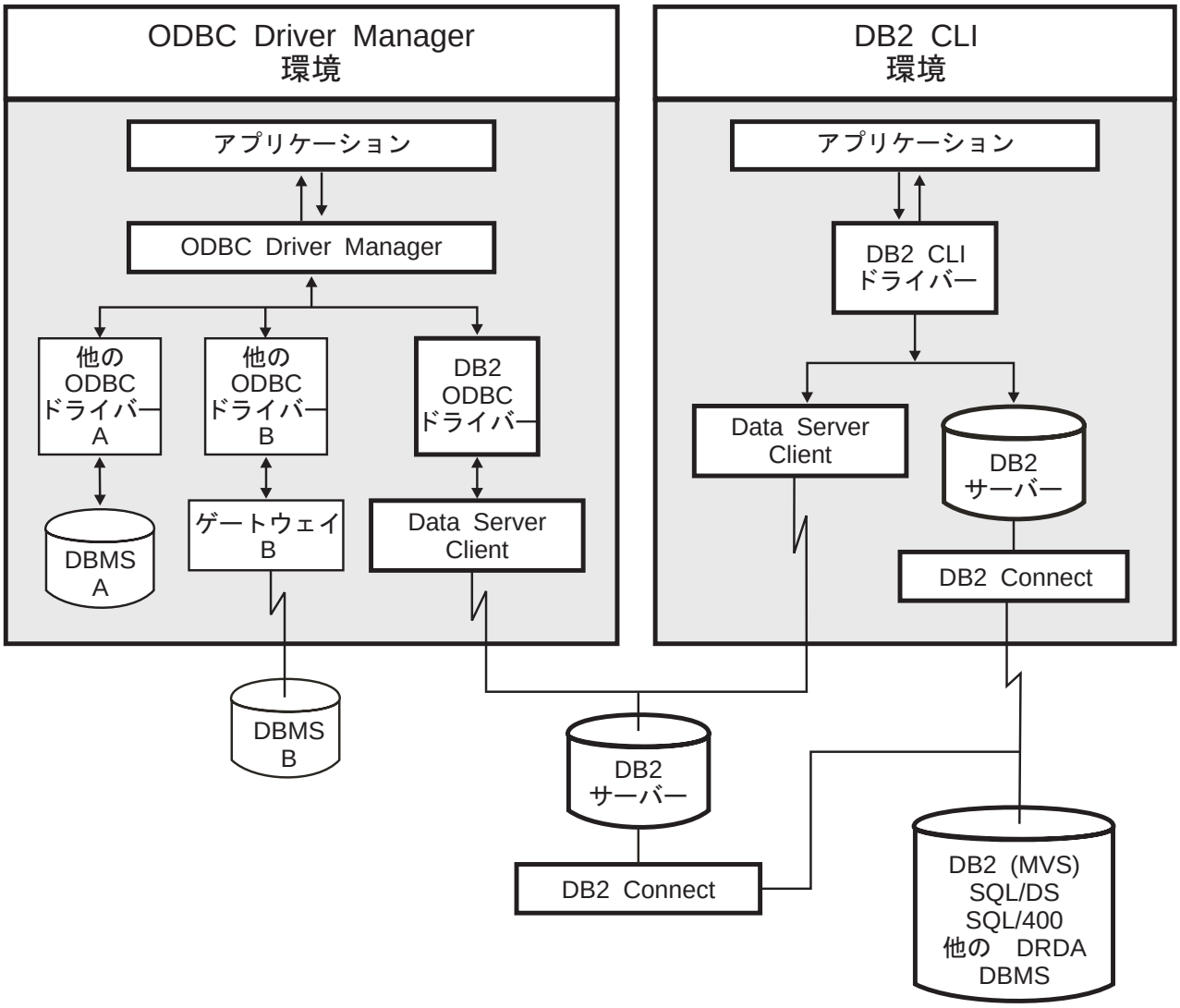


図 1. CLI と ODBC:

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、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、 ODBC SDK は、Microsoft Data Access Components (MDAC) SDK の一部として入手できます。これは、http://www.microsoft.com/downloads 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、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。 5 ページの表 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. CLI ODBC サポート

ODBC フィーチャー	DB2 ODBC Driver	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 CLI 関数	すべて。関数は 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()
SQL データ・タイプ	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_LONGVARBINARY • SQL_LONGVARCHAR • SQL_LONGVARGRAPHIC • SQL_NUMERIC • SQL_REAL • SQL_SMALLINT • SQL_TINYINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_VARBINARY • SQL_VARCHAR • SQL_VARGRAPHIC • SQL_WCHAR

表 1. CLI ODBC サポート (続き)

ODBC フィーチャー	DB2 ODBC Driver	CLI
C データ・タイプ	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_TYPE_DATE • SQL_C_DBCHAR • SQL_C_DBCLOB_LOCATOR • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_SHORT • SQL_C_TYPE_TIME • SQL_C_TYPE_TIMESTAMP • SQL_C_TIMESTAMP_EXT • SQL_C_TINYINT • SQL_C_SBIGINT • SQL_C_UBIGINT • SQL_C_NUMERIC [1] • SQL_C_WCHAR
戻りコード	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	該当しません

注:

1. Windows オペレーティング・システムでのみサポートされます。
2. 以下の SQL データ・タイプは、 ODBC 2.0 との互換性がサポートされています。
   - SQL_DATE
   - SQL_TIME
   - SQL_TIMESTAMP

   データ・タイプ・マッピングを避けるためには、これらの代わりに SQL_TYPE_DATE、SQL_TYPE_TIME、または SQL_TYPE_TIMESTAMP を使用してください。
3. 以下の SQL データ・タイプおよび C データ・タイプは、 ODBC 2.0 との互換性がサポートされています。

- SQL_C_DATE
- SQL_C_TIME
- SQL_C_TIMESTAMP

データ・タイプ・マッピングを避けるためには、これらの代わりに SQL_C_TYPE_DATE、SQL_C_TYPE_TIME、または SQL_C_TYPE_TIMESTAMP を使用してください。

分離レベル

表 2 は、IBM RDBMS 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、使用できる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
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 のフィーチャーと CLI のフィーチャーおよび関数呼び出し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。

第 2 章 IBM Data Server CLI と ODBC ドライバー

IBM Data Server Client および IBM Data Server Runtime Client には、CLI アプリケーション・プログラミング・インターフェース (API) および ODBC API のためのドライバーがあります。DB2 インフォメーション・センターおよび DB2 資料全体で、このドライバーは通常、IBM Data Server CLI ドライバーまたは IBM Data Server CLI/ODBC ドライバーとして参照されています。

DB2 バージョン 9 以降には、IBM Data Server Driver for ODBC and CLI という別の CLI および ODBC ドライバーもあります。IBM Data Server Driver for ODBC and CLI は、CLI および ODBC API に対する実行時サポートを提供します。しかし、このドライバーは別個にインストールおよび構成され、CLI および ODBC API サポートに加えて、接続性などの DB2 クライアント機能のサブセットをサポートします。

DB2 クライアントの一部になっている CLI および ODBC ドライバーに適用される情報は、通常 IBM Data Server Driver for ODBC and CLI にも適用されます。しかし、IBM Data Server Driver for ODBC and CLI に固有な制約事項や機能もあります。IBM Data Server Driver for ODBC and CLI のみに適用される情報では、このドライバーのフルネームを使用して、DB2 クライアントに付属の ODBC および CLI ドライバーに適用される一般情報と区別します。

- IBM Data Server Driver for ODBC and CLI について詳しくは、『IBM Data Server Driver for ODBC and CLI の概要』を参照してください。

IBM Data Server Driver for ODBC and CLI の概要

IBM Data Server Driver for ODBC and CLI は、CLI アプリケーション・プログラミング・インターフェース (API) および ODBC API に対する実行時サポートを提供します。IBM Data Server Client と IBM Data Server Runtime Client はどちらも CLI および ODBC API をサポートしますが、このドライバーは IBM Data Server Client と IBM Data Server Runtime Client のいずれにも含まれていません。これは別個に入手およびインストールされ、IBM Data Server Client 機能のサブセットをサポートします。

IBM Data Server Driver for ODBC and CLI の利点

- このドライバーの占有スペースは IBM Data Server Client および IBM Data Server Runtime Client に比べてかなり少量です。
- 1 つのマシンにこのドライバーを複数インストールすることができます。
- このドライバーは、IBM Data Server Client がすでにインストールされているマシンにもインストールできます。
- ドライバーをデータベース・アプリケーションのインストール・パッケージに含めて、アプリケーションとともにドライバーを再配布することができます。特定の条件のもとで、著作権使用料なしでドライバーをデータベース・アプリケーションとともに再配布できます。

- このドライバーは NFS がマウントされているファイル・システムに常駐可能です。

IBM Data Server Driver for ODBC and CLI の機能

IBM Data Server Driver for ODBC and CLI は次のような機能を提供します。

- CLI API の実行時サポート
- ODBC API の実行時サポート
- XA API の実行時サポート
- データベース接続
- DB2 対話機能コール・レベル・インターフェース (db2cli) のサポート
- LDAP データベース・ディレクトリーのサポート
- トレース、ロギング、および診断のサポート
- 43 ページの『IBM Data Server Driver for ODBC and CLI の制限事項』を参照してください。

IBM Data Server Driver for ODBC and CLI の入手

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client および IBM Data Server Runtime Client には含まれません。インターネットからダウンロードするか、DB2 バージョン 9 インストール CD から入手してください。

手順

以下のソースから IBM Data Server Driver for ODBC and CLI を入手できます。

- IBM Support Fix Central Web サイト (http://www-933.ibm.com/support/fixcentral/) にアクセスする。データ・サーバーのクライアントおよびドライバーの各パッケージは、「Information Management」製品グループで「IBM Data Server Client Packages」製品を選択すると表示されます。適切な「Installed Version」および「Platform」を選択し、「Continue」をクリックします。次の画面で再度「Continue」をクリックすると、 IBM Data Server Driver for ODBC and CLI を含め、ご使用のプラットフォームに使用可能な、クライアントおよびドライバーの全パッケージのリストが表示されます。

  または

- DB2 インストール CD からドライバーをコピーする。

ドライバーは Windows オペレーティング・システムでは「`ibm_data_server_driver_for_odbc_cli.zip`」、その他のオペレーティング・システムでは「`ibm_data_server_driver_for_odbc_cli.tar.Z`」という名前の圧縮ファイルの中にあります。

IBM Data Server Driver for ODBC and CLI のインストール

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client にも IBM Data Server Runtime Client にも含まれません。したがって、別個にインストールする必要があります。

始める前に

IBM Data Server Driver for ODBC and CLI をインストールするには、ドライバーが含まれる圧縮ファイルを取得する必要があります。 10 ページの『IBM Data Server Driver for ODBC and CLI の入手』を参照してください。

このタスクについて

IBM Data Server Driver for ODBC and CLI のインストール・プログラムは存在しません。手動でドライバーをインストールする必要があります。

手順

1. インターネットまたは DB2 バージョン 9 インストール CD から、ドライバーが入っている圧縮ファイルをターゲット・マシンにコピーします。
2. そのファイルを、ターゲット・マシン上の選択したインストール・ディレクトリー内に解凍します。
3. オプション: 圧縮ファイルを削除します。

例

以下の条件のもとで IBM Data Server Driver for ODBC and CLI をインストールする場合、

- ターゲット・マシンのオペレーティング・システムは AIX®
- DB2 バージョン 9 の CD がターゲット・マシンでマウントされている

以下の手順に従ってください。

1. ドライバーのインストール場所に、`$HOME/db2_cli_odbc_driver` というディレクトリーを作成します。
2. インストール CD 上で、圧縮ファイル `ibm_data_server_driver_for_odbc_cli.tar.Z` を見つけます。
3. `ibm_data_server_driver_for_odbc_cli.tar.Z` をインストール・ディレクトリー `$HOME/db2_cli_odbc_driver` にコピーします。
4. 以下のようにして `ibm_data_server_driver_for_odbc_cli.tar.Z` を解凍します。

   ```
   cd $HOME/db2_cli_odbc_driver
   uncompress ibm_data_server_driver_for_odbc_cli.tar.Z
   tar -xvf ibm_data_server_driver_for_odbc_cli.tar
   ```

5. `ibm_data_server_driver_for_odbc_cli.tar.Z` を削除します。
6. ドライバーを NFS ファイル・システム上にインストールした場合は、以下の要件が満たされていることを確認してください。
   - UNIX または Linux オペレーティング・システムでは、`db2dump` および `db2` ディレクトリーが書き込み可能になっている必要があります。あるいは、**diagpath** パラメーターで参照したパスが書き込み可能でなければなりません。
   - ホストまたは i5/OS® データ・サーバーが直接アクセスされる場合、必ず `license` ディレクトリーを書き込み可能にしてください。

IBM Data Server Driver for ODBC and CLI の複数のコピーを同じマシンにインストールする

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client にも IBM Data Server Runtime Client にも含まれません。したがって、別個にインストールする必要があります。

IBM Data Server Driver for ODBC and CLI の複数のコピーを同じマシンにインストールできます。

互いに異なるバージョンのドライバーを必要とする 2 つのデータベース・アプリケーションが同一のマシンに存在する場合には、この方法が適しています。

始める前に

IBM Data Server Driver for ODBC and CLI の複数のコピーを同じマシンにインストールするには、ドライバーが含まれる圧縮ファイルを取得する必要があります。10 ページの『IBM Data Server Driver for ODBC and CLI の入手』を参照してください。

手順

インストール対象の IBM Data Server Driver for ODBC and CLI の各コピーごとに、以下を行います。

1. 固有のターゲット・インストール・ディレクトリーを作成します。
2. 10 ページの『IBM Data Server Driver for ODBC and CLI のインストール』に略述されているインストール手順に従います。
3. アプリケーションがドライバーの正しいコピーを使用していることを確認します。 **LD_LIBRARY_PATH** 環境変数に頼ると、正しくないドライバーを間違えてロードしてしまう可能性があるので、そうすることは避けてください。明示的にターゲット・インストール・ディレクトリーから、ドライバーを動的にロードしてください。

例

以下の条件のもとで IBM Data Server Driver for ODBC and CLI の 2 つのコピーをインストールする場合、

- ターゲット・マシンのオペレーティング・システムは AIX
- DB2 バージョン 9 の CD がターゲット・マシンでマウントされている

以下の手順に従ってください。

1. ドライバーのインストール場所に、`$HOME/db2_cli_odbc_driver1` および `$HOME/db2_cli_odbc_driver2` という 2 つのディレクトリーを作成します。
2. インストール CD 上で、ドライバーが入っている圧縮ファイルを見つけます。このシナリオでは、ファイルは `ibm_data_server_driver_for_odbc_cli.tar.Z` という名前です。

3. ibm_data_server_driver_for_odbc_cli.tar.Z をインストール・ディレクトリー $HOME/db2_cli_odbc_driver1 と $HOME/db2_cli_odbc_driver2 にコピーします。
4. それぞれのディレクトリー内で、以下のようにして ibm_data_server_driver_for_odbc_cli.tar.Z を解凍します。

```
cd $HOME/db2_cli_odbc_driver1
uncompress ibm_data_server_driver_for_odbc_cli.tar.Z
tar -xvf ibm_data_server_driver_for_odbc_cli.tar
cd $HOME/db2_cli_odbc_driver2
uncompress ibm_data_server_driver_for_odbc_cli.tar.Z
tar -xvf ibm_data_server_driver_for_odbc_cli.tar
```

5. ibm_data_server_driver_for_odbc_cli.tar.Z を削除します。

既存の DB2 クライアントが存在するマシンに IBM Data Server Driver for ODBC and CLI をインストールする

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client または IBM Data Server Runtime Client には含まれません。したがって、別個にインストールする必要があります。

IBM Data Server Client または IBM Data Server Runtime Client が既にインストール済みのマシンに、IBM Data Server Driver for ODBC and CLI の 1 つ以上のコピーをインストールすることができます。 IBM Data Server Client を使って開発したいくつかの ODBC または CLI データベース・アプリケーションを IBM Data Server Driver for ODBC and CLI とともにデプロイすることを計画している場合には、この方法が適しています。こうすれば、開発環境と同じマシンのドライバーでデータベース・アプリケーションをテストできるためです。

始める前に

IBM Data Server Driver for ODBC and CLI を IBM Data Server Client または IBM Data Server Runtime Client と同じマシンにインストールするには、以下を行う必要があります。

- ドライバーが格納されている圧縮ファイルを入手する。
  - 10 ページの『IBM Data Server Driver for ODBC and CLI の入手』を参照してください。

このタスクについて

IBM Data Server Client または IBM Data Server Runtime Client が既にインストール済みのマシンに IBM Data Server Driver for ODBC and CLI の 1 つ以上のコピーをインストールする手順は、IBM Data Server Client がまだインストールされていないマシンにドライバーをインストールする手順と同じです。

手順

10 ページの『IBM Data Server Driver for ODBC and CLI のインストール』および 12 ページの『IBM Data Server Driver for ODBC and CLI の複数のコピーを同じマシンにインストールする』を参照してください。

次のタスク

アプリケーションがドライバーの正しいコピーを使用していることを確認します。LD_LIBRARY_PATH 環境変数に頼ると、正しくないドライバーを間違えてロードしてしまう可能性があるので、そうすることは避けてください。明示的にターゲット・インストール・ディレクトリーから、ドライバーを動的にロードしてください。

IBM Data Server Driver for ODBC and CLI の構成

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client にも IBM Data Server Runtime Client にも含まれません。したがって、別個にインストールして構成する必要があります。アプリケーションが IBM Data Server Driver for ODBC and CLI を正常に使用できるようにするには、このドライバーとデータベース・アプリケーション実行時環境のソフトウェア・コンポーネントを構成する必要があります。

始める前に

IBM Data Server Driver for ODBC and CLI を構成し、アプリケーション環境をこのドライバー用に構成するには、以下が必要です。

- ドライバーの 1 つまたは複数のコピーがインストール済みであること。

  10 ページの『IBM Data Server Driver for ODBC and CLI のインストール』を参照してください。

手順

IBM Data Server Driver for ODBC and CLI を構成し、このドライバーを使用するように IBM Data Server Driver for ODBC and CLI アプリケーションの実行時環境を構成するには、以下のようにします。

1. db2cli.ini 初期設定ファイルを更新して、データ・ソース名、ユーザー名、パフォーマンス・オプション、接続オプションなどのドライバーの動作の局面を構成します。

   db2cli.ini ファイルのロケーションは、Microsoft ODBC Driver Manager が使用されているかどうか、使用されているデータ・ソース名 (DSN) のタイプ、インストールされているクライアントまたはドライバーのタイプ、およびレジストリー変数 **DB2CLIINIPATH** が設定されているかどうかに基づいて異なる場合があります。

   - 15 ページの『db2cli.ini 初期設定ファイル』を参照してください。

   IBM Data Server Driver for ODBC and CLI には、コマンド行プロセッサー (CLP) のサポートはありません。この理由で、CLP コマンド「db2 update CLI cfg」を使用して CLI の構成を更新することはできません。手動で「db2cli.ini」初期設定ファイルを更新しなければなりません。

   IBM Data Server Driver for ODBC and CLI のコピーが複数インストールされている場合、それぞれのドライバー・コピーに対応する db2cli.ini ファイルが存在します。該当するドライバーのコピーに関する情報を db2cli.ini に追加してください。

2. アプリケーション環境変数を構成します。
   - 18 ページの『IBM Data Server Driver for ODBC and CLI の環境変数の構成』を参照してください。
3. Microsoft Distributed Transaction Coordinator (DTC) によって管理されるトランザクションに参加するアプリケーションの場合に限り、ドライバーを DTC に登録する必要があります。
   - 22 ページの『IBM Data Server Driver for ODBC and CLI を Microsoft DTC に登録する』を参照してください。
4. Microsoft ODBC ドライバー・マネージャーを使用する ODBC アプリケーションの場合に限り、ドライバーを Microsoft ドライバー・マネージャーに登録する必要があります。
   - 23 ページの『IBM Data Server Driver for ODBC and CLI を Microsoft ODBC ドライバー・マネージャーに登録する』を参照してください。

db2cli.ini 初期設定ファイル

CLI/ODBC 初期設定ファイル (db2cli.ini) には、 CLI とこの製品を使うアプリケーションの動作を構成する場合に使用できる、さまざまなキーワードと値が入っています。

キーワードは、データベース別名と関連しており、そのデータベースにアクセスするすべての CLI および ODBC アプリケーションに影響を与えます。

作業を開始する助けとして db2cli.ini.sample サンプル構成ファイルが組み込まれています。db2cli.ini.sample ファイルに基づいて db2cli.ini ファイルを作成し、同じ場所にそれを保管することができます。サンプル構成ファイルの場所は、ご使用のドライバーのタイプおよびプラットフォームによって異なります。

IBM Data Server Client、IBM Data Server Runtime Client、または IBM Data Server Driver Package の場合、サンプル構成ファイルは次のいずれかのパスに作成されます。

- AIX、HP-UX、Linux、または Solaris オペレーティング・システムの場合: *installation_path*/cfg
- Windows XP および Windows Server 2003 の場合: C:¥Documents and Settings¥All Users¥Application Data¥IBM¥DB2¥*driver_copy_name*¥cfg
- Windows Vista および Windows Server 2008 の場合: C:¥ProgramData¥IBM¥DB2¥*driver_copy_name*¥cfg

例えば、 IBM Data Server Driver Package for Windows XP を使用していて、データ・サーバー・ドライバーのコピー名が IBMDBCL1 である場合、db2cli.ini.sample ファイルは C:¥Documents and Settings¥All Users¥Application Data¥IBM¥DB2¥IBMDBCL1¥cfg ディレクトリーに作成されます。

IBM Data Server Driver for ODBC and CLI の場合、サンプル構成ファイルは次のいずれかのパスに作成されます。

- AIX、HP-UX、Linux、または Solaris オペレーティング・システムの場合: *installation_path*/cfg
- Windows の場合: *installation_path*¥cfg

ここで、*installation_path* はドライバー・ファイルが抽出されたファイル・パスです。

例えば、IBM Data Server Driver for ODBC and CLI for Windows Vista を使用していて、ドライバーが C:¥IBMDB2¥CLIDRIVER¥V97FP3 ディレクトリーにインストールされている場合、db2cli.ini.sample ファイルは C:¥IBMDB2¥CLIDRIVER¥V97FP3¥cfg ディレクトリーに作成されます。

Windows オペレーティング・システム上でユーザー DSN を構成するために ODBC Driver Manager が使用されている場合、db2cli.ini ファイルは Documents and Settings¥User Name に作成されます。ここで、*User Name* はユーザー・ディレクトリーの名前を表します。

環境変数 **DB2CLIINIPATH** を使用して、db2cli.ini ファイルの別の場所を指定することができます。

構成キーワードを使用すると、以下のことが可能になります。

- データ・ソース名、ユーザー名、およびパスワードなどの一般的なフィーチャーを構成する。
- パフォーマンスに影響を及ぼすオプションを設定する。
- ワイルドカード文字などの照会パラメーターを指示する。
- さまざまな ODBC アプリケーション用にパッチまたは作業環境を設定する。
- コード・ページと IBM GRAPHIC データ・タイプなどの接続に関連したその他のフィーチャーを設定する。
- アプリケーションによって指定されるデフォルト接続オプションをオーバーライドする。例えば、アプリケーションが SQL_ATTR_ANSI_APP 接続属性を設定することによって CLI ドライバーに対して Unicode サポートを要求している場合でも、db2cli.ini ファイルの中で **DisableUnicode**=1 が設定されていると、CLI ドライバーはそのアプリケーションに Unicode サポートを提供しません。

  注: db2cli.ini ファイルの中で設定されている CLI/ODBC 構成キーワードが、SQLDriverConnect() 接続ストリングに含まれるキーワードと矛盾する場合、SQLDriverConnect() キーワードが優先されます。

db2cli.ini 初期設定ファイルは、 CLI 構成オプション用の値を保管している ASCII ファイルです。作業を開始する助けとして、サンプル・ファイルが組み込まれています。ほとんどの CLI/ODBC 構成キーワードは db2cli.ini 初期設定ファイル内に設定されますが、一部のキーワードはその代わりに、SQLDriverConnect() への接続ストリング内にキーワード情報を指定することによって設定されます。

ファイル内には、ユーザーが構成を希望するデータベース (データ・ソース) ごとに 1 つのセクションがあります。必要であれば、すべてのデータベース接続に影響を与える共通セクションもあります。

COMMON セクションには、 CLI/ODBC ドライバーを介したすべてのデータベース接続に適用するキーワードのみ含まれています。それには以下のキーワードが含まれます。

- **CheckForFork**
- **DiagPath**

- **DisableMultiThread**
- **JDBCTrace**
- **JDBCTraceFlush**
- **JDBCTracePathName**
- **QueryTimeoutInterval**
- **ReadCommonSectionOnNullConnect**
- **Trace**
- **TraceComm**
- **TraceErrImmediate**
- **TraceFileName**
- **TraceFlush**
- **TraceFlushOnError**
- **TraceLocks**
- **TracePathName**
- **TracePIDList**
- **TracePIDTID**
- **TraceRefreshInterval**
- **TraceStmtOnly**
- **TraceTime**
- **TraceTimeStamp**

他のすべてのキーワードはデータベース固有のセクションに置かれるようになっています。

注: 構成キーワードは COMMON セクション中で有効になりますが、すべてのデータベース接続に適用されます。

db2cli.ini ファイルの COMMON セクションは、次の語で始まります。

```
[COMMON]
```

共通キーワードを設定する前に、クライアントからのすべての CLI/ODBC 接続にこの設定が与える影響を評価するのは重要なことです。例えば、**TRACE** などのキーワードは、DB2 に接続している CLI/ODBC アプリケーションのうち 1 つだけをトラブルシューティングしようとしている場合でも、これらのすべてのアプリケーションに関する情報をそのクライアントで生成します。

それぞれのデータベースの特定のセクションは、必ず大括弧で囲まれたデータ・ソース名 (DSN) の名前で始まります。

[*data source name***]**

これをセクション・ヘッダー と呼びます。

パラメーターを設定するには、キーワードとその関連キーワード値を次の形式で指定します。

KeywordName =*keywordValue*

- 各データベースのすべてのキーワードとその関連値は、そのデータベースのセクション・ヘッダーの下になければなりません。
- データベース固有のセクションに **DBAlias** キーワードが含まれていない場合は、接続が確立される際にはデータ・ソース名がデータベース別名として使用されます。各セクションのキーワード設定値は、該当するデータベース別名だけに適用されます。
- キーワードは大文字小文字の区別はありません。しかし、その値が文字ベースのものであれば値にその区別がある場合もあります。
- .INI ファイルにデータベースがない場合、これらのキーワードのデフォルト値が有効になっています。
- 新しい行の先頭位置にセミコロンを入れると、注釈行になります。
- ブランク行は許可されています。
- 1 つのキーワードに重複項目があると、最初の項目が使用されます (警告は与えられません)。

2 つのデータベース別名セクションがある .INI サンプル・ファイルを次に示します。

```
; This is a comment line.
[MYDB22]
AutoCommit=0
TableType="'TABLE','SYSTEM TABLE'"

; This is another comment line.
[MYDB2MVS]
CurrentSQLID=SAAID
TableType="'TABLE'"
SchemaList="'USER1',CURRENT SQLID,'USER2'"
```

db2cli.ini ファイルはすべてのプラットフォームで手動で編集できますが、使用できるなら **UPDATE CLI CONFIGURATION** コマンドを使用することをお勧めします。手作業で db2cli.ini ファイルを編集する場合、最後の項目の後にブランク行を追加してください。

IBM Data Server Driver for ODBC and CLI の環境変数の構成

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client にも IBM Data Server Runtime Client にも含まれません。したがって、別個にインストールして構成する必要があります。 IBM Data Server Driver for ODBC and CLI を使用するには、2 つのタイプの環境変数を設定する必要が生じることがあります。1 つはいくつかの DB2 レジストリー変数と置き換わっている環境変数で、もう 1 つはドライバー・ライブラリーの場所をアプリケーションに知らせる環境変数です。

始める前に

IBM Data Server Driver for ODBC and CLI の環境変数を構成するには、ドライバーの 1 つ以上のコピーがインストールされている必要があります。 10 ページの『IBM Data Server Driver for ODBC and CLI のインストール』を参照してください。

制約事項

複数のバージョンの IBM Data Server Driver for ODBC and CLI が同じマシンにインストールされている場合や、他の DB2 バージョン 9 製品が同じマシンにインストールされている場合は、環境変数を設定する (例えば、IBM Data Server Driver for ODBC and CLI ライブラリーを指すように **LIBPATH** または **LD_LIBRARY_PATH** を設定する) と、既存のアプリケーションが中断する可能性があります。環境変数を設定する際には、その環境の有効範囲で実行しているすべてのアプリケーションにとって適切であることを確認してください。

64 ビット の UNIX システムおよび Linux システム上の IBM Data Server Driver for ODBC and CLI には、32 ビット・ドライバー・ライブラリーもパッケージされていて、32 ビット CLI アプリケーションをサポートしています。UNIX システムおよび Linux システムでは、32 ビットと 64 ビットの両方ではなく一方のライブラリーをインスタンスと関連付けることができます。**LIBPATH** または **LD_LIBRARY_PATH** を、IBM Data Server Driver for ODBC and CLI の `lib32` または `lib64` のいずれかのライブラリー・ディレクトリーに設定できます。`lib` ディレクトリーから事前設定ソフト・リンクを使用して `lib64` ライブラリーにアクセスすることもできます。Windows システム上の IBM Data Server Driver for ODBC and CLI では、同じ `bin` ディレクトリー内に 32 ビットと 64 ビットの両方の必要なランタイム DLL が含まれています。

手順

IBM Data Server Driver for ODBC and CLI の環境変数を構成するには、以下のようにします。

1. オプション: 該当する DB2 環境変数を、それに相当する DB2 レジストリー変数に準じた設定にします。

   IBM Data Server Driver for ODBC and CLI には、コマンド行プロセッサー (CLP) のサポートはありません。この理由で、**db2set** CLP コマンドを使用して DB2 レジストリー変数を構成することはできません。必須の DB2 レジストリー変数は、環境変数に置き換えられています。

   DB2 レジストリー変数の代わりに使用できる環境変数のリストについては、 20 ページの『IBM Data Server Driver for ODBC and CLI によってサポートされる環境変数』を参照してください。
2. オプション: ローカル環境変数 **DB2_CLI_DRIVER_INSTALL_PATH** を、ドライバーのインストール先ディレクトリーに設定できます。

   IBM Data Server Driver for ODBC and CLI のコピーが複数インストールされている場合、**DB2_CLI_DRIVER_INSTALL_PATH** には必ず該当するドライバーのコピーの場所を指定してください。**DB2_CLI_DRIVER_INSTALL_PATH** 変数を設定すると、IBM Data Server Driver for ODBC and CLI がドライバーのインストール場所として、**DB2_CLI_DRIVER_INSTALL_PATH** 変数で指定されたディレクトリーを使用するよう強制されます。 例えば、以下のようにします。

   ```
   export DB2_CLI_DRIVER_INSTALL_PATH=/home/db2inst1/db2clidriver/clidriver
   ```

   ここで、`/home/db2inst1/db2clidriver` は CLI ドライバーがインストールされているインストール・パスです。

3. オプション: 環境変数 **LIBPATH** (AIX オペレーティング・システムの場合)、**SHLIB_PATH** (HP-UX システムの場合)、または **LD_LIBRARY_PATH** (他の UNIX および Linux システムの場合) を、ドライバーがインストールされている lib ディレクトリーに設定します。 例えば、以下のようにします (AIX システムの場合)。

   ```
   export LIBPATH=/home/db2inst1/db2clidriver/clidriver/lib
   ```

   IBM Data Server Driver for ODBC and CLI のコピーが複数インストールされている場合、**LIBPATH** または **LD_LIBRARY_PATH** には、必ず該当するドライバーのコピーの場所を指定してください。**LIBPATH** 変数または **LD_LIBRARY_PATH** 変数を、システムにインストールされている IBM Data Server Driver for ODBC and CLI の複数のコピーに設定しないでください。**LIBPATH** 変数または **LD_LIBRARY_PATH** 変数を、lib32 と lib64 の両方 (または lib) のライブラリー・ディレクトリーに設定しないでください。

   アプリケーションが、ドライバーのライブラリー (Windows システムの場合は db2cli.dll、その他のシステムの場合は libdb2.a) に静的にリンクするか、または完全修飾名を使用してこのライブラリーを動的にロードする場合は、このステップは必要ありません。

   完全修飾ライブラリー名を使用して、ライブラリーを動的にロードする必要があります。Windows オペレーティング・システムでは、**LOAD_WITH_ALTERED_SEARCH_PATH** パラメーターと、ドライバー DLL へのパスを使用して LoadLibraryEx メソッドを使用する必要があります。
4. オプション: **db2level**、**db2cli**、および **db2trc** のようなユーティリティーを使用する必要がある場合は、すべてのシステム内のドライバー・インストールの bin ディレクトリーが組み込まれるように **PATH** 環境変数を設定します。UNIX システムおよび Linux システムでは、bin ディレクトリーに加えて、ドライバー・インストールの adm ディレクトリーを **PATH** 環境変数に追加します。 例えば、以下のようにします (すべての UNIX システムおよび Linux システムの場合)。

   ```
   export PATH=/home/db2inst1/db2clidriver/clidriver/bin:/home/db2inst1/db2clidriver/clidriver/adm:$PATH
   ```

IBM Data Server Driver for ODBC and CLI によってサポートされる環境変数:

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client または IBM Data Server Runtime Client には含まれません。したがって、別個にインストールして構成する必要があります。

IBM Data Server Driver for ODBC and CLI はコマンド行プロセッサー (CLP) をサポートしません。つまり、DB2 レジストリー変数を設定する通常の方法 (**db2set** CLP コマンドの使用) は不可能です。その代わり、関連する DB2 レジストリー変数は IBM Data Server Driver for ODBC and CLI では環境変数としてサポートされます。

環境変数として IBM Data Server Driver for ODBC and CLI でサポートされる DB2 レジストリー変数は、以下のとおりです。

表 3. 環境変数としてサポートされている DB2 レジストリー変数

変数のタイプ	変数名
一般変数	**DB2ACCOUNT** **DB2BIDI** **DB2CODEPAGE** **DB2GRAPHICUNICODESERVER** **DB2LOCALE** **DB2TERRITORY**
システム環境変数	**DB2DOMAINLIST**
通信変数	**DB2_FORCE_NLS_CACHE** **DB2SORCVBUF** **DB2SOSNDBUF** **DB2TCP_CLIENT_RCVTIMEOUT**
パフォーマンス変数	**DB2_NO_FORK_CHECK**
その他の変数	**DB2CLIINIPATH** **DB2DSDRIVER_CFG_PATH** **DB2DSDRIVER_CLIENT_HOSTNAME** **DB2_ENABLE_LDAP** **DB2LDAP_BASEDN** **DB2LDAP_CLIENT_PROVIDER** **DB2LDAPHOST** **DB2LDAP_KEEP_CONNECTION** **DB2LDAP_SEARCH_SCOPE** **DB2NOEXITLIST**
診断変数	**DB2_DIAGPATH**
接続変数	**AUTHENTICATION** **PROTOCOL** **PWDPLUGIN** **KRBPLUGIN** **ALTHOSTNAME** **ALTPORT** **INSTANCE** **BIDI**

db2oreg1.exe の概要

db2oreg1.exe ユーティリティーを使用することにより、IBM Data Server Driver for ODBC and CLI の XA ライブラリーの Microsoft Distributed Transaction Coordinator (DTC) への登録と、このドライバーの Microsoft ODBC ドライバー・マネージャーへの登録を行うことができます。db2oreg1.exe ユーティリティーの使用が必要となるのは、Windows オペレーティング・システムの場合のみです。

バージョン 9.7 フィックスパック 4 以降、db2oreg1.exe ユーティリティーは非推奨となり、今後のリリースでは利用できなくなります。代わりに、**db2cli** DB2 対話機能 CLI コマンドを使用してください。

Windows の 64 ビット版のオペレーティング・システムの IBM Data Server クライアント・パッケージでは、32 ビット版の db2oreg1.exe ユーティリティー (db2oreg132.exe) が、64 ビット版の db2oreg1.exe に加えてサポートされています。

db2oreg1.exe ユーティリティーの実行が必要になる条件

以下の場合には、db2oreg1.exe ユーティリティーを実行しなければなりません。

- IBM Data Server Driver for ODBC and CLI を使用するアプリケーションが、DTC によって管理される分散トランザクションに参加する場合、または
- IBM Data Server Driver for ODBC and CLI を使用するアプリケーションが、ODBC データ・ソースに接続する場合

また、db2dsdriver.cfg.sample サンプル構成ファイルおよび db2cli.ini.sample サンプル構成ファイルを作成するために、db2oreg1.exe ユーティリティーを実行することもできます。

db2oreg1.exe ユーティリティーを実行する時

db2oreg1.exe ユーティリティーを使用する場合、以下の時点で実行しなければなりません。

- IBM Data Server Driver for ODBC and CLI をインストールする時、および
- IBM Data Server Driver for ODBC and CLI をアンインストールする時

  ドライバーのインストール後に db2oreg1.exe ユーティリティーを実行すると、このユーティリティーは Windows レジストリーに変更を加えます。ドライバーをアンインストールする場合には、このユーティリティーを再実行して、これらの変更内容を取り消す必要があります。

db2oreg1.exe ユーティリティーの実行方法

- db2oreg1.exe は、IBM Data Server Driver for ODBC and CLI のインストール場所の bin サブディレクトリーにあります。
- db2oreg1.exe ユーティリティーで使用されるパラメーターをリストし、これらのパラメーターの使用法を参照するには、「-h」 オプションを指定してこのユーティリティーを実行してください。

IBM Data Server Driver for ODBC and CLI を Microsoft DTC に登録する

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client または IBM Data Server Runtime Client には含まれません。したがって、別個にインストールして構成する必要があります。

Microsoft Distributed Transaction Coordinator (DTC) によって管理されるトランザクションに参加するデータベース・アプリケーションで IBM Data Server Driver for ODBC and CLI を使用するには、ドライバーを DTCに登録する必要があります。

これに関連したセキュリティー要件の詳細については、以下の Microsoft 記事へのリンクをご覧ください。 Registry Entries Are Required for XA Transaction Support

始める前に

IBM Data Server Driver for ODBC and CLI を DTC に登録するには、ドライバーの 1 つ以上のコピーがインストールされている必要があります。 10 ページの『IBM Data Server Driver for ODBC and CLI のインストール』を参照してください。

制約事項

IBM Data Server Driver for ODBC and CLI を DTC に登録する必要があるのは、ドライバーを使用するアプリケーションが、DTC によって管理されるトランザクションに参加する場合に限ります。

手順

IBM Data Server Driver for ODBC and CLI を DTC に登録するには、インストール済みのドライバーのコピーごとに `db2cli install -setup` コマンドを実行します。
ドライバーのインストール後に `db2cli install -setup` コマンドを実行すると、このユーティリティーは Windows レジストリーに変更を加えます。ドライバーをアンインストールする場合には、この `db2cli install -cleanup` コマンドを実行して、これらの変更内容を取り消す必要があります。

例

例えば、次のコマンドは、Windows レジストリーに IBM Data Server Driver for ODBC and CLI を登録し、アプリケーション・データのパスに構成フォルダーを作成します。

```
> db2cli install –setup

IBM Data Server Driver for ODBC and CLI が正常に登録されました。
構成フォルダーが正常に作成されました。
```

IBM Data Server Driver for ODBC and CLI を Microsoft ODBC ドライバー・マネージャーに登録する

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client にも IBM Data Server Runtime Client にも含まれません。したがって、別個にインストールして構成する必要があります。 ODBC アプリケーションで IBM Data Server Driver for ODBC and CLI を Microsoft ODBC ドライバー・マネージャーとともに使用するには、ドライバーをドライバー・マネージャーに登録する必要があります。

始める前に

IBM Data Server Driver for ODBC and CLI を Microsoft ODBC ドライバー・マネージャーに登録するには、ドライバーの 1 つ以上のコピーがインストールされている必要があります。 10 ページの『IBM Data Server Driver for ODBC and CLI のインストール』を参照してください。

このタスクについて

Microsoft ODBC ドライバー・マネージャーは、IBM Data Server Driver for ODBC and CLI の登録を必要とする唯一の ODBC ドライバー・マネージャーです。その

他の ODBC ドライバー・マネージャーでは、このアクティビティーを行う必要はありません。

手順

IBM Data Server Driver for ODBC and CLI を Microsoft ドライバー・マネージャーに登録するには、インストール済みのドライバーのコピーごとに **db2cli install -setup** コマンドを実行します。

タスクの結果

ドライバーのインストール後に **db2cli install -setup** コマンドを実行すると、このコマンドにより Windows レジストリーが変更されます。ドライバーをアンインストールする場合には、**db2cli install -cleanup** コマンドを実行して、これらの変更内容を取り消してください。

例

例えば、次のコマンドは、Windows レジストリーに IBM Data Server Driver for ODBC and CLI を登録し、アプリケーション・データのパスに構成フォルダーを作成します。

```
> db2cli install –setup

IBM Data Server Driver for ODBC and CLI が正常に登録されました。
構成フォルダーが正常に作成されました。
```

IBM Data Server Driver for ODBC and CLI を使用したデータベースへの接続

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client および IBM Data Server Runtime Client には含まれません。したがって、別個にインストールして構成する必要があります。 IBM Data Server Driver for ODBC and CLI は、ローカル・データベース・ディレクトリーを作成しません。つまり、このドライバーを使用するとき、別の方法で接続情報をアプリケーションに提供する必要があります。

始める前に

IBM Data Server Driver for ODBC and CLI でデータベースに接続するには、以下が必要です。

- 接続先のデータベース、および
- ドライバーの 1 つまたは複数のコピーがインストール済みであること。
  - 詳しくは、 10 ページの『IBM Data Server Driver for ODBC and CLI のインストール』 を参照してください。

このタスクについて

CLI および ODBC データベース・アプリケーションで IBM Data Server Driver for ODBC and CLI を使ってデータベースに接続できるようにするために、接続情報を指定する方法がいくつかあります。CLI 設定が複数の場所で指定されている場合、それらの設定は以下の順序で使用されます。

1. 接続ストリング・パラメーター
2. db2cli.ini ファイル
3. db2dsdriver.cfg ファイル

手順

IBM Data Server Driver for ODBC and CLI の使用時にデータベースに関する接続を構成するには、以下の方法のいずれかを使用します。

- SQLDriverConnect への接続ストリング・パラメーター内でデータベース接続情報を指定します。
  - 詳しくは、 30 ページの『SQLDriverConnect 関数 (CLI) - データ・ソースへの (拡張) 接続』 を参照してください。
- CLI アプリケーションの場合のみ: CLI 構成ファイルの中でデータベース接続情報を指定します。

  IBM Data Server Driver for ODBC and CLI には、コマンド行プロセッサー (CLP) のサポートはありません。この理由で、CLP コマンド「db2 update CLI cfg」を使用して CLI の構成を更新することはできません。手動でdb2cli.ini 初期設定ファイルを更新しなければなりません。

  IBM Data Server Driver for ODBC and CLI のコピーが複数インストールされている場合、それぞれのドライバー・コピーに対応する db2cli.ini ファイルが存在します。該当するドライバーのコピーに関する情報を db2cli.ini に追加してください。

  db2cli.ini ファイルの場所について詳しくは、 15 ページの『db2cli.ini 初期設定ファイル』を参照してください。
- db2dsdriver.cfg 構成ファイルを使用して、接続情報およびパラメーターを提供します。 例えば、db2dsdriver.cfg 構成ファイルで以下の情報を指定して、SQLDriverConnect() に接続ストリングを DSN=myDSN;PWD=*XXXXX* として渡すことができます。

```
<configuration>
  <dsncollection>
    <dsn alias="myDSN" name="sample" host="server.domain.com" port="446">
    </dsn>
  </dsncollection>
  <databases>
    <database name="sample" host="server.domain.com" port="446">
      <parameter name="CommProtocol" value="TCPIP"/>
      <parameter name="UID" value="username"/>
    </database>
  </databases>
</configuration>
```

- ODBC アプリケーションの場合のみ: ODBC データ・ソースとしてデータベースを ODBC ドライバー・マネージャーに登録します。 詳しくは、 27 ページの『IBM Data Server Driver for ODBC and CLI を使用するアプリケーションのための ODBC データ・ソースの登録』 を参照してください。
- **FileDSN** CLI/ODBC キーワードを使用して、データベース接続情報が入っているファイルのデータ・ソース名 (DSN) を識別します。 詳しくは、 38 ページの『FileDSN CLI/ODBC 構成キーワード』 を参照してください。

ファイル DSN は、データベース接続情報が入っているファイルです。CLI/ODBC キーワード **SaveFile** を使用することにより、ファイル DSN を作成することができます。 Windows オペレーティング・システムでは、Microsoft ODBC ドライバー・マネージャーを使用して、ファイル DSN を作成できます。

- ローカル・データベース・サーバーの場合のみ: CLI/ODBC キーワード **PROTOCOL** および **INSTANCE** を使用して、以下のようにローカル・データベースを識別します。
  1. CLI/ODBC キーワード **PROTOCOL** の値を `Local` に設定します。
  2. CLI/ODBC キーワード **INSTANCE** を、データベースが置かれているローカル・データベース・サーバーのインスタンス名に設定します。

  詳しくは、39 ページの『Protocol CLI/ODBC 構成キーワード』 および 38 ページの『Instance CLI/ODBC 構成キーワード』を参照してください。

例

ファイル DSN 接続または DSN なし接続を処理する CLI/ODBC キーワードのリストを以下に示します。

- 36 ページの『AltHostName CLI/ODBC 構成キーワード』;
- 36 ページの『AltPort CLI/ODBC 構成キーワード』;
- 36 ページの『Authentication CLI/ODBC 構成キーワード』;
- 37 ページの『BIDI CLI/ODBC 構成キーワード』;
- 38 ページの『FileDSN CLI/ODBC 構成キーワード』;
- 38 ページの『Instance CLI/ODBC 構成キーワード』;
- 38 ページの『Interrupt CLI/ODBC 構成キーワード』;
- 39 ページの『KRBPlugin CLI/ODBC 構成キーワード』;
- 39 ページの『Protocol CLI/ODBC 構成キーワード』;
- 40 ページの『PWDPlugin CLI/ODBC 構成キーワード』;
- 40 ページの『SaveFile CLI/ODBC 構成キーワード』;
- 46 ページの『DiagLevel CLI/ODBC 構成キーワード』;
- 47 ページの『NotifyLevel CLI/ODBC 構成キーワード』;

例として、以下のプロパティーを持つデータベースを想定します。

- サーバー上のデータベースまたはサブシステムの名前は `db1`、
- サーバーの場所は `11.22.33.44`、
- アクセス・ポートは `56789`、
- 転送プロトコルは `TCPIP`

CLI アプリケーション内でこのデータベースへの接続を行うには、以下のアクションのいずれかを実行します。

- 次の情報を含む接続ストリングを使用して、SQLDriverConnect を呼び出します。
  ```
  Database=db1; Protocol=tcpip; Hostname=11.22.33.44; Servicename=56789;
  ```
- 次の例を `db2cli.ini` に追加します。

```
[db1]
Database=db1
Protocol=tcpip
Hostname=11.22.33.44
Servicename=56789
```

ODBC アプリケーションの中でデータベース接続を作成するには、以下のようにします。

1. odbc_db1 という名前の ODBC データ・ソースとしてデータベースをドライバー・マネージャーに登録します。
2. Database=odbc_db1; を含む接続ストリングを使用して SQLConnect を呼び出します。

IBM Data Server Driver for ODBC and CLI を使用するアプリケーションのための ODBC データ・ソースの登録

ODBC データベース・アプリケーションがドライバーを使用できるようになるには、IBM Data Server Driver for ODBC and CLI をインストールして構成する必要があります。このドライバーは、IBM Data Server Client または IBM Data Server Runtime Client には含まれません。

始める前に

データベースを ODBC データ・ソースとして登録し、IBM Data Server Driver for ODBC and CLI をデータベースと関連付けるには、以下の要件を満たす必要があります。

- ODBC アプリケーションの接続先となるデータベース
- ODBC ドライバー・マネージャーがインストール済みであること
- IBM Data Server Driver for ODBC and CLI のコピーが 1 つ以上インストール済みであること
  - 10 ページの『IBM Data Server Driver for ODBC and CLI のインストール』を参照してください。
- ドライバーの 1 つまたは複数のコピーがインストール済みであること。
  - 10 ページの『IBM Data Server Driver for ODBC and CLI のインストール』を参照してください。

このタスクについて

IBM Data Server Driver for ODBC and CLI ライブラリー・ファイルの名前は Windows オペレーティング・システムでは db2app.dll、その他のプラットフォームでは db2app.lib です。このドライバー・ライブラリー・ファイルの格納場所は、ドライバーのインストール・ディレクトリーの「lib」サブディレクトリー内です。

IBM Data Server Driver for ODBC and CLI のコピーが複数インストールされている場合、odbc.ini ファイル内で該当するコピーが識別されていることを確認してください。可能であれば、このドライバーの複数のコピーをインストールすることは避けてください。

手順

この手順は、アプリケーションにどのドライバー・マネージャーを使用しているかによって異なります。

- Microsoft ODBC ドライバー・マネージャーの場合、以下のアクションを実行します。
  1. `db2cli install -setup` コマンドを使用して、IBM Data Server Driver for ODBC and CLI を Microsoft ODBC ドライバー・マネージャーに登録します。 23 ページの『IBM Data Server Driver for ODBC and CLI を Microsoft ODBC ドライバー・マネージャーに登録する』を参照してください。
  2. ODBC データ・ソースとしてデータベースを登録します。 288 ページの『Windows CLI 環境のセットアップ』を参照してください。
- オープン・ソースの ODBC ドライバー・マネージャーの場合、以下のアクションを実行します。
  1. `odbc.ini` ファイルにデータベース情報を追加して、ODBC データ・ソースとしてデータベースを識別します。 284 ページの『ODBC 環境のセットアップ (Linux および UNIX)』を参照してください。
  2. `odbc.ini` ファイルのデータベース・セクションに追加して、IBM Data Server Driver for ODBC and CLI をデータ・ソースに関連付けます。完全修飾されたライブラリー・ファイル名を使用する必要があります。

