IBM DB2 10.1
for Linux, UNIX, and Windows

ADO.NET および OLE DB アプリケーションの開発

SA88-4665-00
(英文原典：SC27-3873-00)

IBM DB2 10.1
for Linux, UNIX, and Windows

ADO.NET および OLE DB アプリケーションの開発

SA88-4665-00
(英文原典：SC27-3873-00)

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、 197 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： SC27-3873-00
IBM DB2 10.1
for Linux, UNIX, and Windows
Developing ADO.NET and OLE DB Applications

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

第 1 章 ADO.NET アプリケーション開発

近年、Microsoft では Windows 用の新規のソフトウェア開発プラットフォームの普及を促進してきました。これは、.NET Framework として知られています。 .NET Framework は、Microsoft の Component Object Model (COM) テクノロジーに代わるものです。主要な .NET Framework 機能は以下のとおりです。

- .NET アプリケーションを 40 以上の異なるプログラミング言語でコーディングできます。 .NET 開発用の最も一般的な言語は C# および Visual Basic .NET です。
- .NET Framework クラス・ライブラリーは、.NET アプリケーションの構築に使用する構築ブロックを提供します。このクラス・ライブラリーは言語に左右されず、オペレーティング・システムおよびアプリケーション・サービスへのインターフェースを提供します。
- ご使用の .NET アプリケーション (言語に関係なく) は、バイトコードの一種である中間言語 (IL) にコンパイルされます。
- Common Language Runtime (CLR) は .NET Framework の核心部で、IL コードをオンザフライにコンパイルし、それを実行します。コンパイルした IL コードを実行すると、CLR はオブジェクトを活動化し、セキュリティー許可を検証し、メモリーを割り振り、それらのオブジェクトを実行します。実行が完了したならメモリーをクリーンアップします。

これらの機能を使用して、.NET Framework は広範囲に及ぶアプリケーション・インプリメンテーション (例えば、Windows フォーム、Web フォーム、および Web サービスなど)、迅速なアプリケーション開発、および安全なアプリケーション・デプロイメントを容易にします。COM および COM+ は、前述のどの機能を実行する場合にも不十分、または煩雑であることが分かっています。

.NET Framework は、ADO.NET を介して大規模なデータ・アクセス・サポートを提供します。 ADO.NET は接続アクセスと切断アクセスの両方をサポートします。ADO.NET での切断データ・アクセスのキー・コンポーネントは DataSet クラスです。このクラスのインスタンスは、アプリケーションのメモリー内にあるデータベース・キャッシュの役割をします。

接続アクセスと切断アクセスのどちらの場合でも、アプリケーションはデータ・プロバイダーとして知られるものを介してデータベースを使用します。さまざまなデータベース製品 (DB2® for Windows を含む) には、独自の .NET データ・プロバイダーが含まれています。

.NET データ・プロバイダーは、以下の基本クラスのインプリメンテーションを特色としています。

- Connection: データベース接続を確立および管理します。
- Command: SQL ステートメントをデータベースに対して実行します。
- DataReader: データベースから結果セット・データを読み取って、戻します。

- DataAdapter: DataSet インスタンスをデータベースにリンクします。 DataAdapter インスタンスを介して、DataSet はデータベース表データの読み取りと書き込みを行えます。

Microsoft では、2 つのデータ・プロバイダー (OLE DB .NET Data Provider および ODBC .NET Data Provider) を提供しています。OLE DB .NET Data Provider は、ADO.NET 要求を IBM® OLE DB Provider に送信する (COM interop モジュールによって) ブリッジ・プロバイダーです。 ODBC .NET Data Provider は、ADO.NET 要求を IBM ODBC ドライバーに送信するブリッジ・プロバイダーです。これらの .NET データ・プロバイダーは、DB2 ファミリー・データベースへのアクセスにはお勧めしません。 IBM Data Server Provider for .NET は、ハイパフォーマンスの、管理された ADO.NET データ・プロバイダーです。 DB2 ファミリー・データベースとともに使用する場合は、この .NET データ・プロバイダーがお勧めです。 IBM Data Server Provider for .NET を使用する ADO.NET データベース・アクセスは、OLE DB および ODBC .NET ブリッジ・プロバイダーと比べて、制限が少なく、極めて良好なパフォーマンスを提供します。

.NET アプリケーションのデプロイ (Windows)

.NET アプリケーションのデプロイメントを簡単に行えるように、IBM では IBM Data Server Driver Package を用意しています。これは、大規模なデプロイメント・シナリオで使用するのに理想的な、スモール・フットプリント・クライアントです。 IBM Data Server Driver Package のフィーチャー以外の追加フィーチャーが必要な場合は、代わりとして IBM Data Server Runtime Client を使用できます。

始める前に

- デプロイメントの前に、.NET アプリケーションを構築する必要があります。これは、Visual Studio またはコマンド行のいずれかを使用して行います。 .NET アプリケーションの構築について詳しくは、関連タスクを参照してください。
- .NET アプリケーションの構築に使用するコンピューター、また .NET アプリケーションをデプロイするコンピューターには、 3 ページの『サポートされる .NET 開発ソフトウェア』で説明されているように、サポートされるバージョンの Windows オペレーティング・システムおよび他のソフトウェアが必要です。
  - 構築システム
    - Windows オペレーティング・システム
    - Visual Studio
    - .NET Framework 再頒布可能パッケージ
    - .NET Framework Software Development Kit
  - デプロイメント・システム
    - Windows オペレーティング・システム
    - .NET Framework 再頒布可能パッケージ

手順

.NET アプリケーションをデプロイするには、以下のようにします。

1. IBM Data Server Driver Package を、アプリケーションのデプロイ先のコンピューターにインストールします。インストール中に、 IBM Data Server Driver Package のインストールをデフォルト・データベース・クライアント・インターフェース・コピーに設定します。

   注: IBM データ・サーバーに対して実行される既存のデータベース・アプリケーションがあれば、それらはこの新しくインストールされた IBM Data Server Driver Package を使用します。デプロイした .NET アプリケーションをロールアウトする前に、それらのアプリケーションを新規ドライバーに対してテストします。
2. 構築したアプリケーションを、それが実行されるコンピューターにインストールします。

サポートされる .NET 開発ソフトウェア

IBM データ・サーバーに対して実行する .NET アプリケーションを開発してデプロイするには、サポートされている開発ソフトウェアとオペレーティング・システムを使用する必要があります。

.NET Framework 2.0、3.0、3.5、および 4.0 アプリケーションの開発とデプロイのためにサポートされているオペレーティング・システム

- Windows XP、Service Pack 2 (32 ビット版、64 ビット版)
- Windows Server 2003 (32 ビット版、64 ビット版)
- Windows Vista (32 ビット版、64 ビット版)
- Windows Server 2008 (32 ビット版、64 ビット版)
- Windows Server 2008 R2 (64 ビット版)
- Windows 7 (32 ビット版および 64 ビット版)

.NET Framework アプリケーション用にサポートされる開発ソフトウェア

.NET Framework アプリケーションを開発するには、IBM Data Server Client または Driver Package のほかに、以下に挙げるサポートされているツールのいずれかが必要です。

- Visual Studio 2008
- Visual Studio 2010

.NET Framework アプリケーション用にサポートされるデプロイメント・ソフトウェア

.NET Framework アプリケーションをデプロイするには、IBM Data Server Client または Driver Package のほかに、以下に挙げるサポートされるパッケージのいずれかが必要です。ほとんどの場合、以下のうちの 1 つが Windows インストールに含まれています。

- .NET Framework Version 2.0 再頒布可能パッケージ
- .NET Framework Version 3.0 再頒布可能パッケージ

- .NET Framework Version 3.5 再頒布可能パッケージ
- .NET Framework Version 4.0 再頒布可能パッケージ

Visual Studio での DB2 統合

Visual Studio 用の IBM Database Add-In は、Visual Studio 開発環境にシームレスに統合するための機能の集合で、これにより DB2 サーバーと連携して、DB2 プロシージャー、関数、およびオブジェクトを開発できます。

Visual Studio 用の IBM Database Add-In は、DB2 データベースに簡単なインターフェースを提供します。例えば、SQL を使用するのではなく、デザイナーとウィザードを使用してデータベース・オブジェクトの作成を行うことができます。また SQL コードを作成する必要がある場合には、組み込まれている DB2 SQL エディターに以下の機能が備えられています。

- より読みやすくするために色付けされた SQL テキスト
- DB2 スクリプトの入力時のインテリジェント・オートコンプリートを提供するための Microsoft Visual Studio IntelliSense 機能との統合

IBM Database Add-Ins for Visual Studio を使用すると、以下のことが可能です。

- 各種の DB2 開発ツールや管理ツールのオープン
- ソリューション・エクスプローラーでの DB2 プロジェクトの作成および管理
- サーバー・エクスプローラーからの DB2 データ接続のアクセスと管理
- ストアード・プロシージャー、関数、表、ビュー、索引、およびトリガーを作成するスクリプトなどの DB2 スクリプトの作成および変更

Visual Studio 2008 および 2010

Visual Studio 用の IBM Database Add-In は、インストール可能な別個のコンポーネントとして DB2 Client および DB2 サーバーに組み込まれています。 DB2 製品のインストールを終えると、Visual Studio 用の IBM Database Add-In をインストールするためのオプションが表示されます。コンピューター上に Visual Studio をインストールしていない場合には、このアドインはインストールされません。 Visual Studio をインストールすると、DB2 製品のセットアップ・メニューからいつでもアドインをインストールできます。

迅速なアプリケーション開発を可能にするために IBM Database Add-In と Data Server Provider for .NET を使用する方法について詳しくは、IBM Information Management and Visual Studio .NET ゾーン (http://www.ibm.com/developerworks/data/zones/vstudio/index.html) にアクセスしてください。

第 2 章 外部ルーチン

外部ルーチンは、データベース外部の、データベース・サーバーのファイル・システムに存在するプログラミング言語アプリケーションにロジックをインプリメントしたルーチンです。

ルーチンと外部コード・アプリケーションとの関連付けは、ルーチンの CREATE ステートメントの EXTERNAL 節で宣言します。

作成できるのは、外部プロシージャー、外部関数、および外部メソッドです。これらすべては外部プログラミング言語でインプリメントされていますが、それぞれのルーチンの機能タイプには異なるフィーチャーがあります。外部ルーチンのインプリメントを決定する前に、『外部ルーチンの概説』のトピックを読むことによって、外部ルーチンの種類、またそれがどのようにインプリメントされ、どのように使用できるかを理解しておくことが大切です。その知識を基にして、データベース環境内で外部ルーチンを使うタイミングとその使い方に関して十分な情報を得た上での決定が行えるように、関連リンクの宛先のトピックから外部ルーチンの詳細を学習することができます。

ルーチン使用の利点

以下の利点を活用するには、ルーチンを使用します。

SQL インターフェースからの呼び出しが可能なアプリケーション・ロジックのカプセル化 多数の異なるクライアント・アプリケーションが共通の要件を持つ環境では、ルーチンを効果的に使用することによって、コードの再利用、標準化、保守の作業を単純化できます。ルーチンを使用している環境内であれば、共通アプリケーションの動作の一面を変更する必要が生じた場合に、その動作をカプセル化した該当のルーチンを変更するだけで済みます。ルーチンがない場合は、各アプリケーションごとにそのロジックを変更する必要があります。

他のデータベース・オブジェクトへのアクセスの制御
ルーチンを使用して、データベース・オブジェクトへのアクセスを制御することができます。例えば、CREATE TABLE などの特定の SQL ステートメントを発行する権限が基本的に与えられていないユーザーにも、そのステートメントの 1 つ以上の特定インプリメンテーションを含んだルーチンを呼び出す権限を与えることができます。こうして特権をカプセル化することで特権の管理が単純化されます。

ネットワーク・トラフィックの削減によるアプリケーション・パフォーマンスの向上 クライアント・コンピューターでアプリケーションを実行する場合、各 SQL ステートメントは別々にクライアント・コンピューターからデータベース・サーバー・コンピューターに送信されて実行され、結果セットも別々に戻されます。その結果、ネットワーク・トラフィックが肥大化してしまいます。ユーザーとの対話をほとんど必要とせず、データベースとの多くの対話を必要とする処理があれば、ネットワーク・トラフィックの量を最小化す

るためにその処理をサーバー上にインストールしたり、より強力なデータベース・サーバー上で作業を実行できるようにするという処置には意味があります。

SQL の実行の高速化と効率化

ルーチンはデータベース・オブジェクトなので、クライアント・アプリケーションよりも SQL の要求とデータを効率的に送信できます。このため、SQL ステートメントはクライアント・アプリケーションで実行するよりも、ルーチン内で実行する方が高いパフォーマンスを発揮します。 NOT FENCED 節を使って作成されるルーチンは、データベース・マネージャーと同じプロセスで実行されるので、通信に共有メモリーを使用でき、結果としてアプリケーション・パフォーマンスが向上します。

別のプログラミング言語でインプリメントされたロジックとのインターオペラビリティーの実現

コード・モジュールは、それぞれのプログラマーが別々のプログラミング言語でインプリメントできます。また、一般的にコードはできる限り再利用することが望ましいので、DB2 のルーチンは、高度なインターオペラビリティーをサポートしています。

- 1 つのプログラム言語のクライアント・アプリケーションから、別々のプログラム言語でインプリメントしたルーチンを呼び出すことができます。例えば、C クライアント・アプリケーションから、.NET 共通言語ランタイム・ルーチンを呼び出すことができます。
- ルーチンはそのタイプやインプリメンテーションに関係なく、別のルーチンを呼び出すことができます。例えば Java プロシージャーは、組み込み SQL スカラー関数を呼び出すことができます。
- 1 つのオペレーティング・システム上のデータベース・サーバーに作成したルーチンを、別のオペレーティング・システム上で実行する DB2 クライアントから呼び出せます。

これらの利点は、ルーチン使用の数ある利点の中のほんの一部にすぎません。データベース管理者、データベース設計者、データベース・アプリケーション開発者など、さまざまなユーザーがルーチンの使用によって恩恵を受けます。このような理由から、ルーチンには検討に値する多くの用途があります。

各種の機能要件に合わせたさまざまなタイプのルーチンがあり、それぞれのインプリメンテーションの方法もさまざまです。ルーチンのタイプとインプリメンテーションの選択によって、前述の利点をどの程度具体化できるかが決まる場合もあります。ルーチンは基本的に、ロジックをカプセル化するための強力な手法です。この手法を活用すれば、SQL を拡張し、アプリケーションの構造と保守作業を改善し、場合によってはアプリケーションのパフォーマンスを向上させることができます。

外部ルーチンのインプリメンテーション

外部ルーチンのインプリメンテーションは、データベースの外部に存在するプログラミング言語のコードによってルーチンのロジックが定義されるインプリメンテーションです。他のルーチンのインプリメンテーションと同様に、外部インプリメンテーションのルーチンも CREATE ステートメントを実行することによりデータベースに作成されます。

コンパイルされたライブラリーに保管されるルーチンのロジックは、データベース・サーバーの特別なディレクトリー・パスにあります。ルーチン名と外部コード・アプリケーションとの関連付けは、 CREATE ステートメントの EXTERNAL 節で宣言します。

外部ルーチンは、サポートされているどの外部ルーチン・プログラミング言語でも作成できます。 『外部ルーチンの開発でサポートされている API およびプログラミング言語』を参照してください。

外部ルーチンのインプリメンテーションは、SQL ルーチンのインプリメンテーションよりもいくらか複雑かもしれません。しかし、選択したインプリメンテーション・プログラミング言語の全機能とパフォーマンスを活用できるという点で非常に強力です。また、外部関数には、データベースの外部 (つまり、ネットワークやファイル・システムなど) に存在するエンティティーにアクセスして操作を実行できるというメリットもあります。 DB2 データベースとの対話はそれほど必要としないものの、大量のロジックや非常に複雑なロジックを組み込む必要があるルーチンの場合は、外部ルーチンのインプリメンテーションが望ましいと言えます。

例えば、VARCHAR データ・タイプを操作する新しいストリング関数や、DOUBLE データ・タイプを操作する複雑な数学関数など、組み込みデータ・タイプの利便性を活用する新しい関数をインプリメントするときには、外部ルーチンを使用するのが理想的です。さらに、外部ルーチンのインプリメンテーションは、E メール送信などの外部アクションを伴うロジックにも最適です。

データ・アクセスよりもプログラミング・ロジックを重視してロジックをカプセル化する必要がある場合、サポートされているいずれかの外部ルーチンのプログラミング言語によるプログラミングが苦にならないのであれば、外部インプリメンテーションのルーチンの作成手順をマスターした時点で、外部ルーチンがいかに強力かをすぐに実感できるはずです。

外部ルーチンの開発でサポートされている API およびプログラミング言語

以下の API および関連したプログラミング言語を使用して、DB2 外部ルーチン (プロシージャーおよび関数) を開発することができます。

- ADO.NET
  - .NET 共通言語ランタイム・プログラミング言語
- CLI
- 組み込み SQL
  - C
  - C++
  - COBOL (プロシージャーでのみサポート)
- JDBC
  - Java
- OLE
  - Visual Basic
  - Visual C++

- この API をサポートするその他のプログラミング言語。
- OLE DB (表関数でのみサポートされる)
  - この API をサポートするプログラミング言語。
- SQLJ
  - Java

外部ルーチンの開発でサポートされている API とプログラミング言語の比較

外部ルーチンのインプリメントを開始する前に、サポートされているさまざまな外部ルーチンのアプリケーション・プログラミング・インターフェース (API) およびプログラミング言語の特性および制限について考慮することは重要です。これによって最初から正しいインプリメンテーションを選ぶことができ、必要なルーチンのフィーチャーを使用することができます。

表 1. 外部ルーチン API とプログラミング言語の比較

API とプログラミング言語	フィーチャー・サポート	パフォーマンス	セキュリティー	スケーラビリティー	制限
SQL (SQL PL を含む)	• SQL は、容易に学習および使用できるハイレベル言語であり、インプリメンテーションを素早く実行できるようにします。 • SQL プロシージャー型言語 (SQL PL) エレメントは、SQL 操作および照会において制御フロー・ロジックを許可します。	• 非常に良い。 • SQL ルーチンは、Java ルーチンよりも高いパフォーマンスを実現します。 • SQL ルーチンのパフォーマンスは、NOT FENCED 節で作成される C および C++ 外部ルーチンと同程度です。	• 非常に安全。 • SQL プロシージャーは、常にデータベース・マネージャーと同じメモリーで稼働します。これは、キーワード NOT FENCED を使ってデフォルトで作成されるルーチンに対応します。	• 高いスケーラビリティー。	• データベース・サーバーのファイル・システムにアクセスできません。 • データベースの外部に常駐するアプリケーションを呼び出すことはできません。

表 1. 外部ルーチン API とプログラミング言語の比較 (続き)

API とプログラミング言語	フィーチャー・サポート	パフォーマンス	セキュリティー	スケーラビリティー	制限
組み込み SQL (C および C++ を含む)	• 低いレベルではあるが、強力なプログラミング言語。	• 非常に良い。 • C および C++ ルーチンは、Java ルーチンよりも高いパフォーマンスを実現します。 • NOT FENCED 節で作成される C および C++ 外部ルーチンのパフォーマンスは、SQL ルーチンと同程度です。	• C および C++ ルーチンではプログラミング・エラーが生じやすくなります。 • プログラマーは、ルーチンのインプリメンテーションを単調にして時間を浪費させる共通メモリーおよびポインター操作のエラーを避けるため、C に熟達している必要があります。 • C および C++ ルーチンは、データベース・マネージャーの混乱によって実行時にルーチン内で例外が発生するのを避けるため、FENCED 節および NOT THREADSAFE 節を使用して作成しなければなりません。これらはデフォルトの節です。これらの節を使用することで、パフォーマンスにいくらか悪い影響が出る場合がありますが、安全な実行が保証されます。 ルーチンのセキュリティーを参照してください。	• C および C++ ルーチンが FENCED 節および NOT THREADSAFE 節で作成されている場合、スケーラビリティーは削減されます。これらのルーチンは、データベース・マネージャー・プロセスとは別に、分離した **db2fmp** プロセス内で実行されます。**db2fmp** プロセスは、並行して実行されるルーチンごとに必要になります。	• サポートされるパラメーター引き渡しスタイルは複数存在するので、混乱してしまう可能性があります。ユーザーは、可能な限りパラメーター・スタイル SQL を使用する必要があります。

表 1. 外部ルーチン API とプログラミング言語の比較 (続き)

API とプログラミング言語	フィーチャー・サポート	パフォーマンス	セキュリティー	スケーラビリティー	制限
組み込み SQL (COBOL)	• ビジネス (通常はファイルを取り扱う) アプリケーションの開発に適した高水準プログラミング言語。 • これまでは実動ビジネス・アプリケーション用に広く使用されてきました。しかし、その普及度は減少しています。 • COBOL は、ポインター、および、再帰的な呼び出しをサポートしていないプログラミング言語です。	• COBOL ルーチンは、他の外部ルーチンのインプリメンテーション・オプションを指定して作成されたルーチンと比較してパフォーマンスが劣ります。	• 現時点で情報はありません。	• 現時点で情報はありません。	• 32 ビットの COBOL プロシージャーを 64 ビットの DB2 インスタンス内に作成して呼び出すことができますが、これらのルーチンは 64 ビットの DB2 インスタンス内にある 64 ビットの COBOL プロシージャーと比較してパフォーマンスが劣ります。
JDBC (Java) および SQLJ (Java)	• スタンドアロン・アプリケーション、アプレット、およびサーブレットの開発に適した、高水準のオブジェクト指向プログラミング言語。 • Java オブジェクトおよびデータ・タイプは、データベース接続の確立、SQL ステートメントの実行、およびデータの操作を容易にします。	• Java ルーチンは、C および C++ ルーチンまたは SQL ルーチンと比較してパフォーマンスが劣ります。	• Java ルーチンでは、危険操作の制御は Java 仮想マシン (JVM) によって担われるので、Java ルーチンのほうが C および C++ ルーチンよりも安全です。これにより、信頼性は向上し、1 つの Java ルーチンのコードが、同じプロセス内で実行中の別のルーチンに悪影響を与えることはほとんどありません。	• 優れたスケーラビリティー • FENCED THREADSAFE 節で作成された Java ルーチン (デフォルト) は、スケーラビリティーに優れています。fenced の Java ルーチンはすべて、いくつかの JVM を共有します。特定の db2fmp プロセスの Java ヒープが使い果たされると、システムでは複数の JVM が起用されるからです。	• 危険性を含んだ操作を避けるために、 Java ルーチンから Java Native Interface (JNI) 呼び出しを行うことはできないことになっています。

表 1. 外部ルーチン API とプログラミング言語の比較 (続き)

API とプログラミング言語	フィーチャー・サポート	パフォーマンス	セキュリティー	スケーラビリティー	制限
.NET 共通言語ランタイムのサポートされる言語 (C#、Visual Basic、およびその他を含む)	• 管理対象コードの Microsoft .NET モデルの一部です。 • ソース・コードは Microsoft .NET Framework 共通言語ランタイムで解釈できる中間言語 (IL) バイト・コードにコンパイルされます。 • CLR アセンブリーは、別の .NET プログラム言語のソース・コードからコンパイルしたサブアセンブリーからでもビルドできます。つまり、ユーザーとしては、さまざまな言語で作成したコード・モジュールの再利用と統合が可能になります。	• CLR ルーチンは、実行時にデータベース・マネージャーの割り込みの可能性を最小限に抑えるために、FENCED NOT THREADSAFE 節でのみ作成できます。これにより、パフォーマンスにいくらか悪影響が出る可能性があります。 • デフォルト節値を使用するなら、実行時にデータベース・マネージャーによる割り込みが起きる可能性は小さくなりますが、CLR ルーチンは FENCED として実行する必要があるため、NOT FENCED として指定できる他の外部ルーチンと比べるとわずかに実行速度が遅くなる可能性があります。	• CLR ルーチンは、FENCED NOT THREADSAFE 節でのみ作成できます。このルーチンはデータベース・マネージャーの外の、別個の **db2fmp** プロセスで実行されるので、安全性が確保されます。	• 入手可能な情報はありません。	• 『.NET CLR ルーチンに関する制約事項』のトピックを参照してください。

表 1. 外部ルーチン API とプログラミング言語の比較 (続き)

API とプログラミング言語	フィーチャー・サポート	パフォーマンス	セキュリティー	スケーラビリティー	制限
• OLE	• OLE ルーチンは、Visual C++、Visual Basic、および OLE でサポートされているその他の言語でインプリメントすることができます。	• OLE 自動化ルーチンの速度は、インプリメントに使用する言語によって異なります。一般的にこのルーチンは、OLE C/C++ 以外のルーチンよりも遅いです。 • OLE ルーチンは、FENCED NOT THREADSAFE モードでのみ実行できるため、OLE 自動化ルーチンはスケーラビリティーにはあまり優れていません。	• 入手可能な情報はありません。	• 入手可能な情報はありません。	• 入手可能な情報はありません。

表 1. 外部ルーチン API とプログラミング言語の比較 (続き)

API とプログラミング言語	フィーチャー・サポート	パフォーマンス	セキュリティー	スケーラビリティー	制限
• OLE DB	• OLE DB は、ユーザー定義の表関数の作成に使用できます。 • OLE DB 関数は、外部の OLE DB データ・ソースに接続します。	• OLE DB 関数のパフォーマンスは、OLE DB Provider によって異なります。ただし、一般に OLE DB 関数は、論理的に同等な Java 関数よりパフォーマンスが優れていますが、論理的に同等な C、C++、または SQL 関数より速度が遅くなります。ただし関数が呼び出される場所である照会内の特定の述部を OLE DB Provider で評価することができるので、DB2 データベース・システムが処理しなければならない行数は減ります。多くの場合、それによってパフォーマンスが向上することになります。	• 入手可能な情報はありません。	• 入手可能な情報はありません。	• OLE DB は、ユーザー定義の表関数の作成にのみ使用できます。

外部ルーチンのフィーチャー

外部ルーチンは、一般的なルーチン・フィーチャーの大多数のサポート、および SQL ルーチンでサポートされない追加フィーチャーのサポートを提供します。

以下のフィーチャーは、外部ルーチンには固有のものです。

データベースの外部に常駐するファイル、データ、およびアプリケーションへのアクセス 外部ルーチンは、データベース自体の外部に常駐するデータまたはファイルにアクセスしてそれらを操作することができます。データベースの外部に常駐するアプリケーションを呼び出すこともできます。例えば、データ、ファイル、またはアプリケーションがデータベース・サーバーのファイル・システム内や使用できるネットワーク内に存在することがあります。

さまざまな外部ルーチンのパラメーター・スタイル・オプション

プログラミング言語での外部ルーチンのインプリメンテーションは、パラメーター・スタイルの選択を使用することによって行うことができます。選択したプログラミング言語の、希望するパラメーター・スタイルが存在する場合でも、時折、選択が存在します。一部のパラメーター・スタイルは、ルーチン・ロジック内で役立つ可能性のある、dbinfo 構造という構造内のルーチンとの間の追加データベースおよびルーチンのプロパティー情報の引き渡しをサポートします。

スクラッチパッドを使用した次回の外部関数の呼び出しまでの状態の保存

外部ユーザー定義関数は、値のセットに対する次回の関数呼び出しまでの状態の保存のサポートを提供します。これは、スクラッチパッドと呼ばれる構造を使用して行われます。これは、集約値を戻す関数と、バッファーの初期化といったロジックの初期セットアップを必要とする関数の両方において便利です。

呼び出しタイプは個別の外部関数の呼び出しを識別する

外部ユーザー定義関数は、値のセットに対して複数回呼び出されます。それぞれの呼び出しは、関数ロジック内で参照可能な呼び出しタイプの値で識別されます。例えば、ある関数の最初の呼び出し、データのフェッチ呼び出し、および最終呼び出しに関して特別な呼び出しタイプが存在します。呼び出しタイプは、特定のロジックを特定の呼び出しタイプに関連付けることができるため便利です。

外部スカラー関数

外部スカラー関数は、外部のプログラム言語でロジックをインプリメントしたスカラー関数です。

こうした関数を作成して、既存の SQL 関数のセットを拡張するために使用できますし、DB2 組み込み関数 (例えば、LENGTH や COUNT) と同じやり方で呼び出すことができます。つまり、SQL ステートメント内で式が有効な場所であればどこからでも参照できるということです。

外部スカラー関数のロジックの実行は DB2 データベース・サーバーで行われます。また、組み込み関数やユーザー定義の SQL スカラー関数とは異なり、外部関数のロジックはデータベース・サーバー・ファイル・システムへのアクセス、システム呼び出しの実行、またはネットワークへのアクセスが可能です。