タスクの結果

ODBC ドライバー・マネージャーを使用して Microsoft ODBC データ・ソースを作成する場合は常に、ODBC データ ソース アドミニストレータを手動で開き、`db2cli.ini` ファイルまたは `db2dsdriver.cfg` ファイルの内容を確認して、データ・ソースを作成する必要があります。 `db2cli.ini` ファイルまたは `db2dsdriver.cfg` ファイルを読み込んで、Microsoft ODBC データ・ソースを作成するコマンド行ユーティリティーはありません。**db2cli** コマンドには、**registerdsn** コマンド・パラメーターを使用して Microsoft ODBC データ・ソースを作成するための追加のオプションが備えられています。このパラメーターには、以下の機能があります。

- データ・ソース項目がカタログされたデータベースとして `db2cli.ini` ファイルや `db2dsdriver.cfg` ファイル、またはローカル・データベース・ディレクトリーで使用可能な場合は、Microsoft システム ODBC データ・ソースまたはユーザー ODBC データ・ソースを登録します。
- Microsoft データ ソース アドミニストレータに既に登録されている DB2 システム・データ・ソースまたはユーザー・データ・ソースをすべてリストします。
- Microsoft データ ソース アドミニストレータに既に登録されているシステム・データ・ソースまたはユーザー・データ・ソースを削除します。

注: db2cli registerdsn コマンドは、Microsoft Windows オペレーティング・システムでのみサポートされています。

例

以下の条件の下で、オープン・ソースのドライバー・マネージャーに ODBC データ・ソースを登録する場合を想定します。

- ターゲット・データベース・サーバーのオペレーティング・システムが AIX である。
- IBM Data Server Driver for ODBC and CLI の 2 つのコピーが以下の場所にインストール済みである
  - $HOME/db2_cli_odbc_driver1 と
  - $HOME/db2_cli_odbc_driver2
- 以下のような 2 つの ODBC データベース・アプリケーションが存在する
  - ODBCapp_A
    - ODBCapp_A が 2 つのデータ・ソース db1 および db2 に接続
    - アプリケーションが $HOME/db2_cli_odbc_driver1 にインストールされたドライバー・コピーを使用する必要がある
  - ODBCapp_B
    - ODBCapp_B がデータ・ソース db3 に接続
    - アプリケーションが $HOME/db2_cli_odbc_driver2 にインストールされたドライバー・コピーを使用する必要がある

オープン・ソースのドライバー・マネージャーに ODBC データ・ソースを登録する場合は、例えば次の項目を odbc.ini ファイルに追加します。

```
[db1]
Driver=$HOME/db2_cli_odbc_driver1/lib/libdb2.a
Description=First ODBC data source for ODBCapp1,
       using the first copy of the IBM Data Server Driver for ODBC and CLI

[db2]
Driver=$HOME/db2_cli_odbc_driver1/lib/libdb.a
Description=Second ODBC data source for ODBCapp1,
       using the first copy of the IBM Data Server Driver for ODBC and CLI

[db3]
Driver=$HOME/db2_cli_odbc_driver2/lib/libdb2.a
Description=First ODBC data source for ODBCapp2,
       using the second copy of the IBM Data Server Driver for ODBC and CLI
```

Microsoft ODBC データ・ソース (ODBC ドライバー・マネージャーを使用) を作成する際は毎回、ユーザーは ODBC データ ソース アドミニストレータを手動で開き、db2cli.ini ファイルまたは db2dsdriver.cfg ファイルの内容を確認して、データ・ソースを作成する必要があります。 db2cli.ini ファイルまたは db2dsdriver.cfg ファイルを読み込んで、Microsoft ODBC データ・ソースを作成するコマンド行ユーティリティーはありません。この問題に対処するため、**db2cli** には、新しい **registerdsn** コマンド・パラメーターを使って Microsoft ODBC データ・ソースを作成するための追加オプションが備えられています。このパラメーターには、以下の機能があります。

- db2cli.ini ファイルまたは db2dsdriver.cfg ファイルでデータ・ソース項目が使用可能な場合には、Microsoft システム ODBC データ・ソースまたはユーザー ODBC データ・ソースを登録します。
- db2cli.ini ファイルまたは db2dsdriver.cfg ファイルで使用可能なデータ・ソースすべてを同時に登録します。データ・ソースは、システム・データ・ソースまたはユーザー・データ・ソースのいずれかとして登録できます。

- Microsoft データ ソース アドミニストレータに既に登録されているすべての DB2 システム・データ・ソースまたはユーザー・データ・ソースをリスト表示します。
- Microsoft データ ソース アドミニストレータに既に登録されているシステム・データ・ソースまたはユーザー・データ・ソースを削除します。

注: db2cli registerdsn がサポートされているのは、Microsoft Windows プラットフォーム上のみです。

IBM Data Server Driver for ODBC and CLI でのセキュリティー・プラグインの使用

セキュリティー・プラグインは、認証セキュリティー・サービスを提供する、動的にロード可能なライブラリーです。

手順

IBM Data Server Driver for ODBC and CLI でのセキュリティー・プラグインの使用は、IBM Data Server Client または IBM Data Server Runtime Client でのセキュリティー・プラグインの使用と同じです。
DB2 インフォメーション・センターおよび DB2 資料全体で、セキュリティー・プラグインの使用に関する箇所を読む際には、IBM Data Server Driver for ODBC and CLI は IBM Data Server Client のようなものと考えてください。IBM Data Server Client でのセキュリティー・プラグインの使用に関する詳細は、IBM Data Server Driver for ODBC and CLI でのセキュリティー・プラグインの使用にも適用されます。

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

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

仕様:

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

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

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

構文

汎用

```
SQLRETURN  SQLDriverConnect (
        SQLHDBC       ConnectionHandle,              /* hdbc */
        SQLHWND       WindowHandle,                  /* hwnd */
        SQLCHAR       *InConnectionString,           /* szConnStrIn */
        SQLSMALLINT   InConnectionStringLength,      /* cbConnStrIn */
```

```
SQLCHAR       *OutConnectionString,          /* szConnStrOut */
SQLSMALLINT   OutConnectionStringCapacity,   /* cbConnStrOutMax */
SQLSMALLINT   *OutConnectionStringLengthPtr, /* pcbConnStrOut */
SQLUSMALLINT  DriverCompletion);             /* fDriverCompletion */
```

関数引数

表 4. SQLDriverConnect 引数

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

使用法

InConnectionString 引数

要求の接続ストリングは、以下の構文になります。

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

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

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

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

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

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

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

OutConnectionString 引数

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

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

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

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

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

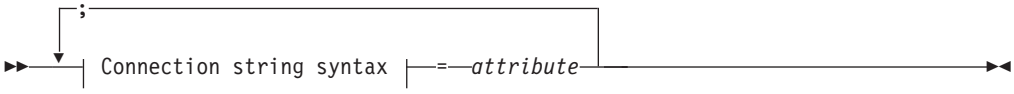
attribute-value ::= {attribute-value-list} | ?
(中括弧はリテラルであり、CLI によって返されます。)
attribute-value-list ::= character-string [:localized-character
string] | character-string [:localized-character string], attribute-value-list

説明

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

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

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



Connection string syntax

各キーワードに関連付けられている属性は、次のとおりです。

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

UID 許可名 (ユーザー ID)。

PWD 許可名に対応するパスワード。ユーザー ID のパスワードがないと、空の値が指定されます (PWD=;)。

NEWPWD
パスワード変更要求の一部として使用する新規パスワード。アプリケーションは、NEWPWD=anewpass; などで、使用する新しいストリングを指定する

か、または NEWPWD=; を指定して CLI ドライバーによって生成されるダイアログ・ボックスが新しいパスワードの入力を要求するようにすることができます。 (*DriverCompletion* 引数には SQL_DRIVER_NOPROMPT 以外の値を指定。)

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

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

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

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

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

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

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

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

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

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLDriverConnectW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 184 ページの『Unicode 関数 (CLI)』 を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA_FOUND
- SQL_INVALID_HANDLE
- SQL_ERROR

診断

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

表 5. SQLDriverConnect SQLSTATE

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

制限

なし。

例

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

ファイル DSN 接続または DSN なしでの接続用の CLI/ODBC キーワード

AltHostName CLI/ODBC 構成キーワード:

HOSTNAME で指定された 1 次サーバーと通信できない場合に使用される代替ホスト名を指定します (クライアント・リルート)。

db2cli.ini キーワード構文:

AltHostName = *完全修飾された代替ホスト名 | ノードの IP アドレス*

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このパラメーターは、データベースの代替サーバーの常駐場所を示す、ノードの完全修飾ホスト名または IP アドレスを指定します。

1 次サーバーが代替サーバー情報を戻す場合、その情報はこの AltHostName 設定値をオーバーライドします。なお、このキーワードは読み取り専用です。つまり、1 次サーバーから受け取った代替サーバー情報によって db2cli.ini が更新されることはありません。

AltPort CLI/ODBC 構成キーワード:

HOSTNAME および PORT で指定された 1 次サーバーと通信できない場合に使用される代替ポートを指定します (クライアント・リルート)。

db2cli.ini キーワード構文:

AltPort = *ポート番号*

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このパラメーターは、データベースの代替サーバーの常駐場所を示す、データベース・マネージャー・インスタンスの代替サーバーのポート番号を指定します。

1 次サーバーが代替サーバー情報を戻す場合、その情報はこの AltPort 設定値をオーバーライドします。なお、このキーワードは読み取り専用です。つまり、1 次サーバーから受け取った代替サーバー情報によって db2cli.ini が更新されることはありません。

Authentication CLI/ODBC 構成キーワード:

ファイル DSN 接続または DSN なし接続で使用される認証タイプを指定します。

db2cli.ini キーワード構文:

Authentication = CERTIFICATE | SERVER | SERVER_ENCRYPT | SERVER_ENCRYPT_AES | DATA_ENCRYPT | KERBEROS | GSSPLUGIN

デフォルト設定:

SERVER

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このオプションを設定するとき、以下のオプションもまた設定する必要があります。

- **Database**
- **Protocol**

Protocol=IPC の場合には、以下のオプションもまた設定する必要があります。

- **Instance**

Protocol=TCPIP の場合には、以下のオプションもまた設定する必要があります。

- **Port**
- **Hostname**

Kerberos を指定する場合には、オプションで **KRBPlugin** を指定することもできます。**KRBPlugin** を指定しない場合、デフォルト・プラグイン IBMkrb5 が使用されます。

DB2 バージョン 9.7 フィックスパック 6 以降、APAR PM53450 以降が適用された DB2 for z/OS バージョン 10 との接続に証明書認証を使用できるようになっています。CERTIFICATE 認証タイプは、DB2 バージョン 9.7 フィックスパック 6 からサポートされています。この認証タイプでは、SSL クライアント認証を使用でき、データベース・クライアントでデータベース・パスワードを提供する必要がありません。証明書ベースの認証を構成して認証情報を指定すると、他の方法 (db2dsdriver.cfg 構成ファイル、db2cli.ini 構成ファイル、または接続ストリングに指定するという方法) ではパスワードを指定できなくなります。 CERTIFICATE を指定する場合は、CLI 構成ファイル db2cli.ini またはデータ・サーバー・ドライバー構成ファイル db2dsdriver.cfg に、新しいラベル・パラメーター SSLCLientLabel を指定する必要もあります。

BIDI CLI/ODBC 構成キーワード:

DB2 for z/OS に接続される場合に BIDI コード・ページを指定します。

db2cli.ini キーワード構文:

BIDI = *code page*

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このオプションを設定する場合は、以下のオプションも設定する必要があります。

- Database
- Protocol=TCPIP
- Hostname
- Port

FileDSN CLI/ODBC 構成キーワード:

データ・ソース用の接続ストリングを構築する基になる DSN ファイルを指定します。

db2cli.ini キーワード構文:

このキーワードは db2cli.ini ファイル内では設定できません。

SQLDriverConnect 内の接続ストリングで、このキーワードの値を以下のように指定できます。

```
FileDSN = file name
```

Instance CLI/ODBC 構成キーワード:

ファイル DSN 接続または DSN なし接続に関するローカル IPC 接続のインスタンス名を指定します。

db2cli.ini キーワード構文:

Instance = *インスタンス名*

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このキーワードを設定するときには、以下のオプションもまた設定する必要があります。

- Database
- Protocol=IPC

Interrupt CLI/ODBC 構成キーワード:

割り込み処理モードを設定します。

db2cli.ini キーワード構文:

Interrupt = **0** | **1** | **2**

デフォルト設定:

1

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このオプションを設定するとき、以下のオプションもまた設定する必要があります。

- Database
- Protocol=IPC

キーワード値の意味は以下のとおりです。

0 処理の割り込みを使用不可にします (SQLCancel 呼び出しによって処理が割り込みされることはありません)。

1 割り込みがサポートされます (デフォルト)。このモードでは、サーバーが割り込みをサポートする場合、割り込みが送信されます。そうでない場合、接続がドロップされます。

INTERRUPT_ENABLED (DB2 Connect™ ゲートウェイ設定) および DB2 レジストリー変数 DB2CONNECT_DISCONNECT_ON_INTERRUPT の設定値は、Interrupt キーワードの設定値 1 よりも優先されます。

2 サーバーの割り込み能力にかかわらず、割り込みによって接続がドロップされます (SQLCancel は接続をドロップします)。

KRBPlugin CLI/ODBC 構成キーワード:

ファイル DSN 接続または DSN なし接続でのクライアント側の認証に使用される Kerberos プラグイン・ライブラリーの名前を指定します。

db2cli.ini キーワード構文:

KRBPlugin = *プラグイン名*

デフォルト設定:

デフォルト値は、UNIX オペレーティング・システムではヌル、Windows オペレーティング・システムでは IBMkrb5 です。

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このパラメーターは、クライアント側の接続認証に使用される Kerberos プラグイン・ライブラリーの名前を指定します。このプラグインは、Kerberos 認証を使ってクライアントが認証されるときに使用されます。

Protocol CLI/ODBC 構成キーワード:

ファイル DSN に対して、または接続頻度の低い DSN で使用される通信プロトコル。

db2cli.ini キーワード構文:

Protocol = **TCPIP** | **TCPIP6** | **TCPIP4** | **IPC** | **LOCAL**

デフォルト設定:

なし

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

TCP/IP はファイル DSN を使用するときにサポートされる唯一のプロトコルです。オプションをストリング TCPIP (スラッシュなし) に設定してください。

このオプションを設定するときには、以下のオプションも設定しなければなりません。

- Database
- ServiceName
- Hostname

Protocol を **IPC** または **LOCAL** に設定することにより、IPC 接続を指定できます。

Protocol = **IPC** | **LOCAL** の場合、Instance キーワードも設定する必要があります。

PWDPlugin CLI/ODBC 構成キーワード:

ファイル DSN 接続または DSN なし接続でのクライアント側の認証に使用されるユーザー ID パスワード・プラグイン・ライブラリーの名前を指定します。

db2cli.ini キーワード構文:

PWDPlugin = *プラグイン名*

デフォルト設定:

デフォルト値はヌルで、DB2 の提供するユーザー ID パスワード・プラグイン・ライブラリーが使用されます。

使用上の注意:

これは、特定のデータ・ソースに関する `db2cli.ini` ファイルの `[Data Source]` セクション内、または接続ストリング内で設定できます。

このパラメーターは、クライアント側の接続認証に使用されるユーザー ID パスワード・プラグイン・ライブラリーの名前を指定します。このプラグインは、SERVER または SERVER_ENCRYPT 認証を使ってクライアントが認証されるときに使用されます。

SaveFile CLI/ODBC 構成キーワード:

既存の正常な接続を確立するために使われたキーワードの属性値を保管するための、DSN ファイルのファイル名を指定します。

db2cli.ini キーワード構文:

このキーワードは `db2cli.ini` ファイル内では設定できません。

SQLDriverConnect 内の接続ストリングで、このキーワードの値を以下のように指定できます。

```
SaveFile = ファイル名
```

IBM Data Server Driver for ODBC and CLI を使って CLI および ODBC アプリケーションを実行する

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client にも IBM Data Server Runtime Client にも含まれません。したがって、別個にインストールして構成する必要があります。このドライバーはいずれかの IBM Data Server Client の機能のサブセットを提供します。 IBM Data Server Driver for ODBC and CLI は、CLI アプリケーション・プログラミング・インターフェース (API)、ODBC API、XA API、およびデータベースへの接続に対する実行時サポートを提供します。

始める前に

IBM Data Server Driver for ODBC and CLI を使ってデータベース・アプリケーションを実行するには、以下が必要です。

- ドライバーの 1 つまたは複数のコピーがインストール済みであること。
  - 10 ページの『IBM Data Server Driver for ODBC and CLI のインストール』を参照してください。

- ドライバー用にアプリケーション環境を構成済みであること。
  - 14 ページの『IBM Data Server Driver for ODBC and CLI の構成』を参照してください。

手順

IBM Data Server Driver for ODBC and CLI 用のアプリケーションを作成するとき、またはアプリケーションを IBM Data Server Driver for ODBC and CLI 用にマイグレーションするときには、以下を行う必要があります。

- ドライバーによってサポートされる CLI、ODBC、および XA API 関数だけをアプリケーションが使用することを確認します。
  - 以下を参照してください。
    - 『IBM Data Server Driver for ODBC and CLI での CLI および ODBC API サポート』
    - 42 ページの『IBM Data Server Driver for ODBC and CLI での XA API サポート』
- ドライバーで制限されている IBM Data Server Client 機能または IBM Data Server Runtime Client 機能をアプリケーションが使用しないことを確認します。
  - 43 ページの『IBM Data Server Driver for ODBC and CLI の制限事項』を参照してください。
-

  32 ビット・バージョンのドライバーは 32 ビット・データベース・アプリケーションで使用し、64 ビット・バージョンのドライバーは 64 ビット・データベース・アプリケーションで使用します。
- 問題を調査するために、ドライバーの提供するトレース、ロギング、および診断サポートを理解しておきます。
  - 44 ページの『IBM Data Server Driver for ODBC and CLI での診断サポート』を参照してください。

IBM Data Server Driver for ODBC and CLI での CLI および ODBC API サポート

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client にも IBM Data Server Runtime Client にも含まれません。したがって、別個にインストールして構成する必要があります。このドライバーはいずれかの IBM Data Server Client の機能のサブセットを提供します。

IBM Data Server Driver for ODBC and CLI は、以下の ODBC および CLI 関数の ANSI および Unicode バージョン (存在する場合) をサポートします。

```
SQLAllocConnect       SQLExtendedPrepare    SQLNumParams
SQLAllocEnv           SQLFetch              SQLNumResultCols
SQLAllocHandle        SQLFetchScroll        SQLParamData
SQLAllocStmt          SQLForeignKeys        SQLParamOptions
SQLBindCol            SQLFreeConnect        SQLPrepare
SQLBindFileToCol      SQLFreeEnv            SQLPrimaryKeys
SQLBindFileToParam    SQLFreeHandle         SQLProcedureColumns
SQLBindParameter      SQLFreeStmt           SQLProcedures
SQLBrowseConnect      SQLGetConnectAttr     SQLPutData
SQLBuildDataLink      SQLGetConnectOption   SQLRowCount
```

```
SQLBulkOperations     SQLGetCursorName      SQLSetColAttributes
SQLCancel             SQLGetData            SQLSetConnectAttr
SQLCloseCursor        SQLGetDataLinkAttr    SQLSetConnectOption
SQLColAttribute       SQLGetDescField       SQLSetConnection
SQLColAttributes      SQLGetDescRec         SQLSetCursorName
SQLColumnPrivileges   SQLGetDiagField       SQLSetDescField
SQLColumns            SQLGetDiagRec         SQLSetDescRec
SQLConnect            SQLGetEnvAttr         SQLSetEnvAttr
SQLCopyDesc           SQLGetFunctions       SQLSetParam
SQLGetInfo            SQLSetPos             SQLDescribeCol
SQLGetLength          SQLSetScrollOptions   SQLDescribeParam
SQLGetPosition        SQLSetStmtAttr        SQLDisconnect
SQLGetSQLCA           SQLSetStmtOption      SQLDriverConnect
SQLGetStmtAttr        SQLSpecialColumns     SQLEndTran
SQLGetStmtOption      SQLStatistics         SQLError
SQLGetSubString       SQLTablePrivileges    SQLExecDirect
SQLGetTypeInfo        SQLTables             SQLExecute
SQLMoreResults        SQLTransact           SQLExtendedBind
SQLNativeSql          SQLExtendedFetch      SQLNextResult
```

IBM Data Server Driver for ODBC and CLI での XA API サポート

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client または IBM Data Server Runtime Client には含まれません。したがって、別個にインストールして構成する必要があります。このドライバーはいずれかの IBM Data Server クライアントの機能のサブセットを提供します。

IBM Data Server Driver for ODBC and CLI は、以下の XA API 関数をサポートします。

xa_open
xa_close
xa_start
xa_end
xa_prepare
xa_commit
xa_rollback
xa_forget
xa_recover

IBM Data Server Driver for ODBC and CLI での LDAP サポート

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client または IBM Data Server Runtime Client には含まれません。したがって、別個にインストールして構成する必要があります。このドライバーはいずれかの IBM Data Server クライアントの機能のサブセットを提供します。

IBM Data Server Driver for ODBC and CLI は LDAP データベース・ディレクトリーをサポートしますが、以下の 1 つの制限があります。

- LDAP キャッシュは単にメモリー内のキャッシュで、ディスクには保存されません。 DB2LDAPCACHE レジストリー変数は無視されます。

IBM Data Server Driver for ODBC and CLI の使用時に LDAP を使用可能にするようデータベース・アプリケーション環境を構成するための手順は、他のシナリオと同じです。ただし、DB2LDAPCACHE レジストリー変数は無視されます。

IBM Data Server Driver for ODBC and CLI の制限事項

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client または IBM Data Server Runtime Client には含まれません。したがって、別個にインストールして構成する必要があります。このドライバーはいずれかの IBM Data Server クライアントの機能のサブセットを提供します。

IBM Data Server Driver for ODBC and CLI は以下に対するランタイム・サポートを提供します。

- DB2 CLI アプリケーション・プログラミング・インターフェース (API)
- ODBC API
- XA API
- データベース接続性
- DB2 対話機能コール・レベル・インターフェース (db2cli)

以下の制限が IBM Data Server Driver for ODBC and CLI に適用されます。

- IBM Data Server Driver for ODBC and CLI が既にインストールされている場合、他のデータベース製品を同じパスにインストールすることはできません。
- Windows オペレーティング・システムの場合、インストールできる IBM Data Server Driver for ODBC and CLI のコピーの最大数は 16 です。
- z/OS サーバーまたは System i® サーバーに接続する場合、DB2 Connect ライセンス・キーを登録する必要があります。(パスポート・アドバンテージ (Passport Advantage®) の配布物からライセンス・ファイル (例えば db2conpe.lic) を取り出し、ドライバーがインストールされたディレクトリーの下の license ディレクトリーにライセンス・ファイルをコピーします。)
- z/OS サーバーに対する XA 接続がサポートされています。ただし、System i サーバーに対する XA 接続はサポートされていません。
- 構成ファイル db2dsdriver.cfg を使用して別名を指定する場合、以下の項目に値を入れる必要があります。
  - <dsncollection> の項目 (alias、name、host、および port)
  - <database> の項目 (name、host、port)

  これらの項目は指定することが必要であり、空にすることはできません。
- CLI/ODBC 構成キーワード DBNAME はサポートされません。
- CLI LOAD ユーティリティー・ステートメント属性 sql_attr_use_load_api はサポートされません。

IBM Data Server Driver for ODBC and CLI でサポートされない機能

- CLI および ODBC アプリケーション開発
- DB2 コマンド行プロセッサー (CLP)
- 管理 API

- CLIENT 認証タイプは、IBM Data Server Driver for ODBC and CLI および IBM Data Server Driver Package のいずれによってもサポートされません。
- インストール・プログラム
  - 手動でドライバーをインストールする必要があります。
    - 10 ページの『IBM Data Server Driver for ODBC and CLI のインストール』を参照してください。
  - 手動でドライバーを構成する必要があります。
    - 14 ページの『IBM Data Server Driver for ODBC and CLI の構成』を参照してください。

IBM Data Server Driver for ODBC and CLI によって制限付きでサポートされる機能

- メッセージは英語でのみ報告されます。
- ローカル・データベース・ディレクトリーはありません。
  - LDAP はサポートされますが、LDAP キャッシュはディスクに保管されません。
    - 42 ページの『IBM Data Server Driver for ODBC and CLI での LDAP サポート』を参照してください。
- すべての診断ユーティリティーを使用できるわけではありません。
  - 『IBM Data Server Driver for ODBC and CLI での診断サポート』を参照してください。

現在の制約事項に関する最新のリストについては、http://www.ibm.com/developerworks/wikis/display/DB2/IBM+Data+Server+Driver+Limitationsを参照してください。

IBM Data Server Driver for ODBC and CLI での診断サポート

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client にも IBM Data Server Runtime Client にも含まれません。したがって、別個にインストールして構成する必要があります。このドライバーはいずれかの IBM Data Server Client の機能のサブセットを提供します。

IBM Data Server Driver for ODBC and CLI には以下のようなトレース、ロギング、および診断ユーティリティーが付属しています。

CLI トレース
IBM Data Server Driver for ODBC and CLI で CLI トレースを使用する方法は、IBM Data Server Client で CLI トレースを使用する方法と同じです。

DB2 トレース
IBM Data Server Driver for ODBC and CLI の使用時に DB2 トレースをオンにするには、`db2trc` ユーティリティーを、ドライバーのインストール・ディレクトリーの `adm` サブディレクトリー (Linux および UNIX の場合) または `bin` サブディレクトリー (Windows の場合) から呼び出す必要があります。

例えば、ドライバーを $HOME/ibm_data_server_driver_for_odbc_cli にインストールした場合には、 $HOME/ibm_data_server_driver_for_odbc_cli/adm ディレクトリーに移動した状態で db2trc を呼び出す必要があります。

IBM Data Server Driver for ODBC and CLI は、Network File System (NFS) にインストールできます。読み取り専用 NFS にドライバーをインストールする場合、DB2 トレースが機能できるようにするためには、環境変数 DB2_DIAGPATH を設定する必要があります。

db2diag ログ・ファイル

IBM Data Server Driver for ODBC and CLI を使用している場合、以下の場所に **db2diag** ログ・ファイルがあります。

- Windows オペレーティング・システムの場合: **db2diag** ログ・ファイルは、*%SYSTEM APP DATA PATH%*¥IBM¥DB2¥*%UNZIPPED PATH%* にあります。例えば、CLI ドライバー *%UNZIPPED PATH%* が D:¥Program Files¥IBM¥clidriver の場合、この CLI ドライバーの db2diag.log パスは *%SYSTEM APP DATA PATH%*¥IBM¥DB2¥D_Program Files_IBM_clidriver になります。 *%SYSTEM APP DATA PATH%* は、Windows 2003、Windows XP の場合は Drive:¥Documents and Settings¥All Users¥Application Data¥、Windows 2008 および Windows Vista の場合は Drive:¥ProgramData¥ です。
- UNIX および Linux オペレーティング・システムの場合: **db2diag** ログ・ファイルは、ドライバーのインストール・ディレクトリーの db2dump サブディレクトリーにあります。

DB2_DIAG 環境変数および DIAGPATH CLI キーワードを使用して、**db2diag** ログ・ファイルの場所を変更することができます。

db2support

IBM Data Server Driver for ODBC and CLI では DB2 コマンド行プロセッサーを使用できないため、CLP ユーティリティーは使用できません。ただし、db2support の実行可能プログラム・バージョンはドライバーで使用できます。

db2support の実行可能プログラム・バージョンは、以下の情報を収集します。

- db2level 出力
- 環境変数、および
- IBM Data Server Driver for ODBC and CLI インストール・ディレクトリーの内容を示すリスト

db2support は、ドライバーのインストール・ディレクトリーの bin サブディレクトリーから呼び出す必要があります。

例えば、ドライバーを $HOME/ibm_data_server_driver_for_odbc_cli にインストールした場合には、$HOME/ibm_data_server_driver_for_odbc_cli/bin ディレクトリーに移動した状態で db2support を呼び出す必要があります。

診断オプションの設定

IBM Data Server Driver for ODBC and CLI はコマンド行プロセッサー (CLP) をサポートしません。つまり、DB2 レジストリー変数を設定する通常の方法 (db2set コマンドの使用) は不可能です。ただし、以下の CLI/ODBC キーワードを使用すれば、診断に関連したレジストリー変数の機能がサポートされます。

- 『DiagLevel CLI/ODBC 構成キーワード』
- 47 ページの『NotifyLevel CLI/ODBC 構成キーワード』
- 47 ページの『DiagPath CLI/ODBC 構成キーワード』

SQLSetEnvAttr および SQSGetEnvAttr の以下の属性も、この機能のサポートのために使用されます。

- SQL_ATTR_DIAGLEVEL
- SQL_ATTR_NOTIFYLEVEL
- SQL_ATTR_DIAGPATH
- 47 ページの『環境属性 (CLI) リスト』を参照してください。

以下の環境変数も、この機能のサポートのために使用されます。

- DB2_DIAGPATH
- 20 ページの『IBM Data Server Driver for ODBC and CLI によってサポートされる環境変数』を参照してください。

CLI/ODBC キーワード DiagPath、属性 SQL_ATTR_DIAGPATH、および環境変数 DB2_DIAGPATH の目的はどれも同じで、診断結果の置き場所を指定することです。しかし、次のような場合には、DB2_DIAGPATH を使用する必要があります。

- IBM Data Server Driver for ODBC and CLI は、Network File System (NFS) にインストールできます。読み取り専用 NFS にドライバーをインストールする場合、DB2 トレースが機能できるようにするためには、環境変数 DB2_DIAGPATH を書き込み可能なディレクトリーに設定する必要があります。

上記の場合を除けば、CLI/ODBC キーワード DiagPath、属性 SQL_ATTR_DIAGPATH、および環境変数 DB2_DIAGPATH は、どれも同じ働きをします。

DiagLevel CLI/ODBC 構成キーワード:

診断レベルを設定します。

db2cli.ini キーワード構文:

DiagLevel = **0** | **1** | **2** | **3** | **4**

デフォルト設定:

3

使用上の注意:

これは、db2cli.ini ファイルの [COMMON] セクションでのみ設定できます。

これは、プロセス全体のための環境ハンドルの割り振り時にのみ適用されます。

これは、データベース・マネージャーのパラメーター DIAGLEVEL と同等です。

NotifyLevel CLI/ODBC 構成キーワード:

診断レベルを設定します。

db2cli.ini キーワード構文:

NotifyLevel = **0** | **1** | **2** | **3** | **4**

デフォルト設定:

3

使用上の注意:

これは、db2cli.ini ファイルの [COMMON] セクションでのみ設定できます。

これは、データベース・マネージャーのパラメーター NOTIFYLEVEL と同等です。

DiagPath CLI/ODBC 構成キーワード:

db2diag ログ・ファイルのパスを設定します。

db2cli.ini キーワード構文:

DiagPath = *既存のディレクトリー*

デフォルト設定:

デフォルト値は、UNIX および Linux オペレーティング・システムでは db2dump ディレクトリー、Windows オペレーティング・システムでは db2 ディレクトリーです。

使用上の注意:

これは、db2cli.ini ファイルの [COMMON] セクションでのみ設定できます。

これは、データベース・マネージャーのパラメーター DIAGPATH と同等です。

環境属性 (CLI) リスト:

SQLSetEnvAttr() を使用して設定できる CLI 環境属性。

ODBC は、SQLSetEnvAttr() を使用したドライバー固有の環境属性の設定をサポートしていません。 CLI アプリケーションのみが、この関数を使って CLI 固有の環境属性を設定することができます。

SQL_ATTR_CONNECTION_POOLING

この属性は、DB2 UDB for Linux, UNIX, and Windows バージョン 8 から使用すべきでない属性となりました。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_CONNECTTYPE

これは、SQL_CONNECTTYPE 属性に取って代わる属性です。このアプリケ

ーションを整合分散環境で実行するか、それとも非整合分散環境で実行するかを指定する 32 ビット整数値。以下の値を指定することができます。

- **SQL_CONCURRENT_TRANS**: アプリケーションを使用して、1 つ以上のデータベースへの並行複数接続を行うことができます。各接続には、それぞれのコミット範囲があります。トランザクションの調整を行わせることはありません。あるアプリケーションが SQLEndTran() 上の環境ハンドルを使用してコミットを発行したが、すべての接続コミットが成功したわけではない場合、そのアプリケーションはリカバリーを行う必要があります。これはデフォルトです。
- **SQL_COORDINATED_TRANS**: アプリケーションは、複数のデータベース接続間でコミットとロールバックを調整できます。このオプション設定は、組み込み SQL のタイプ 2 CONNECT の指定に対応しています。前述の SQL_CONCURRENT_TRANS 設定とは対照的に、アプリケーションは 1 つのデータベースにつき 1 つのオープン接続のみを許可されます。

  注: この接続タイプでは、SQL_ATTR_AUTOCOMMIT 接続オプションのデフォルト値である SQL_AUTOCOMMIT_OFF の設定になります。

この属性をデフォルトから変更する場合、接続を環境ハンドルに対して確立する前にこれを設定する必要があります。

アプリケーションは通常、SQLSetEnvAttr() 関数を呼び出して、この属性を環境属性として設定します。 SQLSetEnvAttr() 関数は、環境ハンドルが割り当てられると同時に呼び出されます。ただし、ODBC アプリケーションは SQLSetEnvAttr() 関数にアクセスできないため、ODBC アプリケーションの場合には、個々の接続ハンドルが割り当てられてから接続が確立されるまでの間に、SQLSetConnectAttr() 関数を使用してこの属性を設定する必要があります。

環境ハンドル上のすべての接続の SQL_ATTR_CONNECTTYPE 設定は、同じでなければなりません。 1 つの環境で同時接続と整合接続の両方を使用することはできません。最初の接続のタイプが、それ以降のすべての接続のタイプを決定します。 SQLSetEnvAttr() は、接続アクティブに接続タイプを変更しようとすると、エラーが返されます。

189 ページの『ConnectType CLI/ODBC 構成キーワード』を使用して、デフォルト接続タイプを設定することもできます。

SQL_ATTR_CONNECTTYPE 属性は、IBM 定義の拡張機能です。

SQL_ATTR_CP_MATCH

この属性は、DB2 データベース・バージョン 8 から使用すべきでない属性となりました。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_DIAGLEVEL

説明 診断レベルを表す 32 ビット整数値。これは、データベース・マネージャーの DIAGLEVEL パラメーターと同等です。

値 有効な値は 0、1、2、3、または 4 (デフォルト値は 3)。

使用上の注意

この属性は、接続ハンドルを作成する前に設定しなければなりません。

SQL_ATTR_DIAGPATH

説明 診断データが格納されるディレクトリーの名前が入っているヌル終了文字ストリングを指すポインター。これは、データベース・マネージャーの DIAGPATH パラメーターと同等です。

値 デフォルト値は、UNIX および Linux オペレーティング・システムでは db2dump ディレクトリー、Windows オペレーティング・システムでは db2 ディレクトリーです。

使用上の注意

この属性は、接続ハンドルを作成する前に設定しなければなりません。

SQL_ATTR_INFO_ACCTSTR

説明 DB2 Connect、または Linux、UNIX、および Windows 用の DB2 データベース製品の使用時に、データ・サーバーに送信されるクライアント・アカウンティング・ストリングを識別するのに使用される、ヌル終了文字ストリングを指すポインター。

値 値を設定すると、サーバーによっては、指定した長さ全体を処理することができず、値を切り捨てる場合があります。DB2 for z/OS および DB2 Universal Database for z/OS and OS/390 サーバーがサポートするのは、最大 200 文字までの長さです。ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

SQL_ATTR_INFO_ACCTSTR 属性は、IBM 定義の拡張機能です。

SQL_ATTR_INFO_APPLNAME

説明 DB2 Connect、または Linux、UNIX、および Windows 用の DB2 データベース製品の使用時に、データ・サーバーに送信されるクライアント・アプリケーション名を識別するのに使用される、ヌル終了文字ストリングを指すポインター。

値 値を設定すると、サーバーによっては、指定した長さ全体を処理することができず、値を切り捨てる場合があります。DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 サーバーがサポートするのは、最大 32 文字までです。ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

SQL_ATTR_INFO_APPLNAME 属性は、IBM 定義の拡張機能です。

SQL_ATTR_INFO_USERID

説明 DB2 Connect、または Linux、UNIX、および Windows 用の DB2

データベース製品の使用時に、データ・サーバーに送信されるクライアント・ユーザー ID を識別するのに使用される、ヌル終了文字ストリングを指すポインター。

値 値を設定すると、サーバーによっては、指定した長さ全体を処理することができず、値を切り捨てる場合があります。DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 サーバーがサポートするのは、最大 16 文字までです。このユーザー ID を認証ユーザー ID と混同しないでください。このユーザー ID は識別目的でのみ使用され、許可のために使われることはありません。ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

SQL_ATTR_INFO_USERID 属性は、IBM 定義の拡張機能です。

SQL_ATTR_INFO_WRKSTNNAME

説明 DB2 Connect、または Linux、UNIX、および Windows 用の DB2 データベース製品の使用時に、データ・サーバーに送信されるクライアント・ワークステーション名を識別するのに使用される、ヌル終了文字ストリングを指すポインター。

値 値を設定すると、サーバーによっては、指定した長さ全体を処理することができず、値を切り捨てる場合があります。DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 サーバーがサポートするのは、最大 18 文字までです。ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

SQL_ATTR_INFO_WRKSTNNAME 属性は、IBM 定義の拡張機能です。

SQL_ATTR_MAXCONN

この属性は、DB2 バージョン 8 から使用すべきでない属性となりました。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_NOTIFYLEVEL

説明 通知レベルを表す 32 ビット整数値。これは、データベース・マネージャーの NOTIFYLEVEL パラメーターと同等です。

値 有効な値は 0、1、2、3、または 4 (デフォルト値は 3)。

使用上の注意

この属性値は、接続ハンドルを作成する前に設定しなければなりません。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_ODBC_VERSION

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

ーションでは、SQLHENV 引数が指定されている関数を呼び出す前に、この環境属性を設定しないと、呼び出しは SQLSTATE HY010 (関数のシーケンス・エラーです。) を戻します。

値 次の値を使用して、この属性値を設定します。

- SQL_OV_ODBC3: この値により、次の ODBC 3.0 (CLI v5) 動作が発生します。
  - CLI は、日付、時刻、およびタイム・スタンプに、ODBC 3.0 (CLI v5) コードを戻し、これらのコードを予期しています。
  - `SQLError()`、`SQLGetDiagField()`、または `SQLGetDiagRec()` 関数が呼び出されると、CLI は ODBC 3.0 (CLI v5) SQLSTATE コードを戻します。
  - `SQLTables()` 関数への呼び出しの *CatalogName* 引数は検索パターンを受け入れます。
- SQL_OV_ODBC2: この値により、次の ODBC 2.x (CLI v2) 動作が発生します。
  - CLI は、日付、時刻、およびタイム・スタンプに ODBC 2.x (CLI v2) コードを戻し、これらのコードを予期しています。
  - `SQLError()`、`SQLGetDiagField()`、または `SQLGetDiagRec()` 関数が呼び出されると、CLI は ODBC 2.0 (CLI v2) SQLSTATE コードを戻します。
  - `SQLTables()` 関数への呼び出しの *CatalogName* 引数は検索パターンを受け入れません。
- SQL_OV_ODBC3_80: この値により、次の ODBC 3.0 (CLI v5) 動作が発生します。
  - CLI は、日付、時刻、およびタイム・スタンプに ODBC 3.x コードを戻し、これらのコードを予期しています。
  - `SQLError()`、`SQLGetDiagField()`、または `SQLGetDiagRec()` 関数が呼び出されると、CLI は ODBC 3.x SQLSTATE コードを戻します。
  - `SQLTables()` 関数への呼び出しの *CatalogName* 引数は検索パターンを受け入れます。

SQL_ATTR_OUTPUT_NTS

説明 出力引数におけるヌル終了の使用を制御する 32 ビット整数値。

値 以下の値を指定することができます。

- **SQL_TRUE**: CLI は、ヌル終了を使用して出力文字ストリングの長さを指示します。
- **SQL_FALSE**: CLI は、ヌル終了を出力文字ストリングに使用しません。

この属性の影響を受ける CLI 関数は、文字ストリング・パラメーターのある環境 (およびその環境で割り振られている接続とステートメント) について呼び出されたすべての関数です。

この属性は、この環境に接続ハンドルが割り振られていないときにのみ、設定することができます。

SQL_ATTR_PROCESSCTL

説明 プロセスのすべての環境と接続に影響を与える、プロセス・レベル属性を設定する 32 ビット・マスク。この属性は、環境ハンドルが割り振られる前に設定する必要があります。

SQLSetEnvAttr() の呼び出しでは、 *EnvironmentHandle* 引数を SQL_NULL_HANDLE に設定する必要があります。この設定は、プロセスの所要時間中はずっと有効です。一般に、この属性を使用するのは、大量の CLI 関数呼び出しが行われる、パフォーマンスの影響を受けやすいアプリケーションの場合だけです。以下のビットのいずれかを設定する前に、アプリケーションとアプリケーションが呼び出すその他のライブラリーが、列挙されている制約事項に従っていることを確認してください。

値

下記の値を組み合わせてビット・マスクを形成できます。

- SQL_PROCESSCTL_NOTHREAD - このビットは、アプリケーションが複数のスレッドを使用しないことを示します。あるいは、アプリケーションが複数のスレッドを使用する場合には、アプリケーションによってすべての DB2 呼び出しがシリアライズされます。これを設定すると、CLI は、CLI への呼び出しをシリアライズするためのシステム呼び出しを行わず、DB2 コンテキスト・タイプを SQL_CTX_ORIGINAL に設定します。
- SQL_PROCESSCTL_NOFORK - このビットは、アプリケーションが子プロセスを fork しないことを示します。デフォルトで、CLI はアプリケーションが子プロセスを fork するかどうかを調べません。しかし、CheckForFork CLI/ODBC 構成キーワードが設定されている場合、CLI はキーワードが有効になっているデータベースに接続するすべてのアプリケーションについて、関数呼び出しごとに現行プロセス ID を調べます。CLI がそのアプリケーションの fork されたプロセスを調べないように、この属性を設定できます。

SQL_ATTR_PROCESSCTL 属性は、IBM 定義の拡張機能です。

SQL_ATTR_RESET_CONNECTION

説明 Windows オペレーティング・システムの接続プールに接続が配置されていることを ODBC Driver Manager が ODBC ドライバーに通知するかどうかを指定する 32 ビットの符号なし整数値。SQL_ATTR_ODBC_VERSION 環境属性が SQL_OV_ODBC3_80 に設定されていると、ODBC Driver Manager は、接続プールに接続を配置する前にこの属性を設定するため、ドライバーが他の接続属性をデフォルト値にリセットできます。

値 指定できるのは以下の値のみです。

- SQL_RESET_CONNECTION_YES (デフォルト): ODBC Driver Manager が ODBC ドライバーに接続が接続プールに配置されていることを通知します。

注: SQL_ATTR_RESET_CONNECTION は、ODBC Driver Manager と ODBC ドライバーの間の通信でのみ使用する必要があります。すべての接

続属性がデフォルト値にリセットされるため、この属性をアプリケーションから設定しないでください。例えば、SQLSetConnectAttr() 関数を使用して設定した接続属性の値は、CLI のデフォルト値にリセットされ、アプリケーションが予期しない動作をする可能性があります。

SQL_ATTR_SYNC_POINT

この属性は、DB2 データベース・バージョン 8 から使用すべきでない属性となりました。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_TRACE

説明 CLI/ODBC トレース機能をオンにするのに使用される、ヌル終了文字ストリングを指すポインター。

値 ストリングには、CLI キーワード **TRACE** および **TRACEPATHNAME** が含まれていなければなりません。例えば、以下のようにします。

```
"TRACE=1; TRACEPATHNAME=<dir>;"
```

使用上の注意

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_TRACENOHEADER

説明 CLI トレース・ファイルにヘッダー情報が含まれるかどうかを指定する 32 ビット整数値。

値 以下の値を指定することができます。

- **0** - ヘッダー情報が CLI トレース・ファイルに含まれます。
- **1** - ヘッダー情報は CLI トレース・ファイルに含まれません。

SQL_ATTR_TRACENOHEADER 属性は、SQL_NULL_HANDLE または有効な環境ハンドルとともに使用することができます。

SQL_ATTR_USE_2BYTES_OCTET_LENGTH

この属性は、DB2 データベース・バージョン 8 から非推奨の属性となっています。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_USE_LIGHT_OUTPUT_SQLDA

この属性を設定することは、接続属性 SQL_ATTR_DESCRIBE_OUTPUT_LEVEL を 0 に設定することに相当します。SQL_ATTR_USE_LIGHT_OUTPUT_SQLDA は推奨されないため、アプリケーションは現在、接続属性 SQL_ATTR_DESCRIBE_OUTPUT_LEVEL を使用する必要があります。

SQL_ATTR_USER_REGISTRY_NAME

説明 この属性は、サーバー上で ID マッピング・サービスを使用するユーザーを認証する際にのみ使用されます。

値 SQL_ATTR_USER_REGISTRY_NAME 属性は、ID マッピング・レ

ジストリーに名前を付けるユーザー定義ストリングに設定されます。名前のフォーマットは、ID マッピング・サービスに応じて変化します。この属性を指定することにより、提供したユーザー名がこのレジストリーにあることがサーバーに通知されます。

この属性を設定した後、次に通常接続を確立しようとするとき、次にトラステッド接続を確立しようとするとき、または次にトラステッド接続でユーザー ID を切り替えようとするときに、値が使用されます。

使用上の注意

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_CONNECTTYPE
この *Attribute* は、SQL_ATTR_CONNECTTYPE に置き換わります。

SQL_MAXCONN
この *Attribute* は、SQL_ATTR_MAXCONN に置き換わります。

SQL_SYNC_POINT
この *Attribute* は、SQL_ATTR_SYNC_POINT に置き換わります。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

IBM Data Server Driver for ODBC and CLI をデータベース・アプリケーションとともにデプロイする

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client にも IBM Data Server Runtime Client にも含まれません。したがって、別個にインストールして構成する必要があります。インストール・プログラムを作成することにより、CLI および ODBC データベース・アプリケーションのデプロイメントを単純化できます。

IBM Data Server Driver for ODBC and CLI を、CLI および ODBC データベース・アプリケーションとともにデプロイするには、このドライバーが含まれる圧縮ファイルを入手し、このドライバーのために必要なインストールと構成のステップをインストール・プログラムに組み込んで実行します。

始める前に

IBM Data Server Driver for ODBC and CLI をアプリケーションとともにデプロイするには、以下が必要です。

- インストール・プログラムなどの、アプリケーションをデプロイするメカニズム (インストール・プログラムは Windows でのみ入手可能)。
- このドライバーが含まれる圧縮ファイルの入手。 10 ページの『IBM Data Server Driver for ODBC and CLI の入手』を参照してください。
- 再配布ライセンス 55 ページの『IBM Data Server Driver for ODBC and CLI のライセンス要件』を参照してください。

制約事項

再配布ライセンスの条件のもとでは、IBM Data Server Driver for ODBC and CLI に含まれる一部のファイルだけが再配布可能です。どのファイルを再配布できるかは、redist.txt ファイルにリストされています。このファイルは、ドライバーが入っている圧縮ファイル (Windows オペレーティング・システムでは ibm_data_server_driver_for_odbc_cli.zip、他のすべてのプラットフォームでは ibm_data_server_driver_for_odbc_cli.tar.Z という名前) の中にあります。

手順

IBM Data Server Driver for ODBC and CLI をインストール・プログラムに取り込むには、以下のようにします。

1. ドライバー・ファイルをインストール・プログラムにコピーします。どのドライバー・ファイルを再配布できるかについては、前述の制限を参照してください。
2. ドライバーをターゲット・マシンにインストールするようインストール・プログラムを設定します。 10 ページの『IBM Data Server Driver for ODBC and CLI のインストール』を参照してください。
3. ターゲット・マシン上の環境を構成するようインストール・プログラムを設定します。 14 ページの『IBM Data Server Driver for ODBC and CLI の構成』を参照してください。

IBM Data Server Driver for ODBC and CLI のライセンス要件

IBM Data Server Driver for ODBC and CLI は、IBM Data Server Client にも IBM Data Server Runtime Client にも含まれません。したがって、別個にインストールして構成する必要があります。

特別なライセンスなしで、IBM Data Server Driver for ODBC and CLI をダウンロードおよびインストールしてお客様の ODBC および CLI アプリケーションでご使用いただけます。

IBM Data Server Driver for ODBC and CLI は、適切なライセンス交付を受けた以下のサーバーに接続することができます。

- DB2 Database for Linux, UNIX, and Windows
- DB2 Connect Server
- InfoSphere® Federation Server
- IBM Informix
- DB2 for z/OS
- IBM DB2 for IBM i
- DB2 Server for VM and VSE

IBM Data Server Driver for ODBC and CLI を使用して DB2 for z/OS、IBM DB2 for IBM i、および DB2 Server for VM and VSE サーバーに接続できるのは、以下の条件が満たされた場合のみです。

- 適切にライセンスを取得した DB2 Connect サーバーを介して接続が確立される、または
- サーバーへの直接接続 (適切にフォーマットされた真正な DB2 Connect ライセンス・ファイルが存在する場合のみ)。ライセンス・ファイルは、DB2 Connect 製品

に付属して配布されています。このライセンス・キー・ファイルは、以下のいずれかの DB2 Connect 製品を購入することによってのみ入手できます。

– DB2 Connect Personal Edition
– DB2 Connect Enterprise Edition
– DB2 Connect Application Server Edition
– DB2 Connect Unlimited Edition for System z®
– DB2 Connect Unlimited Edition for System i

その他のいかなる製品も、必要なライセンス・ファイル、またはこのファイルの存在によって許可されるライセンス権限を提供することはありません。このファイルを改ざん、あるいは無許可で配布することは、ご使用条件に違反します。関係するライセンス・ファイルは、DB2 Connect のアクティベーション・イメージの `/db2/license` ディレクトリーにあります。このライセンスは、データ・サーバー・ドライバー・インストール・パスの `license` サブディレクトリー (例えば *installation_path*`/license`) にコピーしなければなりません。ファイルの名前は製品によって次のように異なります。

– DB2 Connect Personal Edition: `db2conpe.lic`
– DB2 Connect Application Server Edition: `db2consv_as.lic`
– DB2 Connect Enterprise Edition: `db2consv_ee.lic`
– DB2 Connect Unlimited Edition for System i: `db2consv_is.lic`
– DB2 Connect Unlimited Edition for System z: `db2consv_zs.lic`

第 3 章 ODBC Driver Manager

DB2 コール・レベル・インターフェース (CLI) は、DB2 への接続においてさまざまな種類の ODBC ドライバー・マネージャーをサポートしています。

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。 UNIX システム上で ODBC を使用するためには、別個の市販またはオープン・ソースの ODBC Driver Manager が必要です。詳細については、unixODBC の Web サイト (http://www.unixodbc.org)、および unixODBC の配布パッケージに含まれている README ファイルを参照してください。

unixODBC Driver Manager

unixODBC Driver Manager は、サポートされるすべての Linux および UNIX オペレーティング・システム上の DB2 ODBC アプリケーションのためにサポートされるオープン・ソースの ODBC ドライバー・マネージャーです。

サポート・ステートメント

unixODBC Driver Manager と DB2 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 サポート (http://www.ibm.com/software/data/db2/support/db2_9/) に、問題診断の支援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.org 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。

unixODBC ドライバー・マネージャーのセットアップ

Linux または UNIX オペレーティング・システム上で ODBC アプリケーションを実行するには、unixODBC Driver Manager を構成する必要があります。

手順

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.org から、最新の unixODBC ソース・コードをダウンロードします。
2. ソース・ファイルを untar します。 例えば、以下のようにします。

   ```
   gzip -d unixODBC-2.2.11.tar.gz
   tar xf unixODBC-2.2.11.tar
   ```

3. AIX の場合のみ: スレッドを使用できるように C コンパイラーを構成します。

   ```
   export CC=xlc_r
   export CCC=xlC_r
   ```

4. 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 -DBUILD_REAL_64_BIT_MODE
   ```

5. ホーム・ディレクトリーか、デフォルトの /usr/local プレフィックスの下にドライバー・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 -DBUILD_REAL_64_BIT_MODE --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
     ```

6. オプション: 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
   ```

7. 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。

8. AIX のみ: ODBC ドライバーからの共用ライブラリーを解凍し、DB2 が 32 ビットのオペレーティング・システムでは shr.o を、64 ビットのオペレーティング・システムでは shr_64.o を生成するようにします。混乱を避けるために、ファイルの名前を db2.o および db2_64.o に変更します。 unixODBC Driver Manager がドライバーを動的にロードするため、これらのステップは AIX では必須です。
   - 32 ビットのオペレーティング・システムの場合、次のコマンドを発行します。

     ```
     cd INSTHOME/sqllib/lib
     ar -x libdb2.a
     mv shr.o db2.o
     ```

     ここで、*INSTHOME* はインスタンス所有者のホーム・ディレクトリーです。

   - 64 ビットのオペレーティング・システムの場合、次のコマンドを発行します。

     ```
     cd INSTHOME/sqllib/lib
     ar -x -X 64 libdb2.a
     mv shr_64.o db2_64.o
     ```

     ここで、*INSTHOME* はインスタンス所有者のホーム・ディレクトリーです。

   INI ファイルが適切なライブラリーを参照していることを確認してください。

9. オプション: AIX のみ: ドライバー・マネージャーを動的にロードする場合、次のように libodbc.a、 libodbcinst.a、および libodbccr.a を解凍します。

   ```
   ar -x libodbc.a
   ar -x libodbcinst.a
   ar -x libodbccr.a
   ```

これにより、[prefix]/lib/so ディレクトリーに libodbc.so.1、libodbcinst.so.1、および libodbccr.so.1 が生成されます。

10. 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。
11. 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。

    注: ODBC ドライバー用の 64 ビット・アプリケーションをコンパイルする場合は、-DODBC64 オプションを使用して、ドライバー・マネージャーの 64 ビット定義を使用可能にしてください。

Microsoft ODBC ドライバー・マネージャー

Microsoft ODBC ドライバー・マネージャーは、TCP/IP ネットワークを使用してリモート DB2 データベースに接続するときに使用できます。

DataDirect ODBC ドライバー・マネージャー

DB2 用の DataDirect ODBC ドライバー・マネージャーを、DB2 データベースへの接続に使用できます。

制限

UNIX 環境で、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 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、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 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 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。

第 4 章 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。 例えば、以下のようにします。

   ```
   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 引数 (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 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。 例えば、以下のようにします。

   ```
   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* 引数 (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ヌル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);
```

タスクの結果

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。

CLI での初期化と終了の概説

63 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、 81 ページの『第 6 章 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 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。

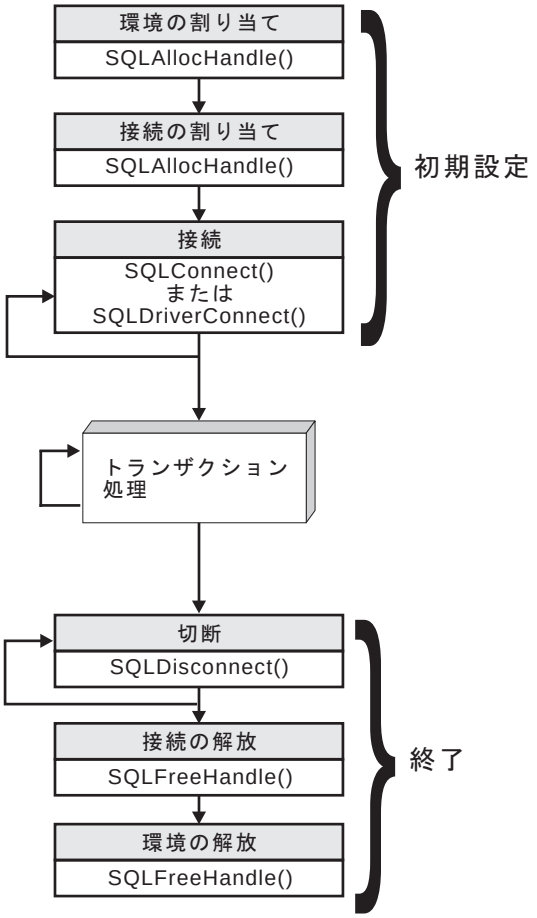


図 2. 初期化タスクと終了タスクの概念説明

CLI でのハンドル

CLI ハンドルとは、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・セーフにすることができます。接続ごとに別々のデータ構造が CLI によって割り振られ、維持されるからです。

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、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。 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 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。

第 5 章 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

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 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、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 (例えば、 C 文字ストリングから SQL INTEGER タイプに) も行います。 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 CLI ドライバーで使用できない。

注:

- GRAPHIC 列および VARGRAPHIC 列は、 Informix データ・サーバーではサポートされません。この制限があるため、SQL_C_DBCHAR (C データ・タイプ) および SQL_GRAPHIC (SQL データ・タイプ) からの変換はサポートされません。GRAPHIC および VARGRAPHIC の代わりに、NCHAR および NVARCHAR データ・タイプと、SQL_C_BINARY および SQL_BINARY との変換を使用してください。
- SQL_XML データ・タイプは、Informix データ・サーバーで使用することはサポートされていません。

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。以下のコード・スニペットでは、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

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。

例えば、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。

注: Informix データ・サーバーに対して CLI を使用する場合、 CHAR 列にはバイナリー値を挿入できません。バイナリー値は LOB 列にのみ挿入できます。これは、STRING タイプへの変換に関する Informix データ・サーバーの制限です。Informix データ・サーバーの場合に、 TIMESTAMP 列を SQL_C_CHAR/SQL_C_WCHAR ストリングにバインドするには、 ODBC エスケープ・シーケンスを使用して入力値を指定するか、またはリテラルとして入力値を指定する必要があります。DATETIME 関数は指定できません。

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 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 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。

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、CLI 関数のストリング引数のさまざまな面を取り扱います。

ストリング引数の長さ

入力ストリングは、関連した長さ引数を保持できます。この引数は、ストリングの正確な長さ (NULL 終止符を除く)、ヌル終了ストリングを示す特殊値 SQL_NTS、または NULL 値を渡す SQL_NULL_DATA のうちのいずれかを示します。長さを SQL_NTS に設定すると、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数で、もう 1 つは CLI が返したストリングの実際の長さを返す出力長さ引数です。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 の引数値を参照してください。

出力長さ引数に NULL ポインターを指定して関数が呼び出される場合、CLI は長さを戻しません。出力データが NULL 値であっても、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、 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。例えば、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、 CLI は 1 回の操作で両方の値を更新します。

ストリングのヌル終了

デフォルトでは、 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 終止符によりヌル終了します。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) データのヌル終了を無効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 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。 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、 CLI にヌル終了の図形および DBCLOB ストリングを挿入することが可能です。

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、 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。

例えば、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、CLI はストリング引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数です。カーソル名が区切られ (引用符で囲まれ) ないと、先行および後続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 UDB のバージョン 8.1 からバージョン 8.1.4 より前の各リリースでは、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 Universal Database バージョン 8.1.4 以降、そうではなくなりました。DB2 UDB 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 より前のリリースの DB2 UDB 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 列からフェッチされる際には、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。

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、
SQL_BLOB、SQL_CLOB、SQL_DBCLOB と表されます。SQL データ・タイプ引数を受け入れたり返したりする 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 ロケーターとファイルの入出力との比較

デフォルトでは、行データは LOB ロケーターによって戻されます。例えば、CLI アプリケーションが出力バッファーを提供していない場合、IBM Data Server Client は結果セットの中の LOB 列ごとに、アプリケーションに代わって LOB ロケーターを要求します。ただし、アプリケーションが、LOB 列に適したサイズのバッファーをバインドする場合は、バッファーで LOB 値が戻されます。

CLI アプリケーションが関数 `SQLGetData()` を呼び出して LOB データを取り出す場合、デフォルトで、サーバーに 1 つの要求を行い、*BufferLength* の大きさが十分である場合に、LOB 全体をメモリーに保管します。*BufferLength* が、LOB 値全体を保留できるほど大きくない場合は、分割してフェッチします。

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ランダム・アクセス・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。

注: IDS データ・サーバーにアクセスする場合、ラージ・バイナリー・オブジェクトのブロック化はサポートされません。

72 ページの図 3 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 CLOB 内でサブストリングを見つけるための入力パラメーターとして使用されて、サブストリングが取得されます。
- 右側は、CLOB がどのようにして直接ファイル内にフェッチされるのかを示しています。

  ファイルはまず CLOB 列にバインドされ、行がフェッチされると、 CLOB 値全体が直接ファイルに転送されます。

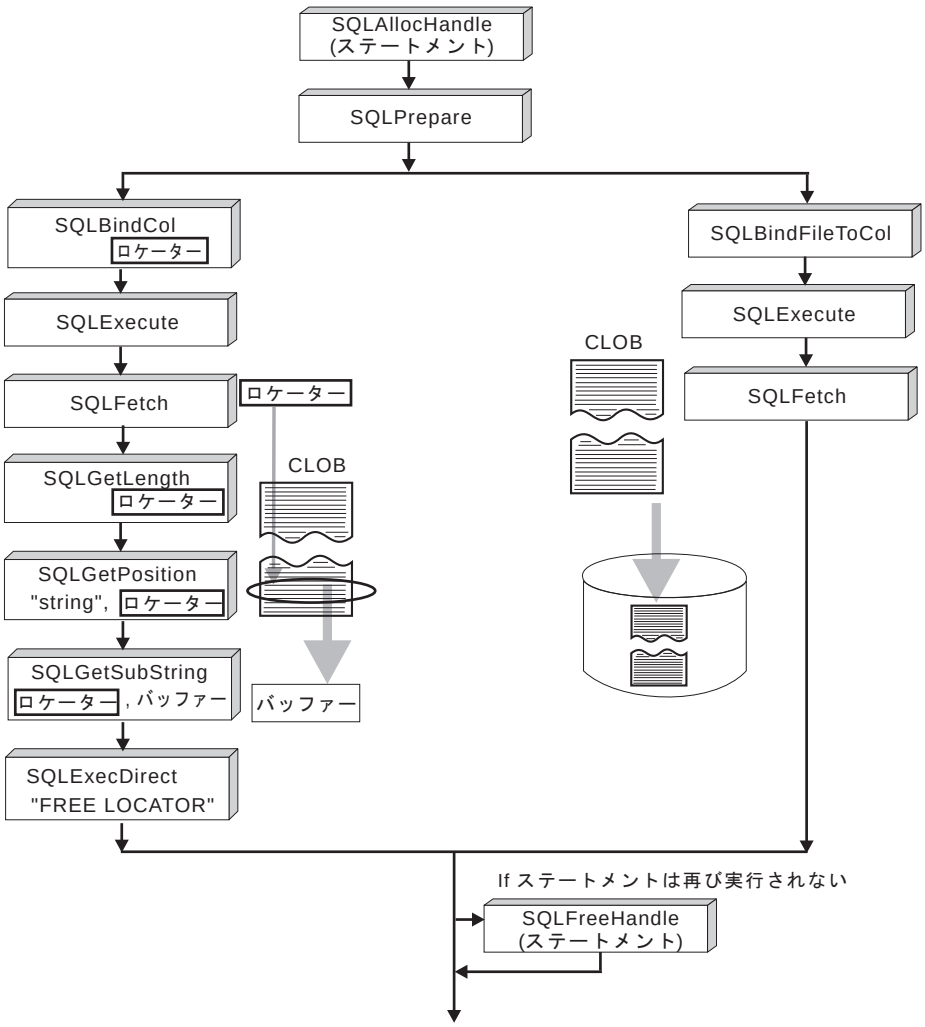


図 3. CLOB データのフェッチ

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・タイプにフェッチします。
- 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。

CLI アプリケーションでは、LOB データを取り出すステートメントでは、デフォルトで、LOB 値を参照する LOB ロケーターとともに行データが戻されます。適切なサイズのバッファーが LOB 列にバインドされている場合、LOB 値は、LOB ロケーターとしてではなくバッファーで戻されます。

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、行がフェッチされ、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 をフェッチする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)。

LOB ロケーターの使用例

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。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))
```

CLI アプリケーションは、次の VALUES ステートメントを使用して、LOB 値を分割して取得することもできます。

```
VALUES (SUBSTR(:locator, :offset, :length))
```

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。 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。実行またはフェッチの際に、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: 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。

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、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。

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。 SQLParamData() が SQL_NEED_DATA を戻す限り、アプリケーションはステップ 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。

DB2 for z/OS では、完了タイプとして SQL_ROLLBACK を指定して SQLEndTran() 関数への呼び出しを行うことは、SQL_ATTR_FORCE_ROLLBACK 接続属性が設定され、StreamPutData 構成キーワードが 1 に設定され、自動コミット・モードが有効になっている場合に許可されます。

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。

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厳密な型判定を提供する、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、構造化タイプ、および参照タイプという 3 つの種類があります。

注: IDS データ・サーバーに対する実行に関し、現在 CLI はユーザー定義タイプ (UDT) をサポートしていません。IDS データ・サーバーに対して UDT を使用すると、「CLI エラー -999 [IBM][CLI Driver][IDS] まだ実装されていません」が返されます。

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  (this is a regular SQL type, not a UDT)
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。構造化タイプの名前または空ストリング (列が構造化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。

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こと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・ハンドル・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。例えば、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);
```

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有しますが、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厳密な型判定を制御するのに役立ち、特殊タイプで明示的に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・タイプで処理するので、 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。

CLI アプリケーションでの XML データの取り扱い - 概要

CLI アプリケーションは、SQL_XML データ・タイプを使用して XML データを検索および保管できます。このデータ・タイプは、整形式 XML 文書を保管する列を定義するために使用する、 DB2 データベースのネイティブ XML データ・タイプに相当します。 SQL_XML タイプは、SQL_C_BINARY、 SQL_C_CHAR、SQL_C_WCHAR、および SQL_C_DBCHAR の各 C タイプにバインドできます。文字タイプの代わりにデフォルトの SQL_C_BINARY タイプを使用して、文字タイプを使用したときのコード・ページ変換から生じるデータ損失または破壊の可能性を回避してください。

XML データを XML 列に保管するには、SQL_XML SQL タイプへの XML 値を含むバイナリー (SQL_C_BINARY) または文字 (SQL_C_CHAR、SQL_C_WCHAR、または SQL_C_DBCHAR) バッファーを SQL_XML SQL タイプにバインドして、INSERT または UPDATE SQL ステートメントを実行します。 XML データをデータベースから検索するには、結果セットをバイナリー (SQL_C_BINARY) または文字 (SQL_C_CHAR、SQL_C_WCHAR、または SQL_C_DBCHAR) タイプにバインドします。エンコードの問題があるため、文字タイプは注意して使用してください。

XML 値が取得されてアプリケーション・データ・バッファーに入れられるとき、DB2 サーバーは XML 値に対する暗黙的なシリアライゼーションを実行して、それ

を保管されている階層フォームからシリアライズされたストリング・フォームに変換します。文字タイプのバッファーでは、XML 値は文字タイプに関連したアプリケーション文字コード・ページに対して暗黙的にシリアライズされます。

デフォルトでは、XML 宣言はシリアライズされた出力ストリングに含まれています。このデフォルトの動作は、SQL_ATTR_XML_DECLARATION ステートメントまたは接続属性を設定することにより、または XMLDeclaration CLI/ODBC 構成キーワードを db2cli.ini ファイル内に設定することにより、変更できます。

CLI アプリケーションで XQuery 式および SQL/XML 関数を発行して実行できます。 SQL/XML 関数は、他の SQL ステートメントと同様に発行および実行できます。大文字小文字の区別のないキーワード **XQUERY** を XQuery 式に接頭部として追加するか、XQuery 式に関連付けられたステートメント・ハンドルの SQL_ATTR_XQUERY_STATEMENT ステートメント属性を設定する必要があります。

注: DB2 バージョン 9.7 フィックスパック 5 以降、SQL_XML データ・タイプは i V7R1 サーバー以降のリリース用の DB2 でサポートされています。

CLI アプリケーションでのデフォルトの XML タイプ処理の変更

CLI は、XML 列およびパラメーター・マーカーを記述するか、またはそこに SQL_C_DEFAULT を指定するとき、デフォルト・タイプが戻されることを予期しないアプリケーションのために互換性を提供する CLI/ODBC 構成キーワードをサポートします。それ以前の CLI および ODBC アプリケーションは、 XML 列またはパラメーターを記述するときに、デフォルトの SQL_XML タイプを認識または予期しないことがあります。いくつかの CLI または ODBC アプリケーションは、 XML 列およびパラメーター・マーカーに対して SQL_C_BINARY 以外のデフォルト・タイプを期待することもあります。これらのタイプのアプリケーションに互換性を提供するために、 CLI は MapXMLDescribe および MapXMLCDefault キーワードをサポートしています。

MapXMLDescribe は、XML 列またはパラメーター・マーカーが記述されている場合にどの SQL データ・タイプが戻されるかを制御します。

MapXMLCDefault は、CLI 関数で XML 列およびパラメーター・マーカーに対して SQL_C_DEFAULT が指定されている場合に使用される C タイプを指定します。

第 6 章 CLI でのトランザクション処理の概説

82 ページの図 4 は、 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。

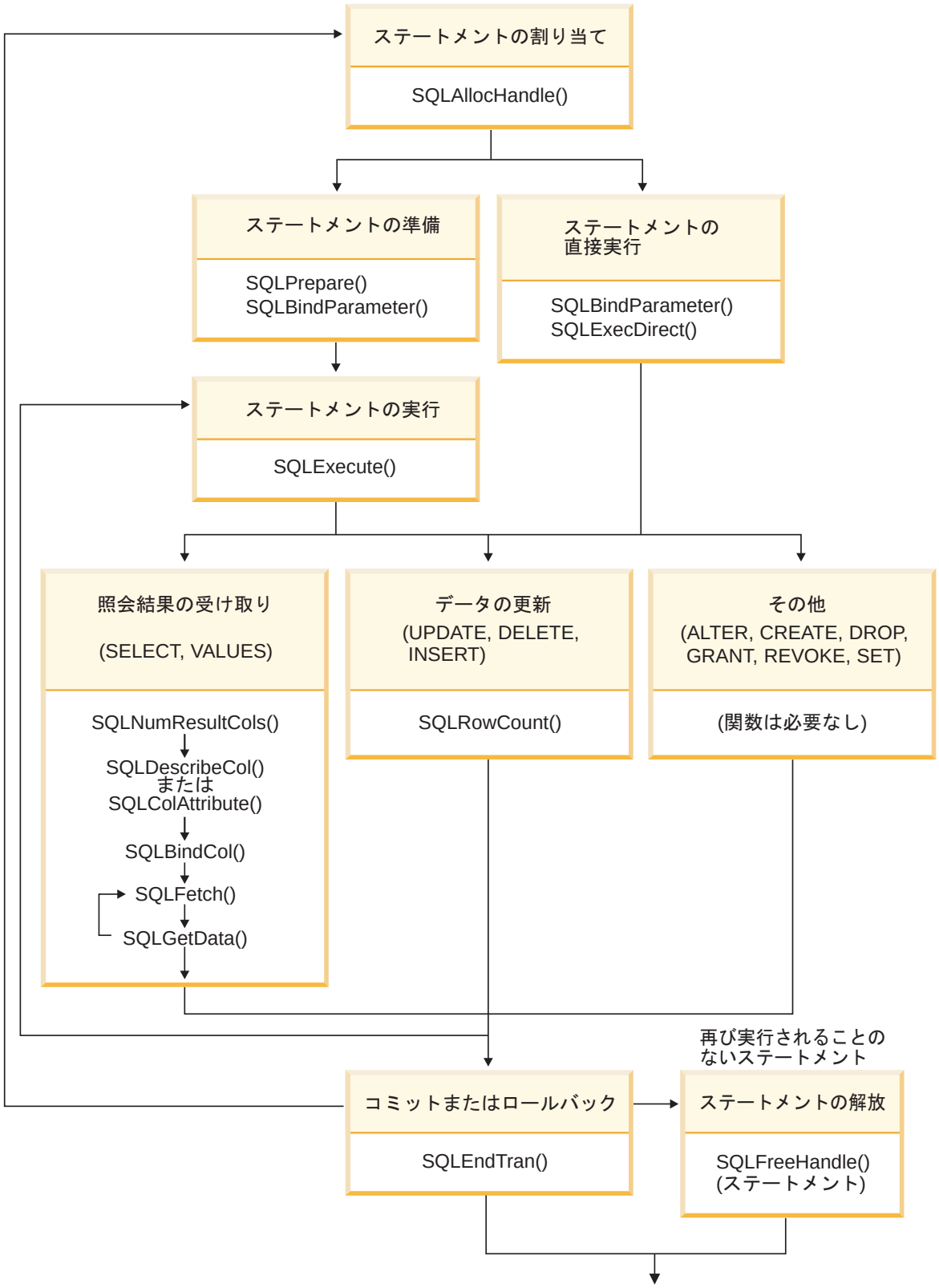


図 4. トランザクション処理

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

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。

始める前に

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。

タスクの結果

環境ハンドル、接続ハンドル、およびステートメント・ハンドルを割り振ると、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。

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 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、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。

始める前に

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数を直接関数に渡します。例えば、以下のようにします。

  ```
  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、次のようにします。

  1. SQLCHAR エレメントの配列 (各エレメントが個々の SQL ステートメントを表す) 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単一の SQLCHAR 変数を初期設定します。例えば、以下のようにします。

     ```
     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')";
     ```

  2. 次の例に示されているように、SQLExecDirect() を呼び出して、ステートメント・ハンドル内の最初のステートメントを発行してから、SQLMoreResults() を呼び出して、ステートメント・ハンドル内の後続のステートメントを発行します。

     ```
     /* Issuing the first SELECT statement of multiple_stmts */
     cliRC = SQLExecDirect (hstmt, multiple_stmts, SQL_NTS);
     /* ... process result-set of first SELECT statement ... */

     /* Issuing the second SELECT statement of multiple_stmts */
     cliRC = SQLMoreResults(hstmt);
     /* ... process result-set of second SELECT statement ... */

     /* Issuing the INSERT statement of multiple_stmts */
     cliRC = SQLMoreResults(hstmt);
     /* cliRC is set to SQL_NO_DATA_FOUND to indicate that */
     /* there are no more SQL statements to issue */
     ```

     ある SQL ステートメントのリストが同じステートメント・ハンドル上で指定されている場合、一度に発行されるステートメントは 1 つだけです。その際には、リスト内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発行されます。

- 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、87 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』を参照してください。

- 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、 140 ページの『CLI/ODBC 静的プロファイル作成による静的 SQL の作成』を参照してください。

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 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。 (組み込み静的 SQL で、ホスト変数が使用できる箇所では、パラメーター・マーカーが使用されます。) CLI は、疑問符 ? で表される名前なしパラメーター・マーカーと、コロンの後に名前が続く形 (例えば、:*name*。ここで *name* は有効な ID) で表される名前付きパラメーター・マーカーをサポートします。名前付きパラメーター・マーカーを使用するには、EnableNamedParameterSupport 構成キーワードを TRUE に設定することによって、名前付きパラメーターの処理を明示的に有効にする必要があります。

パラメーター・マーカーは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、 CLI はファイルの内容をデータベース・サーバーに直接転送します。

アプリケーションが次の場所にパラメーター・マーカーを置くことはできません。

- SELECT リストの中
- 比較述部の両方の式として
- 2 項演算子の両方のオペランドとして
- BETWEEN 演算の第 1 および第 2 オペランドの両方として
- BETWEEN 演算の第 1 および第 3 オペランドの両方として
- IN 演算の式および最初の値の両方として
- 単項の + または - 演算のオペランドとして
- SET FUNCTION 参照の引数として

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、 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 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、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、 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。例えば、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、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、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
- 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)) の自動移植を有効にした場合、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することが望ましい場合があります。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。

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。

始める前に

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

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、保管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。

始める前に

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。

このタスクについて

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・サイズの引数 (*BufferLength*) を使用して、 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_PARAM_BIND_TYPE ステートメント属性を SQL_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。

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。

始める前に

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。例えば、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。例えば、次のようになります。

```
/* 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,
         3, 0, R[0].B, 3, &R.Lb);

/* Parameter C */
rc = SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
         10, 0, R[0].C, 10, &R.Lc);
```

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。 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。

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)。

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。

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、 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。

始める前に

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域に移動します。
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 は相対位置の値を 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. 必要に応じてステップ 4 および 5 を繰り返します。

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フェッチする時に、アプリケーションがパラメーター・データ値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。

始める前に

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。

このタスクについて

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 CLI 関数だけ呼び出せます。

- SQLParamData() および SQLPutData() 関数
- 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、ループを強制終了します。
- SQLGetDiagRec() 関数。

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。

手順

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. 入力データ長ポインターを、実行時に値 SQL_DATA_AT_EXEC が入れられる変数を指すように設定します。例えば、次のようにします。

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

タスクの結果

すべての実行時データ・パラメーターに値が割り当てられると、 SQLParamData() は SQL ステートメントの実行を完了し、元々 SQLExecDirect() または SQLExecute() が作成するはずであった戻り値および診断を返します。

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) または取り消す (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。

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 (整合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。 CLI が手動コミット・モードにある場合、アプリケーションがまだトランザクションになく、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。

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。

アプリケーション内のどの時点でトランザクションを終了するかを決める際には、以下の動作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、従属ステートメントは同一の作業単位の中に保持してください。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 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)。

CLI アプリケーションでの SQL ステートメントの準備および実行

ステートメント・ハンドルを割り振ったら、SQL ステートメントまたは XQuery 式を使用して操作を実行できるようになります。 SQL ステートメントまたは XQuery 式は、実行前に準備しておく必要があります。CLI では、それらを準備して実行するための、2 つの方法が用意されています。つまり、準備および実行操作を個別のステップで実行する方法、および準備および実行操作を結合して 1 つのステップにする方法です。

始める前に

SQL ステートメントまたは XQuery 式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 ステートメントまたは XQuery 式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* 引数として SQL ステートメントまたは XQuery 式を渡すことにより、SQL ステートメントまたは XQuery 式を準備します。

     注: ステートメント属性 SQL_ATTR_XQUERY_STATEMENT がこのステートメント・ハンドルについて SQL_TRUE に設定されていない場合、XQuery 式の前には大/小文字を区別しない "XQUERY" キーワードを付ける必要があります。
  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。 CLI は名前付きパラメーター・マーカー (:name など) に加えて、名前なしパラメーター・マーカー (疑問符 (?) によって表される) もサポートしています。

     注:

     – 名前付きパラメーター・マーカーを使用するには、EnableNamedParameterSupport 構成キーワードを TRUE に設定することによって、名前付きパラメーターの処理を明示的に有効にする必要があります。
     – XQuery 式の場合、式そのものにパラメーター・マーカーを指定することはできません。しかし、XMLQUERY 関数を使用して、パラメーター・マーカーを XQuery 変数にバインドすることができます。次に、バインド済みパラメーター・マーカーの値は、実行のために、XMLQUERY で指定された XQuery 式に渡されます。
  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 ステートメントまたは XQuery 式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または式を準備する必要を省きます。以後の実行時には、準備済みステートメントによって既に生成されたアクセス・プランを使用します。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 ステートメントまたは XQuery 式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。 CLI は名前付きパラメーター・マーカー (:name など) に加えて、名前なしパラメーター・マーカー (疑問符 (?) によって表される) もサポートしています。

     注:

– 名前付きパラメーター・マーカーを使用するには、EnableNamedParameterSupport 構成キーワードを TRUE に設定することによって、名前付きパラメーターの処理を明示的に有効にする必要があります。
– XQuery 式の場合、式そのものにパラメーター・マーカーを指定することはできません。しかし、XMLQUERY 関数を使用して、パラメーター・マーカーを XQuery 変数にバインドすることができます。次に、バインド済みパラメーター・マーカーの値は、実行のために、XMLQUERY で指定された XQuery 式に渡されます。

2. *StatementText* 引数として SQL ステートメントまたは XQuery 式を指定した SQLExecDirect() を呼び出すことにより、SQL ステートメントまたは XQuery 式を準備して実行します。

   注: ステートメント属性 SQL_ATTR_XQUERY_STATEMENT がこのステートメント・ハンドルについて SQL_TRUE に設定されていない場合、XQuery 式の前には大/小文字を区別しない "XQUERY" キーワードを付ける必要があります。

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。

– ステートメントまたは式が一度だけ実行される場合。ステートメントまたは式を実行するのに 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。

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 フィーチャーの名前であり、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。据え置き準備の効果が最も大きいのは、アプリケーションによって生成される照会の応答セットが非常に小さく、個々の要求と応答によるリソース使用が照会データの複数のブロックにわたっていない場合です。

注: 据え置き準備が有効な場合でも、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できるようになるため、

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。

CLI アプリケーションでのコンパウンド SQL (CLI) 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・トラフィックを少なくすることができます。

このタスクについて

- コンパウンド SQL (CLI) 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 (CLI) 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 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 (CLI) 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。 例えば、以下のようにします。

```
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。 例えば、以下のようにします。

```
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。 例えば、以下のようにします。

```
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。 例えば、以下のようにします。

```
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。 例えば、以下のようにします。

```
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。 例えば、以下のようにします。

```
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。 例えば、以下のようにします。

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

8. サブステートメントのハンドルを解放します。 例えば、以下のようにします。

```
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。 例えば、以下のようにします。

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

タスクの結果

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。

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フェッチします。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。 DB2 Universal Database バージョン 8 クライアントの導入によって、更新可能な両方向スクロール・カーソルのサポートがクライアントからサーバーに移りました。つまり、DB2 UDB クライアントのバージョン 8 以上を使用していて、更新可能な両方向スクロール・カーソルが必要なアプリケーションは、サーバーが更新可能な両方向スクロール・カーソルをサポートしていることを確認する必要があります。バージョン 8 以降の DB2 UDB servers on Linux、UNIX および Windows、およびバージョン 7 以降の DB2 for z/OS サーバーは、このフィーチャーをサポートしています。 DB2 for z/OS バージョン 7 またはそれ以降の 3 階層環境で両方向スクロール・カーソルにアクセスするには、ゲートウェイが DB2 UDB バージョン 8 またはそれ以降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 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、次のフェッチ操作により 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 CLI にサポートされています。

スクロール不可
前方スクロールの順方向カーソルは、 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、最小のリソース使用量を必要とします。

スクロール可能
次の 3 つのタイプの両方向スクロール・カーソルが 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。作成されたら、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
静的な両方向スクロール・カーソルとは違って、キー・セット主導の両方向スクロール・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・ドリブン・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・ドリブン・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フェッチするので、キーセット・ドリブン・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フェッチしようとする際に、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 CLI は、サーバーがキーセット・ドリブン・カーソルをサポートしている場合に限り、キーセット・ドリブン・カーソルをサポートします。現在 DB2 UDB バージョン 8 サーバーは更新可能な両方向スクロール・カーソルをサポートしています。そのため、アプリケーションがキー・セット・カーソル機能を必要としていて、現在 DB2 for OS/390 バージョン 6 またはそれ以前、あるいは DB2 for UNIX and Windows バージョン 7 またはそれ以前にアクセスしている場合、そのクライアントを DB2 UDB バージョン 8 以降にアップグレードすることはできません。そのサーバーを、バージョン 8 以降にアップグレードすることは可能です。 IDS データ・サーバーは、キーセット・ドリブン・カーソルの使用をサポートしません。

動的 動的両方向スクロール・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・ドリブン・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・ドリブン・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、動的両方向スクロール・カーソルは、CLI において DB2 for z/OS バージョン 8.1 以降のサーバーにアクセスする場合

のみサポートされます。動的両方向スクロール・カーソルは、 IDS データ・サーバーにアクセスする場合、サポートされません。

注: LOB タイプ、LOB タイプに基づく特殊タイプ、LONG VARCHAR、LONG VARGRAPHIC、 DATALINK、 LOB、 XML タイプ、これらのいずれかのタイプに基づく特殊タイプ、または構造化タイプの列は、両方向スクロール・カーソルの選択リスト内で指定することはできません。 CLI は、カーソル・タイプを両方向スクロールから前方スクロールにダウングレードし、CLI0005W (SQLSTATE 01S02) 警告メッセージを戻します。

カーソル属性

表 1 では、CLI でのカーソルのデフォルト属性をリストします。

表 6. 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・センシティビティー	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
前方スクロール[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 [b]	反映可能	更新可能	値並行処理	スクロール可能

- **a** 前方スクロールは、 FOR UPDATE 節を使用しない両方向スクロール・カーソルのデフォルトの振る舞いです。前方スクロール・カーソルで FOR UPDATE を指定すると、更新可能、ロック並行処理、順方向カーソルが作成されます。
- **b** デフォルトの振る舞いは値並行処理ですが、 DB2 on Linux, UNIX and Windows ではロック並列処理もサポートされます。これによって、ペシミスティック・ロッキングが生じます。

キーセット・ドリブン・カーソルの更新

キーセット・ドリブン・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 節が既に指定されている場合を除いて、CLI ドライバーは FOR UPDATE 節を照会に追加します。デフォルトのキーセット・ドリブン・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。ロック並行処理が明示的に要求された場合、ペシミスティック・ロッキングが使用されて、行が読み取られるとすぐにロックがかかります。このレベルのロッキングは、DB2 on Linux、UNIX および Windows サーバーに対してのみサポートされています。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・ドリブン・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() を介して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。

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、前方スクロール・カーソルと両方向スクロール・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。前方スクロール・カーソルの方が両方向スクロール・カーソルよりリソースの使用量が少なくなります。また両方向スクロール・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両方向スクロール・カーソルの追加フィーチャーを付加する必要がない場合は、順方向カーソルを使用する必要があります。

両方向スクロール・カーソルが必要な場合は、静的カーソル、キーセット・ドリブン・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、リソースの使用量を最小に抑えられます。アプリケーションにキーセット・ドリブン・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 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・ドリブン・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。動的両方向スクロール・カーソルの結果セットの行に対して更新および削除を実行するには、 UPDATE または DELETE ステートメントが、基本表に最低 1 つのユニーク・キーのすべての列を組み込んでいなければなりません。これは、主キーでも他のユニーク・キーでもかまいません。動的カーソルはキーセット・ドリブン・カーソルと比較して、リソースの使用量が多く、また並行性が低くなる可能性もあるため、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する必要がある場合にのみ、動的カーソル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両方向スクロール・カーソルを要求すると、 CLI は動的カーソルは不要であると見な

し、キーセット・ドリブン・カーソルを使用します。この動作により、動的カーソルでの使用リソースの増加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、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 CLI ドライバーの動作を変更する。 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動作が保持されます。 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。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。

両方向スクロール・カーソルがサポートされる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

両方向スクロール・カーソルのサポートは新フィーチャーであるため、DB2 for OS/390 または DB2 for Linux, UNIX and Windows の以前のリリースで動作していた一部の CLI/ODBC アプリケーションでは、動作またはパフォーマンスの変化が起こる可能性があります。両方向スクロール・カーソルを要求したアプリケーションは、両方向スクロール・カーソルがサポートされる前は前方スクロール・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。両方向スクロール・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 7. 両方向スクロール・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	両方向スクロール・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、両方向スクロール・カーソルの要求を前方スクロール・カーソルに自動的にダウングレードします。
DisableKeysetCursor=1	キー・セット主導両方向スクロール・カーソルを無効にします。これは、キーセット・ドリブン・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。

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

フェッチ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、初めてデータのフェッチが行われる前に行セットのサイズを指示し、 2 回目以降のフェッチ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

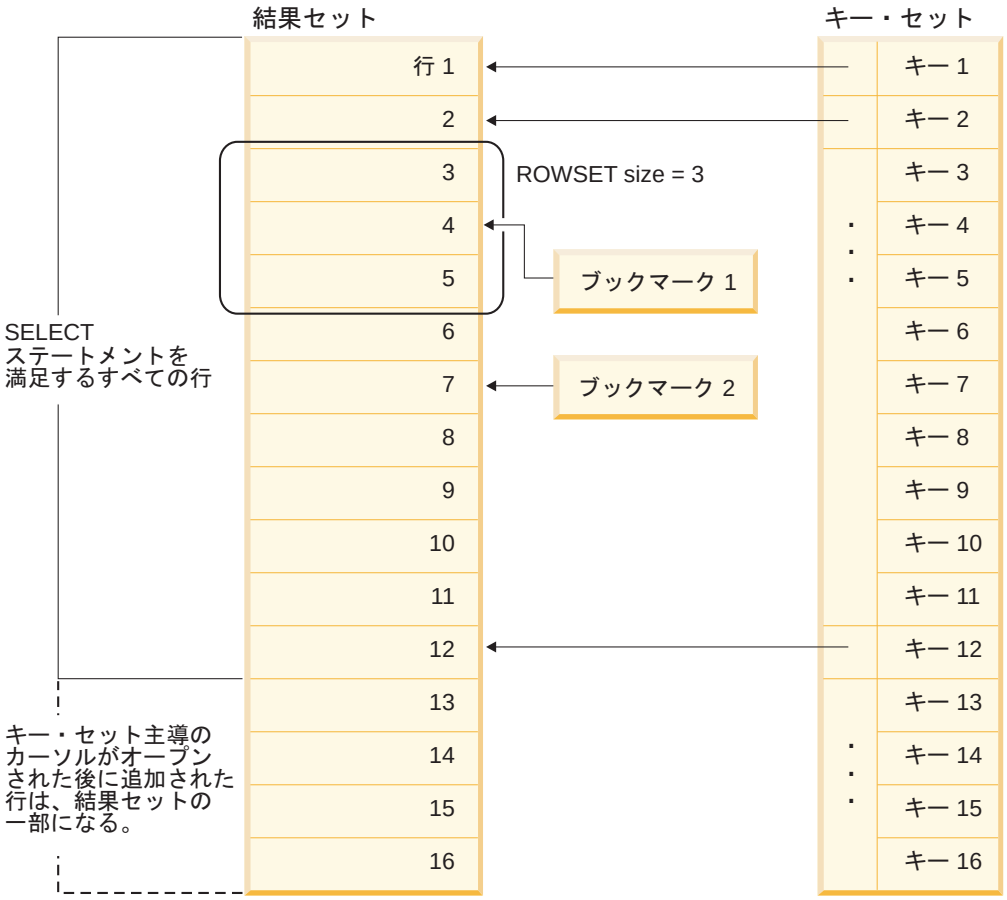
ブックマークといわれる、結果セットにある特定の行への参照を、保管することができます。一度保管すると、アプリケーションは結果セット全体を移動し続けることができ、そして行セットを生成するためブックマークされた行に戻ります。また SQLBulkOperations() による更新や削除を実行する場合にも、ブックマークを使用できます。

キー・セット

キーセット・ドリブン・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・ドリ

ブン・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、結果セット、行セット、ブックマーク、およびキー・セットの間の関係が示されています。



CLI アプリケーションのブックマーク

両方向スクロール・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存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基点として (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。

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。例えば、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 行を基点として 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。

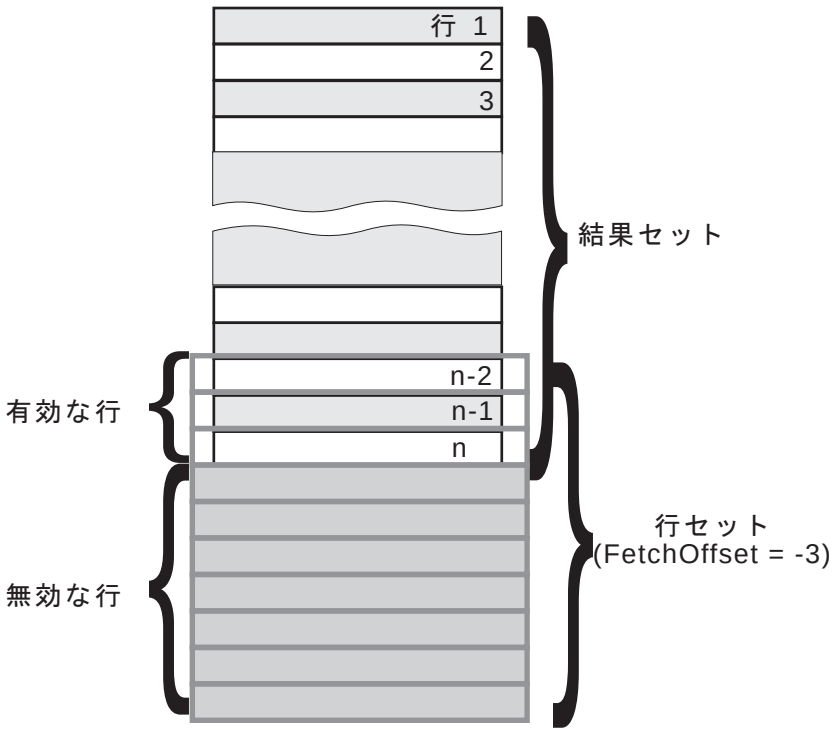


図 5. 一部の行セットの例

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。

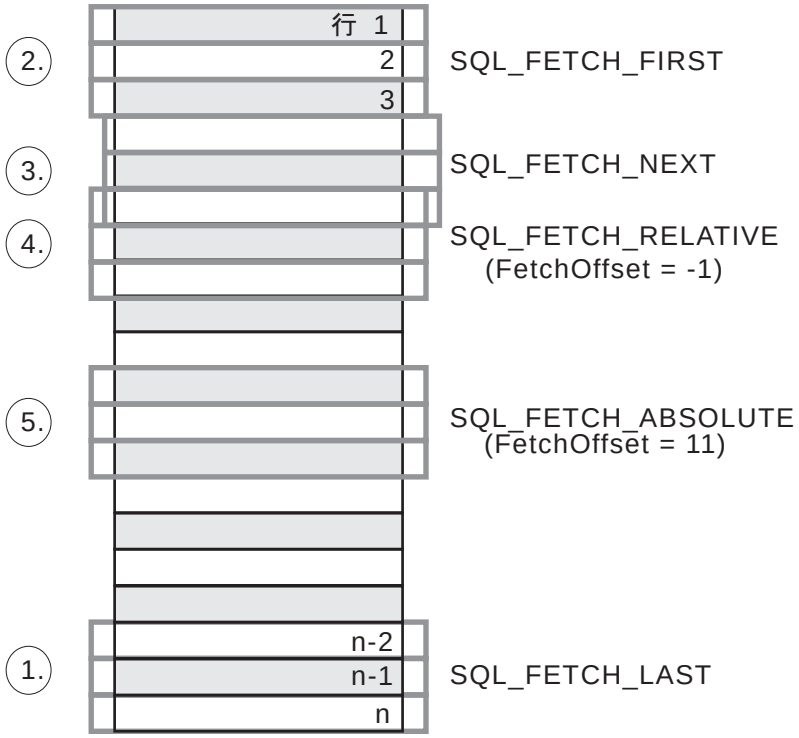


図 6. 行セットの取り出しの例

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。

始める前に

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できるようになります。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、記述のために余計にリソースが使用され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。 例えば、以下のようにします。

```
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数にバインドされます。すなわち、データは、フェッチされるときに、これらの保管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 CLI がこれらの引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。 例えば、以下のようにします。

```
/* 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フェッチでき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。例えば、以下のようにします。

```
/* 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。

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、フェッチ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。フェッチ時には、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。

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

  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 または XML 列にバインドするときに使用します。 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・タイプと長さで保管場所を割り振って、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。データ値が以下のようなラージ可変長データのときは、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、 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、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。例:

```
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。

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。

始める前に

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。

このタスクについて

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順方向カーソルまたは両方向スクロール・カーソル用に、行セットを指定できます。

手順

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ステップ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。例えば、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。例えば、以下のようにします。

```
/* ... */
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 が返されない場合は、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。例えば、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、代わりに前方スクロール・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

取り出す行セット	FetchOrientation 値	スクロール・バー
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。

   例えば、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 行を基点として 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。

CLI アプリケーションでの両方向スクロール・カーソルによるデータの取り出し

両方向スクロール・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、このフィーチャーを使用できます。このトピックでは、両方向スクロール・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。

始める前に

両方向スクロール・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

両方向スクロール・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。 例えば、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. 使用する両方向スクロール・カーソルのタイプを指定します。 SQLSetStmtAttr() を使用して、静的な読み取り専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・ドリブン・カーソルの場合は SQL_CURSOR_KEYSET_DRIVEN に設定してください。 例えば、以下のようにします。

   ```
   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、デフォルトの前方スクロールの順方向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。 例えば、以下のようにします。

   ```
   /* ... */
   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: 両方向スクロール・カーソルと共にブックマークを使用する場合は、SQL_ATTR_USE_BOOKMARKS ステートメント属性を SQL_UB_VARIABLE に設定してください。 例えば、以下のようにします。

```
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フェッチします。
   a. SQLFetchScroll() を呼び出して、結果セットからデータの行セットをフェッチします。 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。 例えば、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。

      例えば、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 行を基点として 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。

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、両方向スクロール・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存できます。データを検索する際に、このフィーチャー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。

始める前に

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両方向スクロール・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

ブックマークと両方向スクロール・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)。 例えば、以下のようにします。

   ```
   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、行セット中の必要な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。 例えば、以下のようにします。

   ```
   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。例えば、*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。例えば、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。

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 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。

始める前に

ブックマークおよび SQLBulkOperations() を使用してバルク・データをフェッチする前に、 CLI アプリケーションを初期化しておいてください。

このタスクについて

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() を呼び出して、SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性をフェッチする行数に設定する。
4. フェッチする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. フェッチする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* 引数に SQL_FETCH_BY_BOOKMARK を指定して SQLBulkOperations() を呼び出し、データをフェッチする。

   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。

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フェッチすることがあります。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、個々のフェッチ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、次のフェッチ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一度にデータの複数行 (行セットと呼ばれます) 配列内へ取り出すことによって、余分なデータ・コピーや余分な SQLBindCol() 呼び出しによるリソースの使用を回避できます。

注: リソースの使用量を少なくする 3 番目の方法はバインドの相対位置を指定することで、この方法は単独でも配列でも使用できます。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。これは行相対位置のバインドでのみ使用できます。

結果セットを配列に取り出す場合、SQLBindCol() を使用して、アプリケーションの配列変数用の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 7 は、列方向バインドの論理ビューです。

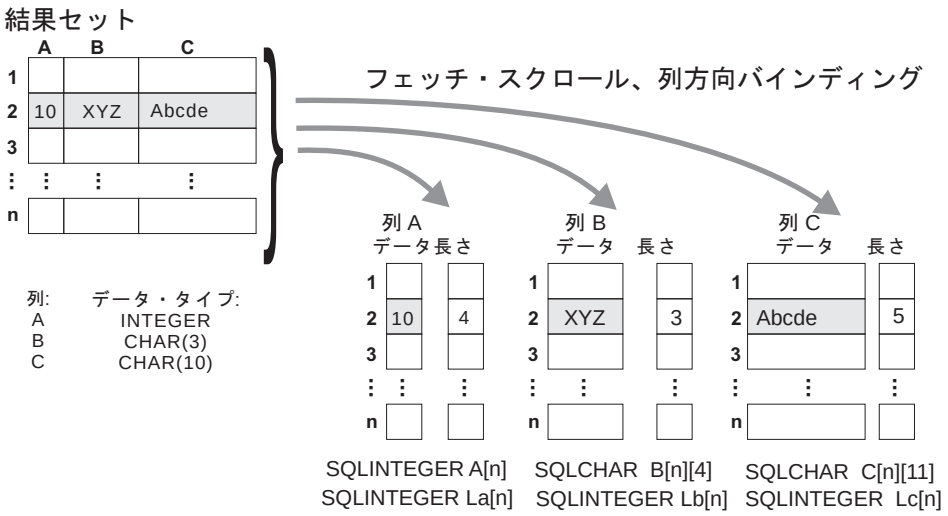


図 7. 列方向バインド

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 8 は、行方向バインドを図示しています。

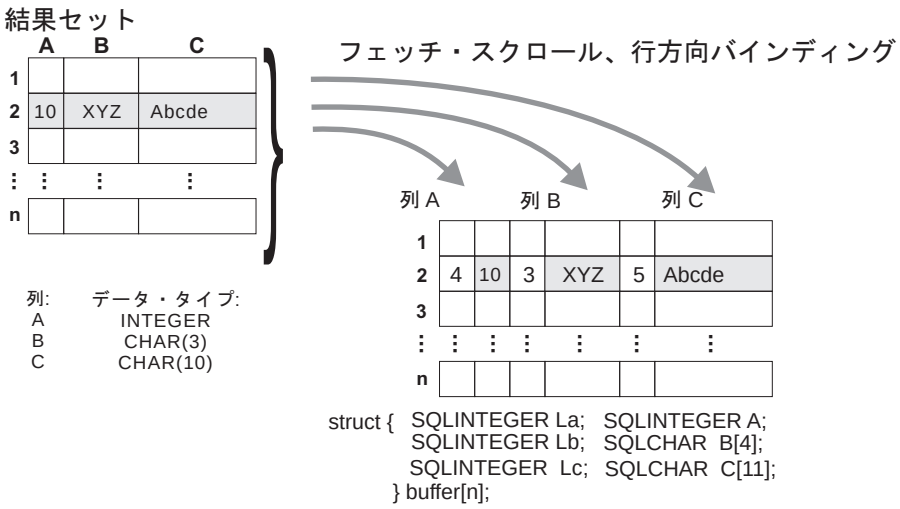


図 8. 行方向バインド

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。

始める前に

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の長さを保管します。
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、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、 CLI は SQLBindCol() の最大バッファー・サイズ引数 (*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。

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。行セット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。

始める前に

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。

   例えば、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、次の例の構造体を割り振ることができます (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。

   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、 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。

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 (例えば、次のフェッチのために)、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、 CLI は列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。

始める前に

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。

このタスクについて

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域に移動します。
2. 通常どおり、SQLFetch() または SQLFetchScroll() を呼び出して、データをフェッチします。返されるデータは、ステップ 1 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 は相対位置の値を 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. 必要に応じてステップ 4 および 5 を繰り返します。

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() のフィーチャーを使用すると、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。

SQLGetData() (SQLFetch() の後に呼び出される) 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。

CLI アプリケーションでの LOB ロケーターによる LOB データのフェッチ

アプリケーションがラージ・オブジェクト・ロケーター (LOB ロケーター) を参照してラージ・オブジェクト値をフェッチする必要が生じる場合がよくあります。

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が例示されています。LOB ロケーターがフェッチされ、それからこのロケーターが CLOB 内でサブストリングの見つけ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。

始める前に

LOB ロケーターを使用して LOB データをフェッチする場合は、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

LOB ロケーターを使用して LOB データをフェッチするには、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。例えば、以下のようにします。