外部スカラー関数は、 SQL データの読み取りはできますが、その変更は行えません。

外部スカラー関数は、 1 つの関数参照で何度も呼び出すことができ、スクラッチパッド (メモリー・バッファー) の使用によって、その呼び出しと呼び出しの間で状態を維持することができます。このような機能は、最初のセットアップ・ロジックが複雑な場合に特に便利です。セットアップ・ロジックを最初の呼び出しで実行するときに、スクラッチパッドを使用して、スカラー関数のそれ以降の呼び出しでアクセスまたは更新するいくつかの値を保管できるからです。

外部スカラー関数のフィーチャー

- SQL ステートメント内で式がサポートされている場所であればどこからでも参照できます。

- スカラー関数の出力は、呼び出し元の SQL ステートメントによって直接処理できます。
- 外部スカラー・ユーザー定義関数の場合は、関数の反復呼び出しの際に、スクラッチパッドを使用して、呼び出しと呼び出しの間で状態を維持できます。
- サーバーで実行されるので、述部で使用するときのパフォーマンスが高くなります。サーバーで関数を候補行に対して適用できる場合は、クライアント・マシンに行を送信する前の時点でその行を考慮の対象から除外できる場合が多いので、サーバーからクライアントに渡す必要のあるデータ量を削減できます。

制限

- スカラー関数内ではトランザクション管理を行えません。つまり、スカラー関数内では COMMIT や ROLLBACK を発行できません。
- 結果セットを戻すことはできません。
- スカラー関数は、入力セットごとに 1 つのスカラー値を戻すようになっています。
- 外部スカラー関数は、一度の呼び出しによる使用を想定していません。むしろ、関数に対する 1 つの参照と 1 つの入力セットを用意して、各入力ごとに関数を一度ずつ呼び出し、そのたびに関数から 1 つのスカラー値が戻される、という設計になっています。スカラー関数を作成するときには、最初の呼び出しで一部のセットアップ作業を行い、その後で呼び出し時にアクセスできる一部の情報を保管するように設計できます。一度の呼び出しだけを必要とする機能には、SQL スカラー関数のほうが適しています。
- 単一パーティション・データベースでは、外部スカラー関数に SQL ステートメントを含めることができます。これらのステートメントは、表のデータの読み取りは行えますが、その変更はできません。データベースに複数のパーティションがある場合、外部スカラー関数に SQL ステートメントを含めることはできません。 SQL スカラー関数は、データの読み取りや変更を行う SQL ステートメントを含めることができます。

一般的な使用法

- DB2 組み込み関数のセットを拡張します。
- 本来 SQL は実行できない SQL ステートメント内のロジックを実行します。
- 副照会としてよく再利用されるスカラー照会を SQL ステートメント内でカプセル化します。例えば、郵便番号を例にあげると、郵便番号が掲載されている都市の表を検索します。

サポートされている言語

- C
- C++
- Java
- OLE
- .NET 共通言語ランタイム言語

注:

1. 集約関数を作成するための機能は限定されています。列関数ともいう集約関数は、一連の類似値 (データ列) を受け取って、1 つの応答を戻します。ユーザー定義集約関数を作成できるのは、組み込み集約関数をソースとする場合だけです。例えば、基本タイプ INTEGER に基づいて特殊タイプ SHOESIZE を定義してある場合は、既存の組み込み集約関数 AVG(INTEGER) をソースとして、AVG(SHOESIZE) という関数を集約関数として定義できます。
2. また、行を戻す関数を作成することもできます。これは、行関数と呼ばれますが、構造化タイプ用の transform 関数としてのみ使用することができます。行関数の出力は単一行です。

外部スカラー関数およびメソッドの処理モデル

FINAL CALL 指定を使用して定義されたメソッドおよびスカラー UDF の処理モデルは以下のとおりです。

FIRST 呼び出し

これは特殊ケースの NORMAL 呼び出しですが、関数を使用して任意の初期処理を実行できるようにするための「最初」の呼び出しを表します。引数が評価されてから、関数に渡されます。通常、この呼び出しでは関数から値が戻されますが、エラーが戻される場合もあります。後者の場合は NORMAL または FINAL 呼び出しは行われません。 FIRST 呼び出しでエラーが戻された場合は、FINAL 呼び出しは行われないため、メソッドまたは UDF は、戻る前に終結処理を行う必要があります。

NORMAL 呼び出し

これは、ステートメントのデータとロジックで示されているとおり、関数の 2 番目から最後から 2 番目までのすべての呼び出しを指します。どの NORMAL 呼び出しでも、引数が評価されてから渡された後で関数から値が戻されることになっています。 NORMAL 呼び出しでエラーが戻された場合、それ以上 NORMAL 呼び出しは行われずに、FINAL 呼び出しが行われます。

FINAL 呼び出し

これは、ステートメントの終わりの処理 (またはカーソルのクローズ) の時点で行われる特殊な呼び出しです。ただし、 FIRST 呼び出しが正常に完了していることを前提とします。 FINAL 呼び出しでは引数値は渡されません。この呼び出しが行われるのは、関数がすべてのリソースを終結処理できるようにするためです。この呼び出しでは関数は値を戻しませんが、エラーを戻すことはあります。

FINAL CALL を指定して定義されていないメソッドまたはスカラー UDF の場合、関数への NORMAL 呼び出しのみが行われ、その場合は通常は、各呼び出しの値が戻されます。 NORMAL 呼び出しでエラーが戻された場合や、ステートメントで別のエラーが生じた場合、その関数に対してはそれ以上呼び出しは行われません。

注: このモデルは、メソッドおよびスカラー UDF の通常のエラー処理を説明しています。システム障害や通信問題が発生した場合、エラー処理モデルによって指示された呼び出しが行われないことがあります。例えば、FENCED UDF の場合、db2udf fenced 処理が何らかの原因で早く終了してしまうと、 DB2 は指示された呼び出しを行うことができません。

外部表関数

ユーザー定義表関数は、表を参照している SQL にその表を引き渡します。

表 UDF 参照は、SELECT ステートメントの FROM 節内でのみ有効です。表関数を使用する際は、次のことに注意してください。

- 表関数は表を送達しますが、DB2 データベース・システムと UDF の間の物理インターフェースは 1 行ずつ行われます。表関数への呼び出しには、OPEN、FETCH、CLOSE、 FIRST、および FINAL の 5 タイプがあります。FIRST および FINAL 呼び出しがあるかどうかは、UDF の定義方法によって決まります。これらの呼び出しの判別には、スカラー関数で使用されるのと同じ*呼び出しタイプ* 機構が使用されます。
- 表関数の CREATE FUNCTION ステートメントの RETURNS 節で定義されたすべての結果列を、戻さなければならないというわけではありません。 CREATE FUNCTION の DBINFO キーワード、および対応する dbinfo 引数によって、特定の表関数参照に必要な列だけを戻すよう最適化できます。
- 戻される個々の列値は、スカラー関数が戻す値と同じ書式です。
- 表関数の CREATE FUNCTION ステートメントには、 CARDINALITY 指定があります。これを指定することにより、定義者は DB2 オプティマイザーに結果の概算のサイズを通知することができ、それによってオプティマイザーは関数が参照されるときにより適切な決定を下すことができます。

  表関数の CARDINALITY として指定された値と関係なく、カーディナリティーが無限の関数、つまり、 FETCH 呼び出しの際に常に行を戻す関数を定義しないよう注意してください。 DB2 データベース・システムでは、照会処理内の触媒として end-of-table 条件を想定する状況が多くあります。GROUP BY や ORDER BY を使用している場合などがそうです。 DB2 データベース・システムは、end-of-table に到達するまで、集合用のグループを作成せず、またすべてのデータがそろうまで、ソートを行うことはできません。そのため、end-of-table 条件 (SQL 状態値 '02000') を決して戻さない表関数では、それを GROUP BY や ORDER BY 節で使用すると、無限処理ループが生じることがあります。

外部表関数の処理モデル

FINAL CALL 指定を使用して定義された表 UDF の処理モデルは以下のとおりです。

FIRST 呼び出し

この呼び出しは最初の OPEN 呼び出しの前に行いますが、その目的は、関数がすべての初期処理を実行できるようにすることにあります。この呼び出しの前に、スクラッチパッドがクリアされます。引数が評価されてから、関数に渡されます。この関数は行を戻しません。この関数がエラーを戻した場合、それ以降この関数への呼び出しは行われません。

OPEN 呼び出し

この呼び出しが行われるのは、スキャンに固有の特別な OPEN 処理を関数で実行できるようにするためです。この呼び出しの前にスクラッチパッド (ある場合) がクリアされることはありません。引数が評価されてから引き渡されます。この関数は、OPEN 呼び出しで行を戻すことはありません。この

関数が OPEN 呼び出しでエラーを戻した場合、FETCH または CLOSE 呼び出しは行われませんが、ステートメントの終わりで FINAL 呼び出しは行われます。

FETCH 呼び出し
FETCH 呼び出しは、表の終わりを意味する SQLSTATE 値が関数から戻されるまで継続して行われます。 UDF は、この呼び出しに対応して、データ行を開発して戻すことになります。引数値が関数に渡されることがありますが、その値は、OPEN のときに渡されたのと同じ値を指しています。したがって、この引数値は現行値でない可能性もあるので、信用することはできません。表関数の次回の呼び出しまで現行値をそのまま維持している必要がある場合、スクラッチパッドを使用してください。この関数は、FETCH 呼び出しでエラーを戻すことはありますが、その場合でも CLOSE 呼び出しを行うことはできます。

CLOSE 呼び出し
この呼び出しが行われるのは、スキャンまたはステートメントの終了時点です。ただし、 OPEN 呼び出しが正常に完了していることを前提とします。どの引数値も現行値ではありません。この関数はエラーを戻すことがあります。

FINAL 呼び出し
FINAL 呼び出しが行われるのは、ステートメントの終了時点です。ただし、 FIRST 呼び出しが正常に完了していることを前提とします。この呼び出しが行われるのは、関数がすべてのリソースを終結処理できるようにするためです。この呼び出しでは関数は値を戻しませんが、エラーを戻すことはあります。

FINAL CALL を指定して定義されていない表 UDF の場合、関数への OPEN、FETCH、および CLOSE 呼び出しのみが行われます。どの OPEN 呼び出しでも、その前にスクラッチパッド (ある場合) がクリアされます。

FINAL CALL を指定して定義された表 UDF と、NO FINAL CALL を指定して定義されたものの違いは、表関数アクセスは「内部寄りの」アクセスとなる結合または副照会に関連したシナリオを見れば明らかになります。例えば、次のようなステートメントがあるとします。

```
SELECT x,y,z,... FROM table_1 as A,
  TABLE(table_func_1(A.col1,...)) as B
  WHERE ...
```

この場合オプティマイザーは、table_1 の各行ごとに table_func_1 のスキャンをオープンします。それは、table_1 の col1 の値 (これが table_func_1 に渡されます) が使用されて表関数スキャンが定義されるからです。

NO FINAL CALL の表 UDF の場合、 table_1 の各行ごとに OPEN、FETCH、FETCH、...、CLOSE の呼び出しシーケンスが繰り返されます。なお、OPEN 呼び出しのたびに、新しいスクラッチパッドが支給されることに注意してください。スキャンの終了時点ごとにさらに別のスキャンがあるかどうかは表関数には分からないので、表関数は、CLOSE 処理中に完全な終結処理を実行する必要があります。そのため、繰り返しの必要な 1 回だけオープンされる重要な処理がある場合には効率が悪くなってしまいます。

FINAL CALL の表 UDF は、一回限りの FIRST 呼び出しおよび一回限りの FINAL 呼び出しの手段になります。これらの呼び出しを使用すれば、表関数のすべてのスキャンを通して初期化と終了の手間を軽減することができます。これまでと同様に外側の表の各行ごとに OPEN、FETCH、FETCH、...、CLOSE の呼び出しは行われますが、 FINAL 呼び出しが出されることが表関数には分かっているので、表関数は CLOSE 呼び出しの時点で一切終結処理 (およびその後の OPEN での再割り振り) をする必要はなくなります。また、表関数リソースは複数のスキャンを対象とすることが主な原因ですが、スキャンのたびにスクラッチパッドがクリアされることはないことにも注意してください。

表 UDF を使用すれば、2 つのさらに別の呼び出しタイプを管理する必要はありますが、その代償として、上記の結合と副照会のシナリオに示されているような大幅な効率化を実現することができます。表関数を FINAL CALL と定義するべきかどうかは、予定している使用法によって決まります。

外部関数とメソッドのスクラッチパッド

スクラッチパッド を使用すれば、次の呼び出し時までユーザー定義関数またはメソッドの状態を保持しておくことができます。

例えば、以下に次回の呼び出しまで状態を保持しておけば便利であることを示す例を 2 つ示します。

1. 正確に言えば、保管状態に依存する関数またはメソッド。

   このような関数またはメソッドの例として、最初の呼び出し時に「1」を戻し、2 回目以降の呼び出しごとに結果を 1 ずつ増分する単純な counter 関数があります。この関数を使用すると、特定の状況下では次のように SELECT 結果の行数を数えることができます。

   ```
   SELECT counter(), a, b+c, ...
     FROM tablex
     WHERE ...
   ```

   関数には、複数の呼び出しにまたがってカウンターの現行値を保管する場所が必要です。それによって、後続の呼び出しでも必ず同じ値が確保されます。その後の呼び出しごとにその値は増加されて、関数の結果として戻されます。

   このタイプのルーチンは限定的なルーチンではありません。つまりその出力が、その SQL 引数の値にのみ依存することはありません。

2. 特定の初期化アクションを実行する機能によってパフォーマンスを改善できる関数またはメソッド。

   このような関数またはメソッドの例として、文書アプリケーションの一部を成す *match* 関数があります。これは、特定の文書に特定のストリングが入っていれば「Y」を、入っていなければ「N」を戻します。

   ```
   SELECT docid, doctitle, docauthor
     FROM docs
     WHERE match('myocardial infarction', docid) = 'Y'
   ```

   このステートメントは、最初の引数で表される特定のテキスト・ストリング値を含む文書すべてを戻します。 *match* が行うことは以下のとおりです。

   - 最初の処理に限り、以下を実行します。

ストリング myocardial infarction が入っていて、しかも DB2 データベース・システムの外部で保存されているすべての文書 ID のリストを文書アプリケーションで検索します。この検索は処理に負荷がかかる処理であるため、関数はこの処理を 1 回だけ行い、検索したリストをその後の呼び出しでの使用に利用しやすい場所に保管します。

- 各呼び出し時には、以下を実行します。

  この最初の呼び出しで保管された文書 ID のリストを用いて、 2 番目の引数として渡された文書 ID がこのリストに載っているかどうかを確認します。

このタイプのルーチンは限定的ルーチンです。その応答は、入力される引数値にのみ依存します。上記の関数は、ある呼び出しから次の呼び出しへ情報を保管できるかどうかによってパフォーマンス (正確さではない) が左右されます。

以下のように、CREATE ステートメントに SCRATCHPAD を指定すれば、上述の 2 つの要件は両方とも満たされます。

```
CREATE FUNCTION counter()
  RETURNS int ... SCRATCHPAD;

CREATE FUNCTION match(varchar(200), char(15))
  RETURNS char(1) ... SCRATCHPAD 10000;
```

SCRATCHPAD キーワードは、ルーチン用のスクラッチパッドを割り振って保持するよう DB2 データベース・システムに指示します。スクラッチパッドのデフォルトのサイズは 100 バイトですが、スクラッチパッド・サイズ (バイト数) を指定することができます。 *match* の例は 10000 バイトの長さです。 DB2 データベース・システムは、スクラッチパッドを最初の呼び出し前のバイナリー数のゼロに初期化します。表関数のスクラッチパッドが定義されている場合に、 NO FINAL CALL (デフォルト) を使用してその表関数が定義されていると、 DB2 データベース・システムは各 OPEN 呼び出しの前にスクラッチパッドをリフレッシュします。表関数オプション FINAL CALL を指定すると、 DB2 データベース・システムは、初期化後のスクラッチパッドの内容を検査も変更もしません。スクラッチパッドを使用して定義されたスカラー関数の場合も、 DB2 データベース・システムは初期化後のスクラッチパッドの内容を検査も変更もしません。各呼び出しごとにスクラッチパッドを指すポインターがルーチンに渡され、 DB2 データベース・システムはそのルーチンの状態情報をスクラッチパッド内に保存します。

したがって *counter* の例の場合、最後に戻された値がスクラッチパッドに保管されます。また *match* の例では、スクラッチパッドが十分に大きい場合は文書のリストをスクラッチパッドに保管し、十分に大きくない場合はリスト用にメモリーを割り振って、取得したメモリーのアドレスをスクラッチパッドに保存します。スクラッチパッドは可変長にすることができます。その長さは、ルーチンの CREATE ステートメント内に定義します。

スクラッチパッドが適用されるのは、ステートメント内のルーチンへの個々の参照に対してのみです。ステートメント内のルーチンに対して複数の参照がある場合、どの参照にもそれ独自のスクラッチパッドがあることになるので、参照同士が互いに通信しあうのにスクラッチパッドを使用することはできません。スクラッチパッドは、単一の DB2 エージェント (エージェントとは、ステートメントのあらゆる側面の処理を実行する DB2 エンティティーのことです) に対してのみ適用されます。エージェント同士がスクラッチパッド情報を共有するのを調整するための「グロー

バル・スクラッチパッド」はありません。このことは、ステートメントを処理するエージェントが DB2 データベース・システムによって複数確立される (単一パーティションまたはマルチパーティション・データベースのどちらかで) 場合は特に重要です。そのような場合、ステートメント内のルーチンへの参照は 1 つしかない場合でも、作業を行うエージェントは複数存在していて、そのおのおのが独自のスクラッチパッドをもつことになります。マルチパーティション・データベースでは、UDF を参照するステートメントは、複数のパーティション上のデータを処理し、各パーティション上で UDF を呼び出しますが、スクラッチパッドは 1 つのパーティションにしか適用されません。結果として、 UDF が実行されるパーティションごとにスクラッチパッドが 1 つずつ存在することになります。

関数が正しく実行されるかどうかが、その関数への参照ごとに 1 つのスクラッチパッドがあるかどうかで決まる場合、その関数を DISALLOW PARALLEL として登録します。これで、関数は 1 つのパーティションでしか実行されなくなるので、関数への 1 つの参照につき必ず 1 つのスクラッチパッドしか存在しないようにすることができます。

UDF またはメソッドは、システム・リソースを必要とする場合があるので、 UDF またはメソッドを定義するときに FINAL CALL キーワードを使用すると便利です。このキーワードは、ステートメント処理の終了時点で UDF またはメソッドを呼び出すよう DB2 データベース・システムに指示するので、 UDF またはメソッドはそのシステム・リソースを解放することができます。ルーチンは獲得したすべてのリソースを解放することが不可欠です。ステートメントが繰り返し呼び出される環境では、小さい不手際は大きい不手際につながることがあり、大きい不手際は DB2 データベースが破壊される原因になることがあります。

スクラッチパッドのサイズは固定されているので、UDF またはメソッド自体にメモリーの割り振りを組み込むことにより、最終呼び出しを利用してメモリーを解放するのも一案です。例えば上記の *match* 関数は、特定のテキスト・ストリングと一致する文書がどのくらいあるかを予測できません。したがって、*match* の定義は次のように行うとよいでしょう。

```
CREATE FUNCTION match(varchar(200), char(15))
  RETURNS char(1) ... SCRATCHPAD 10000 FINAL CALL;
```

UDF またはメソッドがスクラッチパッドを使用していてしかも副照会で参照される場合、もし、UDF またはメソッドに最終呼び出しが指定されているならば、副照会から次の副照会までの間に DB2 データベース・システムは最終呼び出しを行って、スクラッチパッドの内容をリフレッシュすることを決定する場合があります。UDF またはメソッドを副照会で使用している場合は、 FINAL CALL や呼び出しタイプ引数を使用して UDF またはメソッドを定義するか、またはスクラッチパッドの *バイナリー数のゼロ* 状態を必ず検査すれば、リフレッシュが起きないようにすることができます。

FINAL CALL を指定する場合は、UDF またはメソッドがタイプ FIRST の呼び出しを受け取ることに注意してください。これは、永続リソースを獲得して初期化するために使用することができます。

以下は、スクラッチパッドを使って列の項目の平方和を計算する UDF の単純な Java の例です。この例は、列を取り込み、列の先頭から現在行の項目までの累積平方和を含む列を戻します。

```
CREATE FUNCTION SumOfSquares(INTEGER)
RETURNS INTEGER
EXTERNAL NAME 'UDFsrv!SumOfSquares'
DETERMINISTIC
NO EXTERNAL ACTION
FENCED
NOT NULL CALL
LANGUAGE JAVA
PARAMETER STYLE DB2GENERAL
NO SQL
SCRATCHPAD 10
FINAL CALL
DISALLOW PARALLEL
NO DBINFO@


// Sum Of Squares using Scratchpad UDF
public void SumOfSquares(int inColumn,
                         int outSum)
throws Exception
{
  int sum = 0;
  byte[] scratchpad = getScratchpad();

  // variables to read from SCRATCHPAD area
  ByteArrayInputStream byteArrayIn = new ByteArrayInputStream(scratchpad);
  DataInputStream dataIn = new DataInputStream(byteArrayIn);

  // variables to write into SCRATCHPAD area
  byte[] byteArrayCounter;
  int i;
  ByteArrayOutputStream byteArrayOut = new ByteArrayOutputStream(10);
  DataOutputStream dataOut = new DataOutputStream(byteArrayOut);

  switch(getCallType())
  {
    case SQLUDF_FIRST_CALL:
      // initialize data
   sum = (inColumn * inColumn);
      // save data into SCRATCHPAD area
      dataOut.writeInt(sum);
      byteArrayCounter = byteArrayOut.toByteArray();
      for(i = 0; i < byteArrayCounter.length; i++)
      {
        scratchpad[i] = byteArrayCounter[i];
      }
      setScratchpad(scratchpad);
   break;
    case SQLUDF_NORMAL_CALL:
      // read data from SCRATCHPAD area
      sum = dataIn.readInt();
      // work with data
      sum = sum + (inColumn * inColumn);
      // save data into SCRATCHPAD area
      dataOut.writeInt(sum);
      byteArrayCounter = byteArrayOut.toByteArray();
      for(i = 0; i < byteArrayCounter.length; i++)
      {
        scratchpad[i] = byteArrayCounter[i];
      }
      setScratchpad(scratchpad);
break;
  }
  //set the output value
  set(2, sum);
} // SumOfSquares UDF
```

SumOfSquares UDF と同じタスクを実行する組み込み DB2 関数があることに注意してください。この例が選ばれたのは、スクラッチパッドの使用を示すためです。

32 ビット・オペレーティング・システムおよび 64 ビット・オペレーティング・システムでのスクラッチパッド

UDF またはメソッドのコードを 32 ビットと 64 ビットのオペレーティング・システムで相互に移植できるようにするには、 64 ビット値の入ったスクラッチパッドを作成および使用する仕方に気を付ける必要があります。 64 ビット・ポインターや sqlint64 BIGINT 変数などの 1 つ以上の 64 ビット値の入ったスクラッチパッド構造では、明示的な長さ変数を宣言しないようお勧めします。

以下は、スクラッチパッドの構造宣言のサンプルです。

```
struct sql_scratchpad
{
   sqlint32 length;
   char data[100];
};
```

スクラッチパッドの独自の構造の定義では、ルーチンには次のような 2 つの選択肢があります。

1. スクラッチパッド sql_scratchpad 全体を再定義します。この場合、明示的な長さフィールドを組み込む必要があります。以下に例を示します。

   ```
   struct sql_spad
   {
     sqlint32 length;
     sqlint32 int_var;
     sqlint64 bigint_var;
   };
   void SQL_API_FN routine( ..., struct sql_spad* scratchpad, ... )
   {
     /* Use scratchpad */
   }
   ```

2. スクラッチパッド sql_scratchpad のデータ部分だけを再定義します。この場合、長さフィールドは必要ありません。

   ```
   struct spaddata
   {
     sqlint32 int_var;
     sqlint64 bigint_var;
   };
   void SQL_API_FN routine( ..., struct sql_scratchpad* spad, ... )
   {
     struct spaddata* scratchpad = (struct spaddata*)spad->data;
     /* Use scratchpad */
   }
   ```

アプリケーションは、スクラッチパッド内の長さフィールドの値を変更できないので、最初の例に示されているようにルーチンをコーディングしてもあまり有益ではありません。また 2 番目の例も、それぞれ異なるワード・サイズのコンピューター同士で相互に移植できるので、ルーチンを作成するにはこちらのほうがより望ましい方法です。

外部ルーチンの SQL

外部プログラム言語 (C、Visual Basic、C#、Java など) で書かれたどのルーチン内でも SQL を使用することができます。

ルーチン (ストアード・プロシージャー、UDF) の場合の CREATE ステートメントと、メソッドの場合の CREATE TYPE ステートメントには、ルーチンまたはメソッドの SQL アクセス・レベルを定義する節を組み込みます。ルーチンに組み込まれた SQL の特性に基づいて、以下のようなアプリケーション節を選択しなければなりません。

NO SQL
ルーチンには SQL はまったく入りません。

CONTAINS SQL
SQL は入りますが、データの読み取りや書き込みは行いません (例えば、SET SPECIAL REGISTER)。

READS SQL DATA
表からの読み取りを行う SQL は入ります (SELECT、VALUES ステートメント) が、表データは変更しません。

MODIFIES SQL DATA
表を更新する SQL が入ります。これは、ユーザー表を直接 (INSERT、UPDATE、 DELETE ステートメント) または DB2 のカタログ表を暗黙で (DDL ステートメント) 更新します。この節を使用できるのは、ストアード・プロシージャーと SQL 形式の表関数だけです。

実行時に DB2 データベース・システムは、定義されたレベルをルーチンが超えていないかどうかを確認します。例えば、CONTAINS SQL と定義されたルーチンが表からの選択を試みると、 SQL データを読み取ろうとするのでエラー (SQLCODE -579、SQLSTATE 38004) になります。また、ネストされたルーチン参照も、参照を含む同じ SQL レベルか、より厳密な SQL レベルでなければなりません。例えば、SQL データの変更を行うルーチンは、 SQL データの読み取りを行うルーチンを呼び出せますが、 SQL データの読み取り専用のルーチン (READS SQL DATA 節を定義したルーチン) は、 SQL データの変更を行うルーチンを呼び出せません。

ルーチンは、呼び出し元アプリケーションのデータベース接続の有効範囲内で SQL ステートメントを実行します。ルーチンは独自の接続を確立したり、呼び出し元アプリケーションの接続をリセットしたりすることはできません (SQLCODE -751、SQLSTATE 38003)。

MODIFIES SQL DATA と定義されたストアード・プロシージャーだけが、COMMIT および ROLLBACK ステートメントを発行することができます。他のタイプのルーチン (UDF とメソッド) は、 COMMIT も ROLLBACK も発行できません (SQLCODE -751、SQLSTATE 38003)。 MODIFIES SQL DATA と定義されたストアード・プロシージャーはトランザクションのコミットまたはロールバックを行うことはできますが、 COMMIT または ROLLBACK は呼び出し元のアプリケーションから発行して、変更が不用意にコミットされないようにすることをお勧めします。データベースに対してタイプ 2 接続を確立しているアプリケーションからストアード・プロシージャーが呼び出された場合、そのストアード・プロシージャーは COMMIT または ROLLBACK ステートメントを発行することはできません。

また、MODIFIES SQL DATA と定義されたストアード・プロシージャーだけが、独自のセーブポイントを確立して、そのセーブポイント内の独自の作業をロールバックすることができます。他のタイプのルーチン (UDF とメソッド) は、独自のセーブポイントを確立できません。ストアード・プロシージャー内に作成されたセー

ブポイントは、そのストアード・プロシージャーが完了しても解放されません。アプリケーションはそのセーブポイントをロールバックすることができます。同様に、ストアード・プロシージャーも、アプリケーションで定義されたセーブポイントをロールバックすることができます。 DB2 データベース・システムは、ルーチンによって確立されたすべてのセーブポイントを戻る時に暗黙で解放します。

ルーチンは、DB2 から渡された sqlstate 引数に SQLSTATE 値を割り当てることで、正常に完了したかどうかを DB2 データベース・システムに知らせることができます。一部のパラメーター・スタイル (PARAMETER STYLEs JAVA、GENERAL、および GENERAL WITH NULLS) は、SQLSTATE 値の交換をサポートしていません。

ルーチンによって発行された SQL の取り扱いで DB2 データベース・システムにエラーが発生した場合、他のどのアプリケーションに対しても同様に、そのエラーはルーチンに戻されます。通常のユーザー・エラーの場合、ルーチンは選択に応じて代わりのアクションまたは訂正アクションをとることができます。例えば、ルーチンが表への挿入を試みたときに、重複キー・エラー (SQLCODE-813) が戻された場合、選択を行って代わりにその表の既存行を更新することができます。

ただし、DB2 データベース・システムが通常のやり方で先に進むのを妨げるようなもっと重大なエラーが生じることもあります。例えば、デッドロック、データベース・パーティションの障害、またはユーザー割り込みなどがその一例です。このようなエラーの一部は、呼び出し元のアプリケーションまで伝搬されます。作業単位に関連したその他の重大エラーは、トランザクション制御ステートメント (COMMIT または ROLLBACK) の発行を許可された (a) アプリケーションまたは (b) ストアード・プロシージャーのうちの、バックアウトでどちらか先に発生したほうまで到達します。

このようなエラーのいずれかが、ルーチンから発行された SQL の実行中に起きた場合、エラーはルーチンに返されますが、重大エラーが起きたことが DB2 データベース・システムに記憶されます。その場合はさらに、そのルーチンおよびすべての呼び出しルーチンからそれ以後に発行されたすべての SQL は DB2 データベース・システムによって自動的に失敗させられます (SQLCODE -20139、SQLSTATE 51038)。これに対する唯一の例外は、トランザクション制御ステートメントの発行を許可されている最も外側のストアード・プロシージャーにまでしかエラーがバックアウトされない場合です。その場合、そのストアード・プロシージャーは SQL を引き続き発行することができます。

ルーチンは静的および動的の両方の SQL を発行することができますが、どちらの場合も、組み込み SQL を使用するのであればその SQL をプリコンパイルしてバインドする必要があります。静的 SQL の場合にプリコンパイル/バインドのプロセスで使用される情報は、組み込み SQL を使用する他のすべてのクライアント・アプリケーションの場合と同じです。動的 SQL の場合は、**DYNAMICRULES** プリコンパイル/ BIND オプションを使用して、組み込み動的 SQL の現在のスキーマと現在の認証 ID を制御することができます。この動作は、ルーチンとアプリケーションとでは異なります。

ルーチンのパッケージまたはステートメントに対して定義されている分離レベルが順守されます。それに応じて、ルーチンが実行される分離レベルは、呼び出し元のアプリケーションよりも厳密にも寛容にもなります。このことには、呼び出し元の

アプリケーションよりも厳密さの低い分離レベルをもつルーチンを呼び出すときには配慮することが大切です。例えば、反復可能読み取りアプリケーションからカーソル固定関数を呼び出した場合、 UDF は非反復可能読み取り特性を示すことがあります。

呼び出し側のアプリケーションまたはルーチンは、特殊レジスター値に対してルーチンが加えた変更によって影響を受けることはありません。更新可能な特殊レジスターは、呼び出し側からルーチンへと継承されます。更新可能な特殊レジスターに加えられた変更は、呼び出し側には戻されません。更新不能の特殊レジスターには、独自のデフォルト値が与えられます。更新可能および更新不能の特殊レジスターの詳細は、「特殊レジスター」という関連項を参照してください。

ルーチンは、クライアント・アプリケーションと同じやり方でカーソルの OPEN、FETCH、および CLOSE を行うことができます。同じ関数を複数回呼び出す (再帰の場合など) と、そのつど独自のカーソル・インスタンスが与えられます。 UDF とメソッドは、ステートメント呼び出しの完了前にカーソルをクローズする必要があります。そうしないと、エラーが起きます (SQLCODE -472、SQLSTATE 24517)。 UDF またはメソッドを最後に呼び出したときに、オープンしたままになっているすべてのカーソルをクローズするのがよいと思われます。オープンしたままのカーソルは、ストアード・プロシージャーの完了の前にクローズされないと、クライアント・アプリケーションまたは呼び出し側ルーチンに結果セットとして戻されます。

ルーチンに渡された引数が、自動的にホスト変数として扱われることはありません。つまり、ルーチンが SQL 内でホスト変数としてパラメーターを使用するには、独自のホスト変数を宣言して、パラメーター値をそのホスト変数にコピーする必要があるということです。

注: 組み込み SQL ルーチンの場合、**DATETIME** オプションを ISO に設定したうえでプリコンパイルしてバインドする必要があります。

外部ルーチンのパラメーター・スタイル

外部ルーチンのインプリメンテーションは、ルーチンのパラメーター値の交換のための特定の規則に準拠していなければなりません。そのような規則をパラメーター・スタイル と呼びます。

外部ルーチンのパラメーター・スタイルは、PARAMETER STYLE 節を指定してルーチンが作成されると指定されます。パラメーター・スタイルは、パラメーター値が外部ルーチンのインプリメンテーションに渡される仕様および順序を示します。さらに、パラメーター・スタイルは、追加の値が外部ルーチンのインプリメンテーションに渡されたときに取られる動作も指定します。例えば、パラメーター・スタイルの中には、各ルーチンのパラメーター値ごとに、追加された個々の NULL 標識値がルーチンのインプリメンテーションに渡されることを指定するものもあります。これにより、パラメーターの NULL 可能性に関する情報が提供されます (これがないと、ネイティブのプログラミング言語のデータ・タイプで判別することは容易ではありません)。

次の表では、使用可能なパラメーター・スタイル、各パラメーター・スタイルをサポートするルーチンのインプリメンテーション、各パラメーター・スタイルをサポートするルーチンの機能タイプ、およびパラメーター・スタイルの説明のリストを記載しています。

表 2. パラメーター・スタイル

パラメーター・スタイル	サポートされる言語	サポートされるルーチン・タイプ	説明
SQL [1]	• C/C++ • OLE • .NET 共通言語ランタイム言語 • COBOL [2]	• UDF • ストアード・プロシージャー • メソッド	呼び出し時に渡されるパラメーターに加えて、以下の引数が以下に示されている順序でルーチンに渡されます。 • CREATE ステートメント内で宣言された各パラメーターまたは結果ごとの NULL 標識。 • DB2 データベース・システムに戻される SQLSTATE。 • ルーチンの修飾名。 • 個々のルーチン名。 • DB2 データベース・システムに戻される SQL 診断ストリング。 CREATE ステートメントとルーチン・タイプに指定されているオプションに応じて、以下の引数を以下に示されている順序でルーチンに渡すことができます。 • スクラッチパッドのバッファー。 • ルーチンの呼び出しタイプ。 • dbinfo 構造 (データベースに関する情報が入っています)。
DB2SQL [1]	• C/C++ • OLE • .NET 共通言語ランタイム言語 • COBOL	• ストアード・プロシージャー	呼び出し時に渡されるパラメーターに加えて、以下の引数が以下に示されている順序でストアード・プロシージャーに渡されます。 • CALL ステートメント上の各パラメーターごとの NULL 標識の入ったベクトル。 • DB2 データベース・システムに戻される SQLSTATE。 • ストアード・プロシージャーの修飾名。 • 個々のストアード・プロシージャー名。 • DB2 データベース・システムに戻される SQL 診断ストリング。 CREATE PROCEDURE ステートメント内で DBINFO 節を指定すると、 dbinfo 構造 (データベースに関する情報が入っています) がストアード・プロシージャーに渡されます。

表 2. パラメーター・スタイル (続き)

パラメーター・スタイル	サポートされる言語	サポートされるルーチン・タイプ	説明
JAVA	• Java	• UDF • ストアード・プロシージャー	PARAMETER STYLE JAVA ルーチンは、 Java 言語と SQLJ ルーチンの仕様に準拠したパラメーター引き渡し規則に従います。 ストアード・プロシージャーの場合には INOUT および OUT パラメーターは、値を戻しやすくするために単一の項目配列として渡されます。ストアード・プロシージャー用の Java メソッド・シグニチャーには、 IN、OUT、および INOUT パラメーターのほかに、CREATE PROCEDURE ステートメントの DYNAMIC RESULT SETS 節に指定されている各結果セットごとにタイプ ResultSet[] のパラメーターが組み込まれています。 PARAMETER STYLE JAVA の UDF とメソッドの場合、ルーチンの呼び出しに指定されたもの以外の追加引数は渡されません。 PARAMETER STYLE JAVA ルーチンは、DBINFO または PROGRAM TYPE 節をサポートしていません。 UDF の場合、PARAMETER STYLE JAVA を指定できるのは、パラメーターとして構造化データ・タイプを指定しておらず、かつ戻りタイプとして構造化タイプ、CLOB、DBCLOB、または BLOB データ・タイプを指定していない場合だけです (SQLSTATE 429B8)。また、PARAMETER STYLE JAVA UDF は表関数、呼び出しタイプ、またはスクラッチパッドをサポートしていません。
DB2GENERAL	• Java	• UDF • ストアード・プロシージャー • メソッド	このタイプのルーチンは、 Java メソッドで使用するように定義されたパラメーター引き渡し規則に従います。表 UDF やスクラッチパッド付きの UDF を開発したり、 dbinfo 構造にアクセスする必要があったりしない限り、PARAMETER STYLE JAVA を使用することをお勧めします。 PARAMETER STYLE DB2GENERAL ルーチンの場合、ルーチンの呼び出しに指定されたもの以外の追加引数は渡されません。
GENERAL	• C/C++ • .NET 共通言語ランタイム言語 • COBOL	• ストアード・プロシージャー	PARAMETER STYLE GENERAL ストアード・プロシージャーは、呼び出し元のアプリケーションまたはルーチン内の CALL ステートメントからパラメーターを受け取ります。 CREATE PROCEDURE ステートメント内で DBINFO 節を指定すると、dbinfo 構造 (データベースに関する情報が入っています) がストアード・プロシージャーに渡されます。 GENERAL は、 DB2 for z/OS® の SIMPLE ストアード・プロシージャーと同等です。

表 2. パラメーター・スタイル (続き)

パラメーター・スタイル	サポートされる言語	サポートされるルーチン・タイプ	説明
GENERAL WITH NULLS	• C/C++ • .NET 共通言語ランタイム言語 • COBOL	• ストアード・プロシージャー	PARAMETER STYLE GENERAL WITH NULLS ストアード・プロシージャーは、呼び出し元のアプリケーションまたはルーチン内の CALL ステートメントからパラメーターを受け取ります。 CALL ステートメント上の各パラメーターごとの NULL 標識の入ったベクトルもその中に含まれます。 CREATE PROCEDURE ステートメント内で DBINFO 節を指定すると、 dbinfo 構造 (データベースに関する情報が入っています) がストアード・プロシージャーに渡されます。 GENERAL WITH NULLS は、 DB2 for z/OS の SIMPLE WITH NULLS ストアード・プロシージャーと同等です。

注:

1. UDF およびメソッドの場合、PARAMETER STYLE SQL は PARAMETER STYLE DB2SQL と同等です。
2. COBOL を使用できるのは、ストアード・プロシージャーの開発でのみです。
3. .NET 共通言語ランタイム・メソッドはサポートされていません。

ルーチン開発に関するパフォーマンスの考慮

クライアント・アプリケーションを拡張する代わりにルーチンを開発することの大きな利点の 1 つは、パフォーマンスです。

ルーチンのインプリメンテーション用のアプローチを選択するときには、次のようなパフォーマンス上の影響に配慮してください。

NOT FENCED モード

NOT FENCED ルーチンは、データベース・マネージャーと同じプロセス中で稼働します。一般的に、ルーチンを NOT FENCED で実行したほうが、FENCED モードで実行する場合よりもパフォーマンスが改良されます。FENCED ルーチンは、エンジンのアドレス・スペース外部の特別な DB2 プロセスで実行されるからです。

ルーチンを NOT FENCED モードで実行すればルーチンのパフォーマンスの向上は期待できますが、ユーザー・コードによってデータベースやデータベース制御構造が無意識または意識的に破壊される可能性があります。NOT FENCED ルーチンを使用してよいのは、パフォーマンス上の利点を最大化する必要がある場合と、ルーチンをセキュア化する必要がある場合だけです。 (C/C++ ルーチンを NOT FENCED として登録する場合のリスクの評価とその緩和に関する詳細は、『ルーチンのセキュリティーに関する考慮事項』の項を参照してください。) データベース・マネージャーのプロセスで実行するにはルーチンが十分安全でない場合、ルーチンの登録時に FENCED 節を使用します。安全でない可能性があるコードの作成および実行を制限するために、DB2 データベース・システムでは、ユーザーが NOT FENCED ルーチンを作成するには、特殊権限 CREATE_NOT_FENCED_ROUTINE を持っていなければなりません。

NOT FENCED ルーチンの実行中に異常終了が発生する場合、ルーチンが NO SQL として登録されていると、データベース・マネージャーは適切なリカバリーを試行します。しかし、NO SQL として定義されていないルーチンの場合、データベース・マネージャーは失敗します。

ルーチンが GRAPHIC または DBCLOB データを使用する場合は、NOT FENCED ルーチンを WCHARTYPE NOCONVERT オプションでプリコンパイルする必要があります。

FENCED THREADSAFE モード

FENCED THREADSAFE ルーチンは、他のルーチンと同じプロセスで稼働します。具体的には、Java 以外のルーチンはあるプロセスを共有し、Java ルーチンは他の言語で作成されたルーチンとは分離した、別のプロセスを共有します。この分離により、Java ルーチンは、他の言語で作成された、エラーを起こしやすいルーチンから保護されます。また、Java ルーチンのプロセスには JVM が含まれています。これは、メモリー・コストが高くなり、他のルーチン・タイプでは使用されません。 FENCED THREADSAFE ルーチンの複数の呼び出しではリソースを共有するため、それぞれが独自の専用プロセスで実行する FENCED NOT THREADSAFE ルーチンよりもシステムのオーバーヘッドが減ります。

ご使用のルーチンが他のルーチンと同じプロセスで実行しても安全であると感じる場合、それを登録する際に THREADSAFE 節を使用してください。NOT FENCED ルーチンと同様に、C/C++ ルーチンを FENCED THREADSAFE として登録するリスクの評価およびその軽減の詳細については、『ルーチンのセキュリティーに関する考慮事項』のトピックを参照してください。

FENCED THREADSAFE ルーチンが異常終了した場合、このルーチンを実行していたスレッドだけが終了します。プロセス内のその他のルーチンは実行を続けます。ただし、このスレッドが異常終了する原因となった障害によって、プロセス中の他のルーチン・スレッドが悪影響を受けて、トラップ、ハング、またはデータ損傷を起こす可能性があります。あるスレッドが異常終了した後は、そのプロセスは新規のルーチンの呼び出しに使用されません。すべてのアクティブ・ユーザーがこのプロセスでジョブを完了すると、それは終了されます。

Java ルーチンを登録する際に、特に指定されない限り、THREADSAFE であると見なされます。その他の LANGUAGE タイプはすべて、デフォルトで NOT THREADSAFE です。 LANGUAGE OLE および OLE DB を使用するルーチンは THREADSAFE として指定できません。

NOT FENCED ルーチンは THREADSAFE でなければなりません。ルーチンを NOT FENCED NOT THREADSAFE として登録することはできません (SQLCODE -104)。

UNIX のユーザーは、`db2fmp (Java)` または `db2fmp (C)` を探すことにより、Java および C の THREADSAFE プロセスを参照できます。

FENCED NOT THREADSAFE モード

FENCED NOT THREADSAFE ルーチンはそれぞれ、独自の専用プロセスで実行します。多数のルーチンを実行している場合、このことはデータベース・システムのパフォーマンスに悪影響を及ぼす可能性があります。ルーチ

ンが他のルーチンと同じプロセスで実行できるほど安全でない場合は、ルーチンを登録する際に NOT THREADSAFE 節を使用してください。

UNIX では、NOT THREADSAFE プロセスは db2fmp (pid) (pid は fenced モード・プロセスを使用するエージェントのプロセス ID) またはプール NOT THREADSAFE db2fmp の場合は db2fmp (idle) として表示されます。

Java ルーチン

メモリーの所要量の大きい Java ルーチンを実行する予定の場合、そのルーチンを FENCED NOT THREADSAFE と登録することをお勧めします。FENCED THREADSAFE Java ルーチンの呼び出しの場合、DB2 データベース・システムは、ルーチンを十分に実行できる大きさの Java ヒープを持つ、スレッド化された Java fenced モードのプロセスを選択しようとします。独自のプロセス内で大量のヒープを消費するプロセスを分離しないと、マルチスレッドの Java db2fmp プロセスで Java ヒープ不足エラーが生じる可能性があります。このカテゴリーに当てはまらない Java ルーチンの場合、少数の JVM を共有できるスレッド・セーフ・モードにしたほうが FENCED ルーチンの実行は向上します。

NOT FENCED Java ルーチンは現在サポートされていません。 NOT FENCED として定義される Java ルーチンは、FENCED THREADSAFE として定義されているかのように呼び出されます。

C/C++ ルーチン

一般的に、C または C++ ルーチンのほうが Java ルーチンよりも高速ですが、エラー、メモリーの破壊、および破損の可能性が高くなります。というわけで、メモリー操作を実行する機能のために、 THREADSAFE または NOT FENCED モードの登録では C または C++ ルーチンはリスクの高い候補となります。そのようなリスクを軽減するには、セキュア・ルーチンのためのプログラミングの実践方法を順守 (『ルーチンのセキュリティーに関する考慮事項』の項を参照) して、ルーチンを徹底的にテストします。

SQL ルーチン

一般的に、SQL ルーチン、特に SQL プロシージャーのほうが Java ルーチンよりもやはり高速であり、通常は C ルーチンに匹敵するパフォーマンスを備えています。SQL ルーチンは常に NOT FENCED モードで稼働するので、外部ルーチンよりもパフォーマンス上の利点はさらに大きくなります。複雑なロジックを組み込んだ UDF は、一般的に、 SQL で作成するよりも C で作成するほうが実行速度が上がります。ロジックが単純な場合、SQL UDF は外部 UDF に匹敵します。

スクラッチパッド

スクラッチパッドとは、UDF およびメソッドへの割り当てが可能なメモリー・ブロックのことです。スクラッチパッドが適用されるのは、SQL ステートメント内のルーチンへの個々の参照に対してのみです。ステートメント内のルーチンに対して複数の参照がある場合、どの参照もそれ独自のスクラッチパッドをもつことになります。スクラッチパッドを使用すれば、 UDF またはメソッドは次の呼び出し時までその状態を保持しておくことができます。

複雑な初期化を伴う UDF およびメソッドの場合はスクラッチパッドを使用すれば、最初の呼び出しで必要であったすべての値を保管しておいて、以後

のすべての呼び出しでそれを使用することができます。他の UDF やメソッドのロジックでも、呼び出しと呼び出しの間で中間値の保管が必要になる場合があります。

CHAR パラメーターにとって代わる VARCHAR パラメーターの使用
ルーチン定義で CHAR パラメーターの代わりに VARCHAR パラメーターを使用すれば、ルーチンのパフォーマンスを向上させることができます。CHAR データ・タイプではなく VARCHAR データ・タイプを使用すると、パラメーターの引き渡しの前に DB2 データベース・システムによってパラメーターにスペースが埋め込まれなくなります。これで、ネットワークを経由したパラメーターの転送に要する時間が短縮されます。

例えば、クライアント・アプリケーションが CHAR(200) パラメーターを予期するルーチンにストリング "A SHORT STRING" を渡す場合、DB2 データベース・システムはパラメーターに 186 個のスペースを埋め込み、ストリングを NULL で終了してから、200 文字のストリングと NULL 終止符全体をネットワーク経由でルーチンに送信する必要があります。

それと比べて、VARCHAR(200) パラメーターを予期するルーチンに同じストリング "A SHORT STRING" を渡すと、DB2 データベース・システムは単に 14 文字ストリングと NULL 終止符をネットワーク経由で渡します。

ルーチンのセキュリティーに関する考慮事項

ルーチンの開発および配置の作業は、データベース・アプリケーションのパフォーマンスと効率性を大幅に高める機会になります。ただし、データベース管理者がルーチンの配置を正しく管理しないと、セキュリティー上のリスクが生じる可能性もあります。

ここでは、セキュリティー上のリスクについてとそのようなリスクを軽減するための手段について説明します。セキュリティー・リスクの後には、セキュリティーを確認されていないルーチンを安全に配置する方法に関する項が続いています。

セキュリティー・リスク

NOT FENCED ルーチンは、データベース・マネージャー・リソースにアクセスすることができます。
NOT FENCED ルーチンは、データベース・マネージャーと同じプロセスで稼働します。 NOT FENCED ルーチンは、データベース・エンジンと密接なつながりをもっているため、データベース・マネージャーの共有メモリーを無意識または意識的に破壊したり、データベースの制御構造を損壊したりする可能性があります。どちらの支障の場合も、データベース・マネージャーが失敗することになります。また NOT FENCED ルーチンは、データベースとその表を破壊する可能性もあります。

データベース・マネージャーとそのデータベースの保全性を確保するには、NOT FENCED として登録する予定のルーチンを徹底的にスクリーニングする必要があります。そのようなルーチンは、全面的にテストおよびデバッグする必要がありますが、予測しきれない副次効果を示していてはなりません。ルーチンの検査では、メモリー管理と静的変数の使用に対して厳重な注意を払います。破損が生じる可能性が最も高いと言えるのは、コードがメモ

リーを適切に管理しない場合や静的変数を不正に使用する場合です。このような問題は、Java や .NET 以外のプログラミング言語でよく見られます。

NOT FENCED ルーチンを登録するには、
CREATE_NOT_FENCED_ROUTINE 権限が必要です。
CREATE_NOT_FENCED_ROUTINE 権限を付与する場合、付与された人はデータベース・マネージャーとそのすべてのリソースに無制限にアクセスできるようになることに注意してください。

FENCED THREADSAFE ルーチンは、他の FENCED THREADSAFE ルーチン内のメモリーにアクセスできます。

FENCED THREADSAFE ルーチンは、共有プロセス内のスレッドとして稼働します。このルーチンはいずれも、同一プロセス内の他のルーチン・スレッドによって使用されるメモリーを読み取ることができます。したがって、1 つのスレッド化ルーチンがスレッド化プロセス中の他のルーチンから機密データを収集することが可能になります。 1 つのプロセスを共有している場合、メモリー管理が徹底していないルーチン・スレッドが他のルーチン・スレッドを破壊したり、スレッド化プロセス全体が破損する原因になったりするという別のリスクも付随します。

他の FENCED THREADSAFE ルーチンの保全性を確保するには、
FENCED THREADSAFE として登録する予定のルーチンを徹底的にスクリーニングする必要があります。そのようなルーチンは、全面的にテストおよびデバッグする必要がありますが、予測しきれない副次効果を示していてはなりません。ルーチンの検査では、メモリー管理と静的変数の使用に対して厳重な注意を払います。それは、破損が起きる可能性の最も高い箇所であるからです。Java 以外の言語の場合は特にそうです。

FENCED THREADSAFE ルーチンを登録するには、
CREATE_EXTERNAL_ROUTINE 権限が必要です。
CREATE_EXTERNAL_ROUTINE 権限を付与する場合、付与された人は他の FENCED THREADSAFE ルーチンのメモリーをモニターまたは破壊できるようになることに注意してください。

fenced プロセスの所有者からのデータベース・サーバーへの書き込みアクセス権限は、データベース・マネージャーの破壊を起こす可能性があります。

fenced プロセスの実行に使用されるユーザー ID は、 **db2icrt** (インスタンスの作成) または **db2iupdt** (インスタンスの更新) システム・コマンドで定義されます。そのユーザー ID は、ルーチンのライブラリーとクラスが保管されているディレクトリー (UNIX 環境では `sqllib/function`、Windows 環境では `sqllib¥function`) への書き込みアクセス権限を持っていてはなりません。そのユーザー ID は、データベース・サーバー上のデータベースやオペレーティング・システムに対する読み取りまたは書き込みのアクセス権限ももっていてはなりません (それが不可能な場合は、少なくとも重要なファイルやディレクトリーに対する読み取りまたは書き込みのアクセス権限を与えないようにします)。

fenced プロセスの所有者がデータベース・サーバー上の各種のクリティカル・リソースへの書き込みアクセス権限を持っていると、システム破壊が起きる可能性があります。例えば、データベース管理者が、ルーチンを独自のプロセス内に囲い込み fenced することにより破壊の可能性を回避できると考えて、未知のソースから受け取った FENCED NOT THREADSAFE など

のルーチンを登録したとします。ただし、fenced プロセスを所有しているユーザー ID は、 sqllib/function ディレクトリーへの書き込みアクセス権限を持っているとします。ユーザーがこのルーチンを呼び出すと、ユーザーが気付かないうちに sqllib/function 内のライブラリーは、 NOT FENCED と登録されている代替バージョンのルーチン本体で上書きされてしまいます。後者のルーチンはデータベース・マネージャー全体への無制限のアクセス権限を持っているので、データベース表の機密情報の分散、データベースの破壊、認証情報の収集、またはデータベース・マネージャーの破壊を行う可能性があります。

fenced プロセスを所有するユーザー ID に、データベース・サーバー上の重要なファイルやディレクトリー (特に sqllib/function およびデータベースのデータ・ディレクトリー) への書き込みアクセス権限が付与されることがないようにしてください。

ルーチンのライブラリーおよびクラスのぜい弱性

ルーチンのライブラリーおよびクラスを保管しているディレクトリーへのアクセスが制御されていないと、ルーチンのライブラリーとクラスの削除や上書きが可能になってしまいます。前の項で述べたとおり、NOT FENCED ルーチン本体が好ましくない (またはコーディングに不備のある) ルーチンに置き換えられると、データベース・サーバーとそのリソースの安定性、保全性、およびプライバシーが著しく損なわれる可能性があります。

ルーチンの保全性を保護するには、ルーチンのライブラリーとクラスが置かれたディレクトリーへのアクセスを管理しなければなりません。できる限り少数のユーザーにしか、そのディレクトリーとファイルにアクセスできないようにしてください。そのようなディレクトリーへの書き込みアクセス権限を割り当てるときは、その権限によって該当ユーザー ID の所有者はデータベース・マネージャーとそのすべてのリソースに無制限にアクセスできるようになることに注意してください。

セキュリティーに疑いがあるルーチンの展開

未知のソースからルーチンを取得してしまった場合、その作成、登録、呼び出しを行う前に、必ずその機能を綿密に調べてください。そのルーチンを徹底的にテストして不測の副次効果が発生しないことを確認したのでない限り、FENCED および NOT THREADSAFE として登録することをお勧めします。

安全なルーチンの基準を満たさないルーチンを配置する必要が生じた場合、そのルーチンを FENCED および NOT THREADSAFE で登録します。データベースの保全性が必ず維持されるようにするには、 FENCED および NOT THREADSAFE ルーチンに関しては以下のとおりにしなければなりません。

- 他のルーチンと共有されない別個の DB2 プロセスで実行します。そうすれば、異常終了してもデータベース・マネージャーは影響を受けません。
- データベースによって使用されるメモリーとは別のメモリーを使用します。そうすれば、値の割り当てで間違いがあっても、データベース・マネージャーは影響を受けません。

ルーチンのコード・ページに関する考慮事項

文字データは、ルーチンの作成時に使用された PARAMETER CCSID オプションによって示されるコード・ページで外部ルーチンに渡されます。同様に、ルーチンから出力される文字ストリングも、 PARAMETER CCSID オプションによって示されるコード・ページを使用しているものとデータベースでは見なされます。

例えば、コード・ページ C を使用しているクライアント・プログラムが、コード・ページ S のセクションにアクセスし、そのセクションがコード・ページ R のルーチンを呼び出すとすると、以下のようになります。

1. SQL ステートメントを呼び出すと、入力文字データは、クライアント・アプリケーションのコード・ページ (C) からセクションのコード・ページ (S) に変換されます。 FOR BIT DATA として使用されるデータの BLOB の変換は行われません。
2. ルーチンのコード・ページがセクションのコード・ページと異なっている場合は、ルーチンが呼び出される前に、入力文字データ (BLOB と FOR BIT DATA を除く) がルーチンのコード・ページ (R) に変換されます。

   サーバー・ルーチンのプリコンパイル、コンパイル、バインドを実行するときには、ルーチンの呼び出し時に使用するコード・ページ (R) を使用することを強くお勧めします。ただしこれは、すべてのケースで可能であるとは限りません。例えば、Windows 環境では Unicode データベースを作成することができます。しかし Windows 環境に Unicode コード・ページがなければ、ルーチンを作成するアプリケーションを Windows のコード・ページでプリコンパイル、コンパイル、およびバインドする必要があります。プリコンパイラーが理解できない特殊な区切り文字がアプリケーションにない場合は、ルーチンは正常に作動します。
3. ルーチンが終了すると、データベース・マネージャーはすべての出力文字データを、必要に応じて、ルーチン・コード・ページ (R) からセクション・コード・ページ (S) へ変換します。実行中にルーチンでエラーが生じた場合のルーチンからの SQLSTATE と診断メッセージも、ルーチン・コード・ページからセクション・コード・ページに変換されます。 BLOB または FOR BIT DATA の文字ストリングでは変換は行われません。
4. ステートメントが終了すると、出力文字データはセクション・コード・ページ (S) から元のクライアント・アプリケーションのコード・ページ (C) に変換されます。 FOR BIT DATA として使用された BLOB またはデータの変換は行われません。