```
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フェッチします。

```
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。例えば、以下のようにします。

```
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・ストリング内の検索ストリングの位置を入手します。検索ストリングを LOB ロケーターによって表すこともできます。例えば、以下のようにします。

```
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。例えば、以下のようにします。

```
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、 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。例えば、次のようになります。

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

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

CLI アプリケーション内での XML データ検索

表の XML 列からデータを選択するとき、出力データはシリアライズされたストリング・フォーマットとなります。

XML データでは、SQLBindCol() を使用して照会結果セット内の列をアプリケーション変数にバインドするとき、アプリケーション変数のデータ・タイプを SQL_C_BINARY、SQL_C_CHAR、SQL_C_DBCHAR、または SQL_C_WCHAR として指定できます。 XML 列から結果セットを検索するとき、アプリケーション変数を SQL_C_BINARY タイプにバインドすることをお勧めします。文字タイプにバインドすると、コード・ページ変換によりデータ損失が生じる可能性があります。データ損失は、ソース・コード・ページ内の文字をターゲット・コード・ページで表現できないときに生じることがあります。変数を SQL_C_BINARY C タイプにバインドすることにより、これらの問題を回避できます。

XML データは、内部エンコード・データとしてアプリケーションに戻されます。 CLI は、データのエンコード方式を次のように決定します。

- C タイプが SQL_C_BINARY の場合、CLI はデータを UTF-8 コード化スキームで戻します。
- C タイプが SQL_C_CHAR または SQL_C_DBCHAR の場合、 CLI はデータをアプリケーション・コード・ページのコード化スキームで戻します。
- C タイプが SQL_C_WCHAR の場合、CLI はデータを UCS-2 コード化スキームで戻します。

データベース・サーバーは、データをアプリケーションに戻す前に、そのデータに対する暗黙的なシリアライゼーションを実行します。 XMLSERIALIZE 関数を呼び出すことにより、XML データを特定のデータ・タイプに明示的にシリアライズすることができます。ただし、XMLSERIALIZE によって文字タイプに明示的にシリアライズするとエンコードの問題が生じることがあるので、暗黙的なシリアライゼーションが推奨されています。

次の例は、XML データを XML 列から検索して、バイナリー・アプリケーション変数にする方法を示しています。

```
char xmlBuffer[10240];
// xmlBuffer is used to hold the retrieved XML document
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

length = sizeof (xmlBuffer);
SQLExecute (hStmt, "SELECT deptdoc FROM dept WHERE id='001'", SQL_NTS);
SQLBindCol (hStmt, 1, SQL_C_BINARY, xmlBuffer, &length, NULL);
SQLFetch (hStmt);
SQLCloseCursor (hStmt);
// xmlBuffer now contains a valid XML document encoded in UTF-8
```

データの挿入

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。

始める前に

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。

このタスクについて

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する行数に設定する。
4. 挿入する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。

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。

始める前に

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。

注: IDS データ・サーバーにアクセスする場合、IBM DB2 **LOAD** ユーティリティー用の CLI **LOAD** インターフェースは、サポートされません。

このタスクについて

注: バージョン 9.7 フィックスパック 4 以降、このフィーチャーは CLI の非同期処理フィーチャーでも使用できるようになりました。

- 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。
- 挿入されるデータの準備済み SQL ステートメントに SELECT 節が含まれる場合、パラメーター・マーカーはサポートされません。
- 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 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。
- 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。例えば、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。
- 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。

   例えば、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、以下の CLI 関数を除く CLI 関数は呼び出せません (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必要なロード・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. オプション: LOAD API の ANYORDER ファイル・タイプ修飾子オプションを使用すると、ロードのパフォーマンスを向上できる可能性があります。ファイル・タイプ修飾子オプション ANYORDER を指定するには、SQLSetStmtAttr() でステートメント属性 SQL_ATTR_LOAD_MODIFIED_BY を設定します。

   例えば、次の呼び出しは、CLI LOAD に対して anyorder ファイル・タイプ修飾子を指定します。

   ```
   char *filemod="anyorder";
   SQLSetStmtAttr (hstmt, SQL_ATTR_LOAD_MODIFIED_BY,
                   (SQLPOINTER) filemod, SQL_NTS);
   ```

4. 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);
   ```

5. 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)。
6. オプション: CLI **LOAD** 操作の後に、その操作によって処理された行数を照会するには、以下のステートメント属性を使用します。
   - 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。

これらのステートメント属性を使用して、CLI **LOAD** によって処理された行数を照会するには、アプリケーションは CLI **LOAD** を行う前に SQLSetStmtAttr を呼び出し、値が格納されるメモリー内の場所へのポインターを渡しておく必要があります。

例えば、ステップ 1 と同様に、ステートメント属性 SQL_ATTR_USE_LOAD_API を指定して SQLSetStmtAttr を呼び出すことによって CLI **LOAD** をオンにした後に、SQLSetStmtAttr を呼び出し、値が格納されるメモリー内の場所へのポインターを渡してから、INSERT を実行して CLI **LOAD** を行います。

```
int *rowsLoaded;
int *rowsDeleted;

rowsLoaded = (int *)malloc(sizeof(int));
if (rowsLoaded == NULL)
{
  // Handle any memory allocation failure by malloc
}
rc = SQLSetStmtAttr(hstmt, SQL_ATTR_LOAD_ROWS_LOADED_PTR, rowsLoaded,
SQL_IS_POINTER);

rowsDeleted = (int *)malloc(sizeof(int));
if (rowsLoaded == NULL)
{
  // Handle any memory allocation failure by malloc
}
rc = SQLSetStmtAttr(hstmt, SQL_ATTR_LOAD_ROWS_DELETED_PTR, rowsDeleted,
SQL_IS_POINTER);
```

CLI **LOAD** を行った後に、以下のようにしてステートメント属性値を取得できます。

```
printf("¥n  Value of SQL_ATTR_LOAD_ROWS_LOADED_PTR is %d", *rowsLoaded);
printf("¥n  Value of SQL_ATTR_LOAD_ROWS_DELETED_PTR is %d", *rowsDeleted);
```

また、以下の例のように SQLGetStmtAttr を呼び出して、ステートメント属性値を取得することもできます。INSERT ステートメントを CLI **LOAD** に対して実行する前に、SQLSetStmtAttr を呼び出して、値が格納されるメモリー内の場所へのポインターを渡しておく必要があることに注意してください。

```
int *pStmtAttrValue;

rc = SQLGetStmtAttr(hstmt,
                    SQL_ATTR_LOAD_ROWS_LOADED_PTR,
                    &pStmtAttrValue,
                    sizeof(pStmtAttrValue),
                    NULL);
printf("¥n  Value of SQL_ATTR_LOAD_ROWS_LOADED_PTR  is %d", *pStmtAttrValue);

rc = SQLGetStmtAttr(hstmt,
                    SQL_ATTR_LOAD_ROWS_DELETED_PTR,
                    &pStmtAttrValue,
                    sizeof(pStmtAttrValue),
                    NULL);
printf("¥n  Value of SQL_ATTR_LOAD_ROWS_DELETED_PTR is %d", *pStmtAttrValue);
```

CLI アプリケーションでの XML 列の挿入および更新

表の XML 列でデータを更新または挿入するとき、入力データはシリアライズされたストリング・フォーマットでなければなりません。

XML データでは、SQLBindParameter() を使用してパラメーター・マーカーを入力データ・バッファーにバインドするとき、入力データ・バッファーのデータ・タイプを SQL_C_BINARY、SQL_C_CHAR、SQL_C_DBCHAR、または SQL_C_WCHAR として指定できます。

SQL_C_BINARY タイプの XML データを含むデータ・バッファーをバインドするとき、 CLI はその XML データを内部エンコード・データとして処理します。これは文字タイプを使用する場合に、リソースの追加使用と文字変換の際のデータ損失の可能性を回避できるため、より推奨される方法です。

重要: XML データがアプリケーション・コード・ページのコード化スキームではないコード化スキームおよび CCSID でエンコードされた場合、内部エンコードをデータに含めて、そのデータを SQL_C_BINARY としてバインドすることにより文字変換を防止する必要があります。

SQL_C_CHAR、SQL_C_DBCHAR、または SQL_C_WCHAR タイプの XML データを含むデータ・バッファーをバインドするとき、 CLI はその XML データを外部エンコード・データとして処理します。 CLI は、データのエンコード方式を次のように決定します。

- C タイプが SQL_C_WCHAR の場合、CLI はデータが UCS-2 としてエンコードされていると想定します。
- C タイプが SQL_C_CHAR または SQL_C_DBCHAR の場合、 CLI はデータがアプリケーション・コード・ページのコード化スキームでエンコードされていると想定します。

データベース・サーバーがデータを XML 列に保管する前にそれを暗黙的に構文解析するようにするには、 SQLBindParameter() 内のパラメーター・マーカーのデータ・タイプを SQL_XML として指定する必要があります。

XMLPARSE を使用して文字タイプを明示的に構文解析すると、エンコードの問題が生じることがあるので、暗黙的な構文解析が推奨されています。

次の例は、推奨される SQL_C_BINARY タイプを使用して、XML 列内の XML データを更新する方法を示しています。

```
char xmlBuffer[10240];
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

// xmlBuffer contains an internally encoded XML document that is to replace
// the existing XML document
length = strlen (xmlBuffer);
SQLPrepare (hStmt, "UPDATE dept SET deptdoc = ? WHERE id = '001'", SQL_NTS);
SQLBindParameter (hStmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_XML, 0, 0,
                  xmlBuffer, 10240, &length);
SQLExecute (hStmt);
```

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。

始める前に

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、デフォルト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。

   フェッチ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 つはフェッチに 1 つは更新に使用します。フェッチ・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フェッチします。
5. WHERE CURRENT 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。例えば、以下のようにします。

   ```
   sprintf((char *)stmtPositionedUpdate,
           "UPDATE org SET location = 'Toronto' WHERE CURRENT of %s",
           cursorName);
   ```
6. フェッチ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。

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。

始める前に

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。

このタスクについて

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する行数に設定する。
4. 更新する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. 更新する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なされたことを検証する。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フェッチされます。

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。

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。

始める前に

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。

このタスクについて

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する行数に設定する。
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. 削除する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。

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。

始める前に

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。

このタスクについて

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・タイプと引数の長さに一致することを確認します。 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。 例えば、以下のようにします。

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

   または

   ```
   SQLCHAR *stmt = (SQLCHAR *)"CALL OUT_LANGUAGE (:language)";
   ```

   要確認: 名前付きパラメーター・マーカー (:language など) を使用するには、**EnableNamedParameterSupport** 構成キーワードを TRUE に設定することによって、名前付きパラメーターの処理を明示的に有効にする必要があります。

   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節の区切り文字 { } を含めます。例えば、以下のようにします。

```
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。例えば、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。

   注: CALL プロシージャー・ステートメントは、SQL_ATTR_PARAMSET_SIZE 属性を使用したパラメーター・マーカーの配列入力をサポートしません。
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、順方向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、列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。

タスクの結果

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:

値が ISO 形式で戻されない場合、DB2 CLI アプリケーションに戻されるプロシージャー結果セットの中で、DATETYPE データ・タイプ値の数値の月日の部分が逆になります。例えば、ローカル形式が代わりに使用される場合に、これが発生する可能性があります。 DATETYPE データ・タイプ値の情報がクライアント・アプリケーションによって確実に正しく解釈されるようにするには、ロケールに依存しない DATETIME 形式 (例えば ISO) を使用するデータベースにプロシージャーをバインドする必要があります。例えば、以下のようにします。

```
db2set DB2_SQLROUTINE_PREPOPTS="DATETIME ISO"
```

注:

データベースの作成またはアップグレード時に、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。

匿名ブロックからの結果セット

バージョン 9.7 フィックスパック 2 以降、CLI は、BEGIN COMPOUND ではなく BEGIN で始まる SQL ステートメントがアプリケーションによって送信されると、結果セットの準備を整えます。 CLI はサーバーから戻されるカーソルをインタープリットして、アプリケーションが結果セットを取り出せるようにします。

例 1: SQLExecDirect を使用する

```
opt caller on
opt echo on

quickc 1 1 sample

SQLAllocStmt 1 1
getmem 1 1 SQL_C_LONG

SQLExecDirect 1 "drop table t1" -3
SQLExecDirect 1 "create table t1 (c1 int)" -3
SQLExecDirect 1 "insert into t1 values (10)" -3
SQLExecDirect 1 "insert into t1 values (20)" -3
SQLExecDirect 1 "insert into t1 values (30)" -3

SQLExecDirect 1 "begin declare c1 cursor with return to client with hold
for select c1 from t1; end" -3
SQLBindCol 1 1 sql_c_long 1
FetchAll 1

SQLFreeStmt 1 SQL_DROP
SQLTransact 1 1 SQL_COMMIT

killenv 1
```

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 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 8. 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。
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。

CLI/ODBC 静的プロファイル作成による静的 SQL の作成

CLI/ODBC 静的プロファイル作成フィーチャーにより、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・ベースの許可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。

このタスクについて

- 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 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 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。例えば、"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・ユーザーが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検証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 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 (例えば、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 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 = qualified_package_name`
   - `StaticCapFile = capture_file_name`

   例:

   ```
   [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 (この例では MySchema がスキーマ名)。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。

こ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 (例えば、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 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・モードを無効にするか、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、以下の構成キーワードを設定して行います。

   - StaticMode = MATCH
   - StaticCapFile = *capture_file_name*

   例:

   ```
   [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、その情報を使用して、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っていることを検証する必要があります。

9. アプリケーションを実行します。

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
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

フィールド	説明
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。

組み込み SQL と 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 CLI を使用できます。アプリケーション開発者が、CLI カタログ関数の使いやすさを活用し、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するというシナリオを考えてみてください。 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、 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。
  - 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、 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 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。例えば、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、 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。

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。

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。

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。

このタスクについて

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 とお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。

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。

`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、解放する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。

手順

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、 CLI アプリケーションの SQLFreeHandle() を使用します。

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、 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、 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。

第 7 章 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。

始める前に

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。

第 8 章 DB2 Connect を介したトラステッド接続

一部の DB2 データベース・サーバーはトラステッド・コンテキストをサポートしています。*トラステッド・コンテキスト* は、特に、クライアント・アプリケーションがトラステッド接続を作成できる条件をデータベース管理者が定義できるようにします。*トラステッド接続* は通常の接続では不可能なことを実行できます。

トラステッド接続には、暗黙的および明示的という 2 つのタイプがあります。接続の作成時に、明示的または暗黙的なトラステッド接続を取得するか、あるいは通常の接続を取得するかは、表 9 に要約されているように、トラステッド接続を要求するかどうか、さらにはサーバーのトラステッド・コンテキストに定義された基準をその接続が満たしているかどうかに依存します。

表 9. 種々のアクションの組み合わせにより生じる接続のタイプ

	接続がトラステッド接続になるためのサーバーの基準を満たす場合	**接続がトラステッド接続になるためのサーバーの基準を満たさない場合**
トラステッド接続となるように要求する	明示的トラステッド接続	通常の接続および警告 SQL20360W (SQLSTATE 01679) が戻される。
トラステッド接続となるように要求しない	暗黙的トラステッド接続	通常の接続

暗黙的トラステッド接続 は、接続の使用時にユーザーに一時的なロール特権を付与するという以外は、通常の接続と同じです。 (該当する場合) 付与されるロール特権はトラステッド・コンテキストで指定され、その結果、接続は信頼できるものとなります。

暗黙的トラステッド接続は、DB2 Connect を使用して接続する任意のアプリケーションによって作成できます。暗黙的トラステッド接続の作成や使用の方法は、通常の接続と同じです。つまり、既存のアプリケーションが DB2 Connect を介して接続している限りは、そのアプリケーションが暗黙的トラステッド接続を活用するためにコードを変更する必要はありません。

明示的トラステッド接続 は、暗黙的トラステッド接続と同じ仕方で、ユーザーに対して一時的なロール特権を付与します。加えて、明示的トラステッド接続では、その接続を介してアクションを実行する場合に使用する許可 ID を変更することができます。明示的トラステッド接続での許可 ID の変更は、『*ユーザーの切り替え* 』と呼ばれます。トラステッド接続を作成できるようにするトラステッド・コンテキストの一部として、切り替え可能な許可 ID と、切り替え時に指定の許可 ID でパスワードが必要かどうかが定義されています。

ユーザーの切り替えを行うことにより、複数のユーザーでの接続共用の処理使用量を大幅に削減できます。特に、パスワードが不要なユーザー名の場合はデータベース・サーバーで許可 ID の認証を行わないため、その効果は顕著になります。しかしこのフィーチャーの使用時には、必ずご使用のアプリケーションで、許可 ID を

検証および認証することなく、その許可 ID に切り替えることができないようにしなければなりません。そのようにしないと、システムにセキュリティー・ホールを作ってしまいます。

明示的トラステッド接続を作成して、CLI または JDBC を使用した DB2 Connect を介して接続する際に (確立された XA 接続を含む)、ユーザーを切り替えることができます。明示的トラステッド接続の作成とユーザーの切り替えには、特別な接続属性の設定が必要です。つまり、明示的トラステッド接続の利点を生かすには、既存のアプリケーションを変更する必要があります。

前述した相違点以外は、(明示的または暗黙的のどちらでも) トラステッド接続は通常の接続と同じ方法で使用できます。しかし明示的トラステッド接続を切断するには、接続が壊れているか切断状態にある場合であっても、それを必ず明示的に切断しなければなりません。そのようにしないと、その接続が使用しているリソースが解放されない可能性があります。暗黙的トラステッド接続の場合には、これは問題となりません。

注:

1. 明示的トラステッド接続では、クライアント認証を使用しないでください。これは、暗黙的トラステッド接続には当てはまりません。
2. 明示的トラステッド接続を使用するアプリケーションは、パスワードで保護され、許可されたユーザーだけがアクセス可能な機密保護機能のあるマシンで実行してください。これは、暗黙的トラステッド接続には当てはまりません。

CLI を使用したトラステッド接続の作成および終了

CLI による接続時に明示的トラステッド接続を作成できるのは、接続しているデータベース・サーバーがそれを許可するように構成されている場合です。

始める前に

この手順においては、XA トランザクション・マネージャーを使用していないと想定します。 XA トランザクション・マネージャーを使用している場合に唯一必要となるのは、トランザクション・マネージャーが xa_open を呼び出す際に構成値 TCTX を TRUE に設定するよう構成されているのを確認することです。そのように設定されていると、任意の接続を明示的トラステッド接続にすることが可能になります。接続が明示的トラステッド接続であることを確認するには、ステップ 3 を参照してください。

- 接続先のデータベースでトラステッド・コンテキストがサポートされている必要があります。
- クライアントを信頼可能として認識するトラステッド・コンテキストが定義されている必要があります。
- トラステッド・コンテキストで指定されているシステム許可 ID を把握していなければなりません。トラステッド接続のシステム許可 ID は、接続の作成時にユーザー名としてサーバーに提供した許可 ID です。接続が特定のトラステッド・コンテキストによって信頼可能であるとされるには、システム許可 ID がそのト

ラステッド・コンテキストで指定されている必要があります。有効なシステム許可 ID とその ID のパスワードについては、セキュリティー管理者に尋ねてください。

このタスクについて

この手順の例では、C 言語を使用し、*conn* は有効ではあるものの、まだ接続されていない接続ハンドルへのポインターとします。変数、*rc* は SQLRETURN というデータ・タイプであるとします。

手順

1. 通常の接続で設定している接続属性に加え、SQLSetConnectAttr 関数を呼び出して接続属性 SQL_ATTR_USE_TRUSTED_CONTEXT を SQL_TRUE に設定します。

```
rc = SQLSetConnectAttr(
    conn,
    SQL_ATTR_USE_TRUSTED_CONTEXT, SQL_TRUE, SQL_IS_INTEGER
);
```

2. 例えば SQLConnect 関数を呼び出し、通常の接続と同じようにしてデータベースに接続します。 ユーザー名としてシステム許可 ID を、パスワードとして許可 ID のパスワードを使用します。必ずエラーと警告を確認します。特に、表 10 にリストされている項目について確認してください。

表 10. トラステッド接続を作成できなかったことを示すエラー

SQLCODE	SQLSTATE	意味
SQL20360W	01679	接続はトラステッド接続として確立できませんでした。代わりに通常の接続として確立されました。

   ユーザーに報告すべきエラーまたは警告がなければ、明示的トラステッド接続が確立されます。

3. オプション: SQLGetConnectAttr 関数を使用して接続属性 SQL_ATTR_USE_TRUSTED_CONTEXT の値を検査すると、確立された接続が明示的トラステッド接続であることを検証できます。 SQL_TRUE に設定されている場合には、接続は明示的トラステッド接続です。
4. 接続の使用を終了する場合には、接続が壊れているか切断状態にある場合であっても、必ず明示的に切断する必要があります。 明示的トラステッド接続を明示的に切断しないと、接続が使用している一部のリソースが解放されない可能性があります。

タスクの結果

注:

1. 明示的トラステッド接続では、クライアント認証を使用しないでください。これは、暗黙的トラステッド接続には当てはまりません。
2. 明示的トラステッド接続を使用するアプリケーションは、パスワードで保護され、許可されたユーザーだけがアクセス可能な機密保護機能のあるコンピューターでのみ実行してください。これは、暗黙的トラステッド接続には当てはまりません。

CLI を使用したトラステッド接続のユーザーの切り替え

コマンド行インターフェース (CLI) を使用して、明示的トラステッド接続のユーザーを切り替えることができます。

トラステッド接続を使用したユーザーの切り替えに関する説明については、関連リンクのトピックを参照してください。

始める前に

- 接続は明示的トラステッド接続として正常に作成されたものでなければなりません。
- 明示的トラステッド接続はトランザクションに存在することはできません。
- 明示的トラステッド接続を作成できるトラステッド・コンテキストでは、切り替えようとしている許可 ID への切り替えが許可されるように構成されている必要があります。

このタスクについて

この手順の例では、C 言語を使用し、*conn* は接続済み明示的トラステッド接続へのポインターとします。変数、*rc* は SQLRETURN というデータ・タイプであるとします。変数 *newuser* は切り替えるユーザーの許可 ID を保持する文字ストリングへのポインターとします。変数 *passwd* はその許可 ID のパスワードを含む文字ストリングへのポインターとします。

手順

1. SQL_ATTR_TRUSTED_CONTEXT_USERID 属性を設定するために SQLSetConnectAttr 関数を呼び出します。 切り替える許可 ID を設定します。

   ```
   rc = SQLSetConnectAttr(
       conn,
       SQL_ATTR_TRUSTED_CONTEXT_USERID, newuser, SQL_NTS
   );
   //Check for errors
   ```

   必ずエラーと警告を確認します。特に、表 11 にリストされている項目について確認してください。

   表 11. ユーザーの切り替え時に新しい許可 ID を設定できなかったことを示すエラー

SQLCODE	意味
CLI0106E	この接続は、接続されていません。
CLI0197E	この接続は、トラステッド接続ではありません。
CLI0124E	提供された値に問題があります。例えば、NULL でないこと、または長すぎないことを確認してください。
CLI0196E	接続が、ユーザーの切り替えを妨げる作業単位に関係しています。ユーザーを切り替えられるようにするには、接続はトランザクションに存在することはできません。

2. オプション: (このトラステッド接続が許可されるトラステッド・コンテキストで、切り替えようとしている許可 ID のパスワードを必要としない場合、このステップはオプションとなります。)

SQL_ATTR_TRUSTED_CONTEXT_PASSWORD 属性を設定するために SQLSetConnectAttr 関数を呼び出します。 新しい許可 ID のパスワードを設定します。

```
rc = SQLSetConnectAttr(
    conn,
    SQL_ATTR_TRUSTED_CONTEXT_PASSWORD, passwd, SQL_NTS
);
//Check for errors
```

必ずエラーと警告を確認します。 154 ページの表 11 と表 12 の両方にリストされている項目について確認してください。

表 12. ユーザーの切り替え時にパスワードを設定できなかったことを示すエラー

SQLCODE	意味
CLI0198E	属性 SQL_ATTR_TRUSTED_CONTEXT_USERID がまだ設定されていません。

3. 通常の接続として続行します。 XA トランザクション・マネージャーを使用している場合には、次の要求の一部としてユーザー切り替えが試行されます。使用していない場合には、データベースにアクセスする関数 (例えば SQLExecDirect) の次の呼び出しを開始する直前に、このユーザー切り替えが試行されます。どちらの場合であっても、通常確認するエラーと警告に加えて、表 13 にリストされているエラーも確かめてください。 表 13 にあるエラーは、ユーザー切り替えが失敗したことを示しています。

表 13. ユーザーの切り替えが失敗したことを示すエラー

SQLCODE	意味
SQL1046N	このトラステッド接続が許可されているトラステッド・コンテキストは、切り替えようとしている許可 ID への切り替えが許可されるように構成されていません。トラステッド・コンテキストを変更しない限りは、この許可 ID への切り替えはできません。
SQL30082N	提供されたパスワードが、切り替えている許可 ID に対して正しくありません。
ネイティブ・エラー -20361 を伴う SQL0969N	一部のデータベース・レベル制約には、ユーザーへの切り替えを妨げるものがあります。

ユーザー切り替えが失敗すると、別のユーザーへの切り替えが正常に行われるまでは接続は未接続の状態になります。未接続状態にあるトラステッド接続でユーザーを切り替えることができますが、未接続の状態ではデータベース・サーバーにはアクセスできません。未接続状態にある接続は、ユーザーの切り替えが正常に行われるまではそのままの状態です。

次のタスク

注:

1. **重要:** パスワードを提供しないでユーザーを切り替えると、データベース・サーバーの認証がバイパスされます。ご使用のアプリケーションが既に許可 ID を検証して認証済みでない限りは、アプリケーションがパスワードなしでその許可 ID に切り替えられないようにしてください。そのようにしないと、セキュリティー・ホールができてしまいます。

2. SQL_ATTR_TRUSTED_CONTEXT_USERID 属性に NULL 値を指定するのは、トラステッド・コンテキストのシステム許可 ID (明示的トラステッド接続が作成された際に使用したユーザー ID) を指定するのと同じです。
3. 明示的トラステッド接続で SQL_ATTR_TRUSTED_CONTEXT_USERID 接続属性の値を正常に設定すると、接続は即時にリセットされます。リセットされると、元の接続の接続属性を使用して新しい接続が作成されたかのようになります。接続属性の値を、システム許可 ID、NULL、または属性が現在保持しているのと同じ値に設定しても、リセットが生じます。
4. SQL_ATTR_TRUSTED_CONTEXT_PASSWORD 属性が設定されると、トラステッド接続が許可されているトラステッド・コンテキストがその許可 ID の切り替えユーザーに関して認証を必要としない場合であっても、ユーザー切り替えプロセスでパスワードが認証されます。結果として、不要な処理時間が発生します。この規則は、トラステッド・コンテキストのシステム許可 ID には当てはまりません。トラステッド・コンテキストのシステム許可 ID が、その切り替え時に認証を必要としない場合には、パスワードが提供されても認証は行われません。

第 9 章 CLI アプリケーションの記述子

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。

例えば、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。例えば、ある表からフェッチ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 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、 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)
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。

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。例えば、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インプリメンテーション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。

以下にヘッダー・フィールドをリストします。

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

注:

[a] 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)。 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。

記述子レコード内のフィールドは 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。記述子レコードのフィールドは、次のとおりです。

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

SQL_DESC_CARDINALITY

SQL_DESC_CARDINALITY_PTR

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ヌル終了であると見なす。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 (複数行フェッチ専用)	複数行のフェッチで、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 (複数行フェッチ専用)	(なし)
SQL_DESC_CARDINALITY_PTR	(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。

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、必ず整合性検査を求められます。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「記述子情報が矛盾します。」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、 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検証する。
- SQL_DESC_CONCISE_TYPE フィールドが時間データ・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証する。

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。

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、インプリメンテーション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。以下の例では、アプリケーション行記述子を明示的に割り当てます。

```
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。複数行のフェッチを可能にするには、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値がありません。これらのフィールドはステートメントの準備または実行時に設定されます。

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 の自動移植を行うと、リソースの使用量が増えます。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。

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 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

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フェッチします。

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、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、CLI ではサポートされていません。)

例えば、以下の呼び出しを使用すると、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。例えば、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)。

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 つ以上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フェッチします。
  - 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。

第 10 章 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 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フェッチすることができ、通常どおり `SQLFetch()` によって列データを処理します。 CLI カタログ関数は、次のとおりです。

- `SQLColumnPrivileges()`
- `SQLColumns()`
- `SQLExtendedProcedures()` (DB2 バージョン 9.7 フィックスパック 1 以降)
- `SQLExtendedProcedureColumns()` (DB2 バージョン 9.7 フィックスパック 1 以降)
- `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。

注: デフォルトでは、IDS データ・サーバーは、システム・カタログにあるスキーマ情報 (表名および列名など) を小文字で返します。一方、DB2 データ・サーバーは、スキーマ情報を大文字で返します。

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ります。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。

CLI アプリケーションのカタログ関数の入力引数

すべてのカタログ関数には、入力引数リストに *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数は、普通の引数として処理される場合と、パターン値引数として処理される場合があります。普通の引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、いくつかの入力引数でパターン値を受け入れるものがあります。例えば、`SQLColumnPrivileges()` は、 *SchemaName* および *TableName* を普通の引数として扱い、 *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 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 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数があると、基本照会内の WHERE 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。

これらの引数値は、WHERE 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。

例えば、以下の呼び出しは先頭が「ST」の表をすべてフェッチします。

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

第 11 章 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、以下の主題について説明します。

- KEEPDYNAMIC サポート
- 共通接続属性
- 共通ステートメント属性
- ステートメント・ハンドルの再利用
- バインドおよび SQLGetData()
- カタログ関数の使用を制限する
- 関数生成による結果セットの列名
- ODBC アプリケーションからロードされる CLI 固有の関数
- グローバル動的ステートメント・キャッシュ
- データ追加および検索の最適化
- ラージ・オブジェクト・データの最適化
- オブジェクト ID の大/小文字の区別
- SQLDriverConnect() と SQLConnect()
- ステートメント・スキャンをオフにする
- 複数のロールバックにわたりカーソルを保持する
- コンパウンド SQL サブステートメントの準備 (PREPARE)
- ユーザー定義タイプおよびキャスト
- ネットワーク・フロー削減のための準備処理の据え置き

KEEPDYNAMIC 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維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 CLI/ODBC アプリケーションは、DB2 for z/OS and OS/390 バージョン 7 以降のサーバー上で **KEEPDYNAMIC** 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** を使用できるようにするには、以下のステップを完了します。

1. DB2 for z/OS and OS/390 サーバー上で動的ステートメント・キャッシュを使用できるようにします (DB2 for z/OS and 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 Database for Linux, UNIX, and 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. 以下の例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 (ステップ 2 で作成された **KeepDynamic = 1**、**CurrentPackageSet = コレクション名**) を設定します。例:

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

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

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

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

KEEPDYNAMIC および構成の詳細については、DB2 for z/OS and OS/390 サーバー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、CLI アプリケーションで設定できます。

- SQL_ATTR_AUTOCOMMIT - 各コミット要求が余分なネットワーク・フローを生成するので、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 にしておきます。

  注: デフォルト値は SQL_AUTOCOMMIT_ON です。

- SQL_ATTR_TXN_ISOLATION - この接続属性により、接続またはステートメントが動作する分離レベルが決定されます。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。アプリケーションは、データの整合性を保ちつつ、並行性を最大にす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、CLI アプリケーションで設定できます。

- SQL_ATTR_MAX_ROWS - この属性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。非常に大きな結果セットが不用意に生成されて、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) が処理を実行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 for z/OS and OS/390 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 FIRST n ROWS ONLY』 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 FIRST n ROWS ONLY』 節をサポートしない DBMS の場合、サーバー側で 『OPTIMIZE FOR

n ROWS』 節を使用した全結果セットが生成されますが、CLI は、クライアント上で行をカウントし、SQL_ATTR_MAX_ROWS 行までをフェッチします。

- SQL_ATTR_CURSOR_HOLD - このステートメント属性は、 CLI がこのステートメントのカーソルを WITH HOLD 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の属性を SQL_CURSOR_HOLD_OFF に設定しておくと、サーバーはステートメント・ハンドルに関連するリソースをより適切に活用することができます。この属性を適切に設定して使用すれば OS/390 および z/OS 上での効率が大幅に向上します。

  注: 多くの ODBC アプリケーションは、コミット後にカーソル位置が保持されることをデフォルトの動作として想定しています。

- SQL_ATTR_TXN_ISOLATION - 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をデフォルトの分離レベルのままにしておくのでなく、必要に応じた分離レベルに設定すると、CLI はステートメント・ハンドルに関連するリソースをより適切に活用することができます。これは、接続されている 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た上でのみ試行する必要があります。

  アプリケーションは、並行性を最大にするように、可能な限り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、CLI ドライバーは、そのハンドルの基礎となるデータ構造体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 (例えば、SQL_ATTR_PARAMSET_SIZE) を明示的にリセットする必要もあります。さもないと、そのステートメント・ハンドルを使用する以後のすべてのステートメントにより、それらの属性が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、結果セットに対して、アプリケーション変数または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、 ラージ・オブジェクト・データの最適化を参照)。データ値が以下のようなラージ可変長データのときは、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、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すし、また、DBMS カタログ表は非常に大きくなる可能性があります。全般に、カタログ関数を呼び出す回数を制限し、また戻される行数を制限することを試行してください。

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に対して、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に対して、表名またはパターン
- 詳細な列情報を戻すカタログ関数に対して、列名またはパターン

開発とテストのときは数百の表を使用するデータ・ソースを対象にしていたとしても、数千もの表を使用するデータベースに対して実行される可能性があることを忘れないでください。アプリケーションを開発する際には、この可能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 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。 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。

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。 CLI 関数は、必要な

箇所で HSTMT 引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 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、準備済みの動的 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

配列を用いて、パラメーターをバインドしたり、あるいは、検索したデータを取り出す方式は、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

LONG 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。

また、LOB 値をファイルに直接フェッチすることもできますし、 LOB パラメーター値をファイルから直接読み取ることもできます。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リソース使用量を節約することができます。

オブジェクト ID の大/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、そのオブジェクト ID が区切り文字で区切られて記述されていない場合、カタログ表には大文字で保管されます。区切り文字で区切られて記述された名前を用いてオブジェクト 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。

オブジェクト ID が SQL ステートメント内で参照されると、オブジェクト ID が区切られていなければ、*大文字小文字を区別しない* で処理されます。

例えば、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)
```

この例の 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 CLI カタログ関数の引数は、オブジェクトの名前を*大文字小文字の区別*があるものとして処理します。 (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 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を照会するのに、アプリケーション自体のダイアログ・ボックスを使用している場合、ユーザーが接続ストリングに追加の接続オプションを指定できる場合があります。また、このストリングは保管され、それ以降の接続では、デフォルト値として使用できます。

ステートメント・スキャンをオフにする

CLI はデフォルト設定で、ベンダー・エスケープ節シーケンスを見つけるために、各 SQL ステートメントをスキャンします。

アプリケーションがベンダー・エスケープ節シーケンスを含む 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、 CLI がベンダー・エスケープ節を見つけるためにスキャンを実行することがないようにします。

複数のロールバックにわたり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。 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。

例えば、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されてしまいます。アプリケーションは、同じデータベースに対する複数の接続を使用して、オープン・カーソルを行うステートメントを分離させることができます。カーソルが個別のトランザクション内にあるため、1 つのステートメントのロールバックが、他のステートメントのカーソルに影響を与えないためです。

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 (挿入など) 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 ロケーターに関して何も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。ある接続による更新により、その接続が (COMMIT または ROLLBACK によって) ロックを解放するまで、他の接続は容易にロック待機状態となり得ます。

コンパウンド SQL サブステートメントの準備

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で、サブステートメントを準備(PREPARE)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 (EXECUTE) 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 (UDT) 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・マーカーまたはその UDT のいずれかをキャストする必要があります。

例えば、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

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 (EXECUTE) 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。これは、非常に小さい応答セットを伴う複数の照会をアプリケーションが生成するような場合に最も効率的です。ネットワークを介して流れる要求と応答のためのリソース利用が処理時間のかなりの割合を占めるためです。 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、コスト削減になります。

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発生しません。

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のフィーチャーをオンにした場合、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_CHAINING_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。

制約事項: DB2 CLI はコンパウンド SQL (コンパイル済み) ステートメントまたはコンパウンド SQL (インライン) ステートメントでの配列入力チェーニングをサポートしていません。

第 12 章 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 ストリング引数の代わりに Unicode ストリング引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 CLI 構成キーワードによって、 1 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 リテラルと区別される、GRAPHIC/VARGRAPHIC および CHAR/VARCHAR リテラルが含まれます。 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。

Unicode 関数 (CLI)

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

- SQLBrowseConnect
- SQLColAttribute
- SQLColAttributes
- SQLColumnPrivileges
- SQLColumns
- SQLConnect
- SQLCreateDb
- SQLDataSources
- SQLDescribeCol
- SQLDriverConnect
- SQLDropDb
- SQLError
- SQLExecDirect
- SQLExtendedPrepare
- SQLExtendedProcedures
- SQLExtendedProcedureColumns
- SQLForeignKeys
- SQLGetConnectAttr
- SQLGetConnectOption
- SQLGetCursorName
- SQLGetDescField
- SQLGetDescRec
- SQLGetDiagField
- SQLGetDiagRec

- SQLGetInfo
- SQLGetPosition
- SQLGetStmtAttr
- SQLNativeSQL
- SQLPrepare
- SQLPrimaryKeys
- SQLProcedureColumns
- SQLProcedures
- SQLReloadConfig
- SQLSetConnectAttr
- SQLSetConnectOption
- SQLSetCursorName
- SQLSetDescField
- SQLSetStmtAttr
- SQLSpecialColumns
- SQLStatistics
- SQLTablePrivileges
- SQLTables

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

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

ANSI 関数から Unicode 関数へのマッピング

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

細は、ANSI バージョンの CLI Unicode 関数を参照してください。

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、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、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、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。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 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 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、例えば `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 15. 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	**Linux および UNIX**
Microsoft ODBC Driver Manager	UTF-16*	該当しません
unixODBC Driver Manager	UCS-2	UCS-2
DataDirect Connect for ODBC Driver Manager	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。		

第 13 章 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 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 IBM Power Systems™ データベース・サーバー

  注: ホストまたは IBM Power Systems 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。

ConnectType CLI/ODBC 構成キーワード

アプリケーションをリモート作業単位で実行するか、それとも分散作業単位で実行するかを制御します。

db2cli.ini キーワード構文:

ConnectType = **1** | **2**

デフォルト設定:

リモート作業単位。

同等の環境または接続属性:

SQL_ATTR_CONNECTTYPE

使用上の注意:

このオプションによって、デフォルトの接続タイプを指定できます。オプションは、次のとおりです。

- 1 = リモート作業単位。それぞれのコミット範囲がある、複数の同時接続。並行トランザクションは整合されていません。これはデフォルトです。
- 2 = 分散作業単位。複数のデータベースが同じ分散作業単位の下で参加する、整合接続。

最初の接続は、同じ環境ハンドルの下に割り振られた他のすべて接続の接続タイプを決定します。

このキーワードは、環境または接続属性よりも優先されます。

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

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。

- SQL_ATTR_CONNECTTYPE 環境属性を設定しなければなりません。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、複数の同時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_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。しかし、ODBC アプリケーションは `SQLSetEnvAttr()` にアクセ

スできないので、個々の接続ハンドルが割り振られてから確立されるまでの間に、 SQLSetConnectAttr() を使用してこの属性を設定しなければなりません。

環境ハンドル上のすべての接続の SQL_ATTR_CONNECTTYPE 設定は、同じでなければなりません。環境では、並行接続と整合接続を混合して使うことはできません。最初の接続のタイプが決まると、それ以降のすべての接続のタイプはそれに従います。 SQLSetEnvAttr() は、接続アクティブに接続タイプを変更しようとすると、エラーが返されます。

- SQL_ATTR_CONNECTTYPE が SQL_COORDINATED_TRANS に設定されている場合は、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ために、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。
- 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

192 ページの図 9 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。

193 ページの図 10 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。

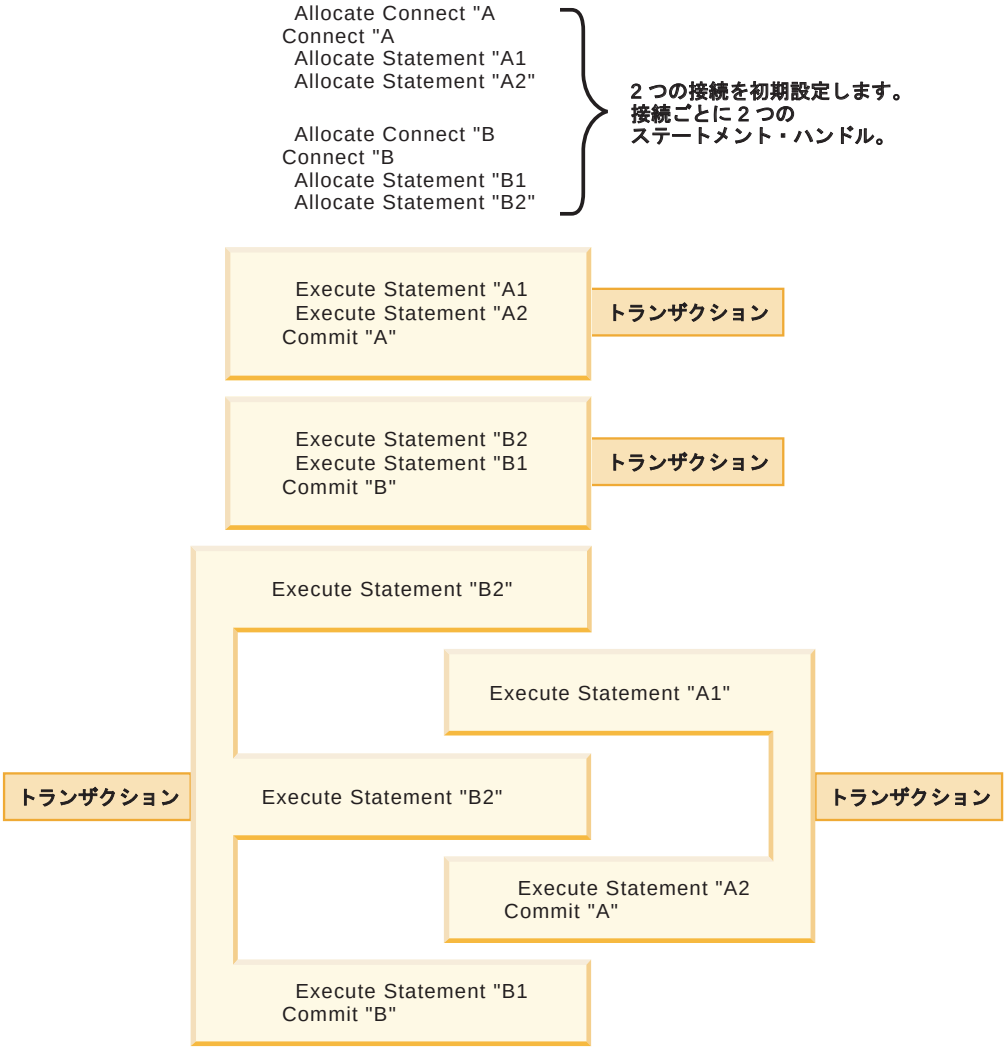


図 9. 並行トランザクションを使用した複数接続

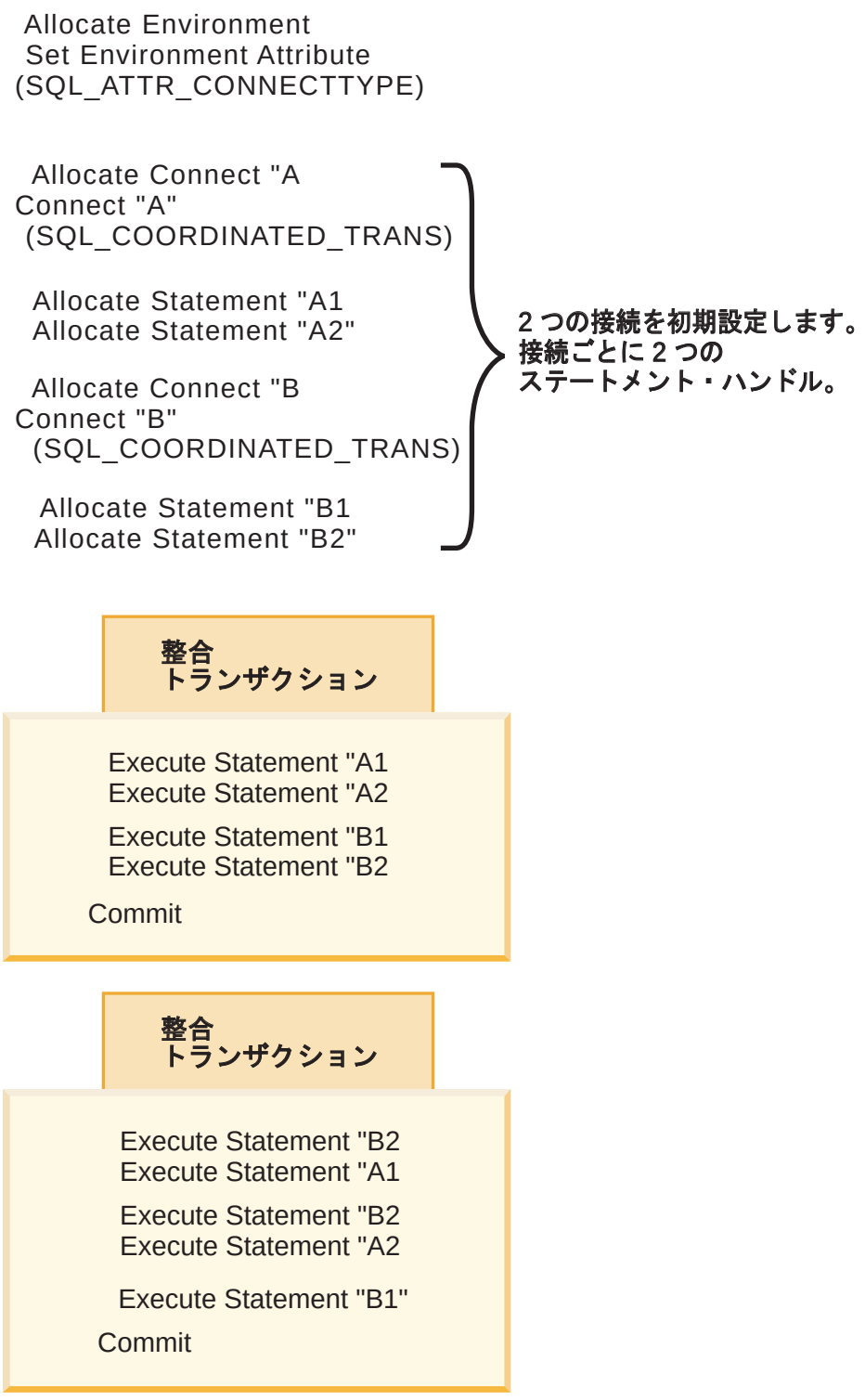


図 10. 整合トランザクションを使用した複数接続

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。

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® 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 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 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。結果として、CLI/ODBC ドライバーではトランザクションが終了していることを認識しなくなるため、アプリケーションは終了前に次のタスクを行う必要があります。
  - 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。

第 14 章 CLI 関数の非同期実行

CLI は CLI 関数のサブセットを非同期に実行できます。これらの関数については、関数の呼び出し後、その関数が実行を終える前に、CLI ドライバーは制御をアプリケーションに戻します。

非同期実行は、通常は要求をサーバーに送信した後に応答を待機する関数で使用できます。関数は呼び出されるたび、実行を完了するまでに SQL_STILL_EXECUTING を戻します。実行の完了時には異なる値 (例えば SQL_SUCCESS) を戻します。非同期に実行中の関数は、応答を待機する代わりに、制御をアプリケーションに戻します。その場合アプリケーションは、SQL_STILL_EXECUTING 以外の戻りコードが戻されるまで、他のタスクを実行し、関数をポーリングできます。非同期に実行できる関数のリストは、SQL_ATTR_ASYNC_ENABLE 接続またはステートメントの属性を参照してください。

アプリケーションが CLI 関数を非同期に実行するためには、アプリケーションには以下の関数呼び出しが組み込まれていなければなりません。

1. 非同期呼び出しがサポートされているかを確認するために SQL_ASYNC_MODE オプションを指定した、関数 `SQLGetInfo()` の呼び出し。
2. 非同期呼び出しがサポートされていることが確認された後、それを使用できるようにする SQL_ATTR_ASYNC_ENABLE 属性を指定した、`SQLSetConnectAttr()` または `SQLSetStmtAttr()` の呼び出し。
3. 非同期実行をサポートする関数の呼び出し、および非同期関数のポーリング。アプリケーションが非同期に実行できる関数を呼び出すと、次の 2 つの事柄のいずれかが生じることがあります。
   - 関数の非同期実行に利点がない場合、CLI はそれを同期的に実行して通常の戻りコード (SQL_STILL_EXECUTING 以外) を戻します。この場合、アプリケーションは非同期モードが使用できない場合のように実行します。
   - CLI は何らかの最小限の処理 (引数にエラーがないかの検査など) を実行した後、ステートメントをサーバーに渡します。この短い処理が完了すると、戻りコード SQL_STILL_EXECUTING がアプリケーションに戻されます。

非同期実行中に呼び出せる関数

関数が非同期に呼び出されると、元の関数が SQL_STILL_EXECUTING 以外のコードを戻すまでは、元の関数、`SQLAllocHandle()`、`SQLCancel()`、`SQLGetDiagField()`、または `SQLGetDiagRec()` だけが、*StatementHandle* に関連したステートメントまたは接続で呼び出せます。 *StatementHandle* または *StatementHandle* に関連した接続で他の関数を呼び出すと、SQL_ERROR が SQLSTATE HY010 (関数のシーケンス・エラーです。) を伴って戻されます。

関数の非同期実行中の診断情報

`SQLGetDiagField()` は、非同期関数実行を行うステートメント・ハンドルで呼び出されると、以下の値を戻します。

- SQL_DIAG_CURSOR_ROW_COUNT、SQL_DIAG_DYNAMIC_FUNCTION、SQL_DIAG_DYNAMIC_FUNCTION_CODE、および SQL_DIAG_ROW_COUNT ヘッダー・フィールドの値は、未定義です。
- SQL_DIAG_NUMBER ヘッダー・フィールドは 0 を戻します。
- SQL_DIAG_RETURN_CODE ヘッダー・フィールドは SQL_STILL_EXECUTING を戻します。
- すべてのレコード・フィールドは SQL_NO_DATA を戻します。

SQLGetDiagRec() は、非同期関数実行を行うステートメント・ハンドルで呼び出されると、常に SQL_NO_DATA を戻します。

非同期関数呼び出しの取り消し

アプリケーションは、SQLCancel() を呼び出すことによって、非同期に実行している関数の取り消し要求を発行できます。すでに実行を完了している関数は、取り消せません。

SQLCancel() 呼び出しからの戻りコードは、非同期関数の実行が停止したかどうかではなく、取り消し要求が受け取られたかどうかを示します。

関数が取り消されたかどうかを識別する唯一の方法は、元の引数を使用してそれをもう一度呼び出すことです。

- 取り消しが成功した場合、関数は SQL_ERROR および SQLSTATE HY008 (操作が取り消されました。) を戻します。
- 取り消しが成功しなかった場合、関数は SQL_ERROR、SQLSTATE HY008 以外の値を戻します。例えば、関数は SQL_STILL_EXECUTING を戻すかもしれません。

CLI アプリケーションで関数を非同期に実行する

CLI アプリケーションで関数を非同期に実行することは、CLI での、大きなプログラミング作業の一部です。非同期関数を使用できるようにし、それらの関数を処理するタスクとして、非同期実行がサポートされることの確認、非同期実行のためのアプリケーションの初期化、および非同期実行を利用するための関数の処理があります。

始める前に

非同期実行のための CLI アプリケーションのセットアップを開始する前に、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。

このタスクについて

注: バージョン 9.7 フィックスパック 4 以降、このフィーチャーは CLI のロード処理フィーチャーでも使用できるようになりました。

アプリケーションは、1 つの接続において、非同期モードで実行されるアクティブな関数を、最大 1 つ持つことができます。非同期モードが接続レベルで有効な場合、すでに割り振られているすべてのステートメントだけでなく、その接続で将来

割り振られるステートメント・ハンドルも、非同期実行に関して有効となります。

手順

1. SQLGetInfo() を、*InfoType* SQL_ASYNC_MODE を指定して呼び出し、その関数が非同期に呼び出せることを確認します。例えば、以下のようにします。

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

   SQLGetInfo() 関数の呼び出しは、次のいずれかの値を戻します。

   - SQL_AM_STATEMENT: 非同期実行を、ステートメント・レベルでオン/オフにできます。
   - SQL_AM_CONNECTION: 非同期実行を、接続レベルでオン/オフにできます。
   - SQL_AM_NONE: 非同期実行はサポートされません。ご使用のアプリケーションは、非同期実行のためにセットアップできません。これは、以下の 2 つの理由のいずれかのために戻されます。
     - データ・ソース自体が非同期実行をサポートしません。
     - CLI/ODBC の構成キーワード ASYNCENABLE が、特に非同期実行を無効にする設定になっていました。

2. SQLGetInfo() からの戻り値が SQL_AM_STATEMENT または SQL_AM_CONNECTION のいずれかである場合には、SQLSetStmtAttr() または SQLSetConnectAttr() を使用して SQL_ATTR_ASYNC_ENABLE 属性を設定し、アプリケーションの非同期実行を有効にします。

   - 戻り値が SQL_AM_STATEMENT である場合は、SQLSetStmtAttr() を使用して SQL_ATTR_ASYNC_ENABLE を SQL_ASYNC_ENABLE_ON に設定します。例えば、以下のようにします。

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

   - 戻り値が SQL_AM_CONNECTION の場合は、SQLSetConnectAttr() を使用して SQL_ATTR_ASYNC_ENABLE を SQL_ASYNC_ENABLE_ON に設定します。例えば、以下のようにします。

```
/* Set connection level asynchronous execution on */
rc = SQLSetConnectAttr( hstmt, /* Connection handle */
                        SQL_ATTR_ASYNC_ENABLE,
                        (SQLPOINTER) SQL_ASYNC_ENABLE_ON,
                        0);
```

3. 非同期実行をサポートする関数を呼び出し、非同期関数をポーリングします。非同期に実行できる関数のリストは、SQL_ATTR_ASYNC_ENABLE 接続またはステートメントの属性を参照してください。

   アプリケーションは、関数を初めて呼び出すのに使用したのと同じ引数を使用して、繰り返し呼び出すことによって、関数が完了したかどうかを判別します。戻りコード SQL_STILL_EXECUTING は、それがまだ完了していないことを示

し、他の値は完了したことを示します。 SQL_STILL_EXECUTING 以外の値は、同期的に実行された場合に戻されるのと同じ戻りコードです。

以下の例は、可能性のある両方の結果を考慮に入れた、一般的な while ループを例示しています。

```
while ( (rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS) ) == SQL_STILL_EXECUTING)
{
    /* Other processing can be performed here, between each call to
     * see if SQLExecDirect() has finished running asynchronously.
     * This section will never run if CLI runs the function
     * synchronously.
     */
}
/* The application continues at this point when SQLExecDirect() */
/* has finished running. */
```

第 15 章 マルチスレッド CLI アプリケーション

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、 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・セーフを保証しています。つまり、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、 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。例えば、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、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 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

CLI アプリケーション内に別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、(例えば、長期実行照会をキャンセルするために) `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 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、 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 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

CLI で割り振られるリソースは、スレッド・セーフ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 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。例えば、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・セーフティーは 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。

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・タスクの数を制限する場合には、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 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。 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 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・バッファーを共有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。例えば、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 つ以上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。

前記のアプリケーション・モデルでは、この問題が回避されます。接続上でスレッドが実行を一度開始すると、スレッド間でアプリケーション・リソースを共有しないからです。

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。

CLI 最初に 呼び出しを実行する場合

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 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。 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。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、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 CLI 関数 (例えば、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、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 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。

第 16 章 CLI アプリケーションでのベンダー・エスケープ節

X/Open SQL CAE 仕様では、**エスケープ節**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。 CLI と ODBC の両方とも、X/Open で定義されているベンダー・エスケープ節をサポートしています。

現在では、エスケープ節は、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。 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、エスケープ節を使うとアプリケーションの可搬性が高くなります。

CLI は、エスケープ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 (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*)--
```

アプリケーションは、これからは、現在の ODBC 標準に従って短縮構文だけを使用するようにしてください。

短縮されたエスケープ節の構文

エスケープ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。

例えば、SELECT * FROM EMPLOYEE WHERE HIREDATE={d '1994-03-29'} 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。

CLI は、選択ステートメントを 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
```

例えば、CLI が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 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一致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節は、次のとおりです。

```
{escape 'escape-character'}
```

escape-character は、SQL ESCAPE 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。 SELECT Name FROM Customers WHERE Growth LIKE '1_¥%'{escape '¥'} 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つ Name から、すべての値を選択することになります。

さまざまなベンダー DBMS 製品間の可搬性には関係のないアプリケーションの場合、 SQL ESCAPE 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ために、アプリケーションで SQL_LIKE_ESCAPE_CLAUSE 情報タイプを指定して SQLGetInfo() を呼び出せます。

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ン・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、オプション・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。 **?=** がエスケープ節にあると、 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。 CLI は、プロシージャー引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節を使用する必要があります。例えば、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。

例えば、CLI が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。

例えば、CLI が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。

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・エスケープ節を使用して定義されます。各関数は、エスケープ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。

- ストリング関数
- 数値関数
- 日時関数
- システム関数
- 変換関数

以下の節にでてくる表には、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・フォーマット。

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、CLI でサポートされ、 ODBC でベンダー・エスケープ節を使用して定義されます。

- スカラー関数に対する引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 16. ストリング・スカラー関数

ストリング・スカラー関数	説明	関数をサポートするサーバー
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 Database for Linux, UNIX, and Windows
CHAR(code)	*code* で指定された ASCII コード値がある文字を戻します。 *code* の値は、0 から 255 までででなければなりません。それ以外は、戻り値は NULL です。	DB2 Database for Linux, UNIX, and Windows
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, IBM DB2 for IBM i

表 16. ストリング・スカラー関数 (続き)

ストリング・スカラー関数	説明	関数をサポートするサーバー
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
LCASE(string_exp)	*string_exp* のすべての大文字を小文字に変換します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE
LEFT(string_exp,count)	*string_exp* の文字の左端の *count* を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Server for VM and VSE には後書きブランクが組み込まれています。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
LTRIM(string_exp)	先行ブランクを除いて *string_exp* の文字を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i

表 16. ストリング・スカラー関数 (続き)

ストリング・スカラー関数	説明	関数をサポートするサーバー
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 Database for Linux, UNIX, and Windows, DB2 for z/OS,
RIGHT(string_exp, count)	*string_exp* の文字の右端の count を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
UCASE(string_exp)	*string_exp* のすべての小文字を大文字に変換します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i

数値関数

この節に示す数値関数は、CLI でサポートされ、 ODBC でベンダー・エスケープ節を使用して定義されます。

- *numeric_exp* として示される引数は、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 17. 数値スカラー関数

数値スカラー関数	説明	関数をサポートするサーバー
ABS(numeric_exp)	*numeric_exp* の絶対値を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
CEILING(numeric_exp)	*numeric_exp* 以上の最短整数を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i

表 17. 数値スカラー関数 (続き)

数値スカラー関数	説明	関数をサポートするサーバー
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
DEGREES(numeric_exp)	*numeric_exp* ラジアンから変換された度数を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
EXP(double_exp)	*double_exp* の指数値を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
FLOOR(numeric_exp)	*numeric_exp* 以下の最大整数を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
LOG(double_exp)	*double_exp* の自然対数を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
LOG10(double_exp)	*double_exp* の 10 を底とする対数 (常用対数) を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
PI()	πの定数値を浮動小数点値として戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
POWER(numeric_exp, integer_exp)	値 *numeric_exp* の *integer_exp* 乗を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i

表 17. 数値スカラー関数 (続き)

数値スカラー関数	説明	関数をサポートするサーバー
RADIANS(numeric_exp)	*numeric_exp* 度から変換されたラジアン数を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
RAND([integer_exp])	*integer_exp* をオプションのシード値として使用してランダム浮動小数点値を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
SQRT(double_exp)	*double_exp* の平方根を戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i

日時関数

この節に示す日時関数は、CLI でサポートされ、 ODBC でベンダー・エスケープ節を使用して定義されます。

- *timestamp_exp* として示される引数は、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 18. 日時スカラー関数

日時スカラー関数	説明	関数をサポートするサーバー
CURDATE()	現在日付を日付値として戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
CURTIME()	地方時刻を時刻値として戻します。	DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i

表 18. 日時スカラー関数 (続き)

日時スカラー関数	説明	関数をサポートするサーバー
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 Database for Linux, UNIX, and Windows, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, IBM DB2 for IBM i

表 18. 日時スカラー関数 (続き)

日時スカラー関数	説明	関数をサポートするサーバー
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows
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、1/1000000000 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 Database for Linux, UNIX, and Windows

表 18. 日時スカラー関数 (続き)

日時スカラー関数	説明	関数をサポートするサーバー
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、1/1000000000 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 Database for Linux, UNIX, and Windows
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 Database for Linux, UNIX, and Windows, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i

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できる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可能になります。データベースがバージョン 7 以前で作成されている場合、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できるようにするには、**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、CLI でサポートされ、 ODBC でベンダー・エスケープ節を使用して定義されます。

- *exp* として示される引数は、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* として示される引数は、リテラル定数です。

表 19. システム・スカラー関数

システム・スカラー関数	説明	関数をサポートするサーバー
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
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 Database for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i

変換関数

変換関数は CLI によってサポートされており、ベンダー・エスケープ節を使用して 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 20. 変換関数

変換スカラー関数	**説明**	**関数をサポートするサーバー**
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 Database for Linux, UNIX, and Windows

第 17 章 IBM データ・サーバー上の高可用性のための非 Java クライアント・サポート

DB2 Database for Linux, UNIX, and Windows、DB2 for z/OS、または IBM Informix に接続するクライアント・アプリケーションは、これらのデータ・サーバーの高可用性フィーチャーを簡単に活用できます。

クライアント・アプリケーションでは、以下の高可用性フィーチャーを使用できます。

- 自動クライアント・リルート

  自動クライアント・リルート機能は、すべての IBM データ・サーバーで使用できます。自動クライアント・リルートでは、データ・サーバーから提供される情報を使用して、障害が起こったサーバーから代替サーバーへクライアント・アプリケーションをリダイレクトします。自動クライアント・リルートによって、アプリケーションは最小限の中断で処理を続行できます。代替サーバーへの処理のリダイレクトをフェイルオーバーと呼びます。

  DB2 for z/OS データ・サーバーへの接続では、自動クライアント・リルートはワークロード・バランシング・フィーチャーの一部となります。通常 DB2 for z/OS では、ワークロード・バランシングなしで自動クライアント・リルートを使用できません。

- クライアント・アフィニティー

  クライアント・アフィニティーは、クライアントによって完全に制御されるフェイルオーバー・ソリューションです。これは、特定の 1 次サーバーへ接続する必要がある状態を対象としています。 1 次サーバーへの接続中に障害が発生した場合、クライアント・アフィニティーを使用して、代替サーバーへのフェイルオーバーの特定の順序を実行します。

  データ共有グループのすべてのメンバーは並行してデータにアクセスできるため、クライアント・アフィニティーは DB2 for z/OS データ共有環境には適用されません。データ共有は、DB2 for z/OS に対する高可用性の推奨ソリューションです。

- ワークロード・バランシング

  ワークロード・バランシングは、すべての IBM データ・サーバーで使用できます。ワークロード・バランシングによって、IBM Informix 高可用性クラスター、DB2 for z/OS データ共有グループ、または DB2 Database for Linux, UNIX, and Windows DB2 pureScale® インスタンス内のサーバー間で処理が効率的に分散されます。

以下の表は、これらのフィーチャーに関するサーバー・サイド情報へのリンクの一覧です。

表 21. 高可用性に関するサーバー・サイド情報

データ・サーバー	関連したトピック
DB2 Database for Linux, UNIX, and Windows	• DB2 pureScale: DB2 pureScale Feature ロードマップ資料 • 自動クライアント・リルート: 自動クライアント・リルートのロードマップ
IBM Informix	接続マネージャーによるクラスター接続の管理
DB2 for z/OS	データ共有グループとの通信

重要: DB2 for z/OS への接続に関して、この情報では DB2 for z/OS への直接接続について扱われます。 DB2 Connect サーバーを介する接続の高可用性に関する情報は、DB2 Connect の資料を参照してください。

DB2 Database for Linux, UNIX, and Windows への接続の高可用性のための非 Java クライアント・サポート

DB2 Database for Linux, UNIX, and Windows サーバーは、ワークロード・バランシングと自動クライアント・リルートにより、クライアント・アプリケーションに高可用性を提供します。このサポートは、Java クライアント (JDBC、SQLJ、または pureQuery) だけではなく、非 Java クライアント (ODBC、CLI、.NET、OLE DB、PHP、Ruby、または組み込み SQL) を使用するアプリケーションでも使用できます。

非 Java クライアントでは、高可用性サポートを利用するために、以下のクライアントまたはクライアント・パッケージの 1 つを使用する必要があります。

- IBM Data Server Client
- IBM Data Server Runtime Client
- IBM Data Server Driver Package
- IBM Data Server Driver for ODBC and CLI

DB2 Database for Linux, UNIX, and Windows サーバーへの接続に対する高可用性サポートには、以下のものが含まれます。

自動クライアント・リルート

このサポートによって、クライアントが代替サーバーを介してデータベースへ再接続しようとして失敗してもリカバリーすることができます。別のサーバーへ再接続することをフェイルオーバーと呼びます。自動クライアント・リルート用のクライアント・サポートは、 DB2 Database for Linux, UNIX, and Windows に接続する非 Java クライアントの場合、デフォルトで使用可能です。

サーバーにおいては、以下のいずれかの方法で自動クライアント・リルート機能を提供することができます。

- DB2 pureScale インスタンスで、複数のサーバーが構成されている場合。データベースへ接続すると、DB2 pureScale インスタンス内の 1 メンバーに接続することになります。フェイルオーバー時には、DB2 pureScale インスタンス内の別のメンバーに再接続します。この環境では、クライアントは TCP/IP を使用して DB2 pureScale インスタンスに接続する必要があります。

- データベースに対して定義されている、DB2 pureScale インスタンスおよび代替サーバーの場合。フェイルオーバー時には、まず DB2 pureScale インスタンス内の別のメンバーに再接続します。サーバーへのフェイルオーバーが試行されるのは、 DB2 pureScale インスタンス内の全メンバーが使用不可である場合に限ります。
- 1 次サーバーに対して 1 つの DB2 pureScale インスタンスが定義されており、サーバーに対して別の DB2 pureScale インスタンスが定義されている場合。フェイルオーバー時には、まず 1 次 DB2 pureScale インスタンス内の別のメンバーに再接続します。代替 DB2 pureScale インスタンスへのフェイルオーバーが試行されるのは、 1 次 DB2 pureScale インスタンス内の全メンバーが使用不可である場合に限ります。
- データベースが単一サーバー上で定義されている場合。このデータベースの構成に、代替サーバーの指定を含めます。フェイルオーバー時には、代替サーバーに再接続します。

代替グループは、現在のグループへの接続を再確立できなかった場合の、自動クライアント・リルートに関する追加のフェイルオーバー・メカニズムです。*グループ* は、DB2 インスタンスで作成されるデータベースです。DB2 pureScale またはパーティション・データベース環境では、データベースに対して参加しているすべてのデータベース・サーバーも、グループと見なされます。アプリケーションが明示的に接続しているデータベースは、*1 次グループ* と呼ばれます。

CLI または .NET クライアント・アプリケーションの場合、自動クライアント・リルートのフェイルオーバーは、*シームレス* にすることも、*非シームレス* にすることもできます。非シームレスなフェイルオーバーでは、クライアント・アプリケーションが別のサーバーへ再接続するときに、フェイルオーバー (代替サーバーへの接続) が発生したことを示すエラーが、毎回アプリケーションに戻されます。シームレス・フェイルオーバーでは、トランザクションの最初の SQL ステートメントの実行中に、接続障害が発生し、代替サーバーまたは代替グループへの再接続が完了しても、クライアントはエラーを戻しません。

DB2 pureScale インスタンスでは、自動クライアント・リルート・サポートを、ワークロード・バランシングなしで使用することもワークロード・バランシングと共に使用することもできます。

ワークロード・バランシング

ワークロード・バランシングによって、DB2 pureScale インスタンスの可用性を高めることができます。

ワークロード・バランシングを使用すると、DB2 pureScale インスタンスは、メンバー間で効率的に処理を分散させることができます。

非 Java クライアントは任意のオペレーティング・システム上でワークロード・バランシングをサポートします。クライアントから DB2 pureScale インスタンスへの接続には、TCP/IP を使用する必要があります。

ワークロード・バランシングが使用可能になると、クライアントはサーバー・リストを使用して、DB2 pureScale インスタンスのメンバーに関する状況情報を頻繁に取得します。クライアントはサーバー・リストをキャッシュに入れ、キャッシュ内のその情報を使用して、次のトランザクションを送付すべきメンバーを決定します。

非 Java クライアントの場合、サーバー・リストはアプリケーション・プロセス内にキャッシュされます。そのプロセス内にある各接続の間でのみ、ワークロード・バランシングのために共有されます。

DB2 Database for Linux, UNIX, and Windows は 2 つのタイプのワークロード・バランシングをサポートします。

接続レベルのワークロード・バランシング

接続レベルのワークロード・バランシングは、接続境界で実行されます。非 Java クライアントの場合のみサポートされます。接続レベルのワークロード・バランシングのクライアント・サポートは、 DB2 Database for Linux, UNIX, and Windows に接続する非 Java クライアントの場合、デフォルトで使用可能です。

接続レベルのロード・バランシングは、接続期間が短い場合に非常に有効です。

トランザクション・レベルのワークロード・バランシング

トランザクション・レベルのワークロード・バランシングは、トランザクション境界で実行されます。トランザクション・レベルのワークロード・バランシングのクライアント・サポートは、 DB2 Database for Linux, UNIX, and Windows に接続するクライアントの場合、デフォルトでは使用不可に設定されています。

トランザクション・レベルのロード・バランシングは、接続期間が長い場合に非常に有効です。

クライアント・アフィニティー

クライアント・アフィニティーは、クライアントによって完全に制御される自動クライアント・リルートのソリューションです。これは、特定の 1 次サーバーへ接続する必要がある状態を対象としています。1 次サーバーへの接続中に障害が発生した場合、クライアント・アフィニティーを使用して、代替サーバーへのフェイルオーバーの特定の順序を実行します。

非 Java クライアント用の DB2 Database for Linux, UNIX, and Windows 自動クライアント・リルート・サポートの構成

DB2 Database for Linux, UNIX, and Windows データベースへの接続の場合、非 Java クライアント上の自動クライアント・リルート・サポートの構成のプロセスは、DB2 pureScale 環境の内外にかかわらず同一です。

非 Java クライアントの場合、自動クライアント・リルート機能はデフォルトで使用可能になります。DB2 pureScale 環境以外の環境では 1 次サーバーに接続する必要があります。DB2 pureScale 環境では、DB2 pureScale インスタンスに接続する必要があります。

サーバーとの接続が最初に正常に確立された時点で、クライアントは、すべての利用可能な代替サーバーのリストをサーバーから取得します。クライアントはこのリストをメモリーに格納します。最初の接続に失敗すると、クライアントは、`db2dsdriver.cfg` ファイル内の `<acr>` セクションの `<alternateserverlist>` タグの下で定義されている代替サーバーのリストを探します。

db2dsdriver.cfg ファイル内の <acr> セクションで代替サーバーが定義されていない場合、サーバーへの最初の正常な接続のときに、クライアントはローカル・キャッシュ・ファイル srvrlst.xml を作成します。クライアントは、利用可能な代替サーバーを記載したサーバーのリストでこのファイルを更新します。新しい接続が行われてサーバーのリストがクライアントの srvrlst.xml ファイルの内容と異なるときには常に、このファイルはリフレッシュされます。

クライアントが srvrlst.xml ファイルを使用して代替サーバーを探すときに、レコードを **db2diag** ログ・ファイルに書き込みます。このログをモニターして、初期サーバー接続が失敗する頻度を判別できます。

表 1 では、非 Java アプリケーションの接続を確立するための基本設定について説明しています。

表 22. 非 Java アプリケーションで、 DB2 Database for Linux, UNIX, and Windows データベースへの接続を確立するための基本設定

クライアントの設定	DB2 pureScale 環境外の値	DB2 pureScale 環境の場合の値
接続アドレス:		
データベース・ホスト[1]	1 次サーバーの IP アドレス。	DB2 pureScale インスタンスのメンバーの IP アドレス[2]
データベース・ポート[1]	1 次サーバーの SQL ポート番号。	DB2 pureScale インスタンスのメンバーの SQL ポート番号[2]
データベース名[1]	データベース名。	データベース名。

表 22. 非 Java アプリケーションで、 DB2 Database for Linux, UNIX, and Windows データベースへの接続を確立するための基本設定 (続き)

クライアントの設定	DB2 pureScale 環境外の値	DB2 pureScale 環境の場合の値

注:

1. 使用するクライアントに応じて、接続情報は以下のいずれかのソース内で定義されます。
   - いずれかのデータ・サーバー・ドライバー、または IBM Data Server Client か IBM Data Server Runtime Client を使用する CLI アプリケーションかオープン・ソース・アプリケーションを使用している場合、以下のようになります。
     - ホスト、ポート、およびデータベース情報がアプリケーション内の接続ストリング内で提供される場合、DB2 データベース・システムはその情報を使用します。
     - ホスト、ポート、およびデータベース情報がアプリケーション内の接続ストリング内で提供されない場合、ドライバーは db2cli.ini ファイルを使用し、この情報が db2cli.ini ファイル内で提供されている場合、DB2 データベース・システムはその情報を使用します。
     - ホスト、ポート、およびデータベース情報がアプリケーション内の接続ストリングおよび db2cli.ini ファイルのどちらでも提供されない場合、DB2 データベース・システムは db2dsdriver 構成ファイル内の情報を使用します。
   - .NET アプリケーション、または組み込み SQL を IBM Data Server Client または IBM Data Server Runtime Client と共に使用するアプリケーションを使用している場合、接続情報は db2dsdriver 構成ファイル以外のソースから提供されます。データベース・カタログ、接続ストリング、db2cli.ini ファイル、または .NET オブジェクト・プロパティーなどがソースになる可能性があります。
2. また、 Websphere Application Server Network Deployment またはマルチホーム DNS などのディストリビューターを使用して、データベースへの最初の接続を確立することもできます。
   - ディストリビューターを使用する場合は、ディストリビューターの IP アドレスおよびポート番号を指定します。ディストリビューターは、現行のワークロード分布を分析し、その情報を使用して DB2 pureScale インスタンスのいずれかのメンバーに接続要求を転送します。
   - マルチホーム DNS を使用する場合は、 DB2 pureScale インスタンスのいずれかのメンバーの IP アドレスおよびポート番号に解決可能な、IP アドレスおよびポート番号を指定します。マルチホーム DNS 処理では、単純なラウンドロビン選択方式やメンバーのワークロード分散などの基準に基づいてメンバーが選択されます。

db2dsdriver.cfg ファイル内の構成キーワードまたはレジストリー変数を設定して、自動クライアント・リルートの動作を詳細化できます。表 2 の構成キーワードを使用して、自動クライアント・リルートを制御できます。クライアント・アフィニティーが使用可能でない場合について、キーワードが説明されています。

db2dsdriver.cfg ファイルを変更すると、CLI アプリケーションは、SQLReloadConfig 関数を開始して、<acr> セクション内のすべての代替サーバーの項目を妥当性検査できるようになります。

表 23. 自動クライアント・リルートの動作を制御するための設定

db2dsdriver 構成ファイルの <acr> セクション内の要素	値
enableAcr パラメーター	自動クライアント・リルートが有効かどうかを指定します。デフォルトは `true` です。
enableSeamlessAcr パラメーター	シームレス・フェイルオーバーが可能かどうかを指定します。enableAcr が `true` に設定されている場合、 enableSeamlessAcr のデフォルトは `true` です。enableSeamlessACR は、グループまたはクラスター内のメンバーにのみ適用されます。
enableAlternateGroupSeamlessACR パラメーター	グループ間のシームレス・フェイルオーバーまたは非シームレス・フェイルオーバーの動作を指定します。デフォルトは `false` です。このパラメーターは、<acr> セクションの <alternategroup> 要素内で定義する必要があります。このパラメーターを `true` に設定するには、 enableSeamlessACR を `true` に設定する必要もあります。このパラメーターを true に設定しても、enableSeamlessACR の設定に影響はありません。 *alternategroup* セクションのサーバーへの接続が正常に確立されている場合、シームレスまたは非シームレスの動作の規則が適用されたままになります。
acrRetryInterval パラメーター	連続して行われる接続再試行の間に待機する秒数。レジストリー変数 **DB2_CONNRETRIES_INTERVAL** はこの値をオーバーライドします。有効な範囲は 0 から MAX_INT までです。**DB2_CONNRETRIES_INTERVAL** が設定されていない場合、デフォルトは待機なし (0) です。
maxAcrRetries パラメーター	自動クライアント・リルートのための接続再試行の最大数。レジストリー変数 **DB2_MAX_CLIENT_CONNRETRIES** はこの値をオーバーライドします。**DB2_MAX_CLIENT_CONNRETRIES** が設定されていない場合、デフォルトでは接続の再試行が 10 分間行われます。値 0 は、再接続の試行が 1 回行われることを意味します。

表 23. 自動クライアント・リルートの動作を制御するための設定 (続き)

db2dsdriver 構成ファイルの <acr> セクション内の要素	値
enableAlternateServerListFirstConnect パラメーター	データ・サーバーに対する最初の接続が失敗した場合のみ使用される代替サーバー・リストがあるかどうかを指定します。デフォルトは false です。enableAlternateServerListFirstConnect の値が true の場合、db2dsdriver 構成ファイル内で指定された自動クライアント・リルートに関する他の設定にかかわらず、自動クライアント・リルートとシームレス・フェイルオーバーは暗黙に使用可能になります。このフィーチャーを使用するには、db2dsdriver 構成ファイル内に <alternateserverlist> 要素も必要です。
<alternateserverlist> 要素	データベースへの最初の接続が失敗した場合に接続を試行する代替サーバーを識別するサーバー名とポート番号のセットを指定します。最初の接続後、代替サーバー・リストは使用されません。DB2 pureScale環境では、リストの項目になるのは DB2 pureScaleインスタンスのメンバーです。DB2 pureScale 環境以外の環境では、1 次サーバーに対して 1 つの項目と高可用性災害時リカバリー (HADR) スタンバイ・サーバーに対して 1 つの項目があります。最初の接続後、代替サーバー・リストは使用されません。

表 3 のレジストリー変数は、自動クライアント・リルートの再試行動作を制御します。

表 24. 自動クライアント・リルートの再試行動作を制御するレジストリー変数

レジストリー変数	値
DB2_MAX_CLIENT_CONNRETRIES	自動クライアント・リルートのための接続再試行の最大数。**DB2_CONNRETRIES_INTERVAL** が設定されている場合、デフォルトは 30 です。
DB2_CONNRETRIES_INTERVAL	連続して行われる接続再試行の間の秒数。**DB2_MAX_CLIENT_CONNRETRIES** が設定されている場合、デフォルトは 10 です。

どちらのレジストリー変数も設定せず、**maxAcrRetries** と **acrRetryInterval** も設定しないと、自動クライアント・リルート処理は、最大 10 分間データベースへの接続を再試行し、再試行の間に待機しません。

CLI、OLE DB、および ADO.NET アプリケーションには、接続タイムアウト値を設定できます。この値は、データベースへの接続が確立されるのをクライアント・アプリケーションが待機する秒数を指定します。接続タイムアウト値を、サーバー

への接続に要する最大時間以上の値に設定する必要があります。そうしないと、接続がタイムアウトになり、クライアント・リルートによって代替サーバーに転送される可能性があります。例えば、通常の日にサーバーに接続するために約 10 秒かかり、回線が混雑している日に約 20 秒かかる場合、接続タイムアウト値を 20 秒以上に設定する必要があります。

非 Java クライアントで DB2 Database for Linux, UNIX, and Windows 自動クライアント・リルート・サポートを使用可能にする例

db2dsdriver.cfg 構成ファイル内でいくつかのキーワードの値を設定することによって、DB2 Database for Linux, UNIX, and Windows 自動クライアント・リルート (acr) サポートのために非 Java クライアント・セットアップを微調整することができます。

db2dsdriver.cfg 構成ファイルに代替サーバーのリストを定義していない場合、サーバーへ初めて正常に接続したときに、クライアントはサーバーから、すべての利用可能な代替サーバーのリストを取得します。クライアントはメモリーにリストを格納し、代替サーバーを記載したサーバーのリストが含まれるローカル・キャッシュ・ファイルの srvrlst.xml も作成します。新しい接続が行われてサーバーのリストがクライアントの srvrlst.xml ファイルの内容と異なるときには常に、このファイルはリフレッシュされます。

サンプルのデータベースは、サーバー db2luwa (ポート 446) およびサーバー db2luwb (ポート 446) の 2 つのメンバーを持つ DB2 pureScale インスタンスであると想定します。このデータベースには、代替サーバー db2luwc (ポート 446) が定義されています。

以下の項目を変更して、デフォルトの自動クライアント・リルート・サポートを適切に調整することもできます。

自動クライアント・リルートの特性	db2dsdriver.cfg 構成キーワード	望ましい値
代替サーバーへの接続を再試行する回数	maxAcrRetries	10
再試行の間に待機する秒数	acrRetryInterval	5
データベースへの最初の接続が失敗した場合に別のサーバーを試行するかどうか	enableAlternateServerListFirstConnect	true

自動クライアント・リルートの特性	db2dsdriver.cfg 構成キーワード	望ましい値
データベースへの最初の接続が失敗した場合に試行するサーバーのホスト名とポート番号	<alternateserverlist> db2dsdriver.cfg 構成ファイルに代替サーバーのリストを定義していない場合、サーバーへ初めて正常に接続したときに、クライアントはサーバーから、すべての利用可能な代替サーバーのリストを取得します。クライアントはメモリーにリストを格納し、代替サーバーを記載したサーバーのリストが含まれるローカル・キャッシュ・ファイルの srvrlst.xml も作成します。新しい接続が行われてサーバーのリストがクライアントの srvrlst.xml ファイルの内容と異なるときには常に、このファイルはリフレッシュされます。 srvrlst.xml ファイルは、cfgcache ディレクトリー内の CLIENT_CONFIG_DIR にあります。	ホスト名およびポート番号: • db2luwa.luw.ibm.com、446 • db2luwb.luw.ibm.com、446 • db2luwc.luw.ibm.com、446

これらの変更を自動リルートの動作に実装するには、以下の db2dsdriver.cfg 構成ファイルを使用します。

```
<configuration>
  <dsncollection>
    <dsn alias="sample" name="sample" host="db2luw.luw.ibm.com" port="446">
    </dsn>
  </dsncollection>
  <databases>
    <database name="sample" host="db2luw.luw.ibm.com" port="446">
      <acr>
        <parameter name="enableAcr" value="true">
        </parameter>
        <parameter name=
        <parameter name="maxAcrRetries" value="10">
        </parameter>
        <parameter name="acrRetryInterval" value="5">
        </parameter>
        <parameter name="enableAlternateServerListFirstConnect"
          value="true"></parameter>
        <alternateserverlist>
           <server name="server1" hostname="db2luwa.luw.ibm.com" port="446">
           </server>
           <server name="server2" hostname="db2luwb.luw.ibm.com" port="446">
           </server>
           <server name="server3" hostname="db2luwc.luw.ibm.com" port="446">
           </server>
        </alternateserverlist>
      </acr>
    </database>
  </databases>
</configuration>
```

非 Java クライアント用の DB2 Database for Linux, UNIX, and Windows ワークロード・バランシング・サポートの構成

DB2 pureScale インスタンス内の DB2 Database for Linux, UNIX, and Windows データ・サーバーへの接続については、非 Java クライアントの場合、接続レベルのワ

ークロード・バランシングがデフォルトで使用可能になっています。トランザクション・レベルのワークロード・バランシング機能は、明示的に使用可能に設定する必要があります。

以下の表では、非 Java アプリケーションに接続レベルのワークロード・バランシング・サポートを使用可能にするための基本設定について説明しています。

表 25. 非 Java アプリケーションに DB2 Database for Linux, UNIX, and Windows 接続レベルのワークロード・バランシング・サポートを使用可能にするための基本設定

クライアントの設定	値
接続アドレス:	
データベース・ホスト[1]	DB2 pureScale インスタンスのメンバーの IP アドレス[2]
データベース・ポート[1]	DB2 pureScale インスタンスのメンバーの SQL ポート番号 [2]
データベース名[1]	データベース名

注:

1. 使用するクライアントに応じて、接続情報は以下のいずれかのソース内で定義されます。
   - いずれかのデータ・サーバー・ドライバー、あるいは IBM Data Server Client または IBM Data Server Runtime Client を使用する CLI アプリケーションまたはオープン・ソース・アプリケーションを使用している場合、以下のようになります。
     - ホスト、ポート、およびデータベース情報がアプリケーション内の接続ストリング内で提供される場合、DB2 はその情報を使用します。
     - ホスト、ポート、およびデータベース情報がアプリケーション内の接続ストリング内で提供されない場合、ドライバーは `db2cli.ini` ファイルを使用し、この情報が `db2cli.ini` ファイル内で提供されている場合、DB2 はその情報を使用します。
     - ホスト、ポート、およびデータベース情報がアプリケーション内の接続ストリングおよび `db2cli.ini` ファイルのどちらでも提供されない場合、DB2 は `db2dsdriver.cfg` 構成ファイル内の情報を使用します。
   - .NET アプリケーション、または組み込み SQL を IBM Data Server Client または IBM Data Server Runtime Client と共に使用するアプリケーションを使用している場合、接続情報は `db2dsdriver.cfg` 構成ファイル以外のソースから提供されます。データベース・カタログ、接続ストリング、`db2cli.ini` ファイル、または .NET オブジェクト・プロパティーなどがソースになる可能性があります。
2. また、 WebSphere® Application Server Network Deployment またはマルチホーム DNS などのディストリビューターを使用して、データベースへの最初の接続を確立することもできます。
   - ディストリビューターを使用する場合は、ディストリビューターの IP アドレスおよびポート番号を指定します。ディストリビューターは、現行のワークロード分布を分析し、その情報を使用して DB2 pureScale インスタンスのいずれかのメンバーに接続要求を転送します。
   - マルチホーム DNS を使用する場合は、 DB2 pureScale インスタンスのいずれかのメンバーの IP アドレスおよびポート番号に解決可能な、IP アドレスおよびポート番号を指定します。マルチホーム DNS の処理では、単純なラウンドロビン選択、またはメンバーのワークロード分散などの基準に基づいてメンバーが選択されます。

`db2dsdriver.cfg` ファイル内の以下の構成キーワードを使用して、接続レベルのワークロード・バランシング設定を変更できます。

表 26. 接続レベルのワークロード・バランシングの動作を制御するための設定

db2dsdriver.cfg 構成ファイル内の要素	セクション	値
connectionLevelLoadBalancing パラメーター	<database>	接続レベルのロード・バランシングを有効にするかどうかを指定します。これはデフォルトで true です。

db2dsdriver.cfg ファイル内の以下の構成キーワードを使用して、トランザクション・レベルのワークロード・バランシングを使用可能にし、微調整することができます。

表 27. トランザクション・レベルのワークロード・バランシングの動作を制御するための設定

db2dsdriver.cfg 構成ファイル内の要素	セクション	値
connectionLevelLoadBalancing パラメーター	<database>	トランザクション・レベルのワークロード・バランシングを使用する場合は、true に設定する必要があります。
enableWLB パラメーター	<wlb>	トランザクション・レベルのワークロード・バランシングを有効にするかどうかを指定します。これはデフォルトで false です。
maxTransportIdleTime	<wlb>	アイドル・トランスポートがドロップされる前の最大経過時間を秒数で指定します。デフォルトは 600 です。サポートされる最小値は 0 です。
maxTransportWaitTime	<wlb>	トランスポートが使用可能になるのをクライアントが待機する秒数を指定します。デフォルトは、 -1 (制限なし) です。サポートされる最小値は 0 です。
maxTransports	<wlb>	DB2 pureScale インスタンスに接続する各アプリケーション・プロセスが確立できる物理接続の最大数を指定します。
maxRefreshInterval	<wlb>	サーバー・リストがリフレッシュされる前の最大経過時間を秒数で指定します。デフォルトは 10 です。サポートされる最小値は 0 です。

非 Java クライアントで DB2 Database for Linux, UNIX, and Windows ワークロード・バランシング・サポートを使用可能にする例

DB2 Database for Linux, UNIX, and Windows ワークロード・バランシングを使用するには、DB2 pureScale 環境が必要です。 CLI、.NET、または組み込み SQL アプリケーションで DB2 Database for Linux, UNIX, and Windows ワークロード・バランシング・サポートを使用する前に、 db2dsdriver.cfg 構成ファイルを適切に設定して更新し、 DB2 pureScale 環境のメンバーに接続する必要があります。

以下の例は、 DB2 Database for Linux, UNIX, and Windows ワークロード・バランシング・サポートを活用するように CLI クライアントをセットアップする方法を示しています。

クライアントをセットアップするには、その前にDB2 pureScale インスタンスを構成する必要があります。

以下は、クライアントをセットアップする手順です。

1. トランザクション・レベルのワークロード・バランシングを使用可能にする db2dsdriver.cfg ファイルを作成します。この例では、以下のように想定します。
   - データベースへの初期接続が失敗した場合には、代替サーバーに対して接続を試行する必要がある。

     db2dsdriver.cfg 構成ファイルに代替サーバーのリストを定義していない場合、サーバーへ初めて正常に接続したときに、クライアントはサーバーから、すべての利用可能な代替サーバーのリストを取得します。クライアントはメモリーにリストを格納し、代替サーバーを記載したサーバーのリストが含まれるローカル・キャッシュ・ファイルの srvrlst.xml も作成します。新しい接続が行われてサーバーのリストがクライアントの srvrlst.xml ファイルの内容と異なるときには常に、このファイルはリフレッシュされます。
   - このデータベース用のトランザクション・レベルのワークロード・バランシングでは、物理接続の最大数を 80 にする必要がある。
   - トランザクション・レベルのワークロード・バランシング用の他のすべてのパラメーターについては、デフォルトを使用して接続しても構わない。

   この場合 db2dsdriver.cfg ファイルは以下のようになります。

```
<configuration>
  <dsncollection>
    <dsn alias="LUWDS1" name="LUWDS1" host="luw1ds.toronto.ibm.com"
      port="50000">
    </dsn>
  </dsncollection>
  <databases>
    <!-- In this example, the host and port represent a member of a
         DB2 pureScale instance -->
    <database name="LUWDS1" host="luw.ds1.ibm.com" port="50000">
       <!-- database-specific parameters -->
       <wlb>
          <!-- Enable transaction-level workload balancing -->
          <parameter name="enableWLB" value="true" />
          <!-- maxTransports represents the maximum number of physical
               connections -->
          <parameter name="maxTransports" value="80" />
       </wlb>
       <acr>
         <!-- acr is already enabled by default -->
         <!-- Enable server list for application first connect -->
         <parameter name="enableAlternateServerListFirstConnect"
           value="true" />
         <alternateserverlist>
           <!-- Alternate server 1 -->
           <parameter name="server" value="luw2ds.toronto.ibm.com" />
           <parameter name="port" value="50001" />
           <!-- Alternate server 2 -->
           <parameter name="server" value="luw3ds.toronto.ibm.com" />
```

```
          <parameter name="port" value="50002" />
          <!-- Alternate server 3 -->
          <parameter name="server" value="luw4ds.toronto.ibm.com" />
          <parameter name="port" value="50003" />
        </alternateserverlist>
      </acr>
    </database>
  </databases>
</configuration>
```

2. データベース名 LUWDS1 は DB2 pureScale インスタンスを表すものとします。 CLI アプリケーションで、以下のようなコードを使用して DB2 pureScale インスタンスに接続します。

```
...
   SQLHDBC         hDbc    = SQL_NULL_HDBC;
   SQLRETURN       rc      = SQL_SUCCESS;
   SQLINTEGER      RETCODE = 0;
   char            *ConnStrIn =
                   "DSN=LUWDS1;PWD=mypass";
                /* dsn matches the database name in the configuration file */
   char            ConnStrOut [200];
   SQLSMALLINT     cbConnStrOut;
   int             i;
   char            *token;
...
  /*************************************************************/
  /* Invoke SQLDriverConnect                                   */
  /*************************************************************/
   RETCODE = SQLDriverConnect (hDbc                ,
                               NULL                ,
                               (SQLCHAR *)ConnStrIn ,
                               strlen(ConnStrIn)    ,
                               (SQLCHAR *)ConnStrOut,
                               sizeof(ConnStrOut)   ,
                               &cbConnStrOut        ,
                               SQL_DRIVER_NOPROMPT);
...
```

非 Java クライアントから DB2 Database for Linux, UNIX, and Windows への接続のための自動クライアント・リルートの操作

自動クライアント・リルートは、IBM データ・サーバー・クライアントが DB2 Database for Linux, UNIX, and Windows データベースの 1 次サーバーへの接続を失った場合の、フェイルオーバー・サポートを提供します。自動クライアント・リルートにより、クライアントは、代替サーバーを介してデータベースへの再接続を試行することにより、障害からリカバリーできます。

データベース接続に対して自動クライアント・リルートが使用可能な場合、クライアントが既存の接続で接続障害を検出すると、以下のプロセスが通常発生します。

1. クライアントが既存の接続を使用して SQL ステートメントの実行を試行し、障害を検出します。
2. クライアントは最後に接続が成功した後に戻されるサーバー・リストを使用してアクセス対象のサーバーを識別し、データベースへの再接続を試行します。

   DB2 pureScale環境でない環境の場合、サーバー・リストには 2 つのエントリー (1 次サーバーのエントリーと代替サーバーのエントリー) が含まれます。

DB2 pureScale 環境の場合、サーバー・リストには DB2 pureScale インスタンスの全メンバーのエントリーが含まれています。さらに、代替サーバーがデータベースに定義されている場合は、代替サーバーのエントリーもサーバー・リストに含まれています。 DB2 pureScaleインスタンスのメンバーのエントリーには、処理容量に関する情報があります。クライアント側で、接続レベルのワークロード・バランシングが使用可能になっている場合、クライアントは、その情報を使用して、最も使用率の低いサーバーに接続します。代替サーバーのエントリーには、処理容量に関する情報はありません。代替サーバーへの接続は、 DB2 pureScale のどのメンバーにも接続できない場合にのみ試行されます。

3. 自動クライアント・リルート・プロセスがアプリケーションをデータベースに再接続できる場合、クライアントは新しく確立された接続のための実行環境を再構成します。クライアントは、サーバー情報が更新された新しいサーバー・リストのコピーを受け取ります。エラー SQL30108N がアプリケーションに戻されて、失敗したデータベース接続がリカバリーされ、トランザクションがロールバックされたことが示されます。その後アプリケーションは、ロールバックされた作業を繰り返すことを含めて、将来のリカバリーを担当します。

   失敗した SQL ステートメントがトランザクション内の最初の SQL ステートメントである場合は、自動クライアント・リルートとシームレス・フェイルオーバーが使用可能になり、クライアントが CLI または .NET の場合は、ドライバーは自動クライアント・リルート処理の一部として失敗した SQL 操作をやり直します。接続が成功すると、エラーはアプリケーションに報告されず、トランザクションはロールバックされません。接続の失敗と、それ以後のリカバリーはアプリケーションから隠されます。
4. 自動クライアント・リルートがデータベースに再接続できない場合は、エラー SQL30081N がアプリケーションに戻されます。その後アプリケーションは、接続障害からのリカバリーを担当します (例えば、それ自身でデータベースへの接続を試行します) 。

自動クライアント・リルートは、クライアントが新規接続での接続障害を検出したときにも使用されます。しかし、この場合、再接続が成功すると、失敗したデータベース接続がリカバリーされたことを示すエラーはアプリケーションに戻されません。再接続が失敗すると、エラー SQL30081N が戻されます。

DB2 Database for Linux, UNIX, and Windows への接続のためのトランザクション・レベルのワークロード・バランシングの操作

DB2 Database for Linux, UNIX, and Windows への接続のためのトランザクション・レベルのワークロード・バランシングは、トランザクションの開始時に DB2 pureScale インスタンス内のサーバー間で作業を平衡化することにより、高可用性に寄与します。

クライアントが DB2 Database for Linux, UNIX, and Windows DB2 pureScale インスタンスに接続し、トランザクション・レベルのワークロード・バランシングが有効になっている場合に行われるステップの概要を以下に示します。

1. クライアントが DB2 pureScale インスタンスへの最初の接続を確立すると、クライアントから接続されたメンバーは、 DB2 pureScale インスタンスのメンバーに関するサーバー・リストおよび接続の詳細 (IP アドレス、ポート、および重み) を戻します。

サーバー・リストがクライアントによってキャッシュに入れられます。キャッシュに入れられたサーバー・リストのデフォルト存続期間は、30 秒です。

2. 新規トランザクションの開始時に、クライアントはキャッシュに入れられたサーバー・リストを読み取って、処理容量に余裕のあるサーバーを識別します。そして、その使用率の低いサーバーに関連付けられたアイドル・トランスポートがないかどうかトランスポート・プールを調べます。 (アイドル・トランスポートとは、接続オブジェクトと関連付けられていないトランスポートのことです。)
   - アイドル・トランスポートが使用可能な場合、クライアントは接続オブジェクトをそのトランスポートと関連付けます。
   - ユーザー構成可能タイムアウト期間 (Java クライアントの場合 db2.jcc.maxTransportObjectWaitTime、非 Java クライアントの場合 maxTransportWaitTime) が経過した後、使用可能なアイドル・トランスポートがトランスポート・プールに存在せず、トランスポート・プールがその制限に達しているために新規トランスポートを割り振ることもできない場合、アプリケーションにエラーが戻されます。
3. トランザクションが実行されると、トランザクションはトランスポートに関連付けられたサーバーにアクセスします。
4. トランザクションが終了すると、クライアントは、トランスポートの再利用が接続オブジェクトに引き続き許可されているかどうかについてサーバーで検証します。
5. トランスポートの再利用が許可されている場合、サーバーは、接続オブジェクトの実行環境に適用される特殊レジスターに関する SET ステートメントのリストを返します。