CREATE FUNCTION、CREATE PROCEDURE、および CREATE TYPE ステートメントで DBINFO オプションを使用すれば、ルーチンのコード・ページがルーチンに渡されます。この解説を参考にして、コード・ページを重視するルーチンを多種多様なコード・ページで機能するように作成することができます。

32 ビットおよび 64 ビット・アプリケーションおよびルーチンのサポート

DB2 Database for Linux, UNIX, and Windows は、さまざまなプラットフォームでのアプリケーションおよびルーチン (プロシージャーおよびユーザー定義関数 (UDF) を含む) の開発とデプロイメントのためのサポートを提供します。アプリケ

ーションおよびルーチンを正しく機能させるためには、DB2 データベース・サポートの 32 ビットおよび 64 ビットに関する考慮事項を検討し、それを理解することが重要です。

まず、いくつかの点を明確にしておきましょう。

- 32 ビット・ハードウェア・プラットフォームは 32 ビット・オペレーティング・システムを稼働し、64 ビット・ハードウェア・プラットフォームは 64 ビット・オペレーティング・システムを稼働します。
- DB2 データベースの 32 ビット・インスタンスは 32 ビット・オペレーティング・システムか 64 ビット・オペレーティング・システムのいずれかにインストールできますが、DB2 データベースの 64 ビット・インスタンスは 64 ビット・オペレーティング・システムにしかインストールできません。
- 32 ビット・アプリケーションは、32 ビット・オペレーティング・システムで構築されたアプリケーションのことを指します。
- 64 ビット・アプリケーションは、64 ビット・オペレーティング・システムで構築されたアプリケーションのことを指します。

以下の表は、クライアント・アプリケーションおよびルーチンに関する DB2 データベースの 32 ビットおよび 64 ビットのサポートを、次の推定とともに概略しています。

表 3. 32 ビットまたは 64 ビット・ハードウェア・プラットフォーム上での 32 ビットおよび 64 ビット・アプリケーションの実行に関するサポート

	32 ビット・ハードウェアおよびオペレーティング・システム	**64 ビット・ハードウェアおよびオペレーティング・システム**
32 ビット・アプリケーション	YES	YES
64 ビット・アプリケーション	NO	YES

以下の表は、DB2 クライアント・アプリケーションから DB2 データベース・サーバーへの接続の作成に関するサポートを示しています。

表 4. 32 ビットおよび 64 ビット・クライアントから 32 ビットおよび 64 ビット・サーバーへの接続に関するサポート

	32 ビット・サーバー	**64 ビット・サーバー**
32 ビット・クライアント	YES	YES
64 ビット・アプリケーション	YES	YES

表 5. 32 ビットまたは 64 ビット・ハードウェア・プラットフォーム上での 32 ビットおよび 64 ビット・アプリケーションの実行に関するサポート

	32 ビット・ハードウェアおよびオペレーティング・システム	**64 ビット・ハードウェアおよびオペレーティング・システム**
32 ビット・アプリケーション	YES	YES

表 5. 32 ビットまたは 64 ビット・ハードウェア・プラットフォーム上での 32 ビットおよび 64 ビット・アプリケーションの実行に関するサポート (続き)

	32 ビット・ハードウェアおよびオペレーティング・システム	**64 ビット・ハードウェアおよびオペレーティング・システム**
64 ビット・アプリケーション	NO	YES

以下の表は、DB2 クライアント・アプリケーションから DB2 データベース・サーバーへの接続の作成に関するサポートを示しています。

表 6. 32 ビットおよび 64 ビット・クライアントから 32 ビットおよび 64 ビット・サーバーへの接続に関するサポート

	32 ビット・サーバー	**64 ビット・サーバー**
32 ビット・クライアント	YES	YES
64 ビット・アプリケーション	YES	YES

表 7. 32 ビットおよび 64 ビット・サーバー上での fenced および unfenced プロシージャーおよび UDF の実行に関するサポート

	32 ビット・サーバー	**64 ビット・サーバー**
32 ビットの fenced プロシージャーまたは UDF	YES	YES[1、2、3]
64 ビットの fenced プロシージャーまたは UDF	NO	YES
32 ビットの unfenced プロシージャーまたは UDF	YES	NO[2]
64 ビットの unfenced プロシージャーまたは UDF	NO	YES

注:

1. 64 ビット・サーバー上での 32 ビット・プロシージャーの実行は低速になる場合があります。
2. 32 ビット・ルーチンを 64 ビット・サーバー上で機能させるには、これを FENCED および NOT THREADSAFE として作成する必要があります。
3. Linux/IA-64 データベース・サーバー上で 32 ビット・ルーチンを呼び出すことはできません。

外部ルーチンの 32 ビットと 64 ビットのサポート

32 ビットと 64 ビットの外部ルーチンのサポートは、ルーチンの CREATE ステートメントの FENCED 節または NOT FENCED 節という 2 つ節のいずれかの指定によって決定されます。

外部ルーチンのルーチン本体はプログラミング言語で作成され、ルーチンの呼び出し時にロードされて実行されるライブラリーまたはクラス・ファイルにコンパイルされます。 FENCED または NOT FENCED 節の指定は、データベース・マネージ

ャーとは異なる fenced 環境で外部ルーチンを実行するか、データベース・マネージャーと同じアドレッシング・スペースで実行するかを決定します。後者は、通信に TCPIP の代わりに共有メモリーが使用されるため、より高いパフォーマンスを提供することができます。デフォルトで、ルーチンは、選択される他の節に関係なく常に fenced として作成されます。

以下の表は、同じオペレーティング・システムで実行されている 32 ビットおよび 64 ビットのデータベース・サーバーで fenced および unfenced の 32 ビットおよび 64 ビット・ルーチンを実行するための DB2 データベース・システムのサポートを説明しています。

表 8. 32 ビットおよび 64 ビットの外部ルーチンのサポート

ルーチンのビット幅	32 ビット・サーバー	64 ビット・サーバー
32 ビットの fenced プロシージャーまたは UDF	サポートされる	サポートされる [1]
64 ビットの fenced プロシージャーまたは UDF	サポートされていない [3]	サポートされる
32 ビットの unfenced プロシージャーまたは UDF	サポートされる	サポートされる [1,2]
64 ビットの unfenced プロシージャーまたは UDF	サポートされていない [3]	サポートされる

注:

1. 64 ビット・サーバー上での 32 ビット・ルーチンの実行には 64 ビット・サーバー上での 64 ビット・ルーチンの実行ほどの実行速度はありません。
2. 32 ビット・ルーチンを 64 ビット・サーバー上で機能させるには、これを FENCED および NOT THREADSAFE として作成する必要があります。
3. 64 ビット・アプリケーションおよびルーチンを 32 ビット・アドレッシング・スペースで実行することはできません。

表の中の注目すべき重要な点は、32 ビットの unfenced プロシージャーを 64 ビットの DB2 データベース・サーバーで実行することができないという点です。32 ビットの unfenced ルーチンを 64 ビット・プラットフォームにデプロイしなければならない場合は、カタログする前に、そのルーチンの CREATE ステートメントから NOT FENCED 節を除去してください。

64 ビット・データベース・サーバー上での 32 ビット・ライブラリーを持つルーチンのパフォーマンス

64 ビット DB2 データベース・サーバー上で 32 ビット・ルーチン・ライブラリーを持つルーチンを呼び出すことは可能です。しかしこの場合、64 ビット・サーバー上で 64 ビット・ルーチンを呼び出すほどのパフォーマンスは発揮しません。

パフォーマンスが低下する理由は、64 ビット・サーバー上で 32 ビット・ルーチンを実行しようとするたびにまずそれを 64 ビット・ライブラリーとして呼び出そうとするからです。これが失敗してから、そのライブラリーは 32 ビット・ライブラリーとして呼び出されます。32 ビット・ライブラリーを 64 ビット・ライブラリーとして呼び出そうとして失敗すると、db2diag ログ・ファイルにエラー・メッセージ (SQLCODE -444) が生成されます。

Java クラスのビット幅はそれぞれ異なります。 Java 仮想マシン (JVM) だけが 32 ビットまたは 64 ビットとして分類されます。 DB2 データベース・システムでは JVM を使用するインスタンスと同じビット幅の JVM の使用のみがサポートされます。言い換えると、32 ビット DB2 インスタンスでは 32 ビット JVM だけを使用でき、64 ビット DB2 インスタンスでは 64 ビット JVM だけを使用できるということです。これにより、Java ルーチンは正しく機能し、可能な限り最高のパフォーマンスを発揮できます。

外部ルーチンでの XML データ・タイプのサポート

下記のプログラミング言語で書かれている外部プロシージャーおよび関数は、データ・タイプ XML のパラメーターおよび変数をサポートします。

- C
- C++
- COBOL
- Java
- .NET CLR 言語

OLE および OLEDB 外部ルーチンは、データ・タイプ XML のパラメーターをサポートしません。

XML データ・タイプの値は、CLOB データ・タイプと同じ方法で外部ルーチンのコード中に示されます。

データ・タイプ XML の外部ルーチン・パラメーターを宣言するときは、データベース内でそのルーチンを作成するときに使用する CREATE PROCEDURE および CREATE FUNCTION ステートメントで、XML データ・タイプを CLOB データ・タイプとして保管することを指定する必要があります。 CLOB 値のサイズは、XML パラメーターで表される XML 文書のサイズに近くなければなりません。

次の CREATE PROCEDURE ステートメントは、parm1 という XML パラメーターを使用して C プログラミング言語でインプリメントされた外部プロシージャーの CREATE PROCEDURE ステートメントを示しています。

```
CREATE PROCEDURE myproc(IN parm1 XML AS CLOB(2M), IN parm2 VARCHAR(32000))
LANGUAGE C
FENCED
PARAMETER STYLE SQL
EXTERNAL NAME 'mylib!myproc';
```

次の例に示されているような外部 UDF の作成時にも、それに似た考慮事項が当てはまります。

```
CREATE FUNCTION myfunc (IN parm1 XML AS CLOB(2M))
RETURNS SMALLINT
LANGUAGE C
PARAMETER STYLE SQL
DETERMINISTIC
NOT FENCED
NULL CALL
NO SQL
NO EXTERNAL ACTION
EXTERNAL NAME 'mylib1!myfunc'
```

XML データは、ストアード・プロシージャーに IN、OUT、または INOUT パラメーターとして渡されるときにマテリアライズされます。Java ストアード・プロシージャーを使用している場合、XML 引数の数量とサイズ、および並行に実行されている外部ストアード・プロシージャーの数に基づいて、ヒープ・サイズ (**java_heap_sz** 構成パラメーター) を増やすことが必要になる場合があります。

外部ルーチン・コード内部では、XML パラメーターおよび変数値へのアクセス、その設定、および変更は、データベース・アプリケーションの場合と同じやり方で行われます。

外部ルーチンに関する制約事項

外部ルーチンには以下の制約事項が適用されます。これらの制約事項は、外部ルーチンを開発およびデバッグするときに考慮してください。

すべての外部ルーチンに適用される制約事項:

- 外部ルーチンに新しいスレッドを作成できない。
- 外部関数または外部メソッド内から接続レベル API を呼び出せない。
- キーボードからの入力の受信または標準出力への出力の表示を外部ルーチンから行えない。標準入出力ストリームは使用しないでください。以下に例を示します。
  - 外部 Java ルーチン・コードでは、System.out.println() メソッドを発行しない。
  - 外部 C または C++ ルーチン・コードでは、printf() を発行しない。
  - 外部 COBOL ルーチン・コードでは、display を発行しない。

  外部ルーチンはデータを標準出力に表示できませんが、データベース・サーバー・ファイル・システム上のファイルにデータを書き込むコードをこれに組み込むことは可能です。

  UNIX 環境上で稼働する fenced ルーチンの場合、ファイルの作成先のターゲット・ディレクトリーやファイルそのものが適切な権限をもち、`sqllib/adm/.fenced` ファイルの所有者がその作成や書き込みを行えるようにしなければなりません。 not fenced ルーチンの場合、インスタンス所有者は、ファイルをオープンする場所であるディレクトリーを対象とした作成、読み取り、および書き込みの権限を持っていなければなりません。

  注: DB2 データベース・システムは、ルーチンが実行する外部の入出力と、DB2 データベース・システム独自のトランザクションとの同期を試みません。したがって、例えばトランザクションの処理中に UDF がファイルに書き込みを行った後で、そのトランザクションが何らかの理由でバックアウトされても、そのファイルへの書き込みの探索や取り消しは試みられません。
- 接続に関連したステートメントまたはコマンドは外部ルーチンで実行できない。この制限は、以下のステートメントおよびコマンドに適用されます。
  - **BACKUP DATABASE**
  - CONNECT
  - CONNECT TO
  - CONNECT RESET
  - **CREATE DATABASE**

  - **DROP DATABASE**
  - FORWARD RECOVERY
  - **RESTORE DATABASE**
- ルーチン内でオペレーティング・システム関数を使用することは推奨されていない。基本的にこの関数の使用には制限がありませんが、以下の場合は例外です。
  - ユーザー定義のシグナル・ハンドラーを外部ルーチンにインストールしてはならない。この制約事項を守らないと、外部ルーチン実行時の不測の障害、データベースの異常終了、またはその他の問題を生じることがあります。シグナル・ハンドラーをインストールすると、 JVM for Java ルーチンの操作が妨げられることもあります。
  - 処理を終了するシステム呼び出しを行うと DB2 データベース・システムの処理の 1 つが異常終了し、データベース・システムまたはデータベース・アプリケーションの障害が発生することがあります。

    他のシステム呼び出しによって DB2 データベース・マネージャーの通常の操作が妨害される場合にも、問題が発生することがあります。例えば、ユーザー定義関数が含まれるライブラリーを関数がメモリーからアンロードしようとすると、重大な問題が発生することがあります。システム呼び出しが含まれる外部ルーチンのコーディングとテストには注意してください。
- DB2 pureScale® for Linux バージョン 9.8 フィックスパック 2 以降、unfenced ルーチンでは、結果的に新規プロセスが作成されるようなオペレーティング・システム関数の使用ができなくなりました。これらの関数には、fork()、popen()、および system() などがあります。これらの関数を使用すると、DB2 サーバーとクラスター・キャッシング・ファシリティーとの間の通信に妨害が生じ、ルーチンが SQL0430N エラーを返すことがあります。
- 現行処理を終了させるコマンドを外部ルーチンに入れてはならない。外部ルーチンは、現行プロセスを終了させることなく常に DB2 データベース・マネージャーに制御を戻さなければなりません。
- 特別な場合を除き、データベースがアクティブになっている間に外部ルーチン・ライブラリー、クラス、またはアセンブリーを更新してはならない。 DB2 データベース・マネージャーがアクティブになっている間に更新の必要が生じ、インスタンスの停止と開始ができない場合には、ルーチン用にライブラリー、クラス、またはアセンブリーを別に新規作成します。その後、新しいライブラリー、クラス、またはアセンブリー・ファイルの名前が参照されるように、ALTER ステートメントを使って外部ルーチンの EXTERNAL NAME 節の値を変更します。
- 外部ルーチンで環境変数 **DB2CKPTR** を使用できない。名前が「DB2」で始まるその他のすべての環境変数は、データベース・マネージャーの開始時にキャプチャーされ、外部ルーチンで使用することができます。
- 「DB2」で名前が始まらない環境変数の中には fenced の外部ルーチンで使用できないものもある。例えば、**LIBPATH** 環境変数は使用できません。ただし、この種の変数は not fenced の外部ルーチンでは使用することができます。
- DB2 データベース・マネージャーの開始後に設定された環境変数の値は外部ルーチンで使用できない。
- 外部ルーチン内での保護リソース (一度に 1 つのプロセスによってのみアクセスできるリソース) の使用は制限する必要がある。使用する場合は、2 つの外部ル

ーチンが保護リソースにアクセスしようとする際にデッドロックが発生する可能性ができるだけ小さくなるようにしてください。保護リソースへのアクセス試行中に複数の外部ルーチンでデッドロックが発生すると、DB2 データベース・マネージャーはこの状況を検出または解決することができません。これにより、外部ルーチンのプロセスはハングしてしまいます。

- 外部ルーチン・パラメーターのメモリーを DB2 データベース・サーバーに対して明示的に割り振ってはならない。 DB2 データベース・マネージャーはルーチンに対して、 CREATE ステートメント内のパラメーター宣言に基づいてストレージを自動的に割り振ります。外部ルーチン内のパラメーターのストレージ・ポインターを変えないでください。ポインターを、ローカル作成したストレージ・ポインターに置き換えようとすると、メモリー・リーク、データ破壊、または異常終了が発生する可能性があります。
- 外部ルーチン内で静的データまたはグローバル・データを使用してはならない。DB2 データベース・システムでは、静的変数またはグローバル変数によって使用されたメモリーが、外部ルーチンの次の呼び出しまで不変のままであるとは限りません。UDF およびメソッドの場合はスクラッチパッドを使用すれば、次の呼び出しでも使用できるように値を保管しておくことができます。
- SQL パラメーター値はすべてバッファーに入れられる。これは、その値がコピーされて外部ルーチンに渡されることを意味します。外部ルーチンの入力パラメーターに変更が加えられても、 SQL の値または処理に対してその変更は効力をもちません。ただし、CREATE ステートメントで指定されている以上のデータを外部ルーチンが入力または出力パラメーターに書き込んだ場合、メモリー破壊が発生し、ルーチンは異常終了する可能性があります。
- **LOAD** ユーティリティーは、fenced プロシージャーを参照する列を含む表へのロードをサポートしていません。そのような表に対して **LOAD** コマンドを発行すると、エラー・メッセージ SQL1376N が出されます。この制限を回避するには、ルーチンを unfenced にするように再定義するか、インポート・ユーティリティーを使用することができます。

外部プロシージャーにのみ適用される制約事項

- ネストされたストアード・プロシージャーから結果セットが戻される場合、複数のネスト・レベルにまたがって同一名でカーソルをオープンすることができる。ただし、バージョン 8 より前のアプリケーションは、オープンした最初の結果セットにしかアクセスすることはできません。この制約事項は、別のパッケージ・レベルでオープンされたカーソルには適用されません。

外部関数にのみ適用される制約事項

- 外部関数は結果セットを戻すことはできない。外部関数内でオープンされたカーソルはすべて、関数の最終呼び出しが完了した時点でクローズされなければなりません。
- 外部ルーチンが戻す前に、外部ルーチンでのメモリーの動的割り振りを解放する必要がある。これが行われないと、メモリー・リークが発生したり、DB2 プロセス内でのメモリーの消費が増え続けたりすることになります。これは、データベース・システムでのメモリー不足を引き起こしかねません。

  外部ユーザー定義関数および外部メソッドの場合、複数の関数呼び出しに必要な動的メモリーの割り振りをスクラッチパッドを使用して行える。この目的でスク

ラッチパッドを使用する場合は、CREATE FUNCTION または CREATE METHOD ステートメントで FINAL CALL 属性を指定します。こうすることで、割り振り済みのメモリーを、ルーチンが戻す前に確実に解放できます。

外部ルーチンの作成

プロシージャーと関数を含む外部ルーチンは、他のインプリメンテーションを使用したルーチンと類似した方法で作成されます。しかし、ルーチンのインプリメンテーションでは、ソース・コードのコード化、コンパイル、およびデプロイが必要になるため、さらにいくつかのステップが必要です。

始める前に

- IBM Data Server Client がインストールされていなければなりません。
- データベース・サーバーが、選択したインプリメンテーション・プログラミング言語コンパイラーおよび開発ソフトウェアをサポートするオペレーティング・システムを実行していること。
- 選択したプログラミング言語の必須コンパイラーおよびランタイム・サポートがデータベース・サーバー上にインストールされていること。
- CREATE PROCEDURE、CREATE FUNCTION、または CREATE METHOD ステートメントを実行する権限。

制約事項

外部ルーチンに関連した制約事項のリストについては、以下を参照してください。

- 40 ページの『外部ルーチンに関する制約事項』

このタスクについて

外部ルーチンをインプリメントするのは、以下のような場合です。

- データベースにアクセスするルーチンや、データベースの外部でアクションを実行するルーチンに複雑なロジックをカプセル化したい場合。
- 複数のアプリケーション、CLP、他のルーチン (プロシージャー、関数 (UDF)、メソッド)、トリガーのいずれかから、カプセル化されたロジックを呼び出す必要のある場合。
- そのロジックのコーディングに、SQL や SQL PL ステートメントを使用するよりも、プログラミング言語が最も使いやすいと感じる場合。
- ルーチン・ロジックにデータベース外での操作を実行させたい場合。これは例えば、データベース・サーバー上でのファイルへの書き込みまたはファイルからの読み取り、別のアプリケーションの実行、または SQL および SQL PL ステートメントで表すことができないロジックなどです。

手順

1. 選択したプログラミング言語でルーチン・ロジックをコーディングします。
   - 外部ルーチン、ルーチン・フィーチャー、およびルーチン・フィーチャーのインプリメンテーションの一般情報については、『前提条件』のセクションで参照されているトピックを参照してください。
   - SQL ステートメントの実行をサポートするために必要な、必須ヘッダー・ファイルを使用またはインポートします。

- DB2 SQL データ・タイプにマップするプログラミング言語データ・タイプを使用して、変数とパラメーターを正しく宣言します。

2. 選択したプログラミング言語のパラメーター・スタイルが要求する形式に従って、パラメーターを宣言する必要があります。パラメーターとプロトタイプ宣言の詳細については、以下を参照してください。
   - 26 ページの『外部ルーチンのパラメーター・スタイル』
3. コードからライブラリーまたはクラス・ファイルを作成します。
4. そのライブラリーまたはクラス・ファイルをデータベース・サーバー上の DB2 *function ディレクトリー* にコピーします。 DB2 ルーチンに関連したアセンブリーまたはライブラリーは、 function ディレクトリーに保管することをお勧めします。 function ディレクトリーの詳細については、 CREATE PROCEDURE ステートメントまたは CREATE FUNCTION ステートメントのいずれかの EXTERNAL 節を参照してください。

   アセンブリーをサーバー上の別のディレクトリーにコピーすることもできますが、ルーチンを正常に呼び出すには、アセンブリーの完全修飾パス名をメモしておく必要があります。次のステップでこれが必要になるからです。
5. 該当するルーチン・タイプの SQL 言語 CREATE ステートメント (CREATE PROCEDURE または CREATE FUNCTION) を動的または静的に実行します。
   - 選択した API またはプログラミング言語の該当する値を使用して、LANGUAGE 節を指定します。例として、CLR、C、JAVA が含まれます。
   - PARAMETER STYLE 節に、ルーチン・コードでインプリメントした有効なパラメーター・スタイルの名前を指定します。
   - EXTERNAL 節に、ルーチンと関連したライブラリー、クラス、またはアセンブリー・ファイルの名前を指定します。そのためには、以下のいずれかの値を使用します。
     – ルーチン・ライブラリー、クラス、またはアセンブリー・ファイルの完全修飾パス名。
     – function ディレクトリーを基準にしたルーチン・ライブラリー、クラス、またはアセンブリー・ファイルの相対パス名。

     EXTERNAL 節にライブラリー、クラス、またはアセンブリー・ファイルの完全修飾パス名または相対パス名を指定しない場合、 DB2 はデフォルトで、function ディレクトリー内でそれらの名前を探します。
   - ルーチンがプロシージャーの場合に、DYNAMIC RESULT SETS に数値を指定すると、呼び出し元に 1 つ以上の結果セットが戻されます。
   - ルーチンの特徴を指定するために必要な他の節を指定します。

次のタスク

外部ルーチンを呼び出すには、ルーチンの呼び出しを参照してください。

ルーチンの作成

始める前に

3 つのタイプのルーチン (プロシージャー、UDF、メソッド) には、その作成法に関して多くの共通点があります。例えば、それら 3 つのルーチン・タイプは、同じパ

ラメーター・スタイルをいくつか使用し、各種クライアント・インターフェース (組み込み SQL、CLI、JDBC) を介して SQL の使用をサポートします。また、いずれも他のルーチンを呼び出すことができます。その例として、以下のステップはルーチンを作成する 1 つのアプローチを示しています。

特定のルーチン・タイプだけに用意されている機能もあります。例えば、結果セットはストアード・プロシージャー独特のものであり、スクラッチパッドは UDF およびメソッド独自のものです。開発しようとしているルーチン・タイプに当てはまらないステップに行き当たったら、その後のステップに進んでください。

ルーチンを作成する場合は、事前に以下を決定する必要があります。

- 必要なルーチン・タイプ。
- 作成に使用するプログラム言語。
- ルーチン内に SQL ステートメントが必要な場合にどのインターフェースを使用するか。

『セキュリティー、ライブラリー、およびクラス管理』および『パフォーマンスの考慮事項』の項も参照してください。

手順

ルーチン本体を作成するには、以下を行う必要があります。

1. *外部ルーチンにのみ当てはまります。* 呼び出し側のアプリケーションまたはルーチンからの入力パラメーターを受け入れて、出力パラメーターを宣言します。ルーチンがパラメーターをどのように受け入れるかは、ルーチンの作成に使用するパラメーター・スタイルによって異なります。各パラメーター・スタイルは、ルーチン本体に渡される一連のパラメーターと、パラメーターが渡される順序を定義します。

   例えば、PARAMETER STYLE SQL 用に C で書かれた UDF 本体のシグニチャー (sqludf.h を使用して) を以下に示します。

   ```
   SQL_API_RC SQL_API_FN product ( SQLUDF_DOUBLE *in1,
                                   SQLUDF_DOUBLE *in2,
                                   SQLUDF_DOUBLE *outProduct,
                                   SQLUDF_NULLIND *in1NullInd,
                                   SQLUDF_NULLIND *in2NullInd,
                                   SQLUDF_NULLIND *productNullInd,
                                   SQLUDF_TRAIL_ARGS )
   ```

2. ルーチンが実行するロジックを追加します。ルーチンの本体で使用できる機能には次のようなものがあります。
   - 他のルーチンの呼び出し (ネスティング)、または現在のルーチンの呼び出し (再帰)。
   - SQL (CONTAINS SQL、READS SQL、または MODIFIES SQL) を組み込むように定義されたルーチンでは、ルーチンから SQL ステートメントを発行することができます。呼び出すステートメントのタイプは、ルーチンの登録方法によって制御します。
   - 外部 UDF およびメソッドでは、スクラッチパッドを使用して、複数の呼び出しにまたがって状態を保管します。

- SQL プロシージャーでは、条件ハンドラーを使用して、指定の条件が発生したときの SQL プロシージャーの動作を指定します。そのような条件は、SQLSTATE をベースにして定義することができます。

3. ストアード・プロシージャーにのみ当てはまります。 1 つ以上の結果セットを戻します。呼び出し側のアプリケーションとの間で交換される個々のパラメーターに加えて、複数の結果セットを戻す機能がストアード・プロシージャーに備わっています。 SQL ルーチンと、CLI、ODBC、JDBC、および SQLJ ルーチンとクライアントのみが、結果セットを受け入れることができます。

タスクの結果

ルーチンの作成以外に、ルーチンを呼び出すにはまず登録する必要があります。それには、開発しているルーチンのタイプに合った CREATE ステートメントを使用します。一般的に、ルーチンを作成して登録する順序は問題にはなりません。ただし、ルーチンが自身を参照する SQL を発行する場合は、作成の前にルーチンを登録する必要があります。その場合にバインドを正常に完了するには、ルーチンの登録が事前に完了していなければなりません。

ルーチンのデバッグ

実動サーバーにルーチンを配置する前に、テスト・サーバーでそのルーチンを徹底的にテストしてデバッグする必要があります。

このタスクについて

これは、NOT FENCED として登録する必要のあるルーチンの場合は特に重要です。このルーチンは、データベース・マネージャーのメモリー、そのデータベース、およびデータベース制御構造に無制限でアクセスできるからです。 FENCED THREADSAFE ルーチンにも厳重な注意が必要です。このルーチンは他のルーチンとメモリーを共有するからです。