   クライアントはこれらのステートメントをキャッシュに入れます。そして、接続オブジェクトが新規トランスポートと関連付けられた時に、実行環境を再構成するため、そのステートメントを再生します。
6. その後、クライアントが接続オブジェクトをトランスポートから切り離す必要があると判断した場合、それは切り離されます。
7. 新規で接続が確立されるか、または、30 秒毎もしくはユーザーが構成したインターバルが経過すると、サーバー・リストのクライアント・コピーがリフレッシュされます。
8. 新規トランザクションに、トランザクション・レベルのワークロード・バランシングが必要な場合、クライアントは上述のプロセスを使用して接続オブジェクトをトランスポートと関連付けます。

非 Java クライアントから DB2 Database for Linux, UNIX, and Windows への接続のための代替グループ

バージョン 9.7 フィックスパック 5 およびそれ以降のフィックスパックのリリースの非 Java クライアントの高可用性を向上するために、現在のグループへの接続を再確立できなかった場合、自動クライアント・リルートのための追加のフェイルオーバー・メカニズムとして代替グループを使用します。

デフォルトで、非 Java クライアントは、自動クライアント・リルート (ACR) が使用可能になっています。この機能は、サーバーへの接続を再確立できなかった場合に、現在のグループ内の代替サーバーへの自動フェイルオーバーを提供します。

この ACR 機能に加えて、現在のグループへの接続を再確立できなかった場合のフェイルオーバー・ターゲットとして、*代替グループ* を定義できます。非 Java クライアントに対して代替グループを定義するには、次のようにします。

- db2dsdriver.cfg ファイルの <acr> セクションの <alternategroup> 要素内で、各代替グループに対して<database> 要素を定義します。 <database> 要素内で <parameter> 要素を指定しないでください。パラメーター設定は、1 次グループから継承されます。<alternategroup> 要素内で、複数の <database> 要素を定義できます。 <database> 要素の順序は、フェイルオーバー中に使用される順序です。
- 代替グループへのフェイルオーバー接続からのエラー・メッセージを抑止する場合は、<alternategroup> 要素で enableAlternateGroupSeamlessACR パラメーターを true に設定します。

デフォルトの ACR の再試行期間は 10 分です。代替グループを定義すると、この期間は 2 分に短縮されます。

非 Java クライアントを代替グループに接続するとき、1 次グループの <database> 要素のすべての接続設定およびパラメーター設定は、代替グループのデータベースへの接続によって継承されます。

非 Java クライアントが代替グループのデータベースに接続した後、1 次グループへのフェイルバックは提供されません。1 次グループにもう一度接続するには、アプリケーションまたはクライアントを再始動する必要があります。

代替グループは、ACR およびワークロード・バランシングでのみサポートされています。クライアントのアフィニティーが構成されている場合、代替グループの定義は無視されます。

例

db2dsdriver.cfg ファイルの代替グループの定義の例を次に示します。

```
<dsncollection>
  <dsn alias="mydsn2" name="mydb2" host="myserver2.ibm.com" port="5912">
  ...
</dsncollection>

<databases>
  <database name="mydb2" host="myserver2.ibm.com" port="5912">
   <parameter name="IsolationLevel" value="4"/>
   ...
   <wlb>
     <parameter name="enableWLB" value="true"/>
   </wlb>
   <acr>
    ... (ACR parameters definition)
    <alternateserverlist>
      <server name="server1" hostname="db2luwa.luw.ibm.com" port="5912">
      </server>
      <server name="server2" hostname="db2luwb.luw.ibm.com" port="5912">
      </server>
    </alternateserverlist>
    <alternategroup>
     <parameter name="enableAlternateGroupSeamlessACR" value="true"/>
     <database name="mydb3" host="myserver3.ibm.com" port="5912">
     </database>
     <database name="mydb4" host="myserver4.ibm.com" port="5912">
     </database>
     </alternategroup>   </acr>
```

```
   </database>

   <database name="mydb3" host="myserver3.ibm.com" port="5912">
    <parameter name="IsolationLevel" value="2"/>
    <acr>
     <parameter name="enableACR" value="true"/>
    <alternateserverlist>
      <server name="server4" hostname="db2luwd.luw.ibm.com" port="5912">
      </server>
    </alternateserverlist>
    <alternategroup>
     <parameter name="enableAlternateGroupSeamlessACR" value="true"/>
     <database name="mydb5" host="myserver5.ibm.com" port="5912">
     </database>
     </alternategroup>    </acr>
   ...
   </database>
</databases>
```

次のサンプル・シナリオでは、代替グループに対して自動クライアント・リルートが動作する方法を説明します。代替グループのフェイルオーバーの詳細に焦点を当てるために、現在のグループへの ACR のフェイルオーバーについての詳細はこれらのシナリオでは説明されていません。これらのシナリオは、前述の段落で説明した、db2dsdriver.cfg サンプル・ファイルを使用します。

1 次グループへの最初の接続が失敗する

非 Java クライアントが最初の試行で 1 次グループへの接続に失敗した場合、現在のグループの代替サーバーへの自動クライアント・リルートのフェイルオーバーも失敗します。この例では、クライアントは次のアクションを実行します。

1. クライアントは *mydb2* への接続に失敗します。
2. クライアントは *server1* への接続に失敗します。
3. クライアントは *server2* への接続に失敗します。
4. クライアントは、db2dsdriver.cfg ファイルの <alternategroup> セクションにリストされた代替グループに、このファイルで指定された順序で接続しようとします。
   a. クライアントは *mydb3* への接続に失敗します。
   b. クライアントは正常に *mydb4* に接続します。

mydb4 に接続した後、シームレスまたは非シームレスの動作の規則が適用されたままになります。クライアントが *mydb4* に接続できなかった場合、SQL30081N エラー・メッセージを受け取ります。

1 次サーバーへの後続の接続または既存の接続が失敗する

非 Java クライアントで *mydb2* への接続が切断された場合、現在のグループの代替サーバーへの自動クライアント・リルートのフェイルオーバーも失敗します。この例では、クライアントは次のアクションを実行します。

1. クライアントは *server1* への接続に失敗します。
2. クライアントは *server2* への接続に失敗します。
3. クライアントは、db2dsdriver.cfg ファイルの <alternategroup> セクションにリストされた代替グループに、このファイルで指定された順序で接続しようとします。
   a. クライアントは正常に *mydb3* に接続します。

mydb3 に接続した後、シームレスまたは非シームレスの動作の規則が適用されたままになります。

代替グループへの既存の接続が失敗する

非 Java クライアントが *mydb2* への接続に失敗し、現在のグループの代替サーバーへの自動クライアント・リルートのフェイルオーバーも失敗し、その後 *mydb3* 代替グループへ正常に接続します。

クライアントで *mydb3* への接続が切断された後、*mydb4* に接続しようとします。この例では、クライアントは *mydb4* への接続に失敗します。

クライアントは SQL30081N エラー・メッセージを受け取ります。1 次グループへの接続を再度試行するには、クライアントまたはアプリケーションを再始動する必要があります。

DB2 Database for Linux, UNIX, and Windows サーバーへの接続の高可用性のためのアプリケーション・プログラミング要件

自動クライアント・リルートのフェイルオーバーは、シームレスまたは非シームレスを選択することができます。DB2 Database for Linux, UNIX, and Windows への接続のためのフェイルオーバーがシームレスでない場合は、フェイルオーバーの発生時に戻されるエラーについて説明するコードを追加する必要があります。

フェイルオーバーが非シームレスで、サーバーとの接続が再確立した場合、SQLCODE -4498 (Java クライアントの場合) または SQL30108N (非 Java クライアントの場合) がアプリケーションへ戻されます。現行のトランザクションで発生したすべての処理は、ロールバックされます。アプリケーションにおいて、以下のことが必要になります。

- エラーと共に戻される理由コードを確認します。障害が起きたデータ共有メンバーの特殊レジスター設定を、新規 (フェイルオーバーした) データ共有メンバーへ持ち越すかどうかを決定します。現行のものではない特殊レジスター値はすべてリセットします。
- 直前のトランザクションの間に発生したすべての SQL 操作を実行します。

DB2 Database for Linux, UNIX, and Windows への接続のフェイルオーバーをシームレスにするには、以下の条件を満たしている必要があります。

- アプリケーション・プログラミング言語が Java、CLI、または .NET である。
- トランザクションにおける接続ではない。これは、トランザクションにおける最初の SQL ステートメントが実行された時に、障害が発生したという場合です。
- (トランザクション・レベルのロード・バランシングが有効になっている場合) データ・サーバーは、直前のトランザクションの終わりでトランスポートの再利用を許可する。
- すべてのグローバル・セッション・データがクローズ、またはドロップされている。
- オープン状態の保留カーソルがない。
- アプリケーションが CLI を使用している場合、そのアプリケーションは SQL ステートメントを再生するために直前で呼び出された API の履歴を維持するようド

ライバーに要求するアクションは実行できない。そのようなアクションの例として、実行時にデータを指定すること、コンパウンド SQL を実行すること、または配列入力を使用することがあります。

- アプリケーションがストアード・プロシージャーではない。
- 自動コミットが有効ではない。自動コミットが有効な場合でも、シームレスなフェイルオーバーは可能です。しかし、次の状況は問題を引き起こす可能性があります。データ・サーバーにおいて SQL 処理が正常に実行され、コミットされますが、コミット操作の確認応答がクライアントへ送信される前に、接続またはサーバーに障害が発生したとします。クライアントが接続を再確立するとき、直前にコミットされた SQL ステートメントを再生します。結果として、SQL ステートメントが 2 度実行されたことになります。この状況を回避するため、シームレス・フェイルオーバーを使用可能にする場合、自動コミットをオフにします。

DB2 Database for Linux, UNIX, and Windows に接続するクライアントに関するクライアント・アフィニティー

クライアント・アフィニティーは、自動クライアント・リルート機能を提供するためのクライアント専用の方法です。

CLI、.NET、または Java (IBM Data Server Driver for JDBC and SQLJ Type 4 接続) を使用するアプリケーションでは、クライアント・アフィニティーを使用することができます。すべての再ルーティングは、ドライバーによって制御されます。

クライアント・アフィニティーでは、特定の 1 次サーバーへ接続する必要がある状態を対象としています。 1 次サーバーへの接続中に障害が発生した場合、代替サーバーへのフェイルオーバーに対して特定の順序を実行する必要があります。サーバー・フェイルオーバー機能を使用する自動クライアント・リルートがご使用の環境で機能しない場合に限り、自動クライアント・リルートにクライアント・アフィニティーを使用する必要があります。

クライアント・アフィニティーの構成の一部として、代替サーバーのリスト、および代替サーバーへの接続を試みる順序を指定します。クライアント・アフィニティーの使用中は、アプリケーションによって指定されるホスト名およびポート番号の代わりに、代替サーバーのリストに基づいて接続が確立されます。例えば、アプリケーションが server1 へ接続するよう指定している場合でも、構成プロセスが server2、server3、server1 という順序でサーバーへの試行をするよう指定しているならば、最初の接続は server1 の代わりに server2 へ行われます。

以下の条件が当てはまる場合、クライアント・アフィニティーを使用するフェイルオーバーはシームレスになります。

- トランザクションにおける接続ではない。これは、トランザクションにおける最初の SQL ステートメントが実行された時に、障害が発生したという場合です。
- サーバーに使用中のグローバル一時表がない。
- オープン状態の保留カーソルがない。

クライアント・アフィニティーを使用する時に、障害後に 1 次サーバーが稼働に戻る場合、トランザクション境界で代替サーバーから 1 次サーバーへ接続が戻るよう指定できます。このアクティビティーがフェイルバックです。

DB2 Database for Linux, UNIX, and Windows 接続に関する非 Java クライアントのクライアント・アフィニティーの構成

CLI および .NET アプリケーションでクライアント・アフィニティーのサポートを使用可能にするには、db2dsdriver.cfg 構成ファイル内に値を設定し、クライアント・アフィニティーを使用することを示し、1 次サーバーと代替サーバーを指定します。

次の表に、CLI および .NET アプリケーションのクライアント・アフィニティーを使用可能にするための db2dsdriver.cfg ファイルにおける設定を示します。

表 28. CLI および .NET アプリケーションのクライアント・アフィニティーを使用可能にするための設定

db2dsdriver 構成ファイルの acr セクション内の要素	**値**
enableAcr パラメーター	true
maxAcrRetries パラメーター	自動クライアント・リルートの間に代替サーバーのリストにある各サーバーへの接続が試行される回数。有効な範囲は 0 から MAX_INT までです。値が 0 の場合、再試行回数は 1 回です。デフォルトは 3 です。
acrRetryInterval パラメーター	再試行と再試行の間で待機する秒数。有効な範囲は 0 から MAX_INT までです。デフォルトは、待機なし (0) です。
affinityFailbackInterval パラメーター	1 次サーバーへフェイルバックする最初のトランザクション境界の後で待機する秒数。 1 次サーバーへフェイルバックしたい場合に、この値を設定します。デフォルトは 0 です。つまり、1 次サーバーへのフェイルバックは試行されません。
alternateserverlist	クライアント・アフィニティーを介する自動クライアント・リルートで使用される各サーバーのホスト名およびポート番号を識別する <server> 要素。この要素のうち 1 つが、1 次サーバーを識別する必要があります。これらの要素があるからといって、自動クライアント・リルートがアクティブになるわけではありません。
affinitylist	serverorder 属性を持つ <list> 要素。serverorder 属性の値は、クライアント・アフィニティーを使用した自動クライアント・リルートの間に試行される順序で、サーバーのリストを明示します。<list> 要素にあるサーバーは、<alternateserverlist> の <server> 要素にも定義する必要があります。それぞれが異なるサーバー順序を持つ複数の <list> 要素を指定することもできます。 <affinitylist> 要素があるからといって、自動クライアント・リルートがアクティブになるわけではありません。

表 28. CLI および .NET アプリケーションのクライアント・アフィニティーを使用可能にするための設定 (続き)

db2dsdriver 構成ファイルの acr セクション内の要素	値
client_affinity	各クライアントに対して行うサーバー接続の試行順序を定義する <clientaffinitydefined> 要素、または <clientaffinityroundrobin> 要素。<clientaffinitydefined> 要素を含める場合、サーバー順序を定義する <list> 要素をそれぞれが指定する <client> 要素を定義することによって、サーバー順序を定義します。<clientaffinityroundrobin> 要素を含める場合も、<client> 要素を指定する必要があります。しかし、この場合の <client> 要素は、<list> 要素を指定しません。その代わりに、<client> 要素の順序がサーバー順序を定義します。データベースに接続するすべてのクライアントを <clientaffinitydefined> または <clientaffinityroundrobin> 要素内に指定する必要があります。あるクライアント・マシンに複数のネットワーク・インターフェース・カードがある場合、アフィニティー・リストを導出するために、CLI ドライバーによって自動的にクライアント・ホスト名が発見され、構成ファイル項目との突き合わせが行われます。CLI ドライバーはすべてのネットワーク・インターフェースを取得して、db2dsdriver 構成ファイルで使用可能なホスト名と突き合わせます。ドメイン名のないホスト名が db2dsdriver.cfg で指定されていると、CLI はデフォルトのドメインを使用して解決しようとし、発見したホスト名と突き合わせます。IP アドレスが cfg ファイルのクライアント・アフィニティー・セクションで定義されている場合、アフィニティー・リストを導出するために、CLI ドライバーがそれぞれの IP アドレスを見つけて構成ファイル項目と (ホスト名を) 突き合わせます。
clientaffinitydefined	各クライアントに対する自動クライアント・リルートのサーバー順序を定義する <client> 要素。それぞれの <client> 要素は、クライアントを <affinitylist> 要素から <list> 要素へ関連付ける listname 属性を含んでいます。

表 28. CLI および .NET アプリケーションのクライアント・アフィニティーを使用可能にするための設定 (続き)

db2dsdriver 構成ファイルの acr セクション内の要素	値
clientaffinityroundrobin	<clientaffinityroundrobin> 要素内での順番が、自動クライアント・リルートに対して選択される最初のサーバーを定義する <client> 要素。各 <client> 要素には、索引があります。<clientaffinityroundrobin> 要素内で 1 番目の <client> 要素の索引は 0 になり、2 番目の <client> 要素の索引は 1 になり、以下同様に続きます。 <alternateserverlist> 要素内のサーバーの数が *n* 台、また <client> 要素の <clientaffinityroundrobin> 要素内の索引が *i* だとします。試行される最初のサーバーは、<alternateserverlist> 要素内の索引が *i* mod *n* になるサーバーです。次に試行されるサーバーは、<alternateserverlist> 要素内の索引が (*i* +1 mod *n*) になるサーバーです。以下同様に続きます。

以下の制約事項は、CLI または .NET クライアントのクライアント・アフィニティーの構成に適用されます。

- 1 つのクライアントに対して適格な代替サーバーの数が 24 より大きい場合、エラー SQL1042N が発生します。
- クライアント・アフィニティーが有効な時に、ワークロード・バランシングを有効にすることはできません。これは、enableWLB が true に設定されている場合に、client_affinity 要素を指定すると、エラー SQL5162N が発生するからです。
- <alternateserverlist>、<affinitylist>、または <client_affinity> 要素で必須属性が指定されていない場合、エラー SQL5163N が発生します。
- クライアント・アフィニティーが有効で、<alternateserverlist> 要素が空の場合、エラー SQL5164N が発生します。
- クライアント・アフィニティーが有効で、サーバーへの接続を試行しているクライアントのホスト名が、<client_affinity> サブグループ (<clientaffinitydefined> または <clientaffinityroundrobin>) のどちらにもない場合、または複数のサブグループにある場合、エラー SQL5164N が発生します。
- それぞれのクライアント・マシンに対して、<clientaffinitydefined> セクションまたは <clientaffinityroundrobin> セクションのいずれかに、1 つの項目だけが存在していなければなりません。db2dsdriver.cfg 内に、異なるホスト名によって同一のクライアント・マシンが指定されている複数の項目があると、エラー SQL5162N が生じます。

DB2 Database for Linux, UNIX, and Windows 接続に関する非 Java クライアントのクライアント・アフィニティーを使用可能にする例

CLI アプリケーションまたは .NET アプリケーションで自動クライアント・リルートにおけるクライアント・アフィニティーを使用するには、その前に

db2dsdriver.cfg 構成ファイルの <acr> セクション内に要素を組み込んで、クライアント・アフィニティーを使用することを示し、1 次サーバーと代替サーバーを特定する必要があります。

以下の例では、フェイルバックしないフェイルオーバーのクライアント・アフィニティーを使用可能にする方法が示されます。

db2dsdriver 構成ファイルが以下のようになっているとします。

```
<database name="SAMPLE" host="v33ec065.svl.ibm.com" port="446">
  <acr>
    <parameter name="enableAcr" value="true"/>
    <parameter name="maxAcrRetries" value="1"/>
    <parameter name="acrRetryInterval" value="2"/>
    <alternateserverlist>
      <server name="server1"
              hostname="v33ec067.svl.ibm.com"
              port="446">
      </server>
      <server name="server2"
              hostname="v33ec066.svl.ibm.com"
              port="446">
      </server>
      <server name="server3"
              hostname="v33ec065.svl.ibm.com"
              port="446">
      </server>
    </alternateserverlist>
    <affinitylist>
      <list name="list1" serverorder="server1,server2,server3">
      </list>
      <list name="list2" serverorder="server3,server2,server1">
      </list>
    </affinitylist>
    <clientaffinitydefined>
    <!- this section has specific defined affinities -->
      <client name="client1"
              hostname="appsrv1.svl.ibm.com"
              listname="list2">
      </client>
      <client name="client2"
              hostname="appsrv2.svl.ibm.com"
              listname="list1">
      </client>
    </clientaffinitydefined>
    <clientaffinityroundrobin>
      <client name="client3" hostname="appsrv3.svl.ibm.com">
          <!- This entry is index 0. The number of servers is 3.
              0 mod 3 is 0, so the first that is tried
              during automatic client reroute is the server whose
              index in <alternateserverlist> is 0 (server1).
              The next server has index 1 mod 3, which is 1
              (server2). The final server has index 2 mod 3,
              which is 2 (server3). -->
      </client>
      <client name="client4" hostname="appsrv4.svl.ibm.com">
          <!- This entry is index 1. The number of servers is 3.
              1 mod 3 is 1, so the first that is tried
              during automatic client reroute is the server whose
              index in <alternateserverlist> is 1 (server2).
              The next server has index 2 mod 3, which is 2
              (server3). The final server has index 3 mod 3,
              which is 0 (server1). -->
```

```
      </client>
    </clientaffinityroundrobin>
  </acr>
</database>
```

ホスト名 appsrv4.svl.ibm.com (client4) というクライアントから v33ec065.svl.ibm.com:446 によって識別されるサーバーへの接続中に、通信障害が発生したとします。以下のステップは、クライアント・アフィニティーを使用した自動クライアント・リルートに起こるプロセスを示します。

1. ドライバーは v33ec066.svl.ibm.com:446 (server2) への接続を試みます。
2. v33ec066.svl.ibm.com:446 への接続は失敗します。
3. ドライバーは 2 秒間待機します。
4. ドライバーは v33ec065.svl.ibm.com:446 (server3) への接続を試みます。
5. v33ec065.svl.ibm.com:446 への接続は失敗します。
6. ドライバーは 2 秒間待機します。
7. ドライバーは v33ec067.svl.ibm.com (server1) への接続を試みます。
8. v33ec067.svl.ibm.com への接続は失敗します。
9. ドライバーは 2 秒間待機します。
10. ドライバーはエラー・コード SQL30081N を返します。

以下の例では、フェイルバックするフェイルオーバーのクライアント・アフィニティーを使用可能にする方法が示されます。

db2dsdriver 構成ファイルが以下のようになっているとします。

```
<database name="SAMPLE" host="v33ec065.svl.ibm.com" port="446">
  <acr>
    <parameter name="enableAcr" value="true"/>
    <parameter name="maxAcrRetries" value="1"/>
    <parameter name="acrRetryInterval" value="2"/>
    <parameter name="affinityFailbackInterval" value="300"/>
    <alternateserverlist>
      <server name="server1"
              hostname="v33ec067.svl.ibm.com"
              port="446">
      </server>
      <server name="server2"
              hostname="v33ec066.svl.ibm.com"
              port="446">
      </server>
      <server name="server3"
              hostname="v33ec065.svl.ibm.com"
              port="446">
      </server>
    </alternateserverlist>
    <affinitylist>
      <list name="list1" serverorder="server1,server2,server3">
      </list>
      <list name="list2" serverorder="server3,server2,server1">
      </list>
    </affinitylist>
    <clientaffinitydefined>
    <!- this section has specific defined affinities -->
      <client name="client1"
              hostname="appsrv1.svl.ibm.com"
              listname="list2">
      </client>
      <client name="client2"
```

```
            hostname="appsrv2.svl.ibm.com"
            listname="list1">
      </client>
    </clientaffinitydefined>
    <clientaffinityroundrobin>
      <client name="client3" hostname="appsrv3.svl.ibm.com">
          <!- This entry is index 0. The number of servers is 3.
              0 mod 3 is 0, so the first that is tried
              during automatic client reroute is the server whose
              index in <alternateserverlist> is 0 (server1).
              The next server has index 1 mod 3, which is 1
              (server2). The final server has index 2 mod 3,
              which is 2 (server3). -->
      </client>
      <client name="client4" hostname="appsrv4.svl.ibm.com">
          <!- This entry is index 1. The number of servers is 3.
              1 mod 3 is 1, so the first that is tried
              during automatic client reroute is the server whose
              index in <alternateserverlist> is 1 (server2).
              The next server has index 2 mod 3, which is 2
              (server3). The final server has index 3 mod 3,
              which is 0 (server1). -->
      </client>
    </clientaffinityroundrobin>
  </acr>
</database>
```

クライアント appsrv2.svl.ibm.com (client2) から v33ec065.svl.ibm.com:446 へ接続された後に、データベース管理者が v33ec065.svl.ibm.com:446 によって識別されるサーバーを保守のため停止するとします。以下は、代替サーバーへフェイルオーバーし、保守が完了した後、1 次サーバーへフェイルバックするステップを示します。

1. クライアント appsrv1.svl.ibm.com に代わって、ドライバーは v33ec065.svl.ibm.com:446 へ正常に接続します。
2. データベース管理者は v33ec065.svl.ibm.com:446 を停止します。
3. アプリケーションは、その接続で処理を試みます。
4. ドライバーは v33ec066.svl.ibm.com:446 へ正常にフェイルオーバーします。
5. 200 秒後、処理がコミットされます。
6. ドライバーは、フェイルバック・インターバル (300 秒) が経過したかどうか確認します。経過していない場合、フェイルバックは発生しません。
7. アプリケーションは v33ec066.svl.ibm.com:446 への接続で処理を続けます。
8. 105 秒後、処理がコミットされます。
9. ドライバーは、フェイルバック・インターバル (300 秒) が経過したかどうか確認します。経過していると、v33ec065.svl.ibm.com:446 へのフェイルバックが発生します。

Informix サーバーへの接続の高可用性のための非 Java クライアント・サポート

IBM Informix サーバー上の高可用性クラスター・サポートは、ワークロード・バランシングと自動クライアント・リルートにより、クライアント・アプリケーションに高可用性を提供します。このサポートは、Java クライアント (JDBC、SQLJ、または pureQuery) または非 Java クライアント (ODBC、 CLI、 .NET、 OLE DB、 PHP、 Ruby、または組み込み SQL) を使用するアプリケーションで使用できます。

Java クライアントでは、IBM Informix の高可用性クラスター・サポートを利用するために、IBM Data Server Driver for JDBC and SQLJ Type 4 接続を使用する必要があります。

非 Java クライアントでは、高可用性クラスター・サポートを利用するために、以下のクライアントまたはクライアント・パッケージの 1 つを使用する必要があります。

- IBM Data Server Client
- IBM Data Server Runtime Client
- IBM Data Server Driver Package
- IBM Data Server Driver for ODBC and CLI

IBM Informix サーバーへの接続の高可用性のためのクラスター・サポートには、以下のものが含まれます。

自動クライアント・リルート

このサポートによって、クライアントは、高可用性クラスター内の任意の使用可能なサーバーを介してデータベースへの再接続を試行することにより、障害からリカバリーできます。別のサーバーへ再接続することを*フェイルオーバー*と呼びます。クライアントでのワークロード・バランシングを使用可能に設定することによって、クライアントでの自動クライアント・リルートを使用可能にすることができます。

IBM Informix 環境では、1 次サーバーおよびスタンバイ・サーバーは、接続マネージャーによって制御される高可用性クラスターのメンバーに対応します。複数の接続マネージャーが存在する場合、クライアントはそれらを使用して、1 次サーバーおよび代替サーバーの情報を判別します。クライアントは、最初の接続にのみ、代替の接続マネージャーを使用します。

自動クライアント・リルートのフェイルオーバーは、*シームレス* にすることも、*非シームレス* にすることもできます。非シームレスなフェイルオーバーでは、クライアント・アプリケーションが代替サーバーへ再接続するときに、フェイルオーバー (代替サーバーへの接続) が発生したことを示すため、サーバーがアプリケーションへ毎回エラーを戻します。

Java、CLI、または .NET クライアント・アプリケーションでは、自動クライアント・リルートのフェイルオーバーについて、シームレスまたは非シームレスを選択することができます。シームレスなフェイルオーバーでは、アプリケーションが代替サーバーへ正常に再接続するとき、サーバーがアプリケーションにエラーを戻すことはありません。

ワークロード・バランシング

ワークロード・バランシングによって、IBM Informix 高可用性クラスターの可用性を高めることができます。ワークロード・バランシングが有効になると、クライアントは高可用性クラスターのメンバーに関する状況情報を頻繁に取得します。クライアントはこの情報を元に、次のトランザクションをどのサーバーに転送するべきか判断します。ワークロード・バランシングを使用する場合、IBM Informix 接続マネージャーは、サーバー間で作業が効率的に分散され、1 つのサーバーに障害が発生すると別のサーバーに作業が転送されるようにします。

接続コンセントレーター

このサポートは、IBM Informix へ接続する Java アプリケーションで使用でき

ます。接続コンセントレーターは、多数のワークステーションやウェブ・ユーザーをサポートするために IBM Informix データベース・サーバーで必要とされるリソースの数を削減します。接続コンセントレーターを使用すれば、ほんのわずかな並行した、アクティブな物理接続が必要になるだけで、データベース・サーバーに並行してアクセスする多数のアプリケーションをサポートすることができます。 Java クライアントでワークロード・バランシングを使用可能に設定すると、自動的に接続コンセントレーターを使用することができます。

クライアント・アフィニティー
クライアント・アフィニティーは、クライアントによって完全に制御される自動クライアント・リルートのソリューションです。これは、特定の 1 次サーバーへ接続する必要がある状態を対象としています。 1 次サーバーへの接続中に障害が発生した場合、クライアント・アフィニティーを使用して、代替サーバーへのフェイルオーバーの特定の順序を実行します。

非 Java クライアント用の Informix 高可用性サポートの構成

Informix 高可用性クラスターに接続する非 Java クライアント・アプリケーションを高可用性のために構成するには、接続マネージャーを表すアドレスに接続し、ワークロード・バランシングおよび接続の最大数を使用可能にするプロパティーを設定する必要があります。

IBM Informix への接続の高可用性のために IBM Data Server Driver for JDBC and SQLJ を使用可能にするには、インストール済み環境に 1 つ以上の接続マネージャー、1 つの 1 次サーバー、および 1 つ以上の代替サーバーがなければなりません。

以下の表では、非 Java アプリケーションのための基本設定について説明しています。

表 29. 非 Java アプリケーションで Informix 高可用性サポートを使用可能にするための基本設定

クライアントの設定	値
接続アドレス:	
データベース・ホスト[1]	接続マネージャーの IP アドレス。 248 ページの『接続マネージャーへの接続のためのサーバーとポートのプロパティーの設定』を参照してください。
データベース・ポート[1]	接続マネージャーの SQL ポート番号。 248 ページの『接続マネージャーへの接続のためのサーバーとポートのプロパティーの設定』を参照してください。
データベース名[1]	データベース名

表 29. 非 Java アプリケーションで Informix 高可用性サポートを使用可能にするための基本設定 (続き)

クライアントの設定	値

注:

1. 使用するクライアントに応じて、接続情報は以下のいずれかのソース内で定義されます。
   - いずれかのデータ・サーバー・ドライバー、あるいは IBM Data Server Client または IBM Data Server Runtime Client を使用する CLI アプリケーションまたはオープン・ソース・アプリケーションを使用している場合、以下のようになります。
     - ホスト、ポート、およびデータベース情報がアプリケーション内の接続ストリング内で提供される場合、DB2 はその情報を使用します。
     - ホスト、ポート、およびデータベース情報がアプリケーション内の接続ストリング内で提供されない場合、ドライバーは `db2cli.ini` ファイルを使用し、この情報が `db2cli.ini` ファイル内で提供されている場合、DB2 はその情報を使用します。
     - ホスト、ポート、およびデータベース情報がアプリケーション内の接続ストリングおよび `db2cli.ini` ファイルのどちらでも提供されない場合、DB2 は `db2dsdriver` 構成ファイル内の情報を使用します。
   - .NET アプリケーション、または組み込み SQL を IBM Data Server Client または IBM Data Server Runtime Client と共に使用するアプリケーションを使用している場合、接続情報は `db2dsdriver` 構成ファイル以外のソースから提供されます。データベース・カタログ、接続ストリング、`db2cli.ini` ファイル、または .NET オブジェクト・プロパティーなどがソースになる可能性があります。

Informix 高可用性サポートのワークロード・バランシング機能を適切に調整したい場合は、追加のプロパティーを使用できます。以下の表では、非 Java アプリケーションに関する追加のプロパティーがリストされています。

表 30. 非 Java アプリケーションから Informix データベース・サーバーへの接続のためにワークロード・バランシング・サポートを適切に調整するためのプロパティー

db2dsdriver 構成ファイル内の要素	db2dsdriver ファイル内のセクション	説明
enableWLB	<wlb>	ワークロード・バランシングが使用可能かどうかを指定します。ワークロード・バランシングを使用可能にするには、この値を true に設定します。
maxTransportIdleTime	<wlb>	アイドル・トランスポートがドロップされる前の最大経過時間を秒数で指定します。デフォルトは 600 です。サポートされる最小値は 0 です。
maxTransportWaitTime	<wlb>	トランスポートが使用可能になるのをクライアントが待機する秒数を指定します。デフォルトは、 -1 (制限なし) です。サポートされる最小値は 0 です。
maxTransports	<wlb>	リクエスターが高可用性クラスターに対して行うことができる接続の最大数を指定します。デフォルトは、 -1 (制限なし) です。サポートされる最小値は 1 です。
maxRefreshInterval	<wlb>	サーバー・リストがリフレッシュされる前の最大経過時間を秒数で指定します。デフォルトは 10 です。サポートされる最小値は 0 です。

ワークロード・バランシングを使用する必要があり、自動クライアント・リルート処理に関して戻されるエラーをアプリケーションが処理できない場合は、db2dsdriver.cfg 構成ファイル内に以下のパラメーターを設定します。

表 31. 非 Java アプリケーションから Informix データベース・サーバー への接続のための Sysplex ワークロード・バランシングのみを使用可能にするためのプロパティー

db2dsdriver 構成ファイル内の要素	db2dsdriver ファイル内のセクション	説明	設定する値
enableWLB	<wlb>	ワークロード・バランシングが使用可能かどうかを指定します。	true。enableAcr が True の場合、接続マネージャーはサーバー接続を再試行します。これは、DB2 for z/OS バージョン 9.0 サーバーでサポートされます。enableAcr 値と enableWLB 値が false の場合、サーバー接続は失敗します。
enableAcr	<acr>	自動クライアント・リルートが使用可能かどうかを指定します。CLI または .NET アプリケーションの場合、自動クライアント・リルートを使用可能にすると、シームレス・フェイルオーバーが自動的に使用可能になります。	false
enableSeamlessAcr	<acr>	シームレス・フェイルオーバーが使用可能かどうかを指定します。非 Java アプリケーション間で、シームレス・フェイルオーバーは CLI または .NET アプリケーションだけにサポートされます。true がデフォルトです。	enableAcr が false の場合、この値は false になるので、設定する必要はありません。

接続マネージャーへの接続のためのサーバーとポートのプロパティーの設定

接続マネージャーへの接続のためにサーバーおよびポート番号を設定するには、以下のプロセスに従います。

- 高可用性クラスターが単一の接続マネージャーを使用している場合は、サーバー名とポート番号を、接続マネージャーのサーバー名とポート番号に設定します。
- 高可用性クラスターが複数の接続マネージャーを使用している場合は、以下のようにします。
  1. 主に使用する接続マネージャーのサーバー名とポート番号を指定します。
  2. db2dsdriver.cfg 構成ファイルのデータベース項目の <acr> サブセクション内で、enableAlternateServerListFirstConnect の値を true に設定します。
  3. db2dsdriver.cfg 構成ファイルの <acr> サブセクション内の alternateserverlist 項目に、代替の接続マネージャーのサーバー名とポート番号を設定します。

非 Java クライアントで IDS 高可用性サポートを使用可能にする例

直接 IDS サーバーに接続している CLI、.NET、または組み込み SQL アプリケーションで IDS 高可用性サポートを使用するには、その前に db2dsdriver 構成ファイルを適切な設定にして更新し、接続マネージャーに接続する必要があります。

以下の例は、1 つの接続マネージャーを使用する IDS 高可用性サポートを活用するように CLI クライアントをセットアップする方法を示しています。

クライアントをセットアップするには、その前に接続マネージャーによって制御される 1 つ以上の高可用性クラスターを構成する必要があります。

クライアントをセットアップするには、以下のステップに従います。

1. IDS 高可用性サポートの基本設定を使用して、db2dsdriver.cfg ファイルを作成します。enableWLB を true に設定すると、ワークロード・バランシングと自動クライアント・リルート機能が使用可能になります。

```
<configuration>
  <dsncollection>
    <dsn alias="IDSCM1" name="IDSCM1" host="ids.cm1.ibm.com" port="446">
    </dsn>
  </dsncollection>
  <databases>
    <database name="IDSCM1" host="ids.cm1.ibm.com" port="446">
       <!-- database-specific parameters -->
       <wlb>
          <!-- Enable workload balancing to get
          automatic client reroute
          functionality -->
          <parameter name="enableWLB" value="true" />
          <!-- maxTransports represents the maximum number of transports -->
          <parameter name="maxTransports" value="80" />
       </wlb>
    </database>
  </databases>
  <parameters>
    <parameter name="connectionLevelLoadBalancing" value="true"/>
  </parameters>
</configuration>
```

2. IDSCM1 の DSN 定義が、データベース IDSCM1 に関する接続マネージャーについての接続情報を提供すると仮定します。CLI アプリケーションで、以下のようなコードを使用して接続マネージャーに接続します。

```
...
   SQLHDBC         hDbc    = SQL_NULL_HDBC;
   SQLRETURN       rc      = SQL_SUCCESS;
   SQLINTEGER      RETCODE = 0;
   char            *ConnStrIn =
                   "DSN=IDSCM1;PWD=mypass";
                /* dsn matches the database name in the configuration file */
   char            ConnStrOut [200];
   SQLSMALLINT     cbConnStrOut;
   int             i;
   char            *token;
...
  /*************************************************************/
  /* Invoke SQLDriverConnect                                   */
  /*************************************************************/
   RETCODE = SQLDriverConnect (hDbc                 ,
                               NULL                 ,
```

```
                                    (SQLCHAR *)ConnStrIn ,
                                    strlen(ConnStrIn)    ,
                                    (SQLCHAR *)ConnStrOut,
                                    sizeof(ConnStrOut)   ,
                                    &cbConnStrOut        ,
                                    SQL_DRIVER_NOPROMPT);
...
```

非 Java クライアントから IDS への接続のための自動クライアント・リルートの操作

自動クライアント・リルート・サポートは、IBM データ・サーバー・クライアントが IBM Informix (IDS) 高可用性クラスター内のサーバーへの接続を失う際に、フェイルオーバー・サポートを提供します。自動クライアント・リルートにより、クライアントは、クラスター内の任意の使用可能なサーバーを介してデータベースへの再接続を試行することにより、障害からリカバリーできます。

自動クライアント・リルートは、ワークロード・バランシングが使用可能なときにはデフォルトで使用可能です。

自動クライアント・リルートが使用可能な場合、クライアントが既存の接続で接続障害を検出するときに、以下のプロセスが通常発生します。

1. クライアントが既存の接続を使用して SQL ステートメントの実行を試行し、障害を検出します。
2. クライアントは、接続マネージャーによって戻されるサーバー・リストを使用して、アクセス対象のサーバーを識別し、データベースへの再接続を試行します。
3. 自動クライアント・リルート・プロセスがアプリケーションをデータベースに再接続できる場合、クライアントは新しく確立された接続のための実行環境を再構成します。エラー SQL30108N がアプリケーションに戻されて、失敗したデータベース接続がリカバリーされ、トランザクションがロールバックされたことが示されます。その後アプリケーションは、ロールバックされた作業を繰り返すことを含めて、将来のリカバリーを担当します。

   失敗した SQL ステートメントがトランザクション内の最初の SQL ステートメントである場合は、自動クライアント・リルートとシームレス・フェイルオーバーが使用可能になり、クライアントが CLI または .NET の場合は、ドライバーは自動クライアント・リルート処理の一部として失敗した SQL 操作をやり直します。接続が成功すると、エラーはアプリケーションに報告されず、トランザクションはロールバックされません。接続の失敗と、それ以後のリカバリーはアプリケーションから隠されます。
4. 自動クライアント・リルートがデータベースに再接続できない場合は、エラー SQL30081N がアプリケーションに戻されます。その後アプリケーションは、接続障害からのリカバリーを担当します (例えば、それ自身でデータベースへの接続を試行します) 。

自動クライアント・リルートは、クライアントが新規接続での接続障害を検出したときにも使用されます。しかし、この場合、再接続が成功すると、失敗したデータベース接続がリカバリーされたことを示すエラーはアプリケーションに戻されません。再接続が失敗すると、エラー SQL30081N が戻されます。

非 Java クライアントから Informix への接続のためのワークロード・バランシングの操作

IBM Informix への接続のためのワークロード・バランシング (トランザクション・レベルのワークロード・バランシングとも呼ばれる) は、トランザクションの開始時に高可用性クラスター内のサーバー間で作業を平衡化することにより、高可用性に寄与します。

クライアントが IBM Informix 接続マネージャーに接続し、ワークロード・バランシングが有効になっている場合に行われるステップの概要を以下に示します。

1. クライアントが接続マネージャーの IP アドレスを使用して最初の接続を確立すると、接続マネージャーはクラスター内のサーバーに関するサーバー・リストおよび接続の詳細 (IP アドレス、ポート、および重み) を戻します。

   サーバー・リストがクライアントによってキャッシュに入れられます。キャッシュに入れられたサーバー・リストのデフォルト存続期間は、30 秒です。
2. 新規トランザクションの開始時に、クライアントはキャッシュに入れられたサーバー・リストを読み取って、容量に余裕のあるサーバーを識別します。そして、その容量に余裕のあるサーバーに関連付けられたアイドル・トランスポートがないかどうかトランスポート・プールを調べます。 (アイドル・トランスポートとは、接続オブジェクトと関連付けられていないトランスポートのことです。)
   - アイドル・トランスポートが使用可能な場合、クライアントは接続オブジェクトをそのトランスポートと関連付けます。
   - ユーザー構成可能タイムアウトが経過した後、使用可能なアイドル・トランスポートがトランスポート・プールに存在せず、トランスポート・プールがその制限に達しているために新規トランスポートを割り振ることもできない場合、アプリケーションにエラーが戻されます。
3. トランザクションが実行されると、トランザクションはトランスポートに関連付けられたサーバーにアクセスします。
4. トランザクションが終了すると、クライアントは、トランスポートの再利用が接続オブジェクトに引き続き許可されているかどうかについてサーバーで検証します。
5. トランスポートの再利用が許可されている場合、サーバーは、接続オブジェクトの実行環境に適用される特殊レジスターに関する SET ステートメントのリストを返します。

   クライアントはこれらのステートメントをキャッシュに入れます。そして、接続オブジェクトが新規トランスポートと関連付けられた時に、実行環境を再構成するため、そのステートメントを再生します。
6. その後、クライアントが接続オブジェクトをトランスポートから切り離す必要があると判断した場合、それは切り離されます。
7. サーバー・リストのクライアント・コピーは、新規接続が行われるとき、または 30 秒ごと、またはユーザーが構成した間隔でリフレッシュされます。
8. 新規トランザクションにワークロード・バランシングが必要な場合、クライアントは上述のプロセスを使用して接続オブジェクトをトランスポートと関連付けます。

非 Java クライアントから Informix サーバーへの接続の高可用性のためのアプリケーション・プログラミング要件

自動クライアント・リルートのフェイルオーバーは、シームレスまたは非シームレスを選択することができます。Informix への接続のためのフェイルオーバーがシームレスでない場合は、フェイルオーバーの発生時に戻されるエラーについて説明するコードを追加する必要があります。

フェイルオーバーが非シームレスで、サーバーとの接続が再確立した場合、SQLCODE -4498 (Java クライアントの場合) または SQL30108N (非 Java クライアントの場合) がアプリケーションへ戻されます。現行のトランザクションで発生したすべての処理は、ロールバックされます。アプリケーションにおいて、以下のことが必要になります。

- エラーと共に戻される理由コードを確認します。障害が起きたデータ共有メンバーの特殊レジスター設定を、新規 (フェイルオーバーした) データ共有メンバーへ持ち越すかどうかを決定します。現行のものではない特殊レジスター値はすべてリセットします。
- 直前のトランザクションの間に発生したすべての SQL 操作を実行します。

IBM Informix データベースへの接続中にシームレス・フェイルオーバーが発生するようにするには、以下の条件を満たしている必要があります。

- アプリケーション・プログラミング言語が Java、CLI、または .NET である。
- トランザクションにおける接続ではない。これは、トランザクションにおける最初の SQL ステートメントが実行された時に、障害が発生したという場合です。
- データ・サーバーは、直前のトランザクションの終わりでトランスポートの再利用を許可する必要がある。
- すべてのグローバル・セッション・データがクローズ、またはドロップされている。
- オープン状態の保留カーソルがない。
- アプリケーションが CLI を使用している場合、そのアプリケーションは SQL ステートメントを再生するために直前で呼び出された API の履歴を維持するようドライバーに要求するアクションは実行できない。そのようなアクションの例として、実行時にデータを指定すること、コンパウンド SQL を実行すること、または配列入力を使用することがあります。
- アプリケーションがストアード・プロシージャーではない。
- 自動コミットが有効ではない。自動コミットが有効な場合でも、シームレスなフェイルオーバーは可能です。しかし、次の状況は問題を引き起こす可能性があります。データ・サーバーにおいて SQL 処理が正常に実行され、コミットされますが、コミット操作の確認応答がクライアントへ送信される前に、接続またはサーバーに障害が発生したとします。クライアントが接続を再確立するとき、直前にコミットされた SQL ステートメントを再生します。結果として、SQL ステートメントが 2 度実行されたことになります。この状況を回避するため、シームレス・フェイルオーバーを使用可能にする場合、自動コミットをオフにします。

加えて、アプリケーションの自動コミットを有効にしている場合、シームレス自動クライアント・リルートが成功しないおそれがあります。自動コミットを有効にしていると、ステートメントが複数回、実行およびコミットされる可能性があります。

非 Java クライアントから Informix への接続のためのクライアント・アフィニティー

クライアント・アフィニティーは、自動クライアント・リルート機能を提供するためのクライアント専用の方法です。

CLI、.NET、または Java (IBM Data Server Driver for JDBC and SQLJ Type 4 接続) を使用するアプリケーションでは、クライアント・アフィニティーを使用することができます。すべての再ルーティングは、ドライバーによって制御されます。

クライアント・アフィニティーでは、特定の 1 次サーバーへ接続する必要がある状態を対象としています。 1 次サーバーへの接続中に障害が発生した場合、代替サーバーへのフェイルオーバーに対して特定の順序を実行する必要があります。サーバー・フェイルオーバー機能を使用する自動クライアント・リルートがご使用の環境で機能しない場合に限り、自動クライアント・リルートにクライアント・アフィニティーを使用する必要があります。

クライアント・アフィニティーの構成の一部として、代替サーバーのリスト、および代替サーバーへの接続を試みる順序を指定します。クライアント・アフィニティーの使用中は、アプリケーションによって指定されるホスト名およびポート番号の代わりに、代替サーバーのリストに基づいて接続が確立されます。例えば、アプリケーションが server1 へ接続するよう指定している場合でも、構成プロセスが server2、server3、server1 という順序でサーバーへの試行をするよう指定しているならば、最初の接続は server1 の代わりに server2 へ行われます。

以下の条件が当てはまる場合、クライアント・アフィニティーを使用するフェイルオーバーはシームレスになります。

- トランザクションにおける接続ではない。これは、トランザクションにおける最初の SQL ステートメントが実行された時に、障害が発生したという場合です。
- サーバーに使用中のグローバル一時表がない。
- オープン状態の保留カーソルがない。

クライアント・アフィニティーを使用する時に、障害後に 1 次サーバーが稼働に戻る場合、トランザクション境界で代替サーバーから 1 次サーバーへ接続が戻るよう指定できます。このアクティビティーがフェイルバックです。

Informix 接続に関する非 Java クライアントのクライアント・アフィニティーの構成

CLI および .NET アプリケーションでクライアント・アフィニティーのサポートを使用可能にするには、db2dsdriver.cfg 構成ファイル内に値を設定し、クライアント・アフィニティーを使用することを示し、1 次サーバーと代替サーバーを指定します。

次の表に、CLI および .NET アプリケーションのクライアント・アフィニティーを使用可能にするための db2dsdriver.cfg ファイルにおける設定を示します。

表 32. CLI および .NET アプリケーションのクライアント・アフィニティーを使用可能にするための設定

db2dsdriver.cfg 構成ファイルの acr セクション内の要素	**値**
enableAcr パラメーター	true
maxAcrRetries パラメーター	自動クライアント・リルートの間に代替サーバーのリストにある各サーバーへの接続が試行される回数。有効な範囲は 0 から MAX_INT までです。値が 0 の場合、再試行回数は 1 回です。デフォルトは 3 です。
acrRetryInterval パラメーター	再試行と再試行の間で待機する秒数。有効な範囲は 0 から MAX_INT までです。デフォルトは、待機なし (0) です。
affinityFailbackInterval パラメーター	1 次サーバーへフェイルバックする最初のトランザクション境界の後で待機する秒数。 1 次サーバーへフェイルバックしたい場合に、この値を設定します。デフォルトは 0 です。つまり、1 次サーバーへのフェイルバックは試行されません。
alternateserverlist	クライアント・アフィニティーを介する自動クライアント・リルートで使用される各サーバーのホスト名およびポート番号を識別する <server> 要素。この要素のうち 1 つが、1 次サーバーを識別する必要があります。これらの要素があるからといって、自動クライアント・リルートがアクティブになるわけではありません。
affinitylist	serverorder 属性を持つ <list> 要素。serverorder 属性の値は、クライアント・アフィニティーを使用した自動クライアント・リルートの間に試行される順序で、サーバーのリストを明示します。<list> 要素にあるサーバーは、<alternateserverlist> の <server> 要素にも定義する必要があります。それぞれが異なるサーバー順序を持つ複数の <list> 要素を指定することもできます。 <affinitylist> 要素があるからといって、自動クライアント・リルートがアクティブになるわけではありません。

表 32. CLI および .NET アプリケーションのクライアント・アフィニティーを使用可能にするための設定 (続き)

db2dsdriver.cfg 構成ファイルの acr セクション内の要素	**値**
client_affinity	各クライアントに対して行うサーバー接続の試行順序を定義する <clientaffinitydefined> 要素、または <clientaffinityroundrobin> 要素。<clientaffinitydefined> 要素を含める場合、サーバー順序を定義する <list> 要素をそれぞれが指定する <client> 要素を定義することによって、サーバー順序を定義します。<clientaffinityroundrobin> 要素を含める場合も、<client> 要素を指定する必要があります。しかし、この場合の <client> 要素は、<list> 要素を指定しません。その代わりに、<client> 要素の順序がサーバー順序を定義します。データベースに接続するすべてのクライアントを <clientaffinitydefined> または <clientaffinityroundrobin> 要素内に指定する必要があります。あるクライアント・マシンに複数のネットワーク・インターフェース・カードがある場合、アフィニティー・リストを導出するために、CLI ドライバーによって自動的にクライアント・ホスト名が発見され、構成ファイル項目との突き合わせが行われます。CLI ドライバーはすべてのネットワーク・インターフェースを取得して、db2dsdriver.cfg 構成ファイルで使用可能なホスト名と突き合わせます。ドメイン名のないホスト名が db2dsdriver.cfg で指定されていると、CLI はデフォルトのドメインを使用して解決しようとし、発見したホスト名と突き合わせます。IP アドレスが db2dsdriver.cfg ファイルのクライアント・アフィニティー・セクションで定義されている場合、アフィニティー・リストを導出するために、CLI ドライバーが各 IP アドレスを発見して構成ファイル項目と (ホスト名を) 突き合わせます。
clientaffinitydefined	各クライアントに対する自動クライアント・リルートのサーバー順序を定義する <client> 要素。それぞれの <client> 要素は、クライアントを <affinitylist> 要素から <list> 要素へ関連付ける listname 属性を含んでいます。

表 32. CLI および .NET アプリケーションのクライアント・アフィニティーを使用可能にするための設定 (続き)

db2dsdriver.cfg 構成ファイルの acr セクション内の要素	値
clientaffinityroundrobin	<clientaffinityroundrobin> 要素内での順番が、自動クライアント・リルートに対して選択される最初のサーバーを定義する <client> 要素。各 <client> 要素には、索引があります。<clientaffinityroundrobin> 要素内で 1 番目の <client> 要素の索引は 0 になり、2 番目の <client> 要素の索引は 1 になり、以下同様に続きます。 <alternateserverlist> 要素内のサーバーの数が *n* 台、また <client> 要素の <clientaffinityroundrobin> 要素内の索引が *i* だとします。試行される最初のサーバーは、<alternateserverlist> 要素内の索引が *i* mod *n* になるサーバーです。次に試行されるサーバーは、<alternateserverlist> 要素内の索引が (*i* +1 mod *n*) になるサーバーです。以下同様に続きます。

以下の制約事項は、CLI または .NET クライアントのクライアント・アフィニティーの構成に適用されます。

- 1 つのクライアントに対して適格な代替サーバーの数が 128 より大きい場合、エラー SQL1042N が発生します。
- クライアント・アフィニティーが有効な時に、ワークロード・バランシングを有効にすることはできません。これは、enableWLB が true に設定されている場合に、client_affinity 要素を指定すると、エラー SQL5162N が発生するからです。
- <alternateserverlist>、<affinitylist>、または <client_affinity> 要素で必須属性が指定されていない場合、エラー SQL5163N が発生します。
- クライアント・アフィニティーが有効で、<alternateserverlist> 要素が空の場合、エラー SQL5164N が発生します。
- クライアント・アフィニティーが有効で、サーバーへの接続を試行しているクライアントのホスト名が、<client_affinity> サブグループ (<clientaffinitydefined> または <clientaffinityroundrobin>) のどちらにもない場合、または複数のサブグループにある場合、エラー SQL5164N が発生します。
- それぞれのクライアント・マシンに対して、<clientaffinitydefined> セクションまたは <clientaffinityroundrobin> セクションのいずれかに、1 つの項目だけが存在していなければなりません。db2dsdriver.cfg 内に、異なるホスト名によって同一のクライアント・マシンが指定されている複数の項目があると、エラー SQL5162N が生じます。

Informix 接続に関する非 Java クライアントのクライアント・アフィニティーを使用可能にする例

CLI アプリケーションまたは .NET アプリケーションで自動クライアント・リルートにおけるクライアント・アフィニティーを使用するには、その前に

db2dsdriver.cfg 構成ファイルの <acr> セクション内に要素を組み込んで、クライアント・アフィニティーを使用することを示し、1 次サーバーと代替サーバーを特定する必要があります。

以下の例では、フェイルバックしないフェイルオーバーのクライアント・アフィニティーを使用可能にする方法が示されます。

db2dsdriver 構成ファイルが以下のようになっているとします。

```
<database name="SAMPLE" host="v33ec065.svl.ibm.com" port="446">
  <acr>
    <parameter name="enableAcr" value="true"/>
    <parameter name="maxAcrRetries" value="1"/>
    <parameter name="acrRetryInterval" value="2"/>
    <alternateserverlist>
      <server name="server1"
              hostname="v33ec067.svl.ibm.com"
              port="446">
      </server>
      <server name="server2"
              hostname="v33ec066.svl.ibm.com"
              port="446">
      </server>
      <server name="server3"
              hostname="v33ec065.svl.ibm.com"
              port="446">
      </server>
    </alternateserverlist>
    <affinitylist>
      <list name="list1" serverorder="server1,server2,server3">
      </list>
      <list name="list2" serverorder="server3,server2,server1">
      </list>
    </affinitylist>
    <clientaffinitydefined>
    <!- this section has specific defined affinities -->
      <client name="client1"
              hostname="appsrv1.svl.ibm.com"
              listname="list2">
      </client>
      <client name="client2"
              hostname="appsrv2.svl.ibm.com"
              listname="list1">
      </client>
    </clientaffinitydefined>
    <clientaffinityroundrobin>
      <client name="client3" hostname="appsrv3.svl.ibm.com">
          <!- This entry is index 0. The number of servers is 3.
              0 mod 3 is 0, so the first that is tried
              during automatic client reroute is the server whose
              index in <alternateserverlist> is 0 (server1).
              The next server has index 1 mod 3, which is 1
              (server2). The final server has index 2 mod 3,
              which is 2 (server3). -->
      </client>
      <client name="client4" hostname="appsrv4.svl.ibm.com">
          <!- This entry is index 1. The number of servers is 3.
              1 mod 3 is 1, so the first that is tried
              during automatic client reroute is the server whose
              index in <alternateserverlist> is 1 (server2).
              The next server has index 2 mod 3, which is 2
              (server3). The final server has index 3 mod 3,
              which is 0 (server1). -->
```

```
      </client>
    </clientaffinityroundrobin>
  </acr>
</database>
```

ホスト名 appsrv4.svl.ibm.com (client4) というクライアントから v33ec065.svl.ibm.com:446 によって識別されるサーバーへの接続中に、通信障害が発生したとします。以下のステップは、クライアント・アフィニティーを使用した自動クライアント・リルートに起こるプロセスを示します。

1. ドライバーは v33ec066.svl.ibm.com:446 (server2) への接続を試みます。
2. v33ec066.svl.ibm.com:446 への接続は失敗します。
3. ドライバーは 2 秒間待機します。
4. ドライバーは v33ec065.svl.ibm.com:446 (server3) への接続を試みます。
5. v33ec065.svl.ibm.com:446 への接続は失敗します。
6. ドライバーは 2 秒間待機します。
7. ドライバーは v33ec067.svl.ibm.com (server1) への接続を試みます。
8. v33ec067.svl.ibm.com への接続は失敗します。
9. ドライバーは 2 秒間待機します。
10. ドライバーはエラー・コード SQL30081N を返します。

以下の例では、フェイルバックするフェイルオーバーのクライアント・アフィニティーを使用可能にする方法が示されます。

db2dsdriver 構成ファイルが以下のようになっているとします。

```
<database name="SAMPLE" host="v33ec065.svl.ibm.com" port="446">
  <acr>
    <parameter name="enableAcr" value="true"/>
    <parameter name="maxAcrRetries" value="1"/>
    <parameter name="acrRetryInterval" value="2"/>
    <parameter name="affinityFailbackInterval" value="300"/>
    <alternateserverlist>
      <server name="server1"
              hostname="v33ec067.svl.ibm.com"
              port="446">
      </server>
      <server name="server2"
              hostname="v33ec066.svl.ibm.com"
              port="446">
      </server>
      <server name="server3"
              hostname="v33ec065.svl.ibm.com"
              port="446">
      </server>
    </alternateserverlist>
    <affinitylist>
      <list name="list1" serverorder="server1,server2,server3">
      </list>
      <list name="list2" serverorder="server3,server2,server1">
      </list>
    </affinitylist>
    <clientaffinitydefined>
    <!- this section has specific defined affinities -->
      <client name="client1"
              hostname="appsrv1.svl.ibm.com"
              listname="list2">
      </client>
      <client name="client2"
```

```
            hostname="appsrv2.svl.ibm.com"
            listname="list1">
      </client>
    </clientaffinitydefined>
    <clientaffinityroundrobin>
      <client name="client3" hostname="appsrv3.svl.ibm.com">
          <!- This entry is index 0. The number of servers is 3.
              0 mod 3 is 0, so the first that is tried
              during automatic client reroute is the server whose
              index in <alternateserverlist> is 0 (server1).
              The next server has index 1 mod 3, which is 1
              (server2). The final server has index 2 mod 3,
              which is 2 (server3). -->
      </client>
      <client name="client4" hostname="appsrv4.svl.ibm.com">
          <!- This entry is index 1. The number of servers is 3.
              1 mod 3 is 1, so the first that is tried
              during automatic client reroute is the server whose
              index in <alternateserverlist> is 1 (server2).
              The next server has index 2 mod 3, which is 2
              (server3). The final server has index 3 mod 3,
              which is 0 (server1). -->
      </client>
    </clientaffinityroundrobin>
  </acr>
</database>
```

クライアント appsrv2.svl.ibm.com (client2) から v33ec065.svl.ibm.com:446 へ接続された後に、データベース管理者が v33ec065.svl.ibm.com:446 によって識別されるサーバーを保守のため停止するとします。以下は、代替サーバーへフェイルオーバーし、保守が完了した後、1 次サーバーへフェイルバックするステップを示します。

1. クライアント appsrv1.svl.ibm.com に代わって、ドライバーは v33ec065.svl.ibm.com:446 へ正常に接続します。
2. データベース管理者は v33ec065.svl.ibm.com:446 を停止します。
3. アプリケーションは、その接続で処理を試みます。
4. ドライバーは v33ec066.svl.ibm.com:446 へ正常にフェイルオーバーします。
5. 200 秒後、処理がコミットされます。
6. ドライバーは、フェイルバック・インターバル (300 秒) が経過したかどうか確認します。経過していない場合、フェイルバックは発生しません。
7. アプリケーションは v33ec066.svl.ibm.com:446 への接続で処理を続けます。
8. 105 秒後、処理がコミットされます。
9. ドライバーは、フェイルバック・インターバル (300 秒) が経過したかどうか確認します。経過していると、v33ec065.svl.ibm.com:446 へのフェイルバックが発生します。

DB2 for z/OS サーバーへの接続の高可用性のための非 Java クライアント・サポート

DB2 for z/OS サーバー上の Sysplex ワークロード・バランシング機能は、直接データ共有グループに接続するクライアント・アプリケーションに高可用性を提供します。Sysplex ワークロード・バランシング機能は、ワークロード・バランシングと自動クライアント・リルートの機能を提供します。このサポートは、Java クライアン

ト (JDBC、SQLJ、または pureQuery) または非 Java クライアント (ODBC、 CLI、 .NET、 OLE DB、 PHP、 Ruby、または組み込み SQL) を使用するアプリケーションで使用できます。

Sysplex とは、カスタマー・ワークロードを処理するマルチシステムのハードウェア・コンポーネントおよびソフトウェア・サービスを介して相互に通信および共同で作業する z/OS システムの集合のことです。 Sysplex における z/OS システムの DB2 for z/OSサブシステムを構成し、データ共有グループを形成できます。データ共有によって、複数の DB2 for z/OSサブシステムで稼働する複数のアプリケーションが、並行して同じデータ集合から読み取り、また書き込むことができます。 1 つ以上のカップリング・ファシリティーによって、データ共用グループに対する高速キャッシングおよびロック処理が提供されます。 Sysplex をワークロード・マネージャー (WLM)、動的仮想 IP アドレス (DVIPA)、および Sysplex ディストリビューターと共に使用することによって、クライアントは、TCP/IP を介してネットワーク回復力のある方法で DB2 for z/OSサブシステムにアクセスし、またデータ共用グループ内のメンバー間でアプリケーションのトランザクションをバランスの取れた方法で分散させることがきます。

これらの機能の中心となるのは、データ共有グループが接続境界およびトランザクション境界 (オプション) で戻すサーバー・リストです。このリストには、データ共有グループの各メンバーの IP アドレスおよび WLM の重みが含まれます。この情報を使用して、クライアントは、トランザクションをバランスの取れた方法で分散したり、通信障害が生じた場合に使用するメンバーを識別したりすることができます。

サーバー・リストは、初めて正常に DB2 for z/OS データ・サーバーに接続した時点で戻されます。クライアントはサーバー・リストを受け取ると、そのサーバー・リストにある情報に基づいてデータ共有グループ・メンバーに直接アクセスします。

DB2 for z/OS では、クライアントがデータ共有グループにアクセスするいくつかの方法があります。データ共有グループとの通信をセットアップするアクセス方式によって、Sysplex ワークロード・バランシングが可能かどうか決まります。以下の表は、アクセス方式の一覧と Sysplex ワークロード・バランシングが可能かどうかを示しています。

表 33. データ共有のアクセス方式および Sysplex ワークロード・バランシング

データ共有のアクセス方式[1]	説明	Sysplex ワークロード・バランシングは可能か?
グループ・アクセス	リクエスターは、グループの動的仮想 IP アドレス (DVIPA) を使用して、DB2 for z/OS ロケーションへの最初の接続を行います。少なくとも 1 つのメンバーが開始されていれば、グループ IP アドレスおよび SQL ポートを使用するデータ共有グループへの接続は常に成功します。データ共有グループによって戻されるサーバー・リストには、以下のものが含まれます。 • 現在アクティブで、処理を実行できるメンバーのリスト • 各メンバーの WLM の重み z/OS Sysplex ディストリビューターを使用して、グループ IP アドレスを構成します。Sysplex 外のクライアントには、DB2 ロケーションを表す単一 IP アドレスを Sysplex ディストリビューターが提供します。フォールト・トレランスの提供に加え、接続ロード・バランシングを提供するように Sysplex ディストリビューターを構成できます。	はい
メンバー固有アクセス	リクエスターはロケーション別名を使用して、別名で表されるメンバーの 1 つに最初の接続を行います。少なくとも 1 つのメンバーが開始されていれば、グループ IP アドレスおよび別名 SQL ポートを使用するデータ共有グループへの接続は常に成功します。データ共有グループによって戻されるサーバー・リストには、以下のものが含まれます。 • 現在アクティブで、処理を実行でき、かつ別名として構成されたメンバーのリスト • 各メンバーの WLM の重み リクエスターはこの情報を使用して、ロケーション別名にも関連付けられた最大能力のメンバー (複数可) に接続します。メンバー固有アクセスは、データ共有グループのメンバーのサブセット内で Sysplex ワークロード・バランシングをリクエスターが利用する必要がある場合に使用します。	はい
単一メンバー・アクセス	単一メンバー・アクセスは、リクエスターがデータ共有グループの 1 つのメンバーのみにアクセスする必要がある場合に使用します。単一メンバー・アクセスでは、接続にメンバー固有の IP アドレスを使用します。	いいえ

表 33. データ共有のアクセス方式および *Sysplex* ワークロード・バランシング (続き)

データ共有のアクセス方式[1]	説明	Sysplex ワークロード・バランシングは可能か?

注:

1. データ共有のアクセス方式について詳しくは、http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2z9.doc.dshare/src/tpc/db2z_tcpipaccessmethods.htmを参照してください。

Sysplex ワークロード・バランシングには自動クライアント・リルートが含まれています。 自動クライアント・リルートのサポートにより、クライアントは、Sysplex の任意の使用可能なメンバーを介してデータベースへの再接続を試行することにより、障害からリカバリーできます。別のメンバーへ再接続することをフェイルオーバーと呼びます。

バージョン 9.7 フィックスパック 5 およびそれ以降のフィックスパックのリリースにおいて、代替グループは、現在のグループへの接続を再確立できなかった場合の、自動クライアント・リルートに対する追加のフェイルオーバー・メカニズムです。*グループ* は、Sysplex データ共有環境で作成されるデータベースです。アプリケーションが明示的に接続しているデータベースは、*1 次グループ* と呼ばれます。

Java、CLI、または .NET クライアント・アプリケーションでは、自動クライアント・リルートのフェイルオーバーについて、*シームレス*または*非シームレス*を選択することができます。シームレス・フェイルオーバーでは、アプリケーションが代替サーバーまたは代替グループへ正常に再接続するとき、サーバーがアプリケーションにエラーを戻すことはありません。

クライアントの DB2 Connect サーバーを使用した高可用性のための直接接続のサポート: 高可用性のためのクライアントの直接接続のサポートには、DB2 Connect ライセンスが必要ですが、DB2 Connect サーバーは必要ありません。クライアントは、DB2 for z/OS に直接接続します。DB2 Connect サーバーを使用しているが、クライアントの高可用性のために環境を設定している場合、DB2 for z/OS への直接接続が提供するフィーチャーの一部を使用できません。例えば、シスプレックスによって提供されるトランザクション・レベルのワークロード・バランシングや自動クライアント・リルート機能は使用できません。

クライアント・アフィニティーを使用しないでください。 DB2 for z/OSへ直接接続する高可用性ソリューションとしてクライアント・アフィニティーを使用することはできません。データ共有グループのすべてのメンバーは並行してデータにアクセスできるため、クライアント・アフィニティーは DB2 for z/OS データ共有環境には適用されません。データ共有環境におけるクライアント・アフィニティーの大きな欠点は、データ共有グループ・メンバーに障害が起きてフェイルオーバーが発生した場合に、失敗したメンバーが、フェイルオーバーが発生したメンバー上のトランザクションに深刻な影響を及ぼすロックを保持してしまう危険があることです。

非 Java クライアント用の Sysplex ワークロード・バランシングと自動クライアント・リルートの構成

Sysplex ワークロード・バランシングと自動クライアント・リルート (ACR) を使用するために、直接 DB2 for z/OS に接続している Java アプリケーション以外のクライアント・アプリケーションを構成するには、db2dsdriver.cfg ファイル内のキーワード値を設定します。

これらのキーワード値は、データ共有グループ (グループ・アクセスの場合) またはデータ共有グループのサブセット (メンバー固有アクセスの場合) を表すアドレスへの接続を指定し、Sysplex ワークロード・バランシングおよび自動クライアント・リルートを使用可能にします。

Sysplex ワークロード・バランシングと自動クライアント・リルートは必ず一緒に構成してください。Sysplex ワークロード・バランシングを使用するようにクライアントを構成すると、デフォルトでは自動クライアント・リルートも使用可能になります。したがって、自動クライアント・リルートに関連したキーワード値を変更する必要があるのは、自動クライアント・リルートの操作を調整する場合のみです。

Java クライアント以外のクライアントの場合、Sysplex ワークロード・バランシングを活用するには、以下にリストされたクライアントまたはクライアント・パッケージのいずれかを使用してください。

- IBM Data Server Client
- IBM Data Server Runtime Client
- IBM Data Server Driver Package
- IBM Data Server Driver for ODBC and CLI

重要: Sysplex フィーチャーを使用して、DB2 for z/OS データ共有グループへの直接接続を確立するには、DB2 Connect サーバー製品のインストール済み環境があるか、または DB2 インストール・パスのライセンス・ディレクトリー内に DB2 Connect サーバー・ライセンス・ファイルがある必要があります。

表 1 では、Java アプリケーション以外のアプリケーションで Sysplex ワークロード・バランシングを使用可能にするために必要な、基本的な構成設定について説明しています。

表 34. Java アプリケーション以外のアプリケーションで Sysplex ワークロード・バランシングを使用可能にするための基本設定

データ共有アクセス方式	クライアントの設定	値
グループ・アクセス	db2dsdriver.cfg 構成ファイルの <wlb> セクション内の enableWLB 要素	true
	接続アドレス:	
	データベース・ホスト[1]	データ共有グループのグループ IP アドレスまたはドメイン・ネーム
	データベース・ポート[1]	DB2 ロケーション用の SQL ポート番号
	データベース名[1]	インストール中に定義される DB2 ロケーション名

表 34. Java アプリケーション以外のアプリケーションで Sysplex ワークロード・バランシングを使用可能にするための基本設定 (続き)

データ共有アクセス方式	クライアントの設定	値
メンバー固有アクセス	db2dsdriver.cfg 構成ファイルの <wlb> セクション内の enableWLB 要素	true
	接続アドレス:	
	データベース・ホスト[1]	データ共有グループのグループ IP アドレスまたはドメイン・ネーム
	データベース・ポート[1]	DB2 ロケーション別名用のポート番号
	データベース名[1]	データ共有グループのメンバーのサブセットを表す、DB2 ロケーション別名の名前

表 34. Java アプリケーション以外のアプリケーションで Sysplex ワークロード・バランシングを使用可能にするための基本設定 (続き)

データ共有 アクセス方式	クライアントの設定	値

注:

1. 使用する DB2 製品およびドライバーに応じて、接続情報は以下のソースのいずれかに定義されている可能性があります。
   - IBM Data Server Clientに CLI またはオープン・ソース・アプリケーションが関係するシナリオでは、以下のソースから接続情報を取得できます。
     - ホスト、ポート、およびデータベース情報がアプリケーションの接続ストリング内で提供される場合、CLI ドライバーはその情報を使用して接続を確立します。
     - データベース・カタログからの情報。
     - ホストおよびポート情報がアプリケーションの接続ストリングにもデータベース・カタログにも提供されていない場合、ドライバーは必要な情報を db2cli.ini ファイルで検索します。CLI ドライバーはこの db2cli.ini ファイル内に提供されている情報を使用して、接続を確立します。
     - ホストおよびポート情報がアプリケーションの接続ストリング、データベース・カタログ、または db2cli.ini ファイルのいずれにも提供されていない場合、CLI ドライバーは db2dsdriver.cfg 構成ファイル内の情報を使用します。
   - IBM データ・サーバー・ドライバーに CLI またはオープン・ソース・アプリケーションが関係するシナリオでは、以下のソースから接続情報を取得できます。
     - ホスト、ポート、およびデータベース情報がアプリケーションの接続ストリング内で提供される場合、CLI ドライバーはその情報を使用して接続を確立します。
     - ホストとポートの情報がアプリケーションの接続ストリング内で提供されない場合、ドライバーは db2cli.ini ファイル内で必要な情報を検索し、CLI ドライバーは db2cli.ini ファイル内で提供されている情報を使用して接続を確立します。
     - ホストとポートの情報がアプリケーションの接続ストリングと db2cli.ini ファイルのどちらでも提供されない場合、CLI ドライバーは db2dsdriver.cfg 構成ファイル内の情報を使用します。
   - IBM Data Server Clientに .NET アプリケーションが関係するシナリオでは、以下のソースから接続情報を取得できます。
     - ホスト、ポート、およびデータベース情報がアプリケーションの接続ストリング内で提供される場合、.NET データ・プロバイダーはその情報を使用して接続を確立します。
     - ホスト、ポート、およびデータベース情報が .NET オブジェクト・プロパティーから提供される場合、.NET データ・プロバイダーはその情報を使用して接続を確立します。
     - データベース・カタログからの情報。
     - ホストおよびポート情報がアプリケーションの接続ストリングにもデータベース・カタログにも提供されていない場合、.NET データ・プロバイダーは db2dsdriver.cfg 構成ファイル内の情報を使用します。
   - IBM データ・サーバー・ドライバーに .NET アプリケーションが関係するシナリオでは、以下のソースから接続情報を取得できます。
     - ホスト、ポート、およびデータベース情報がアプリケーションの接続ストリング内で提供される場合、.NET データ・プロバイダーはその情報を使用して接続を確立します。
     - ホスト、ポート、およびデータベース情報が .NET オブジェクト・プロパティーから提供される場合、.NET データ・プロバイダーはその情報を使用して接続を確立します。
     - ホストおよびポート情報がアプリケーションの接続ストリングにもデータベース・カタログにも提供されていない場合、.NET データ・プロバイダーは db2dsdriver.cfg 構成ファイル内の情報を使用します。

Sysplex ワークロード・バランシングを適切に調整したい場合は、追加のプロパティーを使用できます。表 2 では、Java アプリケーション以外のアプリケーションの追加のプロパティーがリストされています。

表 35. Java アプリケーション以外のアプリケーションから DB2 for zOS への直接接続のために Sysplex ワークロード・バランシングを適切に調整するためのプロパティー

db2dsdriver.cfg 構成ファイル内の要素	db2dsdriver.cfg ファイル内のセクション	説明
maxTransportIdleTime	<wlb>	アイドル・トランスポートがドロップされる前の最大経過時間を秒数で指定します。デフォルトは 600 です。サポートされる最小値は 0 です。
maxTransportWaitTime	<wlb>	トランスポートが使用可能になるのをクライアントが待機する秒数を指定します。デフォルト値は -1 で、無制限の待機時間を指定します。サポートされる最小の待機時間は 0 です。
maxTransports	<wlb>	リクエスターがデータ共有グループに対して行うことができる接続の最大数を指定します。
maxRefreshInterval	<wlb>	サーバー・リストがリフレッシュされる前の最大経過時間を秒数で指定します。デフォルトは 30 です。サポートされる最小値は 0 です。

Java クライアント以外のクライアントの場合、自動クライアント・リルート機能はデフォルトで使用可能になります。サーバーとの接続が最初に正常に確立された時点で、クライアントは接続相手のサーバーからすべての利用可能な代替サーバーのリストを取得します。クライアントはメモリーにリストを格納し、代替サーバーを記載したリストが含まれるローカル・キャッシュ・ファイルの srvrlst.xml も作成します。新しい接続が行われて、サーバーから取得した新しいリストがクライアントの srvrlst.xml ファイルの内容と違っているときには、常にこのファイルはリフレッシュされます。

クライアントが srvrlst.xml ファイルを使用して代替サーバーを探すときに、レコードを **db2diag** ログ・ファイルに書き込みます。このログをモニターして、初期サーバー接続が失敗する頻度を判別できます。

db2dsdriver.cfg ファイル内の構成キーワードまたはレジストリー変数を設定して、自動クライアント・リルートの動作を詳細化できます。db2dsdriver.cfg ファイル内の構成キーワードを使用して、自動クライアント・リルートを制御できます。クライアント・アフィニティーが使用可能でない場合について、キーワードが説明されています。

db2dsdriver.cfg ファイルを変更すると、CLI アプリケーションは、SQLReloadConfig 関数を呼び出して、**<acr>** セクション内のすべての代替サーバーの項目を妥当性検査できるようになります。

表 36. 自動クライアント・リルートの動作を制御するための設定

db2dsdriver 構成ファイルの <acr> セクション内の要素	値
enableAcr パラメーター	自動クライアント・リルートが有効かどうかを指定します。デフォルトは true です。DB2 for z/OS データ共有グループに対する自動クライアント・リルートを使用可能にするときに、**enableWLB** パラメーターが有効な場合にのみ、このパラメーターを使用可能にする必要があります。
enableSeamlessAcr パラメーター	シームレス・フェイルオーバーが可能かどうかを指定します。**enableAcr** が true に設定されている場合、**enableSeamlessAcr** のデフォルトは true です。**enableSeamlessACR** パラメーターは、グループまたはクラスター内のメンバーにのみ適用されます。DB2 for z/OS データ共有グループに対する自動クライアント・リルートを使用可能にするときに、**enableWLB** パラメーターが有効でアプリケーションが SQL30108N 例外を処理できる場合は、このパラメーターを使用可能にする必要があります。
acrRetryInterval パラメーター	連続して行われる接続試行間に待機する秒数。レジストリー変数 **DB2_CONNRETRIES_INTERVAL** はこの値をオーバーライドします。有効な範囲は 0 から MAX_INT までです。**DB2_CONNRETRIES_INTERVAL** が設定されていない場合、デフォルトは待機なし (0) です。DB2 for z/OS データ共有グループに対する自動クライアント・リルートを使用可能にする際には、デフォルト値の待機なしをお勧めします。
maxAcrRetries パラメーター	自動クライアント・リルートのための接続試行の最大数。レジストリー変数 **DB2_MAX_CLIENT_CONNRETRIES** はこの値をオーバーライドします。**DB2_MAX_CLIENT_CONNRETRIES** が設定されていない場合、デフォルトでは接続の試行が 10 分間行われます。代替グループが定義されている場合、接続はデフォルトで 2 分間試行されます。値 0 は、再接続の試行が 1 回行われることを意味します。DB2 for z/OS データ共有グループに対する自動クライアント・リルートを使用可能にする際には、maxAcrRetries を 5 以下に設定することをお勧めします。

表 3 のレジストリー変数は、自動クライアント・リルートの再試行動作を制御します。

表 37. 自動クライアント・リルートの再試行動作を制御するレジストリー変数

レジストリー変数	値
DB2_MAX_CLIENT_CONNRETRIES	自動クライアント・リルートのための接続再試行の最大数。**DB2_CONNRETRIES_INTERVAL** 変数が設定されている場合、デフォルトは 30 です。
DB2_CONNRETRIES_INTERVAL	連続して行われる接続再試行の間の秒数。**DB2_MAX_CLIENT_CONNRETRIES** 変数が設定されている場合、デフォルトは 10 です。

DB2 for z/OS データ共有グループに対する自動クライアント・リルートを使用可能にする際には、**maxAcrRetries** キーワードを設定してください。**DB2_MAX_CLIENT_CONNRETRIES** と **DB2_CONNRETRIES_INTERVAL** を設定せず、**maxAcrRetries** と **acrRetryInterval** も設定しないと、自動クライアント・リルートは、最大 10 分間 z/OS グループへの接続を試行し、試行の間に待機しません。

CLI、OLE DB、および ADO.NET アプリケーションの場合、**ConnectionTimeout**、**MemberConnectTimeout**、および **tcipipConnectionTimeout** の 3 つの接続タイムアウト・キーワードがあります。**tcipipConnectionTimeout** パラメーターはネットワーク層で設定され、すべての DB2 接続に影響します。このキーワードと自動クライアント・リルートを併用しないでください。**ConnectionTimeout** キーワードは、DB2 for z/OS データ共有グループへの接続が確立されるのをクライアント・アプリケーションが待機する秒数を指定します。**MemberConnectTimeout** キーワードは、サーバー・リスト内の次の IP アドレスにルーティングする前にクライアント・アプリケーションが待機する秒数を指定します。DB2 for z/OS データ共有グループへの接続に関する自動クライアント・リルートを使用可能にする際には、**MemberConnectTimeout** キーワードを使用して再ルーティング前の待機時間を管理することをお勧めします。デフォルトの **MemberConnectTimeout** 値は 1 秒です。このタイムアウトにより、メンバーが接続を受諾するのを待機する時間の長さが決まります。ほとんどの場合、デフォルト値が適切です。しかし、低速のネットワークで実行している場合は、値が大きくなるよう調整して、不要なネットワーク・タイムアウトが生じないようにする必要があります。

Sysplex ワークロード・バランシングを使用する必要があり、自動クライアント・リルート処理に関して戻されるエラーをアプリケーションが処理できない場合は、db2dsdriver.cfg 構成ファイル内に以下のパラメーターを設定します。

表 38. Java アプリケーション以外のアプリケーションから DB2 for z/OS への接続のための Sysplex ワークロード・バランシングのみを使用可能にするためのプロパティー

db2dsdriver.cfg 構成ファイル内の要素	db2dsdriver.cfg ファイル内のセクション	説明	設定する値
connectionLevelLoadBalancing	<database>	接続レベルのロード・バランシングを有効にするかどうかを指定します。デフォルトでは、**enableWLB** 構成パラメーターが true に設定されている場合、**connectionLevelLoadBalancing** も true に設定されます。このように設定されていない場合、デフォルトの **connectionLevelLoadBalancing** 値は false です。デフォルトを保持することをお勧めします。	true
enableWLB	<wlb>	ワークロード・バランシングが使用可能かどうかを指定します。デフォルトで、false に設定されます。	true

表 38. *Java アプリケーション以外のアプリケーションから DB2 for z/OS への接続のための Sysplex ワークロード・バランシングのみを使用可能にするためのプロパティー (続き)*

db2dsdriver.cfg 構成ファイル内の要素	db2dsdriver.cfg ファイル内のセクション	説明	設定する値
enableAcr	<acr>	自動クライアント・リルートが使用可能かどうかを指定します。CLI または .NET アプリケーションの場合、自動クライアント・リルートを使用可能にすると、シームレス・フェイルオーバーが自動的に使用可能になります。enableWLB が「true」の場合、デフォルトで、enableAcr も同様に「true」に設定されます。そうでない場合、デフォルトは「false」です。アプリケーションがシームレス・フェイルオーバー例外 (SQL30108N) を処理できない場合を除き、デフォルトを保持することをお勧めします。処理できない場合は、enableAcr を「false」に設定できます。	true
enableSeamlessAcr	<acr>	シームレス・フェイルオーバーが使用可能かどうかを指定します。Java アプリケーション以外のアプリケーション間で、シームレス・フェイルオーバーは CLI または .NET アプリケーションだけにサポートされます。デフォルトでは、**enableAcr** 構成パラメーターの値と同じ値に設定されます。	true
enableAlternateGroupSeamlessACR	<acr>	グループ間のシームレス・フェイルオーバーまたは非シームレス・フェイルオーバーの動作を指定します。デフォルトは false です。このパラメーターを true に設定するには、**enableSeamlessACR** 構成パラメーターを true に設定する必要もあります。このパラメーターを true に設定しても、**enableSeamlessACR** の設定に影響はありません。*alternategroup* セクションのサーバーへの接続が正常に確立されている場合、シームレスまたは非シームレスの動作の規則が適用されたままになります。	true

非 Java クライアント・アプリケーションでの DB2 for z/OS Sysplex ワークロード・バランシングと自動クライアント・リルートの使用可能化の例

直接 DB2 for z/OS サーバーに接続している Java アプリケーションを除いた、その他のアプリケーションで Sysplex ワークロード・バランシングと自動クライアント・リルートを使用するには、その前に db2dsdriver.cfg 構成ファイルを適切な設定にして更新し、データ共有グループに接続する必要があります。

クライアントをセットアップするには、その前に以下のサーバー・ソフトウェアを構成する必要があります。

- WLM for z/OS

  ワークロード・バランシングが効率的に作動するには、DB2 作業を種別分類する必要があります。種別は、各トランザクション内の最初の非 SET SQL ステートメントに適用されます。以下の領域間で、作業を種別分類する必要があります。
  - 許可 ID
  - クライアント情報プロパティー
  - ストアード・プロシージャー名

    ストアード・プロシージャー名が種別に使用されるのは、トランザクション内のクライアントによって発行される最初のステートメントが SQL CALL ステートメントである場合のみです。

  種別属性の完全なリストについては、URL http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2z10.doc.perf/src/tpc/db2z_classificationattributes.htm の、種別属性に関する情報を参照してください。
- データ共有のためにセットアップされる DB2 for z/OS

サーバーとの接続が最初に正常に確立された時点で、クライアントは接続相手のサーバーからすべての利用可能な代替サーバーのリストを取得します。クライアントはメモリーにリストを格納し、代替サーバーを記載したリストが含まれるローカル・キャッシュ・ファイルの srvrlst.xml も作成します。新しい接続が行われて、サーバーから取得した新しいリストがクライアントの srvrlst.xml ファイルの内容と違っているときには、常にこのファイルはリフレッシュされます。

以下の項目を変更して、デフォルトの自動クライアント・リルート・フィーチャーをさらに細かく調整できます。

自動クライアント・リルートの特性	db2dsdriver.cfg 構成キーワード	望ましい値
代替サーバーへの接続を試行する回数	**maxAcrRetries**	< 6
試行の間に待機する秒数	**acrRetryInterval**	0 (デフォルト)

以下の例は、Sysplex と自動クライアント・リルートの高可用性サポートを活用するように、Java アプリケーション以外のクライアント・アプリケーションをセットアップする方法を示しています。

1. Sysplex サポートと自動クライアント・リルートの基本設定を使用して、db2dsdriver.cfg ファイルを作成します。enableWLB および enableAcr を true に設定すると、Sysplex ワークロード・バランシングと自動クライアント・リルート機能が使用可能になります。

```
<configuration>
  <dsncollection>
    <dsn alias="DSGROUP1" name="DSGROUP1"
      host="db2a.sysplex1.ibm.com" port="446">
    </dsn>
  </dsncollection>
  <database name="DSGROUP1" host="db2a.sysplex1.ibm.com" port="446">
   <!-- database-specific parameters -->
   <wlb>
      <!-- Enable Sysplex workload balancing to get
      automatic client reroute
      functionality -->
      <parameter name="enableWLB" value="true" />
      <!-- maxTransports represents the maximum number of transports -->
      <parameter name="maxTransports" value="80" />
   </wlb>
   <acr>
      <parameter name="enableAcr" value="true">
      </parameter>
      <parameter name="maxAcrRetries" value="5">
      </parameter>
      <parameter name="acrRetryInterval" value="0">
      </parameter>
   </acr>
  </database>
</configuration>
```

2. データベース名 DSGROUP1 がグループ・アクセス用にセットアップされるデータ共有グループを表すと仮定します。CLI アプリケーションで、以下のようなコードを使用してデータ共有グループに接続します。

```
...
   SQLHDBC         hDbc    = SQL_NULL_HDBC;
   SQLRETURN       rc      = SQL_SUCCESS;
   SQLINTEGER      RETCODE = 0;
   char            *ConnStrIn =
                   "DSN=DSGROUP1;PWD=mypass";
```

```
                /* dsn matches the database name in the configuration file */
   char            ConnStrOut [200];
   SQLSMALLINT     cbConnStrOut;
   int             i;
   char            *token;
...
  /*************************************************************/
  /* Invoke SQLDriverConnect                                   */
  /*************************************************************/
   RETCODE = SQLDriverConnect (hDbc                 ,
                               NULL                 ,
                               (SQLCHAR *)ConnStrIn ,
                               strlen(ConnStrIn)    ,
                               (SQLCHAR *)ConnStrOut,
                               sizeof(ConnStrOut)   ,
                               &cbConnStrOut        ,
                               SQL_DRIVER_NOPROMPT);
...
```

非 Java クライアントから DB2 for z/OS サーバーへの接続のための Sysplex ワークロード・バランシングの操作

DB2 for z/OS への接続のための Sysplex ワークロード・バランシング (トランザクション・レベルのワークロード・バランシングとも呼ばれる) は、トランザクションの開始時にデータ共用グループのメンバー間で作業を平衡化することにより、高可用性に寄与します。

クライアントが DB2 for z/OS Sysplex に接続し、Sysplex ワークロード・バランシングが有効になっている場合に行われるステップの概要を以下に示します。

1. クライアントがグループ IP アドレスと呼ばれるシスプレックス全体にわたる IP アドレスを使用して接続を初めて確立する時、または他の接続オブジェクトによって接続が再利用される時に、サーバーはメンバー・ワークロード分散情報を戻します。

   キャッシュに入れられたサーバー・リストのデフォルト存続期間は、30 秒です。
2. 新規トランザクションの開始時に、クライアントはキャッシュに入れられたサーバー・リストを読み取って、容量に余裕のあるメンバーを識別します。そして、その容量に余裕のあるメンバーに関連付けられたアイドル・トランスポートがないかどうかトランスポート・プールを調べます。 (アイドル・トランスポートとは、接続オブジェクトと関連付けられていないトランスポートのことです。)
   - アイドル・トランスポートが使用可能な場合、クライアントは接続オブジェクトをそのトランスポートと関連付けます。
   - ユーザー構成可能タイムアウトが経過した後、使用可能なアイドル・トランスポートがトランスポート・プールに存在せず、トランスポート・プールがその制限に達しているために新規トランスポートを割り振ることもできない場合、アプリケーションにエラーが戻されます。
3. トランザクションが実行されると、トランザクションはトランスポートに関連付けられたメンバーにアクセスします。
4. トランザクションが終了すると、クライアントは、トランスポートの再利用が接続オブジェクトに引き続き許可されているかどうかについてサーバーで検証します。

5. トランスポートの再利用が許可されている場合、サーバーは、接続オブジェクトの実行環境に適用される特殊レジスターに関する SET ステートメントのリストを返します。

   クライアントはこれらのステートメントをキャッシュに入れます。そして、接続オブジェクトが新規トランスポートと関連付けられた時に、実行環境を再構成するため、そのステートメントを再生します。
6. その後、接続オブジェクトがトランスポートから切り離されます。
7. 新規で接続が確立されたとき、または 30 秒ごとに、サーバー・リストのクライアント・コピーがリフレッシュされます。
8. 新規トランザクションにワークロード・バランシングが必要な場合、クライアントは同じプロセスを使用して接続オブジェクトをトランスポートと関連付けます。

非 Java クライアントから DB2 for z/OS サーバーへの接続のための自動クライアント・リルートの操作

自動クライアント・リルート・サポートは、IBM データ・サーバー・クライアントが DB2 for z/OS Sysplex のメンバーへの接続を失う際に、フェイルオーバー・サポートを提供します。自動クライアント・リルートにより、クライアントは、Sysplex の任意の使用可能なメンバーを介してデータベースへの再接続を試行することにより、障害からリカバリーできます。

Sysplex ワークロード・バランシングが有効な場合、自動クライアント・リルートはデフォルトで使用可能になります。

DB2 Connect ライセンスがある IBM データ・サーバー・クライアントで、自動クライアント・リルートのクライアント・サポートを使用することができます。自動クライアント・リルートを実行するのに、DB2 Connect サーバーは必須ではありません。

DB2 for z/OS への接続の自動クライアント・リルートは以下のように作動します。

1. クライアントからの COMMIT 要求への応答の一部として、データ・サーバーは以下のものを戻します。
   - トランスポートを再利用できるかどうかを指定する標識。保留カーソルのようなリソースが残っていない場合、トランスポートを再利用できます。
   - トランスポートの再利用中に、接続状態を再生するのにクライアントが使用できる SET ステートメント。
2. トランザクションにおける最初の SQL ステートメントが失敗して、かつトランスポートを再利用できる場合には、以下のようになります。
   - アプリケーションにエラーはレポートされません。
   - 失敗した SQL ステートメントが再び実行されます。
   - 論理接続に関連付けられた SET ステートメントが再生され、接続状態がリストアされます。
3. トランザクションにおける最初以外の SQL ステートメントが失敗して、かつトランスポートを再利用できる場合には、以下のようになります。
   - トランザクションがロールバックされます。

- アプリケーションがデータ・サーバーへ再接続されます。
- 論理接続に関連付けられた SET ステートメントが再生され、接続状態がリストアされます。
- SQL エラー -30108 (Java の場合) または SQL30108N (非 Java クライアントの場合) がアプリケーションに戻され、ロールバックしたこと、および再接続に成功したことをアプリケーションに通知します。アプリケーションは、失敗したトランザクションを再試行するコードを組み込む必要があります。

4. トランザクションにおける最初以外の SQL ステートメントが失敗して、かつトランスポートを再利用できない場合には、以下のようになります。
   - 論理接続が初期状態、つまりデフォルトに戻されます。
   - SQL エラー -30081 (Java の場合) または SQL30081N (非 Java クライアントの場合) がアプリケーションへ戻され、再接続が成功しなかったことをアプリケーションに通知します。アプリケーションは、データ・サーバーへ再接続し、接続状態を再確立し、そして失敗したトランザクションを再試行する必要があります。
5. データ共有のメンバー・リストにあるすべてのメンバーへ接続が試行されて、どれも成功していない場合、現状でどれか使用可能なメンバーがあるかどうかを判断するため、データ共有グループに関連付けられた URL を使用して接続が試行されます。

DB2 for z/OS データ共有グループへの接続のためのトランザクション・レベルのワークロード・バランシングの操作

DB2 for z/OS データベースへの接続のためのトランザクション・レベルのワークロード・バランシングは、トランザクションの開始時に DB2 for z/OS データ共有グループ内のサーバー間で作業を平衡化することにより、高可用性に寄与します。

クライアントが DB2 for z/OS サーバーに接続し、トランザクション・レベルのワークロード・バランシングが使用可能になっている場合、以下のステップが行われます。

1. クライアントが、配布されたグループ IP アドレスを使用して DB2 for z/OS データ共有グループへの最初の接続を確立すると、クライアントは DB2 for z/OS データ共有グループのメンバーに関するサーバー・リストおよび接続の詳細 (IP アドレス、ポート、および重み) を戻します。

   サーバー・リストがクライアントによってキャッシュに入れられます。キャッシュに入れられたサーバー・リストのデフォルト存続期間は、30 秒です。
2. 新規トランザクションの開始時に、クライアントはキャッシュに入れられたサーバー・リストを読み取って、処理容量に余裕のあるサーバーを識別します。そして、その使用率の低いサーバーに関連付けられたアイドル・トランスポートがないかどうかトランスポート・プールを調べます。アイドル・トランスポートとは、接続オブジェクトと関連付けられていないトランスポートのことです。
   - アイドル・トランスポートが使用可能な場合、クライアントは接続オブジェクトをそのトランスポートと関連付けます。
   - ユーザー構成可能タイムアウト期間 (Java クライアントの場合 **db2.jcc.maxTransportObjectWaitTime**、その他のクライアントの場合 **maxTransportWaitTime**) が経過した後、使用可能なアイドル・トランスポート

がトランスポート・プールに存在せず、トランスポート・プールがその制限に達しているために新規トランスポートを割り振ることもできない場合、アプリケーションにエラーが戻されます。

3. トランザクションが実行されると、トランザクションはトランスポートに関連付けられたサーバーにアクセスします。
4. トランザクションが終了すると、クライアントは、トランスポートの再利用が接続オブジェクトに引き続き許可されているかどうかについてサーバーで検証します。
5. トランスポートの再利用が許可されている場合、サーバーは、接続オブジェクトの実行環境に適用される特殊レジスターに関する SET ステートメントのリストを返します。

   クライアントはこれらのステートメントをキャッシュに入れます。そして、接続オブジェクトが新規トランスポートと関連付けられた時に、実行環境を再構成するため、そのステートメントを再生します。
6. その後、クライアントが接続オブジェクトをトランスポートから切り離す必要があると判断した場合、それは切り離されます。
7. 新しい接続が確立された時点か、30 秒ごとか、またはユーザーが構成したインターバルごとに、サーバー・リストのクライアント・コピーがリフレッシュされます。
8. 新規トランザクションに、トランザクション・レベルのワークロード・バランシングが必要な場合、クライアントは上述のプロセスを使用して接続オブジェクトをトランスポートと関連付けます。

非 Java クライアントから DB2 for z/OS サーバーへの接続のための代替グループ

バージョン 9.7 フィックスパック 5 およびそれ以降のフィックスパックのリリースの非 Java クライアントの高可用性を向上するために、現在のグループへの接続を再確立できなかった場合、自動クライアント・リルートのための追加のフェイルオーバー・メカニズムとして代替グループを使用します。

デフォルトで、非 Java クライアントは、自動クライアント・リルート (ACR) が使用可能になっています。この機能は、サーバーへの接続を再確立できなかった場合に、現在のグループ内の代替サーバーへの自動フェイルオーバーを提供します。

この ACR 機能に加えて、現在のグループへの接続を再確立できなかった場合のフェイルオーバー・ターゲットとして、*代替グループ* を定義できます。非 Java クライアントに対して代替グループを定義するには、次のようにします。

- `db2dsdriver.cfg` ファイルの <acr> セクションの <alternategroup> 要素内で、1 つの <database> 要素を定義します。 <database> 要素内で <parameter> 要素を指定しないでください。パラメーター設定は、1 次グループから継承されます。
- 代替グループへのフェイルオーバー接続からのエラー・メッセージを抑止する場合は、<alternategroup> 要素で enableAlternateGroupSeamlessACR パラメーターを `true` に設定します。

DB2 for z/OS の場合、代替グループに定義できるデータベースは 1 つだけです。複数のDB2 for z/OS を定義した場合、接続は終了し、クライアントはエラーを返します。

非 Java クライアントを代替グループに接続するとき、1 次グループの <database> 要素のすべての接続設定およびパラメーター設定は、代替グループのデータベースへの接続によって継承されます。

非 Java クライアントが代替グループのデータベースに接続した後、1 次グループへのフェイルバックは提供されません。1 次グループにもう一度接続するには、アプリケーションまたはクライアントを再始動する必要があります。

代替グループは、ACR およびワークロード・バランシングでのみサポートされています。クライアントのアフィニティーが構成されている場合、代替グループの定義は無視されます。

例

db2dsdriver.cfg ファイルの代替グループの定義の例を次に示します。

```
<dsncollection>
  <dsn alias="mydsn2" name="mydb2" host="myserver2.ibm.com" port="5912">
  ...
</dsncollection>

<databases>
  <database name="mydb2" host="myserver2.ibm.com" port="5912">
   <parameter name="IsolationLevel" value="4"/>
   ...
   <wlb>
     <parameter name="enableWLB" value="true"/>
   </wlb>
   <acr>
    ... (ACR parameters definition)
    <alternateserverlist>
      <server name="server1" hostname="db2zosa.luw.ibm.com" port="5912">
      </server>
      <server name="server2" hostname="db2zosb.luw.ibm.com" port="5912">
      </server>
    </alternateserverlist>
    <alternategroup>
     <parameter name="enableAlternateGroupSeamlessACR" value="true"/>
     <database name="mydb3" host="myserver3.ibm.com" port="5912">
     </database>
     </alternategroup>   </acr>
  </database>

  <database name="mydb3" host="myserver3.ibm.com" port="5912">
   <parameter name="IsolationLevel" value="2"/>
   <acr>
    <parameter name="enableACR" value="true"/>
   <alternateserverlist>
     <server name="server4" hostname="db2zosd.luw.ibm.com" port="5912">
     </server>
   </alternateserverlist>
   <alternategroup>
    <parameter name="enableAlternateGroupSeamlessACR" value="true"/>
    <database name="mydb4" host="myserver4.ibm.com" port="5912">
    </database>
    <database name="mydb5" host="myserver5.ibm.com" port="5912">
    </database>
```

```
    </alternategroup>    </acr>
  ...
  </database>
</databases>
```

次のサンプル・シナリオでは、代替グループに対して自動クライアント・リルートが動作する方法を説明します。代替グループのフェイルオーバーの詳細に焦点を当てるために、現在のグループへの ACR のフェイルオーバーについての詳細はこれらのシナリオでは説明されていません。これらのシナリオは、前述の段落で説明した、サンプル db2dsdriver.cfg を使用します。

1 次への最初の接続

非 Java クライアントが最初の試行で 1 次グループへの接続に失敗した場合、現在のグループの代替サーバーへの自動クライアント・リルートのフェイルオーバーも失敗します。この例では、クライアントは次のアクションを実行します。

1. クライアントは *mydb2* への接続に失敗します。
2. クライアントは *server1* への接続に失敗します。
3. クライアントは *server2* への接続に失敗します。
4. クライアントは、db2dsdriver.cfg ファイルの <alternategroup> セクションにリストされた代替グループに、このファイルで指定された順序で接続しようとします。
   a. クライアントは正常に *mydb3* に接続します。

mydb3 に接続した後、シームレスまたは非シームレスの動作の規則が適用されたままになります。クライアントが *mydb3* に接続できなかった場合、SQL30081N エラー・メッセージを受け取ります。