ルーチンの一般的な問題のチェックリスト

ルーチンが正しく稼働していることを確認するには、以下を調べます。

- ルーチンが正しく登録されている。 CREATE ステートメントに指定するパラメーターは、ルーチン本体で処理される引数に一致している必要があります。この点に留意しながら、以下の項目を 1 つずつチェックします。
  - ルーチン本体で使用されている引数のデータ・タイプは、CREATE ステートメントに定義されているパラメーター・タイプに適したタイプである。
  - CREATE ステートメント内の対応する結果に対して定義されているより大きいバイト数がルーチンによって出力変数に書き込まれていない。
  - 対応する CREATE オプションを使用してルーチンが登録されていた場合は、 SCRATCHPAD、FINAL CALL、DBINFO 用のルーチン引数が存在する。
  - 外部ルーチンの場合、CREATE ステートメント内の EXTERNAL NAME 節の値は、ルーチン・ライブラリーおよびエントリー・ポイントに一致していなければならない (大文字小文字の区別はオペレーティング・システムによって異なります)。

– C++ ルーチンの場合、C++ コンパイラーはタイプ修飾をエントリー・ポイント名に適用します。タイプ修飾名を EXTERNAL NAME 節に指定する必要がありますが、そうでない場合、ユーザー・コード内でエントリー・ポイントを `extern "C"` と定義しなければなりません。
– 呼び出し時に指定するルーチン名は、そのルーチンの登録名 (CREATE ステートメント内で定義済みの名前) に一致していなければならない。デフォルトではルーチン ID は英大文字に変換されます。それは、区切り ID には当てはまりません。区切り ID の場合、大文字には変換されずに、大文字小文字が区別されます。

  ルーチンは、CREATE ステートメントに指定されているディレクトリー・パスに置かれなければなりません。ただし、パスを指定しないと、デフォルトで DB2 データベース・システムがそれを探索します。 UDF、メソッド、および fenced プロシージャーの場合にはそれは、`sqllib/function` (UNIX) または `sqllib¥function` (Windows) となります。unfenced プロシージャーの場合にはそれは、`sqllib/function/unfenced` (UNIX) または `sqllib¥function¥unfenced` (Windows) となります。

- 正しい呼び出しシーケンス、プリコンパイル (組み込み SQL の場合)、およびリンク・オプションを使用してルーチンが作成されている。
- アプリケーションが CLI、ODBC、または JDBC を使用して作成されている場合を除き、アプリケーションはデータベースにバインドされている。ルーチンに SQL が入っていて、しかもそのようなインターフェースを使用しない場合、ルーチンをバインドする必要もあります。
- そのルーチンは、クライアント・アプリケーションにエラー情報を正確に戻す。
- FINAL CALL を使用してルーチンが定義されている場合は、使用可能なすべての呼び出しタイプを考慮に入れる。
- ルーチンによって使用されるシステム・リソースが戻される。
- ルーチンを呼び出そうとしたときに、その操作を実行するための十分な特権がないことを示すエラー (SQLCODE -551、SQLSTATE 42501) を受け取った場合は、ルーチンに関する EXECUTE 特権がないことが原因になっている可能性が高いと言えます。この特権をルーチンの呼び出し側に付与できるのは、SECADM 特権、ACCESSCTRL 特権を持つユーザーか、ルーチンに関する EXECUTE WITH GRANT OPTION 特権を持つユーザーです。権限とルーチンについての関連トピックでは、この特権の使用を効率的に管理する方法を詳しく説明しています。

ルーチンのデバッグの技法

ルーチンをデバッグするには、次のような技法を使用します。

- Development Center には、SQL を本体とするプロシージャーと Java プロシージャー用の包括的デバッグ・ツールが備えられています。
- ルーチンから画面に診断データを書き込むことはできません。診断データをファイルに書き込む場合、必ず `¥tmp` などのグローバル・アクセスの可能なディレクトリーに書き込んでください。データベース・マネージャーやデータベースによって使用されるディレクトリーには書き込まないでください。

プロシージャーの場合はその代わりに SQL 表に診断データを書き込むのが安全な方法です。 SQL 表に書き込めるようにするには、 MODIFIES SQL DATA 節を使用して、テストするプロシージャーを登録する必要があります。既存のプロシージャーでデータを SQL 表に書き込む (または書き込みをやめる) 必要がある場合、プロシージャーをいったんドロップしてから、 MODIFIES SQL DATA 節を使用して (または使用しないで) 再登録しなければなりません。プロシージャーをドロップして再登録する場合、事前にその従属関係に配慮してください。

- ルーチンのエントリー・ポイントを直接呼び出す単純なアプリケーションを作成すれば、ルーチンをローカル側でデバッグすることができます。装備されているデバッガーの使用法の詳細は、コンパイラーの資料をご覧ください。

外部ルーチンのライブラリーおよびクラスの管理

外部ルーチンを首尾よく開発し、呼び出すには、外部ルーチンのライブラリーおよびクラス・ファイルを適切にデプロイし、管理する必要があります。

外部ルーチンのライブラリーおよびクラス・ファイルの管理は、外部ルーチンを初めて作成し、ライブラリーおよびクラス・ファイルをデプロイするときに注意を払うことにより、最小限に抑えることができます。

外部ルーチンの管理に関する主な考慮事項は次のとおりです。

- 外部ルーチンのライブラリーおよびクラス・ファイルのデプロイメント
- 外部ルーチン・ライブラリーおよびクラス・ファイルのセキュリティー
- 外部ルーチンのライブラリーおよびクラスの解決
- 外部ルーチンのライブラリーおよびクラス・ファイルに対する変更
- 外部ルーチンのライブラリーおよびクラス・ファイルのバックアップおよびリストア
- すべてのルーチン・ライブラリーが、`sqllib/function` ディレクトリー内にあり、正しいライブラリーにあることを確認する。ルーチン・ライブラリーの最終バージョンを配置するメンバーを選択します。このライブラリーは、**db2iupdt** コマンドが実行された最終メンバーと同じライブラリーです。

システム管理者、データベース管理者、およびデータベース・アプリケーション開発者のすべてが、ルーチンの開発中およびデータベース管理タスク実行時の外部ルーチンのライブラリーおよびクラス・ファイルの機密保護を確保し、それらが正しく保存されることに責任を持つ必要があります。

外部ルーチン・ライブラリーおよびクラスのデプロイメント

外部ルーチン・ライブラリーおよびクラスのデプロイメントは、外部ルーチン・ライブラリーおよびクラスをソース・コードからビルドされた後に、それらをデータベース・サーバーにコピーすることを指します。

外部ルーチン・ライブラリー、クラス、またはアセンブリー・ファイルは、データベース・サーバーの DB2 データベース・システムの function ディレクトリーまたはこのディレクトリーのサブディレクトリーにコピーする必要があります。これは、外部ルーチンのデプロイメントの推奨される位置です。 function ディレクトリ

ーの詳細については、SQL ステートメント (CREATE PROCEDURE または CREATE FUNCTION) のいずれかの EXTERNAL 節の説明を参照してください。

ルーチンのインプリメントに使用する API およびプログラミング言語に応じて、外部ルーチンのクラス、ライブラリー、またはアセンブリーをサーバー上の他のディレクトリーの位置にコピーすることができます。ただし、これは通常お勧めできません。これを実行した場合、ルーチンを正常に呼び出すために、完全修飾パス名に特に注意して、この値が EXTERNAL NAME 節で使用されていることを確認する必要があります。

ライブラリーおよびクラス・ファイルは、ごく普通に入手できるファイル転送ツールを使ってデータベース・サーバーのファイル・システムにコピーすることができます。特別な組み込みプロシージャーを使って (特にこの目的のために設計された)、 Java ルーチンを DB2 クライアントがインストールされているコンピューターから DB2 データベース・サーバーにコピーすることができます。詳しくは、Java ルーチンに関するトピックを参照してください。

該当するルーチン・タイプの SQL 言語 CREATE ステートメント (CREATE PROCEDURE または CREATE FUNCTION) を実行する場合、EXTERNAL NAME 節に特に注意して、該当する節を指定します。

- 選択した API またはプログラミング言語の該当する値を使用して、LANGUAGE 節を指定します。例として、CLR、C、JAVA が含まれます。
- PARAMETER STYLE 節に、ルーチン・コードでインプリメントした有効なパラメーター・スタイルの名前を指定します。
- EXTERNAL 節に、ルーチンと関連したライブラリー、クラス、またはアセンブリー・ファイルの名前を指定します。そのためには、以下のいずれかの値を使用します。
  - ルーチン・ライブラリー、クラス、またはアセンブリー・ファイルの完全修飾パス名。
  - function ディレクトリーを基準にしたルーチン・ライブラリー、クラス、またはアセンブリー・ファイルの相対パス名。

EXTERNAL 節にライブラリー、クラス、またはアセンブリー・ファイルの完全修飾パス名または相対パス名を指定しない場合、 DB2 データベース・システムはデフォルトで、 function ディレクトリー内でそれらの名前を探します。

外部ルーチン・ライブラリーまたはクラス・ファイルのセキュリティー

外部ルーチン・ライブラリーは、データベース・サーバーのファイル・システムに保管され、DB2 データベース・マネージャーではどんな方法でもバックアップまたは保護されることはありません。ルーチンが正常に呼び出され続けるようにするには、ルーチンに関連したライブラリーが、ルーチンの作成に使用される CREATE ステートメントの EXTERNAL 節で指定された位置に存在し続ける必要があります。

ルーチンの作成後に、ルーチン・ライブラリーを移動したり削除しないでください。これを行うと、ルーチンの呼び出しは失敗します。

ルーチン・ライブラリーが意図せずにまたは故意に削除されたり置き換えられたりしないようにするには、データベース・サーバー上のルーチン・ライブラリーを含むディレクトリーへのアクセス、およびルーチン・ライブラリー・ファイルへのアクセスを制限する必要があります。これは、オペレーティング・システムのコマンドを使用して、ディレクトリーおよびファイル権限を設定することによって行うことができます。

外部ルーチンのライブラリーおよびクラスの解決

DB2 外部ルーチン・ライブラリーの解決は、DB2 インスタンス・レベルで実行されます。これは、複数の DB2 データベースが含まれる DB2 インスタンスでは、外部ルーチンを、あるデータベースのルーチン用にすでに使用されている外部ルーチン・ライブラリーを使用する、別のデータベースに作成できることを意味します。

インスタンス・レベルの外部ルーチンの解決では、複数のルーチン定義が単一のライブラリーに関連付けられるようにして、コードの再利用をサポートします。外部ルーチン・ライブラリーをこの方法で再利用せず、代わりに外部ルーチン・ライブラリーのコピーがデータベース・サーバーのファイル・システムに存在する場合、ライブラリー名の競合が発生する可能性があります。このことが特に起きるのは、単一のインスタンス内に複数のデータベースが存在し、各データベース内のルーチンがルーチン本体の独自のライブラリーおよびクラスのそれ自体のコピーに関連付けられている場合です。あるデータベースのルーチンが使用するライブラリーまたはクラスの名前が、 (同じインスタンス内の) 別のデータベースのルーチンが使用するライブラリーまたはクラスの名前と同一の場合には、競合が生じます。

これが起こる可能性を最小限に抑えるには、ルーチン・ライブラリーの単一コピーをインスタンス・レベルの function ディレクトリー (`sqllib/function` ディレクトリー) に保管すること、および各データベースのすべてのルーチン定義の EXTERNAL 節がその固有のライブラリーを参照するようにすることをお勧めします。

機能的に異なる 2 つのルーチン・ライブラリーを同じ名前で作成する必要がある場合は、ライブラリー名が競合する可能性を最小限に抑えるために追加のステップを行うことが重要です。

C、C++、COBOL、および ADO.NET ルーチンの場合:

以下のようにして、ライブラリー名の競合を最小限に抑えるかあるいは解決することができます。

1. ライブラリーを、各データベースの別個のディレクトリーにルーチン本体と共に保管する。
2. 指定したライブラリーの (相対パスの代わりに) 絶対パスを指定する EXTERNAL NAME 節の値を使用してルーチンを作成する。

Java ルーチンの場合:

CLASSPATH 環境変数はインスタンス全体の環境変数であるため、クラス名競合の問題のあるクラス・ファイルを別々のディレクトリーに移動しても、問題の解決にはなりません。 **CLASSPATH** で最初に検出されたクラスが使用されることになります。このため、同じ名前のクラスを参照する 2 つの異なる Java ルーチンがある場合には、このルーチンのいずれかが間違ったクラスを使用します。可能な 2 つの解決策があります。関係するクラスを名前変更するか、各データベースの別個のインスタンスを作成します。

外部ルーチンのライブラリーおよびクラス・ファイルに対する変更

既存の外部ルーチンのロジックに対する変更は、外部ルーチンをデプロイしており、それが実動データベース・システム環境で使用中になってから必要になる場合があります。既存のルーチンに対する変更を行うことは可能ですが、更新のための明確なテークオーバーのポイント・イン・タイムを定義し、ルーチンの並行呼び出しが中断するリスクを最小限に抑えるように、注意深く行うことが重要です。

外部ルーチン・ライブラリーを更新する必要が生じたら、データベース・マネージャーの稼働中に現行ルーチンが使用しているのと同じターゲット・ファイル (例えば、`sqllib/function/foo.a`) にルーチンを再コンパイルおよび再リンクしないでください。ルーチンの現在の呼び出しがルーチン・プロセスのキャッシュ・バージョンにアクセスする場合、基本ライブラリーが置き換えられていると、ルーチンの呼び出しは失敗することがあります。 DB2 データベース・マネージャーの停止と再始動の過程を経ないでルーチンの本体を変更する必要が生じた場合、以下のステップを行ってください。

1. 別のライブラリーまたはクラス・ファイル名を使用して、新規の外部ルーチン・ライブラリーを作成します。
2. それが組み込み SQL ルーチンである場合、**BIND** コマンドを使用してルーチン・パッケージをデータベースにバインドします。
3. ALTER ROUTINE ステートメントを使用してルーチン定義を変更し、EXTERNAL NAME 節が更新されたルーチン・ライブラリーまたはクラスを参照できるようにします。更新されるルーチン本体が、複数のデータベースにカタログされたルーチンによって使用される場合には、このセクションで指示されたアクションを、関係する各データベースについて実行しなければなりません。
4. JAR ファイルに組み入れられた Java ルーチンの更新の場合、 CALL SQLJ.REFRESH_CLASSES() ステートメントを発行して、 DB2 データベース・マネージャーで強制的に新規クラスをロードする必要があります。 Java ルーチン・クラスを更新した後に CALL SQLJ.REFRESH_CLASSES() ステートメントを発行しないと、 DB2 データベース・システムは以前のバージョンのクラスを使用し続けます。 DB2 データベース・システムは、COMMIT または ROLLBACK が生じると、クラスをリフレッシュします。

ルーチン定義が更新されたら、ルーチンの後続の呼び出しはすべて、新規の外部ルーチン・ライブラリーまたはクラスをロードして実行します。

外部ルーチンのライブラリーおよびクラス・ファイルのバックアップおよびリストア

外部ルーチン・ライブラリーは、データベースのバックアップが実行されるときに、他のデータベース・オブジェクトとともにバックアップされません。同様に、データベースがリストアされる際に、外部ルーチン・ライブラリーはリストアされません。

データベースのバックアップおよびリストアの目的がデータベースの再デプロイである場合、外部ルーチンのライブラリー・ファイルを、外部ルーチン・ライブラリーの相対パス名を保存するのと同じ方法で、元のデータベース・サーバーのファイル・システムからターゲット・データベース・サーバーのファイル・システムにコピーする必要があります。

外部ルーチン・ライブラリー管理およびパフォーマンス

外部ルーチン・ライブラリー管理は、DB2 データベース・マネージャーが、ルーチンの使用法に従ってパフォーマンスを向上させる目的で外部ルーチン・ライブラリーを動的にキャッシュするため、ルーチンのパフォーマンスに影響を与えることがあります。

外部ルーチンのパフォーマンスを最適なものにするには、以下の点を考慮してください。

- 各ライブラリー内のルーチン数を可能な限り少数に保ってください。少数の大きな外部ルーチン・ライブラリーよりも多数の小さい外部ルーチン・ライブラリーを用意したほうがよいと思われます。
- 一般的に一緒に呼び出されるルーチンのルーチン関数を、ソース・コード内で 1 つのグループにまとめます。コードを 1 つの外部ルーチン・ライブラリーにコンパイルすると、一般的に呼び出されるルーチンのエントリー・ポイントは互いに近くなるため、データベース・マネージャーはより優れたキャッシング・サポートを提供できるようになります。キャッシング・サポートが向上するのは、単一の外部ルーチン・ライブラリーを一度ロードして、複数の外部ルーチン関数をそのライブラリー内で呼び出すことによって効率が上がるためです。

  C または C++ プログラミング言語でインプリメントされた外部ルーチンの場合、ライブラリーをロードする手間は、 C ルーチンによって首尾一貫して使用されているライブラリーの場合は 1 回しかかかりません。ルーチンを最初に呼び出した後に、そのプロセスの同じスレッドから後続のすべての呼び出しを実行するときには、ルーチンのライブラリーを再ロードする必要がありません。

第 3 章 .NET 共通言語ランタイム (CLR) ルーチン

DB2 データベース・システムにおける共通言語ランタイム (CLR) ルーチンとは、.NET アセンブリーを外部コード本体として参照する外部ルーチンであり、CREATE PROCEDURE ステートメントまたは CREATE FUNCTION ステートメントの実行によって作成します。

CLR ルーチンのコンテキストで重要な用語は、以下のとおりです。

.NET Framework
CLR と .NET Framework クラス・ライブラリーから成る Microsoft アプリケーション開発環境。コード断片の開発と統合のための一貫したプログラミング環境を提供します。

共通言語ランタイム (CLR)
あらゆる .NET Framework アプリケーションのためのランタイム・インタープリター。

中間言語 (IL)
.NET Framework CLR によって解釈されるコンパイル済みバイトコードの一種。すべての .NET 互換言語のソース・コードが IL バイト・コードにコンパイルされます。

アセンブリー
IL バイト・コードを内容とするファイル。ライブラリーか実行可能ファイルのいずれかです。

CLR ルーチンは、 IL アセンブリーにコンパイルできる言語であればどの言語でもインプリメントできます。例えば、Managed C++、C#、Visual Basic、J# などの言語があります。

CLR ルーチンを開発するには、ルーチンの基本と、CLR ルーチンにユニークなフィーチャーや特徴をあらかじめ理解しておくことが重要です。ルーチンと CLR ルーチンの詳細については、以下を参照してください。

- 5 ページの『ルーチン使用の利点』
- 55 ページの『.NET CLR ルーチンでの SQL データ・タイプ表記』
- 57 ページの『.NET CLR ルーチンのパラメーター』
- 60 ページの『.NET CLR プロシージャーからの結果セットの戻り』
- 62 ページの『.NET CLR ルーチンに関する制約事項』
- 74 ページの『.NET CLR ルーチンに関連したエラー』

CLR ルーチンの開発は簡単です。 CLR ルーチンの開発の方法に関する段階的な説明と完全な例については、以下を参照してください。

- 65 ページの『DB2 コマンド・ウィンドウから .NET CLR ルーチンを作成する』
- 77 ページの『C# .NET CLR プロシージャーの例』
- 105 ページの『C# .NET CLR 関数の例』

.NET CLR 言語での外部ルーチン開発のサポート

.NET CLR 言語で外部ルーチンを開発し、それを正常に実行するには、いずれもサポートされているオペレーティング・システム、DB2 データベース・サーバーとクライアントのバージョン、および開発ソフトウェアを使用する必要があります。

.NET CLR 外部ルーチンは、IL アセンブリーにコンパイルできる言語であればどの言語でも Microsoft .NET Framework を使用してインプリメントできます。例えば、Managed C++、C#、Visual Basic、J# などの言語があります。

.NET CLR ルーチンは、以下のオペレーティング・システム上で開発できます。

- Windows 2000
- Windows XP (32 ビット版および 64 ビット版)
- Windows Server 2003 (32 ビット版および 64 ビット版)
- Windows Server 2008 (32 ビット版および 64 ビット版)

.NET CLR ルーチンの開発では、バージョン 9 以降のデータ・サーバー・クライアントがインストールされている必要があります。データベース・サーバーで、DB2 バージョン 9 以降のデータベース製品を実行している必要があります。

Microsoft .NET Framework ソフトウェアのサポートされているバージョンも DB2 データベース・サーバーと同じコンピューター上にインストールされている必要があります。Microsoft .NET Framework は、単独で、または Microsoft .NET Framework Software Development Kit の一部として入手可能です。

.NET CLR ルーチン開発のためのツール

ツールを使用して、DB2 データベースと対話する .NET CLR ルーチンの開発タスクの実行をより高速かつ簡単にできます。

.NET CLR ルーチンは、Microsoft Visual Studio .NET において以下の製品で有効なグラフィック・ツールを使用して開発することができます。

- IBM Database Add-Ins for Microsoft Visual Studio

DB2 データベース・サーバー上での .NET CLR ルーチンの開発にも使用できる DB2 データベース・システム付属のコマンド行インターフェースには次のものがあります。

- DB2 コマンド行プロセッサー (DB2 CLP)
- DB2 コマンド・ウィンドウ

.NET CLR ルーチンの設計

.NET CLR ルーチンを設計するときは、一般的な外部ルーチンの設計上の考慮事項と .NET CLR 固有の設計上の考慮事項の両方を考慮に入れる必要があります。

.NET アプリケーション開発に関する知識と経験、および外部ルーチンに関する一般的な知識。以下のトピックは、必要な前提条件に関する情報の一部を提供しています。

外部ルーチンのフィーチャーおよび使用方法の詳細については、以下を参照してください。

- 6 ページの『外部ルーチンのインプリメンテーション』

.NET CLR ルーチンの特性の詳細については、以下を参照してください。

- 53 ページの『第 3 章 .NET 共通言語ランタイム (CLR) ルーチン』

すでに前提知識がある場合、組み込み SQL ルーチンの設計には、主に .NET CLR ルーチンの固有のフィーチャーおよび特性について習得することが関係しています。

- .NET CLR ルーチンでの SQL ステートメント実行のサポートを提供する組み込みアセンブリー (IBM.Data.DB2)
- .NET CLR ルーチンでサポートされている SQL データ・タイプ
- .NET CLR ルーチンのパラメーター
- .NET CLR ルーチンからの結果セットの戻り
- .NET CLR ルーチンのセキュリティーおよび実行制御モードの設定
- .NET CLR ルーチンに関する制約事項
- .NET CLR プロシージャーからの結果セットの戻り

.NET CLR の特性について理解した後、64 ページの『.NET CLR ルーチンの作成』を参照してください。

.NET CLR ルーチンでの SQL データ・タイプ表記

.NET CLR ルーチンでは、ルーチン・パラメーターとして、SQL ステートメントの実行に使われるパラメーター値として、および変数として、SQL データ・タイプ (データ型) 値を参照できます。ただし、値のアクセス時または取得時にデータの切り捨てや喪失が発生しないよう、適切な IBM SQL データ・タイプ値、IBM Data Server Provider for .NET データ・タイプ値、および .NET Framework データ・タイプ値を使用する必要があります。

.NET CLR ルーチンの作成に使われる CREATE PROCEDURE または CREATE FUNCTION ステートメント内のルーチン・パラメーター指定では、DB2 SQL データ・タイプ値が使用されます。ほとんどの SQL データ・タイプをルーチン・パラメーターとして指定できますが、例外もあります。

SQL ステートメントの実行で使用されるパラメーター値を指定するには、IBM Data Server Provider for .NET オブジェクトを使用する必要があります。 SQL ステートメントを表す `DB2Command` オブジェクトに追加されるパラメーターを表すために、`DB2Parameter` オブジェクトが使用されます。パラメーターのデータ・タイプ値を指定する際には、`IBM.Data.DB2Types` ネーム・スペース (名前空間) にある IBM Data Server Provider for .NET データ・タイプ値を使用する必要があります。`IBM.Data.DB2Types` ネーム・スペースには、サポートされるそれぞれの IBM SQL データ・タイプを表すクラスおよび構造体があります。

SQL データ・タイプ値を一時的に保持する可能性のあるパラメーターとローカル変数に関しては、IBM.Data.DB2Types ネーム・スペースで定義された適切な IBM Data Server Provider for .NET データ・タイプを使用する必要があります。

注: dbinfo 構造は、パラメーターとして CLR の関数やプロシージャーに渡されます。 CLR UDF のスクラッチパッドと呼び出しタイプも、パラメーターとして CLR ルーチンに渡されます。これらのパラメーターに該当する CLR データ・タイプの詳細については、以下の関連トピックを参照してください。

- CLR ルーチンのパラメーター

次の表は、DB2Type データ・タイプ、DB2 データ・タイプ、Informix® データ・タイプ、Microsoft .NET Framework タイプ、および DB2Types クラスと構造体の対応を示しています。

分類	DB2Types クラスと構造体	DB2Type データ・タイプ	DB2 データ・タイプ	Informix データ・タイプ	.NET データ・タイプ
Numeric	DB2Int16	SmallInt	SMALLINT	BOOLEAN, SMALLINT	Int16
Numeric	DB2Int32	Integer	INT	INTEGER、 INT、 SERIAL	Int32
Numeric	DB2Int64	BigInt	BIGINT	BIGINT, BIGSERIAL, INT8, SERIAL8	Int64
Numeric	DB2Real, DB2Real370	Real	REAL	REAL, SMALLFLOAT	Single
Numeric	DB2Double	Double	DOUBLE PRECISION	DECIMAL (≤31), DOUBLE PRECISION	Double
Numeric	DB2Double	Float	FLOAT	DECIMAL (32), FLOAT	Double
Numeric	DB2Decimal	Decimal	DECIMAL	MONEY	Decimal
Numeric	DB2DecimalFloat	DecimalFloat	DECFLOAT (16|34)[1,4]		Decimal
Numeric	DB2Decimal	Numeric	DECIMAL	DECIMAL (≤31), NUMERIC	Decimal
Date/Time	DB2Date	Date	DATE	DATETIME (日付精度)	Datetime
Date/Time	DB2Time	Time	TIME	DATETIME (時刻精度)	TimeSpan
Date/Time	DB2TimeStamp	Timestamp	TIMESTAMP	DATETIME (時刻と日付の精度)	DateTime
XML	DB2Xml	Xml[2]	XML		Byte[]
文字データ	DB2String	Char	CHAR	CHAR	String
文字データ	DB2String	VarChar	VARCHAR	VARCHAR	String
文字データ	DB2String	LongVarChar[1]	LONG VARCHAR	LVARCHAR	String
バイナリー・データ	DB2Binary	Binary	CHAR FOR BIT DATA		Byte[]

1. これらのデータ・タイプは、DB2 .NET 共通言語ランタイム・ルーチンではパラメーターとしてサポートされていません。
2. DB2Type.Xml 型の DB2ParameterClass.ParameterName プロパティーは、String、byte[]、DB2Xml、および XmlReader 型の変数を受け入れることができます。
3. これらのデータ・タイプは、DB2 for z/OS のみに該当します。
4. このデータ・タイプは DB2 for z/OS バージョン 9 以降のリリース、および DB2 for Linux, UNIX, and Windows バージョン 9.5 以降のリリースでのみサポートされます。

分類	DB2Types クラスと構造体	DB2Type データ・タイプ	DB2 データ・タイプ	Informix データ・タイプ	.NET データ・タイプ
バイナリー・データ	DB2Binary	Binary[3]	BINARY		Byte[]
バイナリー・データ	DB2Binary	VarBinary[3]	VARBINARY		Byte[]
バイナリー・データ	DB2Binary	LongVarBinary[1]	LONG VARCHAR FOR BIT DATA		Byte[]
グラフィック・データ	DB2String	Graphic	GRAPHIC		String
グラフィック・データ	DB2String	VarGraphic	VARGRAPHIC		String
グラフィック・データ	DB2String	LongVarGraphic[1]	LONG VARGRAPHIC		String
LOB データ	DB2Clob	Clob	CLOB	CLOB, TEXT	String
LOB データ	DB2Blob	Blob	BLOB	BLOB, BYTE	Byte[]
LOB データ	DB2Clob	DbClob	DBCLOB		String
Row ID	DB2RowId	RowId	ROWID		Byte[]

.NET CLR ルーチンのパラメーター

.NET CLR ルーチンのパラメーター宣言は、サポートされているいずれかのパラメーター・スタイルの要件と、ルーチンで使用している特定の .NET 言語のパラメーター・キーワードの要件を満たしている必要があります。

ルーチンがスクラッチパッドを使用する場合や、 dbinfo 構造を使用する場合や、PROGRAM TYPE MAIN パラメーター・インターフェースを使用する場合には、追加の考慮事項があります。このトピックでは、CLR パラメーターに関するすべての考慮事項を取り上げます。

CLR ルーチンでサポートされているパラメーター・スタイル

ルーチンのパラメーター・スタイルは、ルーチンの作成時にそのルーチンの CREATE ステートメントの EXTERNAL 節で指定する必要があります。外部 CLR ルーチン・コードのインプリメンテーションでは、そのパラメーター・スタイルを正確に反映しなければなりません。 CLR ルーチンでは、以下の DB2 パラメーター・スタイルがサポートされています。

- SQL (プロシージャーと関数に対応)
- GENERAL (プロシージャーにのみ対応)
- GENERAL WITH NULLS (プロシージャーにのみ対応)
- DB2SQL (プロシージャーと関数に対応)

これらのパラメーター・スタイルの詳細については、以下を参照してください。

- 26 ページの『外部ルーチンのパラメーター・スタイル』

CLR ルーチン・パラメーターの NULL 標識

CLR ルーチンに対して選択したパラメーター・スタイルのパラメーターに NULL 標識を指定する必要がある場合は、パラメーター・スタイルが NULL 標識のベクトルを必要とするときに、 NULL 標識を System.Int16 タイプ値または System.Int16[] 値として CLR ルーチンに渡します。

パラメーター・スタイル SQL のように、 NULL 標識を特殊パラメーターとしてルーチンに渡すことが必要なパラメーター・スタイルの場合は、各パラメーターで 1 つの System.Int16 NULL 標識が必要になります。

.NET 言語の場合は、特殊パラメーターの前に、そのパラメーターを値によって渡すのか、参照によって渡すのかを示すキーワードを付ける必要があります。ルーチン・パラメーターに使用するのと同じキーワードを、関連した NULL 標識パラメーターで使用しなければなりません。引数を値によって渡すのか、参照によって渡すのかを示すキーワードについては、次のセクションで詳しく取り上げます。

パラメーター・スタイル SQL や、他のサポートされているパラメーター・スタイルの詳細については、以下を参照してください。

- 26 ページの『外部ルーチンのパラメーター・スタイル』

CLR ルーチンのパラメーターを値によって渡すか、参照によって渡すか

中間言語 (IL) のバイト・コードにコンパイルする .NET 言語のルーチンの場合は、パラメーターを値によって渡すか、参照によって渡すか、入力専用パラメーターか、出力専用パラメーターか、といったパラメーターのプロパティーを示すキーワードをパラメーターの前に置く必要があります。

パラメーター・キーワードは、それぞれの .NET 言語によって異なります。例えば、 C# の場合、パラメーターを参照によって渡すことを示すパラメーター・キーワードは ref ですが、 Visual Basic の場合は、 byRef キーワードによって参照渡しのパラメーターであることを示します。ルーチンの CREATE ステートメントに指定する SQL パラメーターの使用法 (IN、OUT、INOUT) を示すために、キーワードを使用する必要があります。

DB2 ルーチンで .NET 言語ルーチン・パラメーターにパラメーター・キーワードを適用するときには以下の規則が適用されます。

- IN タイプ・パラメーターは、 C# ではパラメーター・キーワードなしで宣言し、 Visual Basic では byVal キーワードで宣言しなければなりません。
- INOUT タイプ・パラメーターは、参照渡しのパラメーターであることを示す言語固有のキーワードによって宣言しなければなりません。 C# の場合、該当するキーワードは ref です。 Visual Basic の場合、該当するキーワードは byRef です。
- OUT タイプ・パラメーターは、出力専用のパラメーターであることを示す言語固有のキーワードによって宣言しなければなりません。 C# の場合は、out キーワードを使用します。 Visual Basic の場合は、 byRef キーワードによってパラメーターを宣言する必要があります。出力専用パラメーターには、ルーチンが呼び出し元に戻る前に値を代入する必要があります。ルーチンが出力専用パラメーターに値を代入しない場合は、 .NET ルーチンのコンパイル時にエラーが発生します。

1 つの出力パラメーター language を戻すルーチンの C# のパラメーター・スタイル SQL のプロシージャーのプロトタイプは、次のようになります。

```
public static void Counter  (out String language,
                             out Int16  languageNullInd,
                             ref String sqlState,
```

```
                    String funcName,
                    String funcSpecName,
                ref String sqlMsgString,
                    Byte[] scratchPad,
                    Int32  callType);
```

ここでは、出力パラメーター language に関連する追加の NULL 標識パラメーター languageNullInd と、 SQLSTATE、ルーチン名、ルーチン固有名、オプションのユーザー定義 SQL エラー・メッセージを渡すパラメーターのために、パラメーター・スタイル SQL をインプリメントしています。また、パラメーターのパラメーター・キーワードを次のように指定しています。

- C# では、入力専用パラメーターにパラメーター・キーワードは必要ありません。
- C# では、'out' キーワードは、変数が出力パラメーター専用であり、その値が呼び出し元によって初期化されていないことを示します。
- C# では、'ref' キーワードは、パラメーターが呼び出し元によって初期化されており、ルーチンがオプションでこの値を変更できることを示します。

.NET 言語のパラメーター・キーワードについては、パラメーターの受け渡しに関する .NET 言語固有の資料を参照してください。

注: DB2 データベース・システムは、すべてのパラメーターに関するメモリーの割り振りを制御し、ルーチンとの間で受け渡しが行われるすべてのパラメーターへの CLR 参照を管理します。

プロシージャーの結果セットのためのパラメーター・マーカーは不要

プロシージャーのプロシージャー宣言内で、呼び出し元に戻される結果セットのためのパラメーター・マーカーは不要です。 CLR ストアード・プロシージャー内部からクローズされないカーソル・ステートメントはすべて、その呼び出し元に結果セットとして戻されます。

CLR ルーチンの結果セットの詳細については、以下を参照してください。

- 60 ページの『.NET CLR プロシージャーからの結果セットの戻り』

CLR パラメーターとしての dbinfo 構造

CLR ルーチンでは、ルーチンとの間で追加のデータベース情報パラメーターを受け渡すための dbinfo 構造を、 IL の dbinfo クラスの使用によってサポートしています。このクラスには、ストリングに関連した長さフィールドを除いて、 C 言語の sqludf_dbinfo 構造にあるすべてのエレメントが含まれています。各ストリングの長さは、 .NET 言語の各ストリングの Length プロパティーによって検出できます。

dbinfo クラスにアクセスするには、単に、対象のルーチンが含まれるファイルに IBM.Data.DB2 アセンブリーを含め、タイプ sqludf_dbinfo のパラメーターを、ルーチンのシグニチャーの中の、使用されているパラメーター・スタイルによって指定される位置に追加します。

CLR パラメーターとしての UDF スクラッチパッド

ユーザー定義関数のためのスクラッチパッドを要求する場合は、指定のサイズの System.Byte[] パラメーターとしてスクラッチパッドをルーチンに渡します。

CLR UDF の呼び出しタイプ・パラメーターまたは最終呼び出しパラメーター

最終呼び出しパラメーターまたは表関数を要求したユーザー定義関数の場合は、呼び出しタイプ・パラメーターを System.Int32 データ・タイプとしてルーチンに渡します。

CLR プロシージャーでサポートされている PROGRAM TYPE MAIN

.NET CLR プロシージャーでは、プログラム・タイプ MAIN がサポートされています。プログラム・タイプ MAIN を使用するプロシージャーを定義する場合は、以下のシグニチャーが必要です。

```
void functionname(Int32 NumParams, Object[] Params)
```

.NET CLR プロシージャーからの結果セットの戻り

呼び出し側のルーチンまたはアプリケーションに結果セットを戻す CLR プロシージャーを開発できます。 CLR 関数 (UDF) から結果セットを戻すことはできません。

始める前に

結果セットの .NET 表現は、DB2DataReader オブジェクトです。このオブジェクトは、 DB2Command オブジェクトのさまざまな実行呼び出しのいずれかから戻すことができます。戻すことができるのは、プロシージャーの戻りの前に Close() メソッドが明示的に呼び出されなかった DB2DataReader オブジェクトです。結果セットが呼び出し元に戻される順序は、 DB2DataReader オブジェクトがインスタンス化された順序と同じです。結果セットを戻すために、関数定義で追加のパラメーターを指定する必要はありません。

CLR ルーチンの作成方法を理解しておくと、CLR プロシージャーから結果を戻すための次の手順のステップを容易に理解できます。

- 65 ページの『DB2 コマンド・ウィンドウから .NET CLR ルーチンを作成する』

手順

CLR プロシージャーから結果セットを戻すには、次のようにします。

1. CLR ルーチンの CREATE PROCEDURE ステートメントでは、他にも適切な節を指定する中で、特に DYNAMIC RESULT SETS 節に、プロシージャーによって戻される結果セットの数と等しい値を指定しなければなりません。
2. プロシージャー宣言内で、呼び出し元に戻される結果セットのためのパラメーター・マーカーは不要です。
3. CLR ルーチンの .NET 言語インプリメンテーションでは、 DB2Connection オブジェクト、DB2Command オブジェクト、 DB2Transaction オブジェクトを作成します。 DB2Transaction オブジェクトは、データベース・トランザクションのロールバックとコミットを担当します。
4. DB2Transaction オブジェクトに対する DB2Command オブジェクトの Transaction プロパティーを初期化します。

5. 戻したい結果セットを定義する DB2Command オブジェクトの CommandText プロパティーにストリング照会を割り当てます。
6. DB2DataReader をインスタンス化し、 DB2Command オブジェクトの ExecuteReader メソッドの呼び出しの結果をそのインスタンスに割り当てます。照会の結果セットは、DB2DataReader オブジェクトに組み込まれます。
7. DB2DataReader オブジェクトの Close() メソッドは、プロシージャーが呼び出し元に戻る前に実行してはなりません。オープンしている DB2DataReader オブジェクトが、結果セットとして呼び出し元に戻されます。

   プロシージャーの戻り時に複数の DB2DataReader がオープンしたままになっていると、それぞれの DB2DataReader が作成順に呼び出し元に戻されます。CREATE PROCEDURE ステートメントで指定した数の結果セットだけが呼び出し元に戻されます。
8. .NET CLR 言語プロシージャーをコンパイルし、 CREATE PROCEDURE ステートメントの EXTERNAL 節で指定するロケーションにアセンブリーをインストールします。 CLR プロシージャーの CREATE PROCEDURE ステートメントをまだ実行していない場合は、実行してください。
9. CLR プロシージャー・アセンブリーを適切な場所にインストールして、CREATE PROCEDURE ステートメントを正常に実行したなら、CALL ステートメントでプロシージャーを呼び出し、結果セットが呼び出し元に戻されるのを確認してください。

CLR ルーチンのセキュリティーおよび実行モード

データベース管理者またはアプリケーション開発者であれば、DB2 外部ルーチンに関連付けられているアセンブリーを、実行時のルーチンのアクションを制限する厄介な改ざんから保護することもできます。 DB2 .NET 共通言語ランタイム (CLR) ルーチンは、実行制御モードの指定をサポートします。この指定により、実行時にルーチンが行えるアクションのタイプを識別できます。実行時に DB2 データベース・システムは、ルーチンがその指定された実行制御モードの有効範囲を越えてアクションを実行しようとしていないかどうかを検出できます。これはアセンブリーが破損していないかどうかを判別するときに役に立ちます。

CLR ルーチンの実行制御モードを設定するには、ルーチンの CREATE ステートメントにオプションの EXECUTION CONTROL 節を指定します。有効なモードは以下のとおりです。

- SAFE
- FILEREAD
- FILEWRITE
- NETWORK
- UNSAFE

既存の CLR ルーチンの実行制御モードに対して変更を加えるには、ALTER PROCEDURE または ALTER FUNCTION ステートメントを実行します。

CLR ルーチンで EXECUTION CONTROL 節が指定されない場合、デフォルトで CLR ルーチンは、制限度の最も高い実行制御モードである SAFE モードを使って実行されます。この実行制御モードを使って作成されるルーチンは、データベー

ス・マネージャーによって制御されるリソースにのみアクセスできます。これより制限度の低い実行制御モードのルーチンでは、ファイルへのアクセス (FILEREAD または FILEWRITE)、または Web ページへのアクセスなどのネットワーク操作の実行 (NETWORK) が可能です。実行制御モード UNSAFE は、ルーチンの振る舞いに制限を課さないことを指定します。 UNSAFE 実行制御モードを使って定義されるルーチンは、バイナリー・コードを実行することができます。

これらのモードは許容アクションの階層を表し、階層内の高いレベルのモードは、それよりも低いレベルの許容アクションを包含します。例えば、実行制御モードが NETWORK のルーチンは、インターネット上の Web ページへのアクセス、ファイルの読み取りおよび書き込み、およびデータベース・マネージャーが制御しているリソースへのアクセスが可能です。できる限り制限度の高い実行制御モードを使用することと、UNSAFE モードの使用を避けることをお勧めします。

CLR ルーチンがその実行制御モードの有効範囲を越えてアクションを試行していることを DB2 データベース・システムが実行時に検出した場合、DB2 データベース・システムはエラー (SQLSTATE 38501) を戻します。

EXECUTION CONTROL 節は、LANGUAGE CLR ルーチンに対してのみ指定することができます。 EXECUTION CONTROL 節の適用度の有効範囲は .NET CLR ルーチンそのものに限られ、他のいかなるルーチンにも (EXECUTION CONTROL 節によって呼び出せるとしても) 拡張されません。

サポートされる実行制御モードの詳細については、該当するルーチン・タイプの CREATE ステートメントの構文を参照してください。

.NET CLR ルーチンに関する制約事項

すべての外部ルーチンまたは特定のルーチン・クラス (プロシージャーや UDF) のインプリメンテーションに当てはまる一般的な制約事項は、 CLR ルーチンにも当てはまります。また、CLR ルーチンだけに該当する制約事項もいくつかあります。ここでは、その種の制約事項を取り上げます。

LANGUAGE CLR 節付きの CREATE METHOD ステートメントはサポートされていない

CLR アセンブリーを参照する DB2 データベース構造化タイプの外部メソッドは作成できません。 LANGUAGE 節に CLR という値を指定した CREATE METHOD ステートメントの使用は、サポートされていません。

CLR プロシージャーは NOT FENCED プロシージャーとしてインプリメントできない

CLR プロシージャーを unfenced プロシージャーとして実行することはできません。 CLR プロシージャーを作成する CREATE PROCEDURE ステートメントでは、 NOT FENCED 節を指定できません。

EXECUTION CONTROL 節がルーチンに含まれているロジックを制限する

EXECUTION CONTROL 節および関連値は、.NET CLR ルーチンで実行できるロジックおよび操作のタイプを決定します。デフォルトでは、EXECUTION CONTROL 節の値は SAFE に設定されます。ファイルの読み取り、ファイルへの書き込みを行ったり、インターネットにアクセスするルーチン・ロジックの場合、EXECUTION CONTROL 節にはデフォルト以外の値および制約の少ない値を指定する必要があります。

CLR ルーチンの 10 進数の最大精度は 29、最大スケールは 28

DB2 データベースの DECIMAL データ・タイプは、31 桁の精度と 28 桁のスケールで表現します。 .NET CLR の System.Decimal データ・タイプは、29 桁の精度と 28 桁のスケールに制限されます。したがって、DB2 の外部 CLR ルーチンで、(2^96)-1 (29 桁の精度と 28 桁のスケールで表せる最大値) より大きい値を持つ System.Decimal データ・タイプに値を割り当てないでください。こうした割り当てが行われると、DB2 データベース・システムで実行時エラー (SQLSTATE 22003、SQLCODE -413) が発生します。ルーチンの CREATE ステートメントの実行時にスケールが 28 桁より大きい DECIMAL データ・タイプ・パラメーターが定義されていると、DB2 データベース・システムでエラー (SQLSTATE 42613、SQLCODE -628) が発生します。

DB2 データベース・システムでサポートされている精度とスケールの最大桁数を利用した 10 進数値をルーチンで操作する必要がある場合は、別のプログラム言語 (Java など) で外部ルーチンをインプリメントしてください。

CLR ルーチンでサポートされていないデータ・タイプ

CLR ルーチンでは、以下の DB2 SQL データ・タイプがサポートされていません。

- LONG VARCHAR
- LONG VARCHAR FOR BIT DATA
- LONG GRAPHIC
- ROWID

64 ビットのインスタンス上での 32 ビットの CLR ルーチンの実行

現時点で、 64 ビットのオペレーティング・システムには .NET Framework をインストールできないので、 CLR ルーチンを 64 ビットのインスタンスで実行することはできません。

.NET CLR はセキュリティー・プラグインのインプリメントに非対応

セキュリティー・プラグイン・ライブラリーのソース・コードのコンパイルおよびリンクにおいて、.NET CLR はサポートされません。

.NET CLR ルーチンの作成

.NET CLR ルーチンの作成は、ルーチンを DB2 データベース・サーバーに定義する CREATE ステートメントを実行する工程と、そのルーチン定義に対応するルーチンの実装を開発する工程から成ります。

始める前に

- 53 ページの『第 3 章 .NET 共通言語ランタイム (CLR) ルーチン』を確認します。
- DB2 バージョン 9 サーバー (インスタンスおよびデータベースを含む) にアクセスしていることを確認します。
- オペレーティング・システムのバージョン・レベルが、DB2 データベース製品でサポートされているものであることを確認します。
- Microsoft .NET 開発ソフトウェアのバージョン・レベルが、.NET CLR ルーチン開発でサポートされているものであることを確認します。 54 ページの『.NET CLR 言語での外部ルーチン開発のサポート』を参照してください。
- CREATE PROCEDURE または CREATE FUNCTION ステートメントを実行する権限。

CLR ルーチンに関連した制約事項のリストについては、以下を参照してください。

- 62 ページの『.NET CLR ルーチンに関する制約事項』

このタスクについて

.NET CLR ルーチンを作成する方法は、以下に従ってください。

- IBM Database Add-Ins for Microsoft Visual Studio に備えられているグラフィック・ツールを使用する
- DB2 コマンド・ウィンドウを使用する

一般に、最も簡単な方法は、IBM Database Add-Ins for Microsoft Visual Studio を使用して、.NET CLR ルーチンを作成する方法です。これが使用できない場合は、DB2 コマンド・ウィンドウがコマンド行インターフェースを介して同様のサポートを提供します。

以下のいずれかのインターフェースからの .NET CLR ルーチンの作成:

手順

- IBM Database Add-Ins for Microsoft Visual Studio もインストールする場合には、Visual Studio .NET。 Add-In がインストールされているなら、DB2 データベース・サーバーで動作する .NET CLR ルーチンを作成するために、Visual Studio .NET に統合されたグラフィック・ツールを使用できます。
- DB2 コマンド・ウィンドウ

次のタスク

DB2 コマンド・ウィンドウから .NET CLR ルーチンを作成するには、以下を参照してください。

- 65 ページの『DB2 コマンド・ウィンドウから .NET CLR ルーチンを作成する』

DB2 コマンド・ウィンドウから .NET CLR ルーチンを作成する

中間言語アセンブリーを参照するプロシージャーと関数の作成方法は、他の外部ルーチンの場合と同じです。

始める前に

- CLR ルーチンのインプリメンテーションに関する知識。 CLR ルーチンの概要や CLR のフィーチャーについては、以下を参照してください。
  - 53 ページの『第 3 章 .NET 共通言語ランタイム (CLR) ルーチン』
- データベース・サーバーが Microsoft .NET Framework をサポートする Windows オペレーティング・システムを実行していること。
- Microsoft .NET Framework ソフトウェアのサポートされているバージョンもそのサーバー上にインストールされている必要があります。 .NET Framework は、単独で、または Microsoft .NET Framework Software Development Kit の一部として入手可能です。
- サポートされている DB2 データベース製品、または IBM Data Server Client をインストールする必要があります。DB2 データベース製品のインストール要件を参照してください。
- 外部ルーチンを作成する CREATE ステートメントの実行権限。 CREATE PROCEDURE ステートメントまたは CREATE FUNCTION ステートメントの実行に必要な特権については、該当するステートメントの詳細を参照してください。

制約事項

CLR ルーチンに関連した制約事項のリストについては、以下を参照してください。

- 62 ページの『.NET CLR ルーチンに関する制約事項』

このタスクについて

.NET 言語で外部ルーチンをインプリメントするのは、以下のような場合です。

- データベースにアクセスするルーチンや、データベースの外部でアクションを実行するルーチンに複雑なロジックをカプセル化したい場合。
- 複数のアプリケーション、CLP、他のルーチン (プロシージャー、関数 (UDF)、メソッド)、トリガーのいずれかから、カプセル化されたロジックを呼び出す必要のある場合。
- そのロジックのコーディングに .NET 言語が最も使いやすいと感じる場合。

手順

1. CLR でサポートされている言語でルーチン・ロジックをコーディングします。
   - .NET CLR ルーチンおよび .NET CLR ルーチン・フィーチャーの概要については、『始める前に』のセクションで示したトピックを参照してください。
   - ルーチンで SQL を実行する場合は、`IBM.Data.DB2` アセンブリーを使用するか、インポートします。
   - DB2 SQL データ・タイプにマップするデータ・タイプを使用して、ホスト変数とパラメーターを正しく宣言します。 DB2 データ・タイプと .NET データ・タイプの間のマッピングについては、以下を参照してください。

– 55 ページの『.NET CLR ルーチンでの SQL データ・タイプ表記』

- DB2 がサポートするパラメーター・スタイルのいずれかを使用し、 .NET CLR ルーチンのパラメーター要件に従って、パラメーターとパラメーターの NULL 標識を宣言する必要があります。また、UDF のスクラッチパッドと DBINFO クラスをパラメーターとして CLR ルーチンに渡します。パラメーターとプロトタイプ宣言の詳細については、以下を参照してください。
  – 57 ページの『.NET CLR ルーチンのパラメーター』
- ルーチンがプロシージャーで、ルーチンの呼び出し元に結果セットを戻したい場合、結果セット用のパラメーターは必要ありません。 CLR ルーチンから結果セットを戻す方法の詳細については、以下を参照してください。
  – 60 ページの『.NET CLR プロシージャーからの結果セットの戻り』
- 必要に応じてルーチンの戻り値を設定します。 CLR スカラー関数の場合は、値を戻す前に戻り値を設定する必要があります。 CLR 表関数の戻りコードは、表関数の呼び出しごとに出力パラメーターとして指定しなければなりません。 CLR プロシージャーは戻り値を戻しません。

2. CLR で実行できる中間言語 (IL) アセンブリーにコードをビルドします。 DB2 データベースにアクセスする CLR .NET ルーチンの作成方法については、以下のトピックを参照してください。
   - 「*ADO.NET および OLE DB アプリケーションの開発*」内の『Common Language Runtime (CLR) .NET ルーチンの構築』
3. そのアセンブリーをデータベース・サーバー上の DB2 *function* ディレクトリーにコピーします。 DB2 ルーチンに関連したアセンブリーまたはライブラリーは、 function ディレクトリーに保管することをお勧めします。 function ディレクトリーの詳細については、 CREATE PROCEDURE ステートメントまたは CREATE FUNCTION ステートメントのいずれかの EXTERNAL 節を参照してください。

   アセンブリーをサーバー上の別のディレクトリーにコピーすることもできますが、ルーチンを正常に呼び出すには、アセンブリーの完全修飾パス名をメモしておく必要があります。次のステップでこれが必要になるからです。
4. 該当するルーチン・タイプの SQL 言語 CREATE ステートメント (CREATE PROCEDURE または CREATE FUNCTION) を動的または静的に実行します。
   - `LANGUAGE` 節に、`CLR` という値を指定します。
   - `PARAMETER STYLE` 節に、ルーチン・コードでインプリメントした有効なパラメーター・スタイルの名前を指定します。
   - `EXTERNAL` 節に、ルーチンと関連したアセンブリーの名前を指定します。そのためには、以下のいずれかの値を使用します。
     – ルーチン・アセンブリーの完全修飾パス名。
     – function ディレクトリーを基準にしたルーチン・アセンブリーの相対パス名。

     EXTERNAL 節にライブラリーの完全修飾パス名または相対パス名を指定しない場合、 DB2 データベース・システムはデフォルトで、 function ディレクトリー内でアセンブリーの名前を探します。

CREATE ステートメントを実行した後、 EXTERNAL 節に指定したアセンブリーを DB2 データベース・システムが見つけられない場合は、理由コード 1 のエラー (SQLCODE -20282) が発生します。

- DYNAMIC RESULT SETS 節に、ルーチンによって戻される結果セットの最大数と等しい整数値を指定します。
- CLR プロシージャーに NOT FENCED 節を指定することはできません。CLR プロシージャーはデフォルトで、FENCED プロシージャーとして実行されます。

.NET CLR ルーチン・コードのビルド

.NET CLR ルーチンのインプリメンテーション・コードを作成したなら、ルーチン・アセンブリーをデプロイしてルーチンを呼び出せるようにするには、その前にインプリメンテーション・コードをビルドする必要があります。 .NET CLR ルーチンの構築に必要なステップは、外部ルーチンの構築に必要なステップに似ていますが、いくつかの点で異なります。

手順

.NET CLR ルーチンをビルドするには、次の 3 とおりの方法があります。

- IBM Database Add-Ins for Microsoft Visual Studio に備えられているグラフィック・ツールを使用する
- DB2 サンプル・バッチ・ファイルを使用する
- DB2 コマンド・ウィンドウからコマンドを入力する

ルーチン用の DB2 サンプル・ビルド・スクリプトおよびバッチ・ファイルは、デフォルトでサポートされるコンパイラーを使用する特定のオペレーティング・システムのためのユーザー作成ルーチンだけでなく、DB2 サンプル・ルーチン (プロシージャーおよびユーザー定義関数) 用に設計されています。
C# および Visual Basic 用の DB2 サンプル・ビルド・スクリプトおよびバッチ・ファイルには個別セットがあります。一般に、.NET CLR ルーチンをビルドするには、グラフィック・ツールまたはビルド・スクリプト (必要に応じて容易に変更できる) を使用するのが最も簡単な方法ですが、DB2 コマンド・ウィンドウからルーチンをビルドする方法も知っておくと、いろいろと役に立ちます。

サンプル・ビルド・スクリプトを使った .NET 共通言語ランタイム (CLR) ルーチン・コードのビルド

.NET 共通言語ランタイム (CLR) ルーチンのソース・コードのビルドは、.NET CLR ルーチン作成のサブタスクです。この作業は DB2 サンプル・バッチ・ファイルを使用することにより、迅速かつ簡単に行うことができます。

サンプル・ビルド・スクリプトは、SQL ステートメントを含むソース・コードにも含まないソース・コードにも使用できます。ビルド・スクリプトはビルド済みアセンブリーの `function` ディレクトリーへのコンパイル、リンク、およびデプロイメントを処理します。

別の方法として、Visual Studio .NET によって .NET CLR ルーチン・コードのビルド作業を単純化するか、または DB2 サンプル・ビルド・スクリプトのステップを手動で行うことができます。次のトピックを参照してください。

- Visual Studio .NET での .NET 共通言語ランタイム (CLR) ルーチンの構築
- DB2 コマンド・ウィンドウによる .NET 共通言語ランタイム (CLR) ルーチンの作成

C# および Visual Basic .NET CLR ルーチンをビルドするためのプログラミング言語固有のサンプル・ビルド・スクリプトには **bldrtn** という名前があります。これを使用してビルドできるサンプル・プログラムと一緒に、次の DB2 ディレクトリーに置かれています。

- C: の場合 `sqllib/samples/cs/`
- C++: の場合 `sqllib/samples/vb/`

bldrtn スクリプトを使用して、プロシージャーとユーザー定義関数の両方を含むソース・コード・ファイルをビルドすることができます。このスクリプトは、以下の処理を行います。

- ユーザーが指定したデータベースとの接続を確立する
- ソース・コードをコンパイルしてリンクし、.DLL というファイル接尾部を持つアセンブリーを生成する
- アセンブリーをデータベース・サーバーの DB2 function ディレクトリーにコピーする

bldrtn スクリプトは、次の 2 つの引数を受け入れます。

- ソース・コード・ファイルの名前 (ファイル接尾部なし)
- 接続が確立される先のデータベースの名前

データベース・パラメーターはオプションです。データベース名を指定しない場合は、プログラムはデフォルトの sample データベースを使用します。ルーチンはデータベースがあるインスタンスと同じインスタンス上で構築する必要があるため、ユーザー ID とパスワードのための引数は不要です。

前提条件

- 必要な .NET CLR ルーチンのオペレーティング・システムおよび開発ソフトウェアの前提要件を満たしている必要があります。 『.NET CLR ルーチンの開発のサポート』を参照してください。
- 1 つ以上のルーチン・インプリメンテーションを含むソース・コード・ファイル
- 現行 DB2 インスタンス内の、ルーチンが作成されるデータベースの名前。