1 次サーバーへの後続の接続または既存の接続

非 Java クライアントで *mydb3* への接続が切断された場合、現在のグループの代替サーバーへの自動クライアント・リルートのフェイルオーバーも失敗します。この例では、クライアントは次のアクションを実行します。

1. クライアントは *server4* への接続に失敗します。
2. クライアントは、db2dsdriver.cfg ファイルの <alternategroup> セクションにリストされた代替グループに、このファイルで指定された順序で接続しようとします。
   a. クライアントは *mydb4* に接続し、<alternategroup> セクションに複数の <database> 要素があるかを判別します。
   b. クライアントは *mydb4* への接続を終了し、SQL0866N エラー・メッセージを返します。

db2dsdriver.cfg ファイルを編集して、*mydb4* または *mydb5* の <database> 要素を削除し、アプリケーションまたはクライアントを再始動します。

代替グループへの既存の接続

非 Java クライアントが *mydb2* への接続に失敗し、現在のグループの代替サーバーへの自動クライアント・リルートのフェイルオーバーも失敗し、その後 *mydb3* 代替グループへ正常に接続します。クライアントで *mydb3* への接続が切断された後、クライアントは SQL30081N エラー・メッセージを受け取ります。

1 次グループへの接続を再度試行するには、クライアントまたはアプリケーションを再始動する必要があります。

非 Java クライアントから DB2 for z/OS サーバーへの接続の高可用性のためのアプリケーション・プログラミング要件

自動クライアント・リルートのフェイルオーバーは、シームレスまたは非シームレスを選択することができます。DB2 for z/OS への接続のためのフェイルオーバーがシームレスでない場合は、フェイルオーバーの発生時に戻されるエラーについて説明するコードを追加する必要があります。

フェイルオーバーがシームレスではなく、サーバーとの接続が再確立した場合、SQLCODE -30108 (SQL30108N) がアプリケーションへ戻されます。現行のトランザクションで発生したすべての処理は、ロールバックされます。アプリケーションにおいて、以下のことが必要になります。

- -30108 エラーと共に戻された理由コードを確認し、障害が起きたデータ共有メンバーの特殊レジスター設定を、新規 (フェイルオーバーした) データ共有メンバーへ持ち越すかどうかを決定します。現行のものではない特殊レジスター値はすべてリセットします。
- 直前のコミット操作以降、発生したすべての SQL 操作を実行します。

DB2 for z/OS への直接接続にシームレス・フェイルオーバーが発生するようにするには、以下の条件を満たしている必要があります。

- アプリケーション言語が Java、CLI、または .NET である。
- トランザクションにおける接続ではない。これは、トランザクションにおける最初の SQL ステートメントが実行された時に、障害が発生したという場合です。
- データ・サーバーは、直前のトランザクションの終わりでトランスポートの再利用を許可する。アプリケーションが KEEPDYNAMIC(YES) を指定してバインドされたため、トランスポートの再利用が許可されない場合は、この条件の例外となります。
- すべてのグローバル・セッション・データがクローズ、またはドロップされている。
- オープン状態の保留カーソルがない。
- アプリケーションが CLI を使用している場合、そのアプリケーションは SQL ステートメントを再生するために直前で呼び出された API の履歴を維持するようドライバーに要求するアクションは実行できない。そのようなアクションの例として、実行時にデータを指定すること、コンパウンド SQL を実行すること、または配列入力を使用することがあります。
- アプリケーションがストアード・プロシージャーではない。
- アプリケーションがフェデレーテッド環境で稼働していない。
- トランザクションが直前のトランザクションの成功に依存する場合、2 フェーズ・コミットが使用される。コミット操作中に障害が発生した場合、クライアントには処理がサーバーでコミットされたのか、ロールバックされたのかを判別する情報がありません。各トランザクションが直前のトランザクションの成功に依存している場合、2 フェーズ・コミットを使用します。 2 フェーズ・コミットは、XA サポートの使用が必要です。

第 18 章 非 Java クライアントでの Sysplex に関する XA サポート

DB2 Connect ライセンスのある IBM データ・サーバー・クライアントおよび非 Java データ・サーバー・ドライバーは、DB2 for z/OS Sysplex に直接アクセスでき、中間層 DB2 Connect サーバーを介さずにネイティブ XA サポートを使用します。

このタイプのクライアント・サイド XA サポートは、単一トランスポート処理モデルを使用するトランザクション・マネージャー専用です。単一トランスポート・モデルでは、単一トランスポート (物理接続) 上のトランザクションは xa_start から xa_end までメンバーに結合されます。トランザクション終了の直後に、xa_prepare(readonly) か、xa_prepare と xa_commit または xa_rollback か、あるいは xa_rollback が続きます。このすべてが単一のアプリケーション・プロセス内で行われなければなりません。このモデルを使用するトランザクション・マネージャーの例には、IBM TXSeries CICS、IBM WebSphere Application Server、および Microsoft Distributed Transaction Coordinator が含まれます。

単一トランスポート処理モデルに関するサポートには、xa_recover によって各リカバリー可能トランザクションのメンバー情報が検索される、未確定トランザクション・リカバリーも含まれます。その場合、指定されたメンバーに xa_commit または xa_rollback を指し向けることができます。

XA サポートを使用可能にするには、xa_open ストリング内で SINGLE_PROCESS パラメーターを使用するか、`db2dsdriver` 構成ファイル内で XA に関する設定を指定します。

非 Java クライアント内の XA サポートには、以下の制約事項があります。

- 以下のトランザクション・マネージャー処理モデルはサポートされていません。
  - 二重トランスポート。このモデルでは、トランスポート A 上のトランザクションは xa_start から xa_end までメンバーに結合されますが、xa_prepare(readonly)、xa_prepare と xa_commit か xa_rollback、または xa_rollback は (おそらく別のアプリケーション・プロセスからの) トランスポート B 上からになります。このモデルを使用するトランザクション・マネージャーの例には、IBM WebSphere MQ および IBM Lotus® Domino® があります。
  - マルチ・トランスポート。このモデルには、同じトランザクションに関して、複数のアプリケーション・プロセスから複数のトランスポートを使用することが含まれます。
- マルチ・トランスポート処理モデルを使用する XA トランザクション・マネージャーの場合、引き続き中間層 DB2 Connect サーバーが必要です。
- クライアントで XA サポートを使用可能にすると、シームレス・フェイルオーバーが自動的に使用不可になります。

重要: 直接 XA サポート用に DB2 for z/OS APAR PK69659 をインストールしなければなりません (Microsoft Distributed Transaction Coordinator などのトランザクション・マネージャーにとって必要)。詳しくは、APAR PK69659 を参照してください。

非 Java クライアントでの Sysplex に関する XA サポートの使用可能化

DB2 for z/OS Sysplex の XA サポートは、有効にされた WLB または Microsoft Distributed Transaction Coordinator、あるいは、インスタンスのないクライアントに使用されている Microsoft Component Services (COM+) のいずれかによって、暗黙的に使用可能になります。 DB2 for z/OS Sysplex にアクセスするクライアントの XA サポートを明示的に使用可能にするには、db2dsdriver 構成ファイルの設定を指定するか、または xa_open ストリングに SINGLE_PROCESS パラメーターを使用します。

始める前に

DB2 for z/OS Sysplex へアクセスするには、DB2 Connect ライセンスが必要になります。

以下のクライアントは、DB2 for z/OS Sysplex へアクセスするアプリケーションのための XA サポートを備えています。

- IBM Data Server Client
- IBM Data Server Runtime Client
- IBM Data Server Driver Package
- IBM Data Server Driver for ODBC and CLI

重要: 直接 XA サポート用に DB2 for z/OS APAR PK69659 をインストールしなければなりません (Microsoft Distributed Transaction Coordinator などのトランザクション・マネージャーにとって必要)。詳しくは、APAR PK69659 を参照してください。

このタスクについて

このタスクでは、IBM データ・サーバー・クライアントおよび非 Java データ・サーバー・ドライバーに対する XA サポートを明示的に使用可能に設定する方法を示します。

制約事項

XA サポートは、単一トランスポート処理モデルを使用するトランザクション・マネージャーでのみ使用可能です。この制約事項について詳しくは、クライアント Sysplex の制約事項に関するトピックを参照してください。

手順

1. インスタンス・ベースのクライアント (IBM データ・サーバー・クライアント) の場合、db2dsdriver 構成ファイルに enableDirectXA パラメーターを設定するか、もしくは xa_open ストリングに SINGLE_PROCESS パラメーターを使用して、XA サポートをオン (true) またはオフ (false) のいずれかに指定します。

2. インスタンスなしのクライアント (IBM データ・サーバー・ドライバー) の場合、Microsoft Distributed Transaction Coordinator または Microsoft Component Services (COM+) に対して XA サポートはデフォルトで使用可能です。サポートされる他のすべてのトランザクション・マネージャーの場合、xa_open ストリングに SINGLE_PROCESS キーワードを設定することによって、XA サポートを使用可能にするかどうかを指定します。 db2dsdriver 構成ファイルでの enableDirectXA の設定は、インスタンスなしのクライアントには適用できません。

タスクの結果

XA サポートが使用可能な場合、アプリケーションは、中間層の DB2 Connect サーバーを介さずに、単一アプリケーション・プロセス内で単一のトランスポートを介して分散トランザクションを実行できます。

例

データベース SAMPLE に対して単一トランスポートの XA サポートを使用可能にします。

```
<database name="SAMPLE" host="v33ec065.my.domain.com" port="446">
   <!-- database-specific parameters -->
   <!—directXA is disabled by default -->
      <parameter name="enableDirectXA" value="true" />
   </parameters>
</database>
```

第 19 章 CLI および ODBC アプリケーションの構築および実行のための開発環境の構成

CLI アプリケーションおよび ODBC アプリケーションを、IBM Data Server Client、IBM Data Server Runtime Client、または IBM Data Server Driver for ODBC and CLI を使用して DB2 データベース・サーバーに対して実行できます。ただし、CLI アプリケーションまたは ODBC アプリケーションをコンパイルするには、IBM Data Server Client が必要です。

手順

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. 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. IBM Data Server Client および Runtime Client のみ: 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、データベース、およびデータベースが置かれているマシンのホスト名をカタログします。

   Windows オペレーティング・システムでは、「CLI/ODBC 設定」GUI を使用して、DB2 データベースをカタログすることができます。
3. オプション: 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 オペレーティング・システムでは、「CLI/ODBC 設定」GUI を使用して、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、CLI/ODBC 構成キーワードを変更します。db2cli.ini ファイルの場所についての情報は、「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『db2cli.ini 初期設定ファイル』を参照してください。

   Windows オペレーティング・システムでは、「CLI/ODBC 設定」GUI を使用して、CLI/ODBC 構成キーワードを設定することができます。

タスクの結果

ステップ 1 から 4 を完了したら、Windows CLI 環境の設定に進むか、Linux または UNIX で ODBC アプリケーションを実行しているのであれば、Linux または UNIX ODBC 環境の設定に進みます。

ODBC 環境のセットアップ (Linux および UNIX)

このトピックでは、Linux および UNIX オペレーティング・システムで、ODBC アプリケーション用に DB2 データベースへのクライアント・アクセスをセットアップする方法について説明します。ご使用のアプリケーションが CLI アプリケーションの場合、実行する必要があるタスクは『*始める前に*』のセクションに記述されているものだけです。これを行えば、環境のセットアップが完了します。

始める前に

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、以下のステップを実行します。

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、例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・システムでは、例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。例えば、 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。

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 Driver Manager	libdb2.*
64 ビット CLI	libdb2.*
64 ビット ODBC Driver Manager	libdb2o.* (AIX の場合は、db2o.o)

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、HP-IPF
- .sl - HP-PA

3. 対応する共有ライブラリーをライブラリー・パスの環境変数に含めることによって、アプリケーション実行環境が確実に ODBC ドライバー・マネージャーへの参照を持つようにします。次の表は、オペレーティング・システムごとのライブラリー名を示しています。

オペレーティング・システム	環境変数	ライブラリー名
AIX	**LIBPATH**	libodbc.a
HP-UX、Linux、および Solaris	**LD_LIBRARY_PATH**	libodbc.so

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フィーチャー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。

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。このドライバー・マネージャーは、サポートされる DB2 プラットフォーム上の ODBC アプリケーションでサポートされています。このトピックでは、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 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/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.org 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

以下の例に示すのは、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-2.2.11.tar.gz
tar xf unixODBC-2.2.11.tar

cd unixODBC-2.2.11

#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-2.2.11.tar.gz
tar xf unixODBC-2.2.11.tar

cd unixODBC-2.2.11

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

以下の例に示すのは、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/user/trc.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 トレースを有効にします。

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、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。

Windows CLI 環境のセットアップ

Windows プラットフォームで、CLI ドライバーを ODBC アプリケーションで使用できるようにするには、その前にこのドライバーを Windows ODBC データ ソース アドミニストレータ (odbcad32.exe) に登録しなければなりません。

始める前に

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。

このタスクについて

CLI ドライバーは、CLI アプリケーション・プログラミング・インターフェース (API) と ODBC API の両方を実装します。Windows 環境で、CLI ドライバーを ODBC アプリケーションで使用できるようにするには、その前にこのドライバーを Windows ODBC データ ソース アドミニストレータ (odbcad32.exe) に登録しなければなりません。Windows 64 ビット・プラットフォームで ODBC データ・ソース・アドミニストレータを使用する場合、デフォルトでは、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。

- 32 ビット・アプリケーション用のデータ・ソースをセットアップするには、%WINDIR%¥SysWOW64¥odbcad32.exe を使用する必要があります。
- 64 ビット・アプリケーション用のデータ・ソースをセットアップするには、%WINDIR%¥System32¥odbcad32.exe を使用する必要があります。

手順

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 CLI/ODBC ドライバーがインストールされたことを検証します。 Windows オペレーティング・システムでは、DB2 データベース製品と共に両方のドライバーがインストールされます。新しいバージョンの Microsoft ODBC Driver Manager が既にインストールされている場合、またはインストールするオプションを手動でクリアした場合、Microsoft ODBC Driver Manager はインストールされません。両方のドライバーがインストールされていることを検証するには、次のアクションを実行します。
   a. コントロール パネルで Microsoft「ODBC データ ソース」アイコンをダブルクリックするか、コマンド行から **odbcad32.exe** コマンドを実行します。
   b. 「**ドライバ**」タブをクリックします。
   c. リストに IBM DB2 ODBC DRIVER - *DB2_Copy_Name* が表示されていることを検証します。 *DB2_Copy_Name* は、使用する DB2 コピー名です。

Microsoft ODBC Driver Manager または IBM Data Server Driver for ODBC and CLI 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、www.microsoft.com からダウンロードできます。

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)、現在のユーザーのみ使用できるよう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。
   - **db2cli** コマンドに **registerdsn** パラメーターを指定して使用します。
     – 次のように、追加する各データ・ソースに対して **db2cli** コマンドを実行します。

       ```
       db2cli registerdsn –add data-source-name
       ```
   - Microsoft ODBC 管理ツールを使用すると、「コントロール パネル」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**」をクリックします。
     c. リストにある IBM DB2 ODBC DRIVER - *DB2_Copy_Name* をダブルクリックします。 *DB2_Copy_Name* は、使用する DB2 コピー名です。
     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。例えば、以下のようにします。

     ```
     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コンピューターでこのスクリプトを実行します。

タスクの結果

Windows CLI 環境の構成後に、Windows ODBC アプリケーションから DB2 データ・ソースにアクセスできるようになります。

Windows CLI アプリケーション用に異なる DB2 コピーを選択する

デフォルトでは、Windows システム上で実行する CLI アプリケーションは、デフォルトの DB2 コピーを使用します。ただし、アプリケーションはシステムにインストールされている任意の DB2 コピーを使用できます。

始める前に

Windows CLI 環境がセットアップ済みであることを確認してください。

手順

CLI アプリケーションが Windows オペレーティング・システム上のさまざまな DB2 コピーに正常にアクセスできるようにするための方法を以下に示します。

- 「**スタート**」 > 「**プログラム**」 > 「**IBM DB2**」 > ***DB2_Copy_Name*** > 「**コマンド行ツール**」 > 「**DB2 コマンド・ウィンドウ**」から、DB2 コマンド・ウィンドウを使用する: コマンド・ウィンドウは、選択されている特定の DB2 コピーのための適切な環境変数を使用して、既にセットアップされています。
- 次のように、コマンド・ウィンドウから `db2envar.bat` を使用します。
  1. コマンド・ウィンドウを開きます。
  2. アプリケーションが使用する DB2 コピーの完全修飾パスを使用して、`db2envar.bat` ファイルを実行します。

     DB2_Copy_install_dir¥bin¥db2envar.bat
  3. 同じコマンド・ウィンドウから、CLI アプリケーションを実行します。

  これにより、選択した DB2 コピーのための環境変数が、`db2envar.bat` を実行したコマンド・ウィンドウにすべてセットアップされます。そのコマンド・ウィンドウがクローズされた後に新規のコマンド・ウィンドウが開かれると、別の DB2 コピーのための `db2envar.bat` が再実行されなければ、 CLI アプリケーションはデフォルトの DB2 コピーに対して実行されます。
- **db2SelectDB2Copy** API を使用する: 動的にリンクされるアプリケーションでは、アプリケーション・プロセス内でいずれかの DB2 DLL をロードする前にこの API を呼び出すことができます。この API は、使用する DB2 コピーをアプリケーションが使用するために必要な環境をセットアップします。 **/delayload** リンク・オプションを使用して、 DB2 DLL のロードを遅らせることができます。例えば、CLI アプリケーションが `db2api.lib` をリンクする場合、次のようにリンカーの **/delayload** オプションを使用して `db2app.dll` のロードを遅らせる必要があります。

  ```
  cl -Zi -MDd -Tp App.C /link /DELAY:nobind /DELAYLOAD:db2app.dll
     advapi32.lib psapi.lib db2api.lib delayimp.lib
  ```

  API を使用するには、`db2ApiInstall.h` を含める必要があります。これにより、アプリケーションは必ず `db2ApiInstall.lib` に静的にリンクします。
- LoadLibraryEx を使用する: LoadLibrary を使用する代わりに、 LoadLibraryEx に LOAD_WITH_ALTERED_SEARCH_PATH パラメーターを指定して呼び出し、使用する DB2 コピーのバージョンに対応する `db2app.dll` をロードできます。例えば、以下のようにします。

```
HMODULE hLib = LoadLibraryEx("c:¥¥sqllib¥¥bin¥¥db2app.dll",
   NULL, LOAD_WITH_ALTERED_SEARCH_PATH);
```

CLI 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アップグレード時、またはフィックスパックのクライアントまたはサーバーへの適用時に、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、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、 db2cli.lst を再バインドする必要があります。

次のコマンドを発行して、db2cli.lst を再バインドします。

Linux および 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、 CLI が DB2 servers on Linux, UNIX, and Windows 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 IBM Power Systems 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。

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 データベース・システムは総称バインド・ファイルを使用しますが、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 データベース・システムは警告を生成することがあります。

次のメッセージに示す警告の例は、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 39. 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	Linux、UNIX、および Windows 上の DB2 サーバーで必要?	ホスト・サーバーで必要?	説明
db2clipk.bnd	SYSSH*xyy*	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
	SYSSN*xyy*	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
	SYSLH*xyy*	はい	はい	動的プレースホルダー - ラージ・パッケージ WITH HOLD
	SYSLN*xyy*	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	SQLL9*vyy*	はい	いいえ	カタログ関数サポート

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
- '*v*' は、DB2 サーバーのバージョンを表します。例えば、E=バージョン 8、F=バージョン 9 となります。
- '*x*' は分離レベルです。0=NC、1=UR、2=CS、3=RS、4=RR となります。
- '*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

例えば、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、Linux、UNIX、および Windows 上の DB2 サーバーでデータベースが作成またはアップグレードされるときかフィックスパックが適用されるときに自動的にバインドされて、これらのタイプのサーバー上にのみ存在します。このバインド・ファイルはサーバー側に存在します。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アップグレード時に SQL1088W (+1088) 警告を受け取った場合には、それを (サーバーから) 手動でバインドすることが必要になります。

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 (例えば、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アップグレードした後に 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。

CLI パッケージのバインド・オプションの制限

いくつかのバインド・オプションは、CLI パッケージを、次のリスト・ファイルにてバインドする場合、有効ではありません。すなわち、db2cli.lst、 ddcsmvs.lst、ddcs400.lst、 ddcsvm.lst、または ddcsvse.lst。 CLI パッケージは CLI、ODBC、JDBC、 OLE DB、 .NET、および ADO の各アプリケーションによって使用されるので、CLI パッケージに対して行われる変更は、これらのタイプのアプリケーションすべてに影響します。したがって、バインド・オプションのサブセットだけが、CLI パッケージのバインド時にデフォルトでサポートされます。サポートされるオプションは、ACTION、 COLLECTION、 CLIPKG、 OWNER、および REPLVER です。 CLI パッケージに影響する他のすべてのバインド・オプションは、無視されます。

CLI パッケージを、デフォルトでサポートされないバインド・オプションを指定して作成するには、COLLECTION バインド・オプションを、デフォルトのコレクション ID (NULLID) とは異なるコレクション ID とともに指定します。すると、指定されたバインド・オプションはいずれも受け入れられます。例えば、CLI パッケージを KEEPDYNAMIC YES バインド・オプション (デフォルトでサポートされない) を指定して作成するには、次のコマンドを発行します。

```
db2 bind @db2cli.lst collection newcolid keepdynamic yes
```

CLI/ODBC アプリケーションが新規コレクションで作成された CLI パッケージにアクセスするには、db2cli.ini 初期設定ファイル内の CurrentPackageSet CLI/ODBC キーワードを、新規コレクション ID に設定します。

特定のコレクション ID ですでに存在する CLI パッケージを上書きするには、次のアクションのいずれかを実行します。

- このコレクション ID に対してバインド・コマンドを発行する前に、既存の CLI パッケージを除去します。
- バインド・コマンドを発行する際に、ACTION REPLACE バインド・オプションを指定します。

第 20 章 CLI アプリケーションの作成

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、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。

このタスクについて

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
```

手順

- **組み込み SQL アプリケーションの作成と実行** 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
     ```

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 ではこのトピック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。

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。例: $HOME/sqllib/$LIB. -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。

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 ではこのトピック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。

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。

+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。

-Ae HP ANSI 拡張モードを有効にします。

-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。

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。

+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。

-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。

$EXTRA_LFLAG
ランタイム・パスを指定します。設定する場合、32 ビットならその値は `-Wl,+b$HOME/sqllib/lib32` であり、64 ビットなら `-Wl,+b$HOME/sqllib/lib64` 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/lib64` 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。

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 ではこのトピック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。

コンパイル・オプション:

gcc C コンパイラー。

$EXTRA_C_FLAGS
以下のフラグのいずれかが入ります。

- `-m31` (Linux for zSeries® で 32 ビット・ライブラリーをビルドする場合のみ)
- `-m32` (Linux for x86、x64 および POWER® で 32 ビット・ライブラリーをビルドする場合)
- `-m64` (Linux for zSeries、 POWER、x64 で 64 ビット・ライブラリーをビルドする場合)
- 値なし (Linux for IA64 で 64 ビット・ライブラリーをビルドする場合)

-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
以下のフラグのいずれかが入ります。

- -m31 (Linux for zSeries で 32 ビット・ライブラリーをビルドする場合のみ)
- -m32 (Linux for x86、x64 および POWER で 32 ビット・ライブラリーをビルドする場合)
- -m64 (Linux for zSeries、 POWER、x64 で 64 ビット・ライブラリーをビルドする場合)
- 値なし (Linux for IA64 で 64 ビット・ライブラリーをビルドする場合)

-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。

$EXTRA_LFLAG
32 ビットの場合、値は「-Wl,-rpath,$DB2PATH/lib32」、64 ビットの場合は「-Wl,-rpath,$DB2PATH/lib64」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。例えば、32 ビットの場合は $HOME/sqllib/lib32、64 ビットの場合は $HOME/sqllib/lib64 のように指定します。

-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。

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は、DB2 ではこのトピック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、以下のように設定されます。

- "v8plusa": Solaris SPARC 上の 32 ビット・アプリケーション
- "v9": Solaris SPARC 上の 64 ビット・アプリケーション
- "sse2": Solaris x64 上の 32 ビット・アプリケーション
- "amd64": Solaris x64 上の 64 ビット・アプリケーション

-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。例えば、32 ビットの場合は $HOME/sqllib/lib32、64 ビットの場合は $HOME/sqllib/lib64 のように指定します。

$EXTRA_LFLAG
実行時の DB2 共有ライブラリーのロケーションを示します。32 ビットの場合、その値は「-R$DB2PATH/lib32」、64 ビットの場合は「-R$DB2PATH/lib64」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。

UNIX での CLI 複数接続アプリケーションの作成

DB2 Database for Linux, UNIX, and Windows 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。

このタスクについて

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" ($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。

オプション・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。オプション・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 とパスワードを指定します。

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。

手順

- 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データベース・パーティションです。

- 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。TCP/IP Listener が実行されていることを確認するには、下記のステップに従います。

  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。サービス・ファイルが見つからない場合、またはサービス・ファイルを読むためのファイル・アクセス権限がない場合は、システム管理者にお問い合わせください。 UNIX および Linux システムでは、サービス・ファイルは /etc/services にあります。

  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。

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。

このタスクについて

バッチ・ファイル bldapp.bat には、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、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 (.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。

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。

手順

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。

   a. 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
      ```

   b. 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
      ```

c. 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
```

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 ではこのトピック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。

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。または、icl (32 ビットおよび 64 ビット・アプリケーション用の Intel C++ Compiler を表す)、あるいは ecl (Itanium 64 ビット・アプリケーション用の Intel C++ Compiler を表す) に設定することもできます。

-Zi デバッグ情報を有効にします。

-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。

-J コンパイラー・オプション。このオプションを使用して、Visual Studio アプリケーションが -J オプションでコンパイルされると、lconv 構造に含まれる char メンバーは、現在のロケールでサポートされていない場合、UCHAR_MAX の値と等しくなります。

リンク・オプション:

link 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。

/delayload:db2app.dll
DB2 API が最初に呼び出されるよりも前には、 db2app.dll がロードされないようにするために使用します。これは、db2SelectDB2Copy API を使用するときに必要です。

db2ApiInstall.lib
コンピューターにインストールされた特定の DB2 コピーを db2SelectDB2Copy API を使用して選択する必要がある場合に、アプリケーションを静的にリンクするためのライブラリー。注: この機能を使用するには、db2app.dll を動的にロードするかまたはコンパイラーの /delayload:db2app.dll オプションを使用して、他のいずれかの DB2 API を呼び出すよりも前に、db2SelectDB2Copy API を呼び出す必要があります。

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。

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。

このタスクについて

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" (%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。

オプション・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。オプション・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 とパスワードを指定します。

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。

手順

- 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データベース・パーティションです。

- 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。TCP/IP Listener が実行されていることを確認するには、下記のステップに従います。

  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。サービス・ファイルが見つからない場合、またはサービス・ファイルを読むためのファイル・アクセス権限がない場合は、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。

構成ファイルを使用した CLI アプリケーションの作成

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 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
   ```

タスクの結果

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。

- 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
  ```

- 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
  ```

- 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。

- 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
  ```

- 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
  ```

- 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
  ```

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

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 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。

4. 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します。

   ```
   db2 -td@ -vf spcreate.db2
   ```

   カタログが終了したら、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

database
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。

第 21 章 CLI ルーチンの作成

UNIX での CLI ルーチンの作成

DB2 Database for Linux, UNIX, and Windows には、DB2 コール・レベル・インターフェース (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、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、これが共有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。

タスクの結果

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

database
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。

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 ではこのトピック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。

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。例: `$HOME/sqllib/$LIB`. `-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。

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 ではこのトピック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。

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。

+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。

+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 位置独立コードを生成します。

-Ae HP ANSI 拡張モードを有効にします。

-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。

$EXTRA_LFLAG
ランタイム・パスを指定します。設定する場合、32 ビットならその値は +b$HOME/sqllib/lib32 であり、64 ビットなら +b$HOME/sqllib/lib64 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/lib64 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。

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 ではこのトピック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。

コンパイル・オプション:

gcc C コンパイラー。

$EXTRA_C_FLAGS
以下のフラグのいずれかが入ります。

- -m31 (Linux for zSeries で 32 ビット・ライブラリーをビルドする場合のみ)
- -m32 (Linux for x86、x64 および POWER で 32 ビット・ライブラリーをビルドする場合)
- -m64 (Linux for zSeries、 POWER、x64 で 64 ビット・ライブラリーをビルドする場合)
- 値なし (Linux for IA64 で 64 ビット・ライブラリーをビルドする場合)

-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
以下のフラグのいずれかが入ります。

- -m31 (Linux for zSeries で 32 ビット・ライブラリーをビルドする場合のみ)
- -m32 (Linux for x86、x64 および POWER で 32 ビット・ライブラリーをビルドする場合)
- -m64 (Linux for zSeries、 POWER、x64 で 64 ビット・ライブラリーをビルドする場合)
- 値なし (Linux for IA64 で 64 ビット・ライブラリーをビルドする場合)

-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。

$EXTRA_LFLAG
実行時の DB2 共有ライブラリーのロケーションを示します。32 ビットの場合、その値は「-Wl,-rpath,$DB2PATH/lib32」です。64 ビットの場合、その値は「-Wl,-rpath,$DB2PATH/lib64」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。例えば、32 ビットの場合は $HOME/sqllib/lib32、64 ビットの場合は $HOME/sqllib/lib64 のように指定します。

-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。

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は、DB2 ではこのトピック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。

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、以下のように設定されます。

- "v8plusa": Solaris SPARC 上の 32 ビット・アプリケーション
- "v9": Solaris SPARC 上の 64 ビット・アプリケーション
- "sse2": Solaris x64 上の 32 ビット・アプリケーション
- "amd64": Solaris x64 上の 64 ビット・アプリケーション

-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。例えば、32 ビットの場合は $HOME/sqllib/lib32、64 ビットの場合は $HOME/sqllib/lib64 のように指定します。

$EXTRA_LFLAG
実行時の DB2 共有ライブラリーのロケーションを示します。32 ビットの場合、その値は「-R$DB2PATH/lib32」、64 ビットの場合は「-R$DB2PATH/lib64」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。

Windows での CLI ルーチンの作成

DB2 Database for Linux, UNIX, and Windows 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、これが共有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。

タスクの結果

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

database
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。

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 ではこのトピック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。

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。または、icl (32 ビットおよび 64 ビット・アプリケーション用の Intel C++ Compiler を表す)、あるいは ecl (Itanium 64 ビット・アプリケーション用の Intel C++ Compiler を表す) に設定することもできます。

-Zi デバッグ情報を有効にします。

-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 を使用してリンクします。

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 ファイルをビルドします。

%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。

/delayload:db2app.dll
DB2 API が最初に呼び出されるよりも前には、 db2app.dll がロードされないようにするために使用します。これは、db2SelectDB2Copy API を使用するときに必要です。

db2ApiInstall.lib
コンピューターにインストールされた特定の DB2 コピーを db2SelectDB2Copy API を使用して選択する必要がある場合に、アプリケーションを静的にリンクするためのライブラリー。注: この機能を使用するには、db2app.dll を動的にロードするかまたはコンパイラーの /delayload:db2app.dll オプションを使用して、他のいずれかの DB2 API を呼び出すよりも前に、db2SelectDB2Copy API を呼び出す必要があります。

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。

付録 A. DB2 技術情報の概説

DB2 技術情報は、さまざまな方法でアクセスすることが可能な、各種形式で入手できます。

DB2 技術情報は、以下のツールと方法を介して利用できます。

- DB2インフォメーション・センター
  - トピック (タスク、概念、およびリファレンス・トピック)
  - サンプル・プログラム
  - チュートリアル
- DB2 資料
  - PDF ファイル (ダウンロード可能)
  - PDF ファイル (DB2 PDF DVD に含まれる)
  - 印刷資料
- コマンド行ヘルプ
  - コマンド・ヘルプ
  - メッセージ・ヘルプ

注: DB2 インフォメーション・センターのトピックは、PDF やハードコピー資料よりも頻繁に更新されます。最新の情報を入手するには、資料の更新が発行されたときにそれをインストールするか、ibm.com にある DB2 インフォメーション・センターを参照してください。

技術資料、ホワイト・ペーパー、IBM Redbooks® 資料などのその他の DB2 技術情報には、オンライン (ibm.com) でアクセスできます。 DB2 Information Management ソフトウェア・ライブラリー・サイト (http://www.ibm.com/software/data/sw-library/) にアクセスしてください。

資料についてのフィードバック

DB2 の資料についてのお客様からの貴重なご意見をお待ちしています。DB2 の資料を改善するための提案については、db2docs@ca.ibm.com まで E メールを送信してください。DB2 の資料チームは、お客様からのフィードバックすべてに目を通しますが、直接お客様に返答することはありません。お客様が関心をお持ちの内容について、可能な限り具体的な例を提供してください。特定のトピックまたはヘルプ・ファイルについてのフィードバックを提供する場合は、そのトピック・タイトルおよび URL を含めてください。

DB2 お客様サポートに連絡する場合には、この E メール・アドレスを使用しないでください。資料を参照しても、DB2 の技術的な問題が解決しない場合は、お近くの IBM サービス・センターにお問い合わせください。

DB2 テクニカル・ライブラリー (ハードコピーまたは PDF 形式)

以下の表は、IBM Publications Center (www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss) から利用できる DB2 ライブラリーについて説明しています。英語および翻訳された DB2 バージョン 10.1 のマニュアル (PDF 形式) は、www.ibm.com/support/docview.wss?rs=71&uid=swg2700947 からダウンロードできます。

この表には印刷資料が入手可能かどうかが示されていますが、国または地域によっては入手できない場合があります。

資料番号は、資料が更新される度に大きくなります。資料を参照する際は、以下にリストされている最新版であることを確認してください。

注: *DB2 インフォメーション・センター*は、PDF やハードコピー資料よりも頻繁に更新されます。

表 40. DB2 の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
管理 API リファレンス	SA88-4671-00	入手可能	2012 年 4 月
管理ルーチンおよびビュー	SA88-4672-00	入手不可	2012 年 4 月
コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻	SA88-4676-00	入手可能	2012 年 4 月
コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻	SA88-4677-00	入手可能	2012 年 4 月
コマンド・リファレンス	SA88-4673-00	入手可能	2012 年 4 月
データベース: 管理の概念および構成リファレンス	SA88-4662-00	入手可能	2012 年 4 月
データ移動ユーティリティー ガイドおよびリファレンス	SA88-4693-00	入手可能	2012 年 4 月
データベースのモニタリング ガイドおよびリファレンス	SA88-4663-00	入手可能	2012 年 4 月
データ・リカバリーと高可用性 ガイドおよびリファレンス	SA88-4694-00	入手可能	2012 年 4 月
データベース・セキュリティー・ガイド	SA88-4695-00	入手可能	2012 年 4 月

表 40. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
DB2 ワークロード管理ガイドおよびリファレンス	SA88-4685-00	入手可能	2012 年 4 月
ADO.NET および OLE DB アプリケーションの開発	SA88-4665-00	入手可能	2012 年 4 月
組み込み SQL アプリケーションの開発	SA88-4666-00	入手可能	2012 年 4 月
Java アプリケーションの開発	SA88-4669-00	入手可能	2012 年 4 月
Perl、PHP、Python および Ruby on Rails アプリケーションの開発	SA88-4670-00	入手不可	2012 年 4 月
SQL および外部ルーチンの開発	SA88-4667-00	入手可能	2012 年 4 月
データベース・アプリケーション開発の基礎	GI88-4279-00	入手可能	2012 年 4 月
DB2 インストールおよび管理 概説 (Linux および Windows 版)	GI88-4280-00	入手可能	2012 年 4 月
グローバリゼーション・ガイド	SA88-4696-00	入手可能	2012 年 4 月
DB2 サーバー機能 インストール	GA88-4679-00	入手可能	2012 年 4 月
IBM データ・サーバー・クライアント機能 インストール	GA88-4680-00	入手不可	2012 年 4 月
メッセージ・リファレンス 第 1 巻	SA88-4688-00	入手不可	2012 年 4 月
メッセージ・リファレンス 第 2 巻	SA88-4689-00	入手不可	2012 年 4 月
Net Search Extender 管理およびユーザーズ・ガイド	SA88-4691-00	入手不可	2012 年 4 月
パーティションおよびクラスタリングのガイド	SA88-4697-00	入手可能	2012 年 4 月
pureXML ガイド	SA88-4686-00	入手可能	2012 年 4 月
Spatial Extender ユーザーズ・ガイドおよびリファレンス	SA88-4690-00	入手不可	2012 年 4 月

表 40. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
SQL プロシージャー言語: アプリケーションのイネーブルメントおよびサポート	SA88-4668-00	入手可能	2012 年 4 月
SQL リファレンス 第 1 巻	SA88-4674-00	入手可能	2012 年 4 月
SQL リファレンス 第 2 巻	SA88-4675-00	入手可能	2012 年 4 月
Text Search ガイド	SA88-4692-00	入手可能	2012 年 4 月
問題判別およびデータベース・パフォーマンスのチューニング	SA88-4664-00	入手可能	2012 年 4 月
DB2 バージョン 10.1 へのアップグレード	SA88-4678-00	入手可能	2012 年 4 月
DB2 バージョン 10.1 の新機能	SA88-4684-00	入手可能	2012 年 4 月
XQuery リファレンス	SA88-4687-00	入手不可	2012 年 4 月

表 41. DB2 Connect 固有の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
DB2 Connect DB2 Connect Personal Edition インストールおよび構成	SA88-4681-00	入手可能	2012 年 4 月
DB2 Connect DB2 Connect サーバー機能インストールおよび構成	SA88-4682-00	入手可能	2012 年 4 月
DB2 Connect ユーザーズ・ガイド	SA88-4683-00	入手可能	2012 年 4 月

コマンド行プロセッサーから SQL 状態ヘルプを表示する

DB2 製品は、SQL ステートメントの結果の原因になったと考えられる条件の SQLSTATE 値を戻します。 SQLSTATE ヘルプは、SQL 状態および SQL 状態クラス・コードの意味を説明します。

手順

SQL 状態ヘルプを開始するには、コマンド行プロセッサーを開いて以下のように入力します。

```
? sqlstate または ? class code
```

ここで、*sqlstate* は有効な 5 桁の SQL 状態を、*class code* は SQL 状態の最初の 2 桁を表します。
例えば、`? 08003` を指定すると SQL 状態 08003 のヘルプが表示され、 `? 08` を指定するとクラス・コード 08 のヘルプが表示されます。

異なるバージョンの DB2 インフォメーション・センターへのアクセス

他のバージョンの DB2 製品の資料は、ibm.com® のそれぞれのインフォメーション・センターにあります。

このタスクについて

DB2 バージョン 10.1 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v10r1 です。

DB2 バージョン 9.8 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/ です。

DB2 バージョン 9.7 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/ です。

DB2 バージョン 9.5 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r5 です。

DB2 バージョン 9.1 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9/ です。

DB2 バージョン 8 のトピックについては、*DB2 インフォメーション・センター* の URL (http://publib.boulder.ibm.com/infocenter/db2luw/v8/) を参照してください。

コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの更新

ローカルにインストールした DB2 インフォメーション・センターは、定期的に更新する必要があります。

始める前に

DB2 バージョン 10.1 インフォメーション・センターが既にインストール済みである必要があります。詳しくは、「*DB2 サーバー機能 インストール*」の『DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール』のトピックを参照してください。インフォメーション・センターのインストールに適用されるすべての前提条件と制約事項は、インフォメーション・センターの更新にも適用されます。

このタスクについて

既存の DB2 インフォメーション・センターは、自動で更新することも手動で更新することもできます。

- 自動更新は、既存のインフォメーション・センターのフィーチャーと言語を更新します。自動更新を使用すると、手動更新と比べて、更新中にインフォメーショ

ン・センターが使用できなくなる時間が短くなるというメリットがあります。さらに、自動更新は、定期的に実行する他のバッチ・ジョブの一部として実行されるように設定することができます。

- 手動更新は、既存のインフォメーション・センターのフィーチャーと言語の更新に使用できます。自動更新は更新処理中のダウン時間を減らすことができますが、フィーチャーまたは言語を追加する場合は手動処理を使用する必要があります。例えば、ローカルのインフォメーション・センターが最初は英語とフランス語でインストールされており、その後ドイツ語もインストールすることにした場合、手動更新でドイツ語をインストールし、同時に、既存のインフォメーション・センターのフィーチャーおよび言語を更新できます。しかし、手動更新ではインフォメーション・センターを手動で停止、更新、再始動する必要があります。更新処理の間はずっと、インフォメーション・センターは使用できなくなります。自動更新処理では、インフォメーション・センターは、更新を行った後に、インフォメーション・センターを再始動するための停止が発生するだけで済みます。

このトピックでは、自動更新のプロセスを詳しく説明しています。手動更新の手順については、『コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの手動更新』のトピックを参照してください。

手順

コンピューターまたはイントラネット・サーバーにインストールされている DB2 インフォメーション・センターを自動更新する手順を以下に示します。

1. Linux オペレーティング・システムの場合、次のようにします。
   a. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは、`/opt/ibm/db2ic/V10.1` ディレクトリーにインストールされています。
   b. インストール・ディレクトリーから `doc/bin` ディレクトリーにナビゲートします。
   c. 次のように `update-ic` スクリプトを実行します。

      ```
      update-ic
      ```
2. Windows オペレーティング・システムの場合、次のようにします。
   a. コマンド・ウィンドウを開きます。
   b. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは、`<Program Files>¥IBM¥DB2 Information Center¥バージョン 10.1` ディレクトリーにインストールされています (`<Program Files>` は「Program Files」ディレクトリーのロケーション)。
   c. インストール・ディレクトリーから `doc¥bin` ディレクトリーにナビゲートします。
   d. 次のように `update-ic.bat` ファイルを実行します。

      ```
      update-ic.bat
      ```

タスクの結果

DB2 インフォメーション・センターが自動的に再始動します。更新が入手可能な場合、インフォメーション・センターに、更新された新しいトピックが表示されます。インフォメーション・センターの更新が入手可能でなかった場合、メッセージがログに追加されます。ログ・ファイルは、doc¥eclipse¥configuration ディレクトリーにあります。ログ・ファイル名はランダムに生成された名前です。例えば、1239053440785.log のようになります。

コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの手動更新

DB2 インフォメーション・センターをローカルにインストールしている場合は、IBM から資料の更新を入手してインストールすることができます。

このタスクについて

ローカルにインストールされた *DB2 インフォメーション・センター* を手動で更新するには、以下のことを行う必要があります。

1. コンピューター上の *DB2 インフォメーション・センター* を停止し、インフォメーション・センターをスタンドアロン・モードで再始動します。インフォメーション・センターをスタンドアロン・モードで実行すると、ネットワーク上の他のユーザーがそのインフォメーション・センターにアクセスできなくなります。これで、更新を適用できるようになります。DB2 インフォメーション・センターのワークステーション・バージョンは、常にスタンドアロン・モードで実行されます。を参照してください。
2. 「更新」機能を使用することにより、どんな更新が利用できるかを確認します。インストールしなければならない更新がある場合は、「更新」機能を使用してそれを入手およびインストールできます。

   注: ご使用の環境において、インターネットに接続されていないマシンに *DB2 インフォメーション・センター* の更新をインストールする必要がある場合、インターネットに接続されていて *DB2 インフォメーション・センター* がインストールされているマシンを使用して、更新サイトをローカル・ファイル・システムにミラーリングしてください。ネットワーク上の多数のユーザーが資料の更新をインストールする場合にも、更新サイトをローカルにミラーリングして、更新サイト用のプロキシーを作成することにより、個々のユーザーが更新を実行するのに要する時間を短縮できます。
   更新パッケージが入手可能な場合、「更新」機能を使用してパッケージを入手します。ただし、「更新」機能は、スタンドアロン・モードでのみ使用できます。
3. スタンドアロンのインフォメーション・センターを停止し、コンピューター上の *DB2 インフォメーション・センター* を再開します。

注: Windows 2008、Windows Vista (およびそれ以上) では、このセクションの後の部分でリストされているコマンドは管理者として実行する必要があります。完全な管理者特権でコマンド・プロンプトまたはグラフィカル・ツールを開くには、ショートカットを右クリックしてから、**「管理者として実行」**を選択します。

手順

コンピューターまたはイントラネット・サーバーにインストール済みの *DB2* インフォメーション・センター を更新するには、以下のようにします。

1. *DB2* インフォメーション・センター を停止します。
   - Windows では、「**スタート**」 > 「**コントロール パネル**」 > 「**管理ツール**」 > 「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**停止**」を選択します。
   - Linux では、以下のコマンドを入力します。

     ```
     /etc/init.d/db2icdv10 stop
     ```
2. インフォメーション・センターをスタンドアロン・モードで開始します。
   - Windows の場合:
     a. コマンド・ウィンドウを開きます。
     b. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、*DB2* インフォメーション・センター は、*Program_Files*¥IBM¥DB2 Information Center¥バージョン 10.1 ディレクトリーにインストールされています (*Program_Files* は Program Files ディレクトリーのロケーション)。
     c. インストール・ディレクトリーから doc¥bin ディレクトリーにナビゲートします。
     d. 次のように help_start.bat ファイルを実行します。

        ```
        help_start.bat
        ```
   - Linux の場合:
     a. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、*DB2* インフォメーション・センター は、/opt/ibm/db2ic/V10.1 ディレクトリーにインストールされています。
     b. インストール・ディレクトリーから doc/bin ディレクトリーにナビゲートします。
     c. 次のように help_start スクリプトを実行します。

        ```
        help_start
        ```

   システムのデフォルト Web ブラウザーが開き、スタンドアロンのインフォメーション・センターが表示されます。
3. 「**更新**」ボタン () をクリックします。(ブラウザーで JavaScript が有効になっている必要があります。) インフォメーション・センターの右側のパネルで、「**更新の検索**」をクリックします。 既存の文書に対する更新のリストが表示されます。
4. インストール・プロセスを開始するには、インストールする更新をチェックして選択し、「**更新のインストール**」をクリックします。
5. インストール・プロセスが完了したら、「**完了**」をクリックします。
6. 次のようにして、スタンドアロンのインフォメーション・センターを停止します。
   - Windows の場合は、インストール・ディレクトリーの doc¥bin ディレクトリーにナビゲートしてから、次のように help_end.bat ファイルを実行します。

```
help_end.bat
```

注: help_end バッチ・ファイルには、help_start バッチ・ファイルを使用して開始したプロセスを安全に停止するのに必要なコマンドが含まれています。help_start.bat は、Ctrl-C や他の方法を使用して停止しないでください。

- Linux の場合は、インストール・ディレクトリーの doc/bin ディレクトリーにナビゲートしてから、次のように help_end スクリプトを実行します。

```
help_end
```

注: help_end スクリプトには、help_start スクリプトを使用して開始したプロセスを安全に停止するのに必要なコマンドが含まれています。他の方法を使用して、help_start スクリプトを停止しないでください。

7. *DB2 インフォメーション・センター* を再開します。
   - Windows では、「**スタート**」 > 「**コントロール パネル**」 > 「**管理ツール**」 > 「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**開始**」を選択します。
   - Linux では、以下のコマンドを入力します。

```
/etc/init.d/db2icdv10 start
```

タスクの結果

更新された *DB2 インフォメーション・センター* に、更新された新しいトピックが表示されます。

DB2 チュートリアル

DB2 チュートリアルは、DB2 データベース製品のさまざまな機能について学習するための支援となります。この演習をとおして段階的に学習することができます。

はじめに

インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/) から、このチュートリアルの XHTML 版を表示できます。

演習の中で、サンプル・データまたはサンプル・コードを使用する場合があります。個々のタスクの前提条件については、チュートリアルを参照してください。

DB2 チュートリアル

チュートリアルを表示するには、タイトルをクリックします。

「*pureXML ガイド*」の『**pureXML®**』
: XML データを保管し、ネイティブ XML データ・ストアに対して基本的な操作を実行できるように、DB2 データベースをセットアップします。

DB2 トラブルシューティング情報

DB2 データベース製品を使用する際に役立つ、トラブルシューティングおよび問題判別に関する広範囲な情報を利用できます。

DB2 の資料

トラブルシューティング情報は、「*問題判別およびデータベース・パフォーマンスのチューニング*」または *DB2 インフォメーション・センター* の『データベースの基本』セクションにあります。ここには、以下の情報が記載されています。

- DB2 診断ツールおよびユーティリティーを使用した、問題の切り分け方法および識別方法に関する情報。
- 最も一般的な問題のうち、いくつかの解決方法。
- DB2 データベース製品で発生する可能性のある、その他の問題の解決に役立つアドバイス。

IBM サポート・ポータル

現在問題が発生していて、考えられる原因とソリューションを見つけるには、IBM サポート・ポータルを参照してください。 Technical Support サイトには、最新の DB2 資料、TechNotes、プログラム診断依頼書 (APAR またはバグ修正)、フィックスパック、およびその他のリソースへのリンクが用意されています。この知識ベースを活用して、問題に対する有効なソリューションを探し出すことができます。

IBM サポート・ポータル (http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows) にアクセスしてください。

ご利用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

適用度: これらのご利用条件は、IBM Web サイトのあらゆるご利用条件に追加で適用されるものです。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。 ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

権利: ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

IBM の商標: IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

付録 B. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。IBM 以外の製品に関する情報は、本書の最初の発行時点で入手可能な情報に基づいており、変更される場合があります。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒103-8510
東京都中央区日本橋箱崎町19番21号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、

利便性もしくは機能性があることをほのめかしたり、保証することはできません。サンプル・プログラムは、現存するままの状態で提供されるものであり、いかなる種類の保証も提供されません。 IBM は、これらのサンプル・プログラムの使用から生ずるいかなる損害に対しても責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

© (*お客様の会社名*) (*西暦年*). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _*年を入れる*_. All rights reserved.

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

以下は、それぞれ各社の商標または登録商標です。

- Linux は、Linus Torvalds の米国およびその他の国における商標です。
- Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。
- UNIX は The Open Group の米国およびその他の国における登録商標です。
- インテル、Intel、Intel ロゴ、Intel Inside、Intel Inside ロゴ、Celeron、Intel SpeedStep、Itanium、Pentium は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。
- Microsoft、 Windows、 Windows NT、および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

［サ行］

[タ行]

[ナ行]

[ハ行]

［マ行］

［ラ行］

［ワ行］

[数字]

A

B

C

D

E

F

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

W

X

Y

［特殊文字］

Printed in Japan

SA88-4676-00

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21

Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