手順

1 つ以上のルーチン・コード・インプリメンテーションを含むソース・コード・ファイルをビルドするには、次のステップを実行します。

1. DB2 コマンド・ウィンドウをオープンする。
2. ソース・コード・ファイルを **bldrtn** スクリプト・ファイルと同じディレクトリーにコピーする。

3. サンプル・データベースにルーチンを作成する場合は、次のように、ビルド・スクリプト名に続いて、ソース・コード・ファイルの名前 (.cs または .vb ファイル拡張子なし) を入力する。

   ```
   bldrtn file-name
   ```

   別のデータベースにルーチンを作成する場合は、次のように、ビルド・スクリプト名、ソース・コード・ファイル名 (ファイル拡張子なし)、データベース名を入力します。

   ```
   bldrtn file-name database-name
   ```

   スクリプトによってソース・コードのコンパイルとリンクが行われ、アセンブリーが生成されます。次に、スクリプトによってアセンブリーがデータベース・サーバーの function ディレクトリーにコピーされます。
4. ルーチン・インプリメンテーションを含むソース・コード・ファイルをビルドしたのが今回が初めてでない場合は、データベースを停止してから再始動することにより、DB2 データベース・システムで新しいバージョンの共有ライブラリーが使用されるようにする。これを行うには、コマンド行で **db2stop** に続けて **db2start** を入力します。

ルーチン共有ライブラリーのビルドとデータベース・サーバー上の function ディレクトリーへのデプロイが正常に終了したなら、C および C++ ルーチンの作成作業に関連したステップを完了する必要があります。

.NET CLR ルーチンの作成には、ソース・コード・ファイルにインプリメントされたルーチンごとに CREATE ステートメントを実行するというステップが含まれます。ルーチンの作成が完了すると、ルーチンを呼び出すことができます。

DB2 コマンド・ウィンドウからの .NET 共通言語ランタイム (CLR) ルーチン・コードのビルド

.NET CLR ルーチンのソース・コードのビルドは、.NET CLR ルーチン作成の副次作業です。この作業は、DB2 コマンド・ウィンドウから手動で行うことができます。ルーチン・コードに SQL ステートメントが含まれているかどうかにかかわらず、同じ手順で進めることができます。この作業ステップには、.NET CLR がサポートするプログラミング言語で作成されたソース・コードを、.DLL ファイル接尾部を持つアセンブリーにコンパイルする処理が含まれます。

始める前に

別の方法として、Visual Studio .NET で行う方法と、DB2 サンプル・ビルド・スクリプトを使用する方法により、.NET CLR ルーチン・コードをビルド作業が簡単になります。次のトピックを参照してください。

- Visual Studio .NET での .NET 共通言語ランタイム (CLR) ルーチンの構築
- サンプル・ビルド・スクリプトを使った .NET 共通言語ランタイム (CLR) ルーチンの構築
- 必須のオペレーティング・システムと .NET CLR ルーチン開発のソフトウェア前提条件が満たされていること。
- 1 つ以上の .NET CLR ルーチン・インプリメンテーションを含む、サポートされている .NET CLR プログラミング言語で作成されたソース・コード。

- 現行 DB2 インスタンス内の、ルーチンが作成されるデータベースの名前。
- .NET CLR ルーチンをビルドするのに必要な、オペレーティング環境固有のコンパイルおよびリンク・オプション。

手順

1 つ以上の .NET CLR ルーチン・コード・インプリメンテーションを含むソース・コード・ファイルをビルドするには、以下のようにします。

1. DB2 コマンド・ウィンドウをオープンする。
2. ソース・コード・ファイルを含むディレクトリーにナビゲートする。
3. ルーチンが作成されるデータベースとの接続を確立する。
4. ソース・コード・ファイルをコンパイルする。
5. ソース・コード・ファイルをリンクして共有ライブラリーを生成する。ここでは、DB2 データベース・システム固有のコンパイルおよびリンク・オプションを使用する必要があります。
6. .DLL ファイル接尾部を持つアセンブリー・ファイルを、データベース・サーバーの DB2 function ディレクトリーにコピーする。
7. ルーチン・インプリメンテーションを含むソース・コード・ファイルをビルドしたのが今回が初めてでない場合は、データベースを停止してから再始動することにより、DB2 データベース・システムで新しいバージョンの共有ライブラリーが使用されるようにする。 これを行うには、**db2stop** コマンドに続けて **db2start** コマンドを実行します。

タスクの結果

ルーチン・ライブラリーのビルドとデプロイが正常に終了したなら、.NET CLR ルーチンの作成作業に関連したステップを完了する必要があります。 .NET CLR ルーチンの作成には、ソース・コード・ファイルにインプリメントされたルーチンごとに CREATE ステートメントを実行するというステップが含まれます。ルーチンを呼び出せるようにするには、このステップも完了しなければなりません。

例

以下は、.NET CLR ソース・コード・ファイルの再ビルドを示した例です。ルーチン・インプリメンテーションを含む、`myVBfile.vb` という名前の Visual Basic コード・ファイルと `myCSfile.cs` という名前の C# コード・ファイルの、両方の場合のステップを示しています。ルーチンは Windows 2000 オペレーティング・システム上でビルドされ、Microsoft .NET Framework 1.1 を使用して 64 ビット・アセンブリーが生成されます。

1. DB2 コマンド・ウィンドウをオープンする。
2. ソース・コード・ファイルを含むディレクトリーにナビゲートする。
3. ルーチンが作成されるデータベースとの接続を確立する。

   ```
   db2 connect to database-name
   ```

4. コンパイルとリンクの推奨オプションを使用してソース・コード・ファイルをコンパイルする (`$DB2PATH` は DB2 インスタンスのインストール・パスです。コマンドを実行する前にこの値を置き換えてください)。

```
C# example
==================
csc /out:myCSfile.dll /target:library
    /reference:$DB2PATH%¥bin¥netf11¥IBM.Data.DB2.dll myCSfile.cs

Visual Basic example
====================
vbc /target:library /libpath:$DB2PATH¥bin¥netf11
    /reference:$DB2PATH¥bin¥netf11¥IBM.Data.DB2.dll
    /reference:System.dll
    /reference:System.Data.dll myVBfile.vb
```

エラーがある場合は、コンパイラーによって出力が生成されます。このステップで、myfile.exp という名前のエクスポート・ファイルが生成されます。

5. 共有ライブラリーをデータベース・サーバーの DB2 function ディレクトリーにコピーする。

```
C# example
====================
rm -f ~HOME/sqllib/function/myCSfile.DLL
cp myCSfile $HOME/sqllib/function/myCSfile.DLL

Visual Basic example
====================
rm -f ~HOME/sqllib/function/myVBfile.DLL
cp myVBfile $HOME/sqllib/function/myVBfile.DLL
```

このステップで、DB2 がルーチン・ライブラリーを探すデフォルト・ディレクトリーにルーチン・ライブラリーが置かれます。ルーチン・ライブラリーのデプロイについての詳細は、.NET CLR ルーチンの作成に関するトピックを参照してください。

6. これは以前にビルドされたルーチン・ソース・コード・ファイルの再ビルドなので、データベースを停止して再始動する。

```
db2stop
db2start
```

.NET CLR ルーチンの構築は一般に、オペレーティング環境固有のサンプル・ビルド・スクリプトを使用して行うのが最も簡単です。このスクリプトは、コマンド行からルーチンを構築する方法の参考として使用できます。

CLR .NET ルーチンのコンパイルとリンクのオプション

Windows 上で Microsoft Visual Basic .NET コンパイラーまたは Microsoft C# コンパイラーのどちらかを使用して、Common Language Runtime (CLR).NET ルーチンを構築する場合に DB2 で使用可能なコンパイルとリンクのオプション。これらは、samples¥.NET¥cs¥bldrtn.bat および samples¥.NET¥vb¥bldrtn.bat バッチ・ファイルの中で示されているものです。

Microsoft C# コンパイラーを使用する場合の bldrtn のコンパイルとリンクのオプション

Microsoft C# コンパイラーを使用したコンパイルとリンクのオプション

csc Microsoft C# コンパイラー

/out:%1.dll /target:library
ダイナミック・リンク・ライブラリーをストアード・プロシージャーのアセンブリー DLL として出力します。

/debug デバッガーを使用します。

/lib: "%DB2PATH%"¥bin¥netf20¥
.NET Framework バージョン 2.0 のライブラリー・パスを使用します。

アプリケーション用にサポートされている .NET Framework のバージョンはいくつかあります。バージョン 2.0、バージョン 3.0、およびバージョン 3.5 です。それぞれにダイナミック・リンク・ライブラリーがあります。.NET Framework バージョン 1.1 の場合には "%DB2PATH%"¥bin¥netf11 サブディレクトリーを使用します。 .NET Framework バージョン 2.0、3.0、および 3.5 の場合には "%DB2PATH%"¥bin¥netf20 サブディレクトリーを使用します。

/reference:IBM.Data.DB2.dll
IBM Data Server Provider for .NET の DB2 ダイナミック・リンク・ライブラリーを使用します。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

Microsoft Visual Basic .NET コンパイラーを使用する場合の bldrtn のコンパイルとリンクのオプション

vbc Microsoft Visual Basic .NET コンパイラー。

/out:%1.dll /target:library
ダイナミック・リンク・ライブラリーをストアード・プロシージャーのアセンブリー DLL として出力します。

/debug デバッガーを使用します。

/libpath:"%DB2PATH%"¥bin¥netf20¥
.NET Framework バージョン 2.0 のライブラリー・パスを使用します。

アプリケーション用にサポートされている .NET Framework のバージョンはいくつかあります。バージョン 2.0、バージョン 3.0、およびバージョン 3.5 です。それぞれにダイナミック・リンク・ライブラリーがあります。.NET Framework バージョン 1.1 の場合には "%DB2PATH%"¥bin¥netf11 サブディレクトリーを使用します。 .NET Framework バージョン 2.0、3.0、および 3.5 の場合には "%DB2PATH%"¥bin¥netf20 サブディレクトリーを使用します。

/reference:IBM.Data.DB2.dll
IBM Data Server Provider for .NET の DB2 ダイナミック・リンク・ライブラリーを使用します。

/reference:System.dll
Microsoft Windows の System ダイナミック・リンク・ライブラリーを参照します。

/reference:System.Data.dll
Microsoft Windows の System Data ダイナミック・リンク・ライブラリーを参照します。

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。

.NET CLR ルーチンのデバッグ

ルーチンの作成やルーチンの呼び出しに失敗する場合、またはルーチンの呼び出し時にそれが期待どおりに動作または実行しない場合は、.NET CLR ルーチンのデバッグが必要となる可能性があります。

このタスクについて

.NET CLR ルーチンをデバッグするときには、以下を考慮に入れてください。

手順

- .NET CLR ルーチン開発用にサポートされるオペレーティング・システムが使用されていることを確認します。
- .NET CLR ルーチン開発用にサポートされる DB2 データベース・サーバーと DB2 クライアントの両方が使用されていることを確認します。
- サポートされる Microsoft .NET Framework 開発ソフトウェアが使用されていることを確認します。
- ルーチンの作成が失敗した場合、以下のようにします。
  - ユーザーが CREATE PROCEDURE または CREATE FUNCTION ステートメントを実行するのに必要な権限および特権を持っていることを確認します。
- ルーチンの呼び出しが失敗した場合、以下のようにします。
  - ユーザーがルーチンを実行するための権限を持っていることを確認します。エラー (SQLCODE -551, SQLSTATE 42501) が発生する場合、呼び出し側に、ルーチンの EXECUTE 特権が付与されていないことが原因である可能性が高いと言えます。この特権を付与できるのは、SECADM 特権、ACCESSCTRL 特権を持つユーザーか、ルーチンに関する EXECUTE WITH GRANT OPTION 特権を持つユーザーです。
  - CREATE ステートメントで使用されるルーチン用のルーチン・パラメーター・シグニチャーが、ルーチン・インプリメンテーションのルーチン・パラメーター・シグニチャーと一致することを確認します。
  - ルーチン・インプリメンテーションで使用されるデータ・タイプが、CREATE ステートメントのルーチン・パラメーター・シグニチャーで指定されたデータ・タイプと互換性があることを確認します。
  - ルーチン・インプリメンテーションでは、パラメーターを (値別または参照別に) 受け渡さなければならないメソッドを示すために使用される .NET CLR 言語固有キーワードが有効であることを確認します。
  - CREATE PROCEDURE または CREATE FUNCTION ステートメントの EXTERNAL 節で指定される値が、ルーチン・インプリメンテーションを含む .NET CLR アセンブリーが、DB2 データベース・サーバーがインストールされているコンピューターのファイル・システム上にある場所と一致することを確認します。

- ルーチンが関数の場合、適用できる呼び出しタイプのすべてがルーチン・インプリメンテーションで正しくプログラムされていることを確認します。 FINAL CALL 節を使用してルーチンが定義されている場合には、これが特に重要です。

- ルーチンが期待どおりに動作しない場合、以下のようにします。
  - ルーチンが診断情報をグローバル・アクセス可能なディレクトリーにあるファイルに出力するように、ルーチンを変更します。診断情報を画面に出力することは、.NET CLR ルーチンからは行えません。 DB2 データベース・マネージャーまたは DB2 データベースによって使用されるディレクトリー内のファイルに出力を送らないでください。
  - ルーチンのエントリー・ポイントを直接呼び出す単純な .NET アプリケーションを作成して、ルーチンをローカル側でデバッグします。 Microsoft Visual Studio .NET でのデバッグ・フィーチャーの使用方法について詳しくは、Microsoft Visual Studio .NET コンパイラーの資料を調べてください。

タスクの結果

.NET CLR ルーチンの作成および呼び出しに関連する共通エラーについて詳しくは、以下を参照してください。

- 『.NET CLR ルーチンに関連したエラー』

.NET CLR ルーチンに関連したエラー

外部ルーチンのインプリメンテーションは基本的に共通ですが、CLR ルーチンに固有の DB2 データベース・システム・エラーもいくつか発生します。

このリファレンスでは、最もよく発生する .NET CLR 関連のエラーを SQLCODE ごと、または動作ごとに取り上げ、それぞれのデバッグのための提案を示します。ルーチンに関連した DB2 データベース・システム・エラーは、以下のように分類できます。

ルーチン作成時のエラー
ルーチンを作成する CREATE ステートメントの実行時に発生するエラーです。

ルーチン実行時のエラー
ルーチンの呼び出し時または実行時に発生するエラーです。

DB2 データベース・システムのルーチンに関連したエラーが DB2 によっていつ検出されるかにかかわりなく、エラー・メッセージのテキストには、エラーの原因と、その問題を解決するためにユーザーが行うべき処置が詳しく説明されています。ルーチン・エラーのシナリオに関するその他の情報については、**db2diag** 診断ログ・ファイルを参照してください。

CLR ルーチン作成時のエラー

SQLCODE -451、SQLSTATE 42815
このエラーは、 LANGUAGE 節に CLR という値を指定した外部メソッド宣言を含んだ CREATE TYPE ステートメントを実行しようとしたときに発生します。現時点で、 CLR アセンブリーを参照する構造化タイプの DB2

外部メソッドは作成できません。 LANGUAGE 節を変更して、そのメソッドでサポートされている言語を指定し、その代替言語でメソッドをインプリメントしてください。

SQLCODE -449、SQLSTATE 42878

CLR ルーチンを作成する CREATE ステートメントの EXTERNAL NAME 節に、無効な形式のライブラリー指定または関数指定が含まれています。 CLR 言語の場合、 EXTERNAL 節の値は、'<a>:<b>!<c>' という厳密な形式でなければなりません。それぞれの意味は、次のとおりです。

- <a> は CLR アセンブリー・ファイルです。このファイルの中に対象のクラスが存在しています。
- <b> はクラスです。このクラスの中に呼び出し対象のメソッドが含まれています。
- <c> は呼び出し対象のメソッドです。

単一引用符、オブジェクト ID、分離文字の間に先行ブランクと末尾ブランクを入れてはなりません (例えば、' <a> ! <b> ' は無効です)。ただし、プラットフォームによっては、パス名とファイル名にブランクを組み込むことは可能です。どんなファイル名についても、短形式の名前 (例: math.dll) と完全修飾パス名 (例: d:¥udfs¥math.dll) のどちらを指定してもかまいません。短形式のファイル名を使用するか、プラットフォームが UNIX であるか、ルーチンが LANGUAGE CLR ルーチンである場合は、対象のファイルが function ディレクトリーに存在している必要があります。プラットフォームが Windows であり、ルーチンが LANGUAGE CLR ルーチンでない場合は、対象のファイルがシステム PATH に存在している必要があります。ファイル名には、常にファイル拡張子 (例: .a (UNIX の場合)、 .dll (Windows の場合)) を付けてください。

CLR ルーチン実行時のエラー

SQLCODE -20282、SQLSTATE 42724、理由コード 1

ルーチンを作成する CREATE ステートメントの EXTERNAL 節で指定した外部アセンブリーが見つかりません。

- EXTERNAL 節で正しいルーチン・アセンブリー名を指定したかどうか、そのアセンブリーが指定のロケーションに存在するかどうかを確認してください。 EXTERNAL 節で対象のアセンブリーの完全修飾パス名を指定していない場合、 DB2 データベース・システムは、そのパス名を、 DB2 データベース・システムの function ディレクトリーを基準にしたアセンブリーの相対パス名と見なします。

SQLCODE -20282、SQLSTATE 42724、理由コード 2

ルーチンを作成する CREATE ステートメントの EXTERNAL 節で指定したロケーションでアセンブリーが見つかりましたが、そのアセンブリーの中に、EXTERNAL 節で指定したクラスに一致するクラスが存在しません。

- EXTERNAL 節で指定したアセンブリー名がルーチンの正しいアセンブリー名かどうか、そのアセンブリーが指定のロケーションに存在するかどうかを確認してください。
- EXTERNAL 節で指定したクラス名が正しいクラス名かどうか、そのクラスが指定のアセンブリーの中に存在するかどうかを確認してください。

SQLCODE -20282、SQLSTATE 42724、理由コード 3

ルーチンを作成する CREATE ステートメントの EXTERNAL 節で指定したロケーションでアセンブリーが見つかり、そのアセンブリーの中に該当するクラス定義が含まれていましたが、ルーチンのメソッド・シグニチャーが、そのルーチンの CREATE ステートメントで指定されたルーチン・シグニチャーと一致しません。

- EXTERNAL 節で指定したアセンブリー名がルーチンの正しいアセンブリー名かどうか、そのアセンブリーが指定のロケーションに存在するかどうかを確認してください。
- EXTERNAL 節で指定したクラス名が正しいクラス名かどうか、そのクラスが指定のアセンブリーの中に存在するかどうかを確認してください。
- パラメーター・スタイルのインプリメンテーションが、ルーチンを作成する CREATE ステートメントで指定したパラメーター・スタイルと一致するかどうかを確認してください。
- パラメーター・インプリメンテーションの順序が、ルーチンを作成する CREATE ステートメントで指定したパラメーター宣言の順序と一致するかどうか、そのパラメーター・スタイルのその他のパラメーター要件を満たしているかどうかを確認してください。
- SQL パラメーターのデータ・タイプが、 CLR .NET でサポートされているデータ・タイプに正しくマップされているかどうかを確認してください。

SQLCODE -4301、SQLSTATE 58004、理由コード 5 または 6

.NET インタープリターを開始しようとしたとき、または .NET インタープリターと通信しようとしたときに、エラーが発生しました。 DB2 データベース・システムが従属の .NET ライブラリーをロードできなかったか [理由コード 5]、 .NET インタープリターの呼び出しが失敗しました [理由コード 6]。

- DB2 インスタンスが .NET のプロシージャーまたは関数を実行するための正しい構成になっているかどうかを確認してください (システム `PATH` に `mscoree.dll` が存在している必要があります)。 `db2clr.dll` が `sqllib/bin` ディレクトリーに存在するかどうか、 `IBM.Data.DB2` がグローバル・アセンブリー・キャッシュにインストールされているかどうかを確認してください。そのいずれかが存在しない場合は、 `.NET Framework` バージョン 1.1 以降がデータベース・サーバーにインストールされているかどうか、そのデータベース・サーバーが DB2 バージョン 8.2 以降のリリースを実行しているかどうかを確認してください。

SQLCODE -4302、SQLSTATE 38501

ルーチンの実行中か、実行準備中か、実行後に、処理できない例外が発生しました。これは、未処理になっていたルーチン・ロジックのプログラミング・エラーの結果か、内部処理エラーの結果であると考えられます。このタイプのエラーの場合、処理できない例外が発生した場所を示す .NET スタック・トレースバックが db2diag ログ・ファイルに書き込まれます。

また、ルーチンに指定された実行モードに対して許可されたアクションの有効範囲を超えるアクションをルーチンが試みた場合にも、このエラーが発生します。この場合、実行制御違反により例外が発生したことを具体的に示す

項目が db2diag ログ・ファイルに書き込まれます。違反が発生した場所を示す例外スタック・トレースバックもこれに含まれます。

ルーチンのアセンブリーが破損していたり、最近変更されたりしたことはないか、判別してください。ルーチンの変更が妥当である場合、この問題は、ルーチンの EXECUTION CONTROL モードの設定が、変更後のロジックに適したものでなくなったために生じている可能性があります。アセンブリーで正しい変更が行われていることが確かな場合、ALTER PROCEDURE または ALTER FUNCTION ステートメントを使用してルーチンの実行モードを必要に応じて変更することができます。詳細については、以下のトピックを参照してください。

- 61 ページの『CLR ルーチンのセキュリティーおよび実行モード』

.NET CLR ルーチンの例

.NET CLR ルーチンを開発する際には、例を参照して、CREATE ステートメントと .NET CLR ルーチン・コードをどのようにすればよいかという感覚をつかむと役に立ちます。

このタスクについて

以下のトピックには、.NET CLR プロシージャーと .NET CLR 関数 (スカラー関数と表関数の両方を含む) の例を記載しています。

.NET CLR プロシージャー

- Visual Basic .NET CLR プロシージャーの例
- C# .NET CLR プロシージャーの例

.NET CLR 関数

- Visual Basic .NET CLR 関数の例
- C# .NET CLR 関数の例

C# .NET CLR プロシージャーの例

プロシージャー (ストアード・プロシージャーともいう) の基礎と .NET 共通言語ランタイム・ルーチンの基本を理解できたら、アプリケーションで CLR プロシージャーをさっそく活用できます。

始める前に

CLR プロシージャーの例を使用した作業を開始する前に、概念について説明している以下のトピックを参照することもできます。

- 53 ページの『第 3 章 .NET 共通言語ランタイム (CLR) ルーチン』
- 65 ページの『DB2 コマンド・ウィンドウから .NET CLR ルーチンを作成する』
- 共通言語ランタイム (CLR) .NET ルーチンのビルド

このタスクについて

このトピックでは、 C# でインプリメントした CLR プロシージャーの例をいくつか紹介します。それぞれの例は、サポートされているパラメーター・スタイル、パ

ラメーター (dbinfo 構造を含む) の受け渡し、結果セットの戻し方などを示しています。 C# の CLR UDF の例については、以下を参照してください。

- 105 ページの『C# .NET CLR 関数の例』

次の例では、SAMPLE データベースに含まれる EMPLOYEE という名前の表を使用しています。

手順

独自の C# CLR プロシージャーを作成するときには、以下の例を参考にしてください。

- 78
- 79
- 80
- 81
- 82
- 83
- 84

例

C# 外部コード・ファイル

以下の例では、C# プロシージャーのさまざまなインプリメンテーションを示しています。それぞれの例は、CREATE PROCEDURE ステートメントと、関連アセンブリーのビルド元プロシージャーの外部 C# コード・インプリメンテーションという 2 つの部分から成っています。

以下の例のプロシージャー・インプリメンテーションに含まれる C# ソース・ファイルは、 gwenProc.cs という名前であり、以下の形式になっています。

```
using System;
using System.IO;
using IBM.Data.DB2;

namespace bizLogic
{
   class empOps
   {          ...
     // C# procedures
              ...
   }
}
```

ファイルの先頭には、このファイルに組み込むものを示します。ファイル内のプロシージャーのいずれかに SQL が含まれる場合は、 IBM.Data.DB2 を含める必要があります。このファイルには、ネーム・スペース宣言を組み込み、プロシージャーを内容とするクラス empOps を組み込みます。ネーム・スペースの使用はオプションです。ネーム・スペースを使用する場合は、CREATE PROCEDURE ステートメントの EXTERNAL 節に指定するアセンブリー・パス名の中にネーム・スペースを入れなければなりません。

ファイルの名前、ネームスペース、特定のプロシージャー・インプリメンテーションを含むクラスの名前をメモしておくことは重要です。各プロシージ

ャーの CREATE PROCEDURE ステートメントの EXTERNAL 節でその情報を指定して、 DB2 がアセンブリーと CLR プロシージャーのクラスを見つけられるようにする必要があるからです。

例 1: C# のパラメーター・スタイル GENERAL のプロシージャー

この例では、以下について説明します。

- パラメーター・スタイル GENERAL のプロシージャーの CREATE PROCEDURE ステートメント
- パラメーター・スタイル GENERAL のプロシージャーの C# コード

このプロシージャーは、従業員 ID と現在のボーナスの額を入力値として取ります。そして、従業員の名前と給与を検索します。現在のボーナスの額がゼロの場合は、従業員の給与に基づいて新しいボーナスを計算し、従業員の氏名と一緒に戻します。従業員が見つからない場合は、空ストリングを戻します。

```
CREATE PROCEDURE setEmpBonusGEN(IN empID CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusGEN
LANGUAGE CLR
PARAMETER STYLE GENERAL
MODIFIES SQL DATA
EXECUTION CONTROL SAFE
FENCED
THREADSAFE
DYNAMIC RESULT SETS 0
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusGEN' ;
```

```
 public static void SetEmpBonusGEN(    String empID,
                                    ref Decimal bonus,
                                    out String empName)
 {
    // Declare local variables
    Decimal salary = 0;

    DB2Command myCommand = DB2Context.GetCommand();
    myCommand.CommandText =
                    "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY "
                  + "FROM EMPLOYEE "
                  + "WHERE EMPNO = '" + empID + '";

    DB2DataReader reader = myCommand.ExecuteReader();

    if (reader.Read())  // If employee record is found
    {
       // Get the employee's full name and salary
       empName = reader.GetString(0) + " " +
                 reader.GetString(1) + ". " +
                 reader.GetString(2);

       salary = reader.GetDecimal(3);

       if (bonus == 0)
       {
          if (salary > 75000)
          {
             bonus = salary * (Decimal)0.025;
          }
          else
          {
             bonus = salary * (Decimal)0.05;
```

```
        }
      }
   }
   else  // Employee not found
   {
      empName = "";  // Set output parameter
   }

   reader.Close();
}
```

例 2: C# のパラメーター・スタイル GENERAL WITH NULLS のプロシージャー

この例では、以下について説明します。

- パラメーター・スタイル GENERAL WITH NULLS のプロシージャーの CREATE PROCEDURE ステートメント
- パラメーター・スタイル GENERAL WITH NULLS のプロシージャーの C# コード

このプロシージャーは、従業員 ID と現在のボーナスの額を入力値として取ります。入力パラメーターが NULL 以外の場合は、従業員の名前と給与を検索します。現在のボーナスの額がゼロの場合は、給与に基づいて新しいボーナスを計算し、従業員の氏名と一緒に戻します。従業員データが見つからない場合は、NULL ストリングと整数を戻します。

```
CREATE PROCEDURE SetEmpbonusGENNULL(IN empID CHAR(6),
                                    INOUT bonus Decimal(9,2),
                                    OUT empName VARCHAR(60))
SPECIFIC SetEmpbonusGENNULL
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
DYNAMIC RESULT SETS 0
MODIFIES SQL DATA
EXECUTION CONTROL SAFE
FENCED
THREADSAFE
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusGENNULL'
;
```

```
public static void SetEmpBonusGENNULL(    String empID,
                                       ref Decimal bonus,
                                       out String empName,
                                           Int16[] NullInds)
{
   Decimal salary = 0;
   if (NullInds[0] == -1) // Check if the input is null
   {
     NullInds[1] = -1;    // Return a NULL bonus value
     empName = "";        // Set output value
     NullInds[2] = -1;    // Return a NULL empName value
   }
   else
   {
      DB2Command myCommand = DB2Context.GetCommand();
       myCommand.CommandText =
                     "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY "
                   + "FROM EMPLOYEE "
                   + "WHERE EMPNO = '" + empID + "'";
      DB2DataReader reader = myCommand.ExecuteReader();

      if (reader.Read())  // If employee record is found
      {
         // Get the employee's full name and salary
         empName = reader.GetString(0) + " "
```

```
            +
                      reader.GetString(1) + ". " +
                      reader.GetString(2);
            salary = reader.GetDecimal(3);

            if (bonus == 0)
            {
              if (salary > 75000)
              {
                 bonus = salary * (Decimal)0.025;
                 NullInds[1] = 0; // Return a non-NULL value
              }
              else
              {
                 bonus = salary * (Decimal)0.05;
                 NullInds[1] = 0; // Return a non-NULL value
              }
            }
          }
          else  // Employee not found
          {
             empName = "*sdq;;         // Set output parameter
             NullInds[2] = -1;     // Return a NULL value
          }

          reader.Close();
       }
    }
```

例 3: C# のパラメーター・スタイル SQL のプロシージャー

この例では、以下について説明します。

- パラメーター・スタイル SQL のプロシージャーの CREATE PROCEDURE ステートメント
- パラメーター・スタイル SQL のプロシージャーの C# コード

このプロシージャーは、従業員 ID と現在のボーナスの額を入力値として取ります。そして、従業員の名前と給与を検索します。現在のボーナスの額がゼロの場合は、給与に基づいて新しいボーナスを計算し、従業員の氏名と一緒に戻します。従業員が見つからない場合は、空ストリングを戻します。

```
CREATE PROCEDURE SetEmpbonusSQL(IN empID CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpbonusSQL
LANGUAGE CLR
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
MODIFIES SQL DATA
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusSQL' ;
```

```
 public static void SetEmpBonusSQL(    String empID,
                                   ref Decimal bonus,
                                   out String empName,
                                       Int16  empIDNullInd,
                                   ref Int16  bonusNullInd,
                                   out Int16  empNameNullInd,
                                   ref string sqlStateate,
                                       string funcName,
                                       string specName,
                                   ref string sqlMessageText)
 {
```

```
    // Declare local host variables
    Decimal salary eq; 0;

    if (empIDNullInd == -1) // Check if the input is null
    {
       bonusNullInd = -1;   // Return a NULL bonus value
       empName = "";
       empNameNullInd = -1; // Return a NULL empName value
    }
    else
    {
       DB2Command myCommand = DB2Context.GetCommand();
       myCommand.CommandText =
                        "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY
                        "
                      + "FROM EMPLOYEE "
                      + "WHERE EMPNO = '" + empID + "'";

       DB2DataReader reader = myCommand.ExecuteReader();

       if (reader.Read())  // If employee record is found
       {
          // Get the employee's full name and salary
          empName = reader.GetString(0) + " "
          +
          reader.GetString(1) + ". " +
          reader.GetString(2);
          empNameNullInd = 0;
          salary = reader.GetDecimal(3);

          if (bonus == 0)
          {
             if (salary > 75000)
             {
                bonus = salary * (Decimal)0.025;
                bonusNullInd = 0;  // Return a non-NULL value
             }
             else
             {
                bonus = salary * (Decimal)0.05;
                bonusNullInd = 0;  // Return a non-NULL value
             }
          }
       }
       else  // Employee not found
       {
          empName = "";         // Set output parameter
          empNameNullInd = -1;  // Return a NULL value
       }

       reader.Close();
    }
 }
```

例 4: 結果セットを戻す C# のパラメーター・スタイル GENERAL のプロシージャー　この例では、以下について説明します。

- 結果セットを戻す外部 C# プロシージャーの CREATE PROCEDURE ステートメント
- 結果セットを戻すパラメーター・スタイル GENERAL のプロシージャーの C# コード

このプロシージャーは、パラメーターとして表の名前を受け入れます。そして、入力パラメーターによって指定されている表の行すべてを含む結果セットを戻します。この処理のために、プロシージャーの戻り時に特定の照会結

果セットの DB2DataReader をオープンにしておきます。具体的には、reader.Close() が実行されなければ、結果セットが戻されるということです。

```
CREATE PROCEDURE ReturnResultSet(IN tableName
                                 VARCHAR(20))
SPECIFIC ReturnResultSet
DYNAMIC RESULT SETS 1
LANGUAGE CLR
PARAMETER STYLE GENERAL
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!ReturnResultSet' ;
```

```
public static void ReturnResultSet(string tableName)
{
   DB2Command myCommand = DB2Context.GetCommand();

   // Set the SQL statement to be executed and execute it.
   myCommand.CommandText = "SELECT * FROM " + tableName;
   DB2DataReader reader = myCommand.ExecuteReader();

   // The DB2DataReader contains the result of the query.
   // This result set can be returned with the procedure,
   // by simply NOT closing the DB2DataReader.
   // Specifically, do NOT execute reader.Close();
}
```

例 5: dbinfo 構造にアクセスする C# のパラメーター・スタイル SQL のプロシージャー この例では、以下について説明します。

- dbinfo 構造にアクセスするプロシージャーの CREATE PROCEDURE ステートメント
- dbinfo 構造にアクセスするパラメーター・スタイル SQL のプロシージャーの C# コード

dbinfo 構造にアクセスするには、 CREATE PROCEDURE ステートメントに DBINFO 節を指定する必要があります。 CREATE PROCEDURE ステートメントの dbinfo 構造にパラメーターは必要ありませんが、外部ルーチン・コードでそのためのパラメーターを作成する必要があります。このプロシージャーは、 dbinfo 構造の dbname フィールドからの現行データベース名の値だけを戻します。

```
CREATE PROCEDURE ReturnDbName(OUT dbName VARCHAR(20))
SPECIFIC ReturnDbName
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE SQL
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
DBINFO
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!ReturnDbName'
;
```

```
public static void ReturnDbName(out string dbName,
                                 out Int16  dbNameNullInd,
                                 ref string sqlState,
                                     string funcName,
                                     string specName,
                                 ref string sqlMessageText,
                                     sqludf_dbinfo dbinfo)
{
```

```
    // Retrieve the current database name from the
    // dbinfo structure and return it.
    // ** Note! ** dbinfo field names are case sensitive
    dbName = dbinfo.dbname;
    dbNameNullInd = 0;  // Return a non-null value;

    // If you want to return a user-defined error in
    // the SQLCA you can specify a 5 digit user-defined
    // sqlStateate and an error message string text.
    // For example:
    //
    //   sqlStateate = "ABCDE";
    //   sqlMessageText = "A user-defined error has occurred"
    //
    //  DB2 returns the above values to the client in the
    //  SQLCA structure.  The values are used to generate a
    //  standard DB2 sqlStateate error.
  }
```

例 6: PROGRAM TYPE MAIN スタイルの C# プロシージャー

この例では、以下について説明します。

- メインプログラム・スタイルを使用したプロシージャーの CREATE PROCEDURE ステートメント
- メインプログラム・スタイルを使用した C# のパラメーター・スタイル GENERAL WITH NULLS のコード

メインプログラム・スタイルでルーチンをインプリメントするには、CREATE PROCEDURE ステートメントの PROGRAM TYPE 節に MAIN という値を指定する必要があります。 CREATE PROCEDURE ステートメントにもパラメーターを指定しますが、コードのインプリメンテーションでは、ルーチンの argc 整数パラメーターと argv パラメーター配列にパラメーターを渡します。

```
CREATE PROCEDURE MainStyle( IN empID CHAR(6),
                            INOUT bonus Decimal(9,2),
                            OUT empName VARCHAR(60))
SPECIFIC MainStyle
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
MODIFIES SQL DATA
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
PROGRAM TYPE MAIN
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!main' ;
```

```
 public static void main(Int32 argc, Object[]
 argv)
 {
   String empID = (String)argv[0];   // argv[0] has nullInd:argv[3]
   Decimal bonus = (Decimal)argv[1]; // argv[1] has nullInd:argv[4]
                                     // argv[2] has nullInd:argv[5]
   Decimal salary = 0;
   Int16[] NullInds = (Int16[])argv[3];

   if ((NullInds[0]) == (Int16)(-1)) // Check if empID is null
   {
      NullInds[1] = (Int16)(-1);  // Return a NULL bonus value
      argv[1] = (String)"";       // Set output parameter empName
      NullInds[2] = (Int16)(-1);  // Return a NULL empName value
      Return;
   }
   else
```

```
{
   DB2Command myCommand = DB2Context.GetCommand();
   myCommand.CommandText =
                    "SELECT FIRSTNME, MIDINIT, LASTNAME, salary "
                  + "FROM EMPLOYEE "
                  + "WHERE EMPNO = '" + empID + "'";

   DB2DataReader reader = myCommand.ExecuteReader();

   if (reader.Read())  // If employee record is found
   {
     // Get the employee's full name and salary
     argv[2] = (String) (reader.GetString(0) + " " +
                         reader.GetString(1) + ".
                         " +
                         reader.GetString(2));
     NullInds[2] = (Int16)0;
     salary = reader.GetDecimal(3);

     if (bonus == 0)
     {
        if (salary > 75000)
        {
           argv[1] = (Decimal)(salary * (Decimal)0.025);
           NullInds[1] = (Int16)(0);  // Return a non-NULL value
        }
        else
        {
           argv[1] = (Decimal)(salary * (Decimal)0.05);
           NullInds[1] = (Int16)(0);  // Return a non-NULL value
        }
     }
   }
   else  // Employee not found
   {
      argv[2] = (String)("");         // Set output parameter
      NullInds[2] = (Int16)(-1);      // Return a NULL value
   }

   reader.Close();
  }
}
```

Visual Basic .NET CLR 関数の例

ユーザー定義関数 (UDF) の基礎と CLR ルーチンの基本を理解できたら、アプリケーションやデータベース環境の中で CLR UDF をさっそく活用できます。このトピックでは、手始めとして CLR UDF の例をいくつか紹介します。

始める前に

CLR UDF の例を使用した作業を開始する前に、概念について説明している以下のトピックを参照することもできます。

- 53 ページの『第 3 章 .NET 共通言語ランタイム (CLR) ルーチン』
- 65 ページの『DB2 コマンド・ウィンドウから .NET CLR ルーチンを作成する』
- 14 ページの『外部スカラー関数』
- 共通言語ランタイム (CLR) .NET ルーチンのビルド

このタスクについて

Visual Basic の CLR プロシージャーの例については、以下を参照してください。

- 90 ページの『Visual Basic .NET CLR プロシージャーの例』

次の例では、SAMPLE データベースに含まれる EMPLOYEE という名前の表を使用しています。

手順

独自の Visual Basic CLR UDF を作成するときには、以下の例を参考にしてください。

- 86
- 86
- 89

例

Visual Basic 外部コード・ファイル

次の例では、Visual Basic UDF のさまざまなインプリメンテーションを示しています。各 UDF ごとに、関連アセンブリーのビルド元になる Visual Basic ソース・コードとともに、 CREATE FUNCTION ステートメントを用意します。以下の例で使用している関数宣言に含まれる Visual Basic ソース・ファイルは、 gwenVbUDF.cs という名前であり、以下の形式になっています。

```
using System;
using System.IO;
using IBM.Data.DB2;

Namespace bizLogic

   ...
   ' Class definitions that contain UDF declarations
   ' and any supporting class definitions
   ...

End Namespace
```

Visual Basic ファイル内のクラスに関数宣言を組み込む必要があります。ネーム・スペースの使用はオプションです。ネーム・スペースを使用する場合は、 CREATE PROCEDURE ステートメントの EXTERNAL 節に指定するアセンブリー・パス名の中にネーム・スペースを入れなければなりません。関数に SQL が含まれる場合は、IBM.Data.DB2. を含める必要があります。

例 1: Visual Basic のパラメーター・スタイル SQL の表関数

この例では、以下について説明します。

- パラメーター・スタイル SQL の表関数の CREATE FUNCTION ステートメント
- パラメーター・スタイル SQL の表関数の Visual Basic コード

この表関数は、データ配列から作成された従業員データの行を含んだ表を戻します。この例には、2 つの関連クラスがあります。 1 つは従業員を表すクラス person であり、もう 1 つはクラス person を使用するルーチン表

UDF を含んだクラス empOps です。従業員の給与情報は、入力パラメーターの値に基づいて更新されます。この例のデータ配列は、表関数を最初に呼び出したときに表関数そのものの中に作成されます。そのような配列は、ファイル・システム上のテキスト・ファイルからデータを読み取ることによっても作成できます。表関数のその後の呼び出しで配列データにアクセスするために、データの値がスクラッチパッドに書き込まれます。

表関数を呼び出すたびに、1 つのレコードが配列から読み取られ、 1 つの行が関数によって戻される表の中に生成されます。行を表の中に生成する処理は、表関数の出力パラメーターを対象の行値に設定するという形で実行されます。表関数の最終呼び出しが行われた後、生成された行の表が戻されます。

```
CREATE FUNCTION TableUDF(double)
RETURNS TABLE (name varchar(20),
               job varchar(20),
               salary double)
EXTERNAL NAME 'gwenVbUDF.dll:bizLogic.empOps!TableUDF'
LANGUAGE CLR
PARAMETER STYLE SQL
NOT DETERMINISTIC
FENCED
SCRATCHPAD 10
FINAL CALL
DISALLOW PARALLEL
NO DBINFO
EXECUTION CONTROL SAFE
```

```
Class Person
' The class Person is a supporting class for
' the table function UDF, tableUDF, below.

  Private name As String
  Private position As String
  Private salary As Int32

  Public Sub New(ByVal newName As String, _
                 ByVal newPosition As String, _
                 ByVal newSalary As Int32)

    name = newName
    position = newPosition
    salary = newSalary
  End Sub

  Public Property GetName() As String
    Get
      Return name
    End Get

    Set (ByVal value As String)
      name = value
    End Set
  End Property

  Public Property GetPosition() As String
    Get
      Return position
    End Get

    Set (ByVal value As String)
      position = value
    End Set
  End Property
```

```
  Public Property GetSalary() As Int32
    Get
      Return salary
    End Get

    Set (ByVal value As Int32)
      salary = value
    End Set
  End Property

End Class

 Class empOps

   Public Shared Sub TableUDF(byVal factor as Double, _
                              byRef name As String, _
                              byRef position As String, _
                              byRef salary As Double, _
                              byVal factorNullInd As Int16, _
                              byRef nameNullInd As Int16, _
                              byRef positionNullInd As Int16, _
                              byRef salaryNullInd As Int16, _
                              byRef sqlState As String, _
                              byVal funcName As String, _
                              byVal specName As String, _
                              byRef sqlMessageText As String, _
                              byVal scratchPad As Byte(), _
                              byVal callType As Int32)

     Dim intRow As Int16

     intRow = 0

     ' Create an array of Person type information
     Dim staff(2) As Person
     staff(0) = New Person("Gwen", "Developer", 10000)
     staff(1) = New Person("Andrew", "Developer", 20000)
     staff(2) = New Person("Liu", "Team Leader", 30000)

     ' Initialize output parameter values and NULL indicators
     salary = 0
     name = position = ""
     nameNullInd = positionNullInd = salaryNullInd = -1

     Select callType
        Case -2   ' Case SQLUDF_TF_FIRST:
        Case -1   ' Case SQLUDF_TF_OPEN:
          intRow = 1
          scratchPad(0) = intRow  ' Write to scratchpad
        Case 0    ' Case SQLUDF_TF_FETCH:
          intRow = scratchPad(0)
          If intRow > staff.Length
             sqlState = "02000"  ' Return an error SQLSTATE
          Else
             ' Generate a row in the output table
             ' based on the staff array data.
             name = staff(intRow).GetName()
             position = staff(intRow).GetPosition()
             salary = (staff(intRow).GetSalary()) * factor
             nameNullInd = 0
             positionNullInd = 0
             salaryNullInd = 0
          End If
          intRow = intRow + 1
          scratchPad(0) = intRow  ' Write scratchpad

        Case 1    ' Case SQLUDF_TF_CLOSE:
```

```
            Case 2    ' Case SQLUDF_TF_FINAL:
          End Select

        End Sub

      End Class
```

例 2: Visual Basic のパラメーター・スタイル SQL のスカラー関数

この例では、以下について説明します。

- パラメーター・スタイル SQL のスカラー関数の CREATE FUNCTION ステートメント
- パラメーター・スタイル SQL のスカラー関数の Visual Basic コード

このスカラー関数は、操作対象の入力値ごとに 1 つのカウント値を戻します。入力値セットの n 番目の桁にある入力値に対する出力スカラー値は n になります。スカラー関数の各呼び出しでは、行または値の入力セット内のそれぞれの行または値に 1 つの呼び出しが関連付けられており、呼び出しのたびにカウントが 1 つずつ増え、カウントの現行値が戻されます。そのカウントはスクラッチパッドのメモリー・バッファー内に保管されるので、スカラー関数の呼び出しと呼び出しの間でカウントの値が保たれるようになっています。

例えば、表を次のように定義している場合は、このスカラー関数を簡単に呼び出すことができます。

```
CREATE TABLE T (i1 INTEGER);
INSERT INTO T VALUES 12, 45, 16, 99;
```

このスカラー関数の呼び出しには、以下のような簡単な照会を使用できます。

```
SELECT my_count(i1) as count, i1 FROM T;
```

この照会の出力は次のようになります。

```
COUNT           I1
-----------     ----------
1               12
2               45
3               16
4               99
```

このスカラー UDF は非常に簡単です。スカラー関数を使用するときには、行のカウントだけを戻す代わりに、データの形式を既存の列に合わせることもできます。例えば、住所列の各値にストリングを付加することや、一連の入力ストリングから複雑なストリングを組み立てることや、中間結果の保管先のデータ・セットに対して複雑な数値評価を行うことなども可能です。

```
CREATE FUNCTION mycount(INTEGER)
RETURNS INTEGER
LANGUAGE CLR
PARAMETER STYLE SQL
NO SQL
SCRATCHPAD 10
FINAL CALL
FENCED
EXECUTION CONTROL SAFE
NOT DETERMINISTIC
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!CountUp';
```

```
Class empOps
  Public Shared Sub CountUp(byVal input As Int32, _
                            byRef outCounter As Int32, _
                            byVal nullIndInput As Int16, _
                            byRef nullIndOutCounter As Int16, _
                            byRef sqlState As String, _
                            byVal qualName As String, _
                            byVal specName As String, _
                            byRef sqlMessageText As String, _
                            byVal scratchPad As Byte(), _
                            byVal callType As Int32)

     Dim counter As Int32
     counter = 1

     Select callType
        case -1           ' case SQLUDF_TF_OPEN_CALL
           scratchPad(0) = counter
           outCounter = counter
           nullIndOutCounter = 0
        case 0             'case SQLUDF_TF_FETCH_CALL:
           counter = scratchPad(0)
           counter = counter + 1
           outCounter = counter
           nullIndOutCounter = 0
           scratchPad(0) = counter
        case 1            'case SQLUDF_CLOSE_CALL:
           counter = scratchPad(0)
           outCounter = counter
           nullIndOutCounter = 0
        case Else          ' Should never enter here
           ' These cases won't occur for the following reasons:
           ' Case -2  (SQLUDF_TF_FIRST)     ->No FINAL CALL in CREATE stmt
           ' Case 2   (SQLUDF_TF_FINAL)     ->No FINAL CALL in CREATE stmt
           ' Case 255 (SQLUDF_TF_FINAL_CRA) ->No SQL used in the function
           '
           ' * Note!*
           ' ---------
           ' The Else is required so that at compile time
           ' out parameter outCounter is always set *
           outCounter = 0
           nullIndOutCounter = -1
     End Select
  End Sub

End Class
```

Visual Basic .NET CLR プロシージャーの例

プロシージャー (ストアード・プロシージャーともいう) の基礎と .NET 共通言語ランタイム・ルーチンの基本を理解できたら、アプリケーションで CLR プロシージャーをさっそく活用できます。このトピックでは、 Visual Basic でインプリメントした CLR プロシージャーの例をいくつか紹介します。それぞれの例は、サポートされているパラメーター・スタイル、パラメーター (dbinfo 構造を含む) の受け渡し、結果セットの戻し方などを示しています。

始める前に

CLR プロシージャーの例を使用した作業を開始する前に、概念について説明している以下のトピックを参照することもできます。

- 53 ページの『第 3 章 .NET 共通言語ランタイム (CLR) ルーチン』
- 65 ページの『DB2 コマンド・ウィンドウから .NET CLR ルーチンを作成する』

- 5 ページの『ルーチン使用の利点』
- 共通言語ランタイム (CLR) .NET ルーチンのビルド

このタスクについて

Visual Basic の CLR UDF の例については、以下を参照してください。

- 85 ページの『Visual Basic .NET CLR 関数の例』

次の例では、SAMPLE データベースに含まれる EMPLOYEE という名前の表を使用しています。

手順

独自の Visual Basic CLR プロシージャーを作成するときには、以下の例を参考にしてください。

- 91
- 92
- 93
- 94
- 95
- 96
- 97

例

Visual Basic 外部コード・ファイル

以下の例では、Visual Basic プロシージャーのさまざまなインプリメンテーションを示しています。それぞれの例は、CREATE PROCEDURE ステートメントと、関連アセンブリーのビルド元プロシージャーの外部 Visual Basic コード・インプリメンテーションという 2 つの部分から成っています。

以下の例のプロシージャー・インプリメンテーションに含まれる Visual Basic ソース・ファイルは、 gwenVbProc.vb という名前であり、以下の形式になっています。

```
using System;
using System.IO;
using IBM.Data.DB2;

Namespace bizLogic

   Class empOps
              ...
     ' Visual Basic procedures
              ...
   End Class
End Namespace
```

ファイルの先頭には、このファイルに組み込むものを示します。ファイル内のプロシージャーのいずれかに SQL が含まれる場合は、 IBM.Data.DB2 を含める必要があります。このファイルには、ネーム・スペース宣言を組み込み、プロシージャーを内容とするクラス empOps を組み込みます。ネーム・スペースの使用はオプションです。ネーム・スペースを使用する場合は、

CREATE PROCEDURE ステートメントの EXTERNAL 節に指定するアセンブリー・パス名の中にネーム・スペースを入れなければなりません。

ファイルの名前、ネームスペース、特定のプロシージャー・インプリメンテーションを含むクラスの名前をメモしておくことは重要です。各プロシージャーの CREATE PROCEDURE ステートメントの EXTERNAL 節でその情報を指定して、 DB2 がアセンブリーと CLR プロシージャーのクラスを見つけられるようにする必要があるからです。

例 1: Visual Basic のパラメーター・スタイル GENERAL のプロシージャー

この例では、以下について説明します。

- パラメーター・スタイル GENERAL のプロシージャーの CREATE PROCEDURE ステートメント
- パラメーター・スタイル GENERAL のプロシージャーの Visual Basic コード

このプロシージャーは、従業員 ID と現在のボーナスの額を入力値として取ります。そして、従業員の名前と給与を検索します。現在のボーナスの額がゼロの場合は、従業員の給与に基づいて新しいボーナスを計算し、従業員の氏名と一緒に戻します。従業員が見つからない場合は、空ストリングを戻します。

```
CREATE PROCEDURE SetEmpBonusGEN(IN empId CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC setEmpBonusGEN
LANGUAGE CLR
PARAMETER STYLE GENERAL
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusGEN'
```

```
  Public Shared Sub SetEmpBonusGEN(ByVal empId As String, _
                                   ByRef bonus As Decimal, _
                                   ByRef empName As String)

     Dim salary As Decimal
     Dim myCommand As DB2Command
     Dim myReader As DB2DataReader

     salary = 0

     myCommand = DB2Context.GetCommand()
     myCommand.CommandText = _
               "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
             + "FROM EMPLOYEE " _
             + "WHERE EMPNO = '" + empId + "'"
     myReader = myCommand.ExecuteReader()

     If myReader.Read()  ' If employee record is found
        ' Get the employee's full name and salary
        empName = myReader.GetString(0) + " " _
                + myReader.GetString(1) + ". " _
                + myReader.GetString(2)

        salary = myReader.GetDecimal(3)

        If bonus = 0
           If salary > 75000
              bonus = salary * 0.025
```

```
            Else
               bonus = salary * 0.05
            End If
         End If
      Else  ' Employee not found
         empName = ""  ' Set output parameter
      End If

      myReader.Close()

   End Sub
```

例 2: Visual Basic のパラメーター・スタイル GENERAL WITH NULLS のプロシージャー

この例では、以下について説明します。

- パラメーター・スタイル GENERAL WITH NULLS のプロシージャーの CREATE PROCEDURE ステートメント
- パラメーター・スタイル GENERAL WITH NULLS のプロシージャーの Visual Basic コード

このプロシージャーは、従業員 ID と現在のボーナスの額を入力値として取ります。入力パラメーターが NULL 以外の場合は、従業員の名前と給与を検索します。現在のボーナスの額がゼロの場合は、給与に基づいて新しいボーナスを計算し、従業員の氏名と一緒に戻します。従業員データが見つからない場合は、NULL ストリングと整数を戻します。

```
CREATE PROCEDURE SetEmpBonusGENNULL(IN empId CHAR(6),
                                    INOUT bonus Decimal(9,2),
                                    OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusGENNULL
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusGENNULL'
```

```
  Public Shared Sub SetEmpBonusGENNULL(ByVal empId As String, _
                                       ByRef bonus As Decimal, _
                                       ByRef empName As String, _
                                       byVal nullInds As Int16())

     Dim salary As Decimal
     Dim myCommand As DB2Command
     Dim myReader As DB2DataReader

     salary = 0

     If nullInds(0) = -1   ' Check if the input is null
        nullInds(1) = -1   ' Return a NULL bonus value
        empName = ""       ' Set output parameter
        nullInds(2) = -1   ' Return a NULL empName value
        Return
     Else
        myCommand = DB2Context.GetCommand()
        myCommand.CommandText = _
                   "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
                 + "FROM EMPLOYEE " _
                 + "WHERE EMPNO = '" + empId + "'"

        myReader = myCommand.ExecuteReader()

        If myReader.Read()  ' If employee record is found
```

```
        ' Get the employee's full name and salary
        empName = myReader.GetString(0) + " "   _
                + myReader.GetString(1) + ". " _
                + myReader.GetString(2)

        salary = myReader.GetDecimal(3)

        If bonus = 0
           If salary > 75000
              bonus = Salary * 0.025
              nullInds(1) = 0 'Return a non-NULL value
           Else
              bonus = salary * 0.05
              nullInds(1) = 0 ' Return a non-NULL value
           End If
        Else  'Employee not found
           empName = ""       ' Set output parameter
           nullInds(2) = -1   ' Return a NULL value
        End If
     End If

     myReader.Close()

  End If

End Sub
```

例 3: Visual Basic のパラメーター・スタイル SQL のプロシージャー

この例では、以下について説明します。

- パラメーター・スタイル SQL のプロシージャーの CREATE PROCEDURE ステートメント
- パラメーター・スタイル SQL のプロシージャーの Visual Basic コード

このプロシージャーは、従業員 ID と現在のボーナスの額を入力値として取ります。そして、従業員の名前と給与を検索します。現在のボーナスの額がゼロの場合は、給与に基づいて新しいボーナスを計算し、従業員の氏名と一緒に戻します。従業員が見つからない場合は、空ストリングを戻します。

```
CREATE PROCEDURE SetEmpBonusSQL(IN empId CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusSQL
LANGUAGE CLR
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusSQL'
```

```
  Public Shared Sub SetEmpBonusSQL(byVal empId As String, _
                                   byRef bonus As Decimal, _
                                   byRef empName As String, _
                                   byVal empIdNullInd As Int16, _
                                   byRef bonusNullInd As Int16, _
                                   byRef empNameNullInd As Int16, _
                                   byRef sqlState As String, _
                                   byVal funcName As String, _
                                   byVal specName As String, _
                                   byRef sqlMessageText As String)

    ' Declare local host variables
    Dim salary As Decimal
    Dim myCommand As DB2Command
    Dim myReader As DB2DataReader
```

```
salary = 0

If empIdNullInd = -1   ' Check if the input is null
   bonusNullInd = -1   ' Return a NULL Bonus value
   empName = ""
   empNameNullInd = -1 ' Return a NULL empName value
Else
  myCommand = DB2Context.GetCommand()
  myCommand.CommandText = _
            "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
          + "FROM EMPLOYEE " _
          + " WHERE EMPNO = '" + empId + "'"

   myReader = myCommand.ExecuteReader()

   If myReader.Read()  ' If employee record is found
      '  Get the employee's full name and salary
      empName = myReader.GetString(0) + " "
              + myReader.GetString(1) _
              + ". " +  myReader.GetString(2)
      empNameNullInd = 0
      salary = myReader.GetDecimal(3)

      If bonus = 0
         If salary > 75000
            bonus = salary * 0.025
            bonusNullInd = 0  ' Return a non-NULL value
         Else
            bonus = salary * 0.05
            bonusNullInd = 0  ' Return a non-NULL value
         End If
      End If
   Else  ' Employee not found
      empName = ""            ' Set output parameter
      empNameNullInd = -1     ' Return a NULL value
   End If

   myReader.Close()
End If

End Sub
```

例 4: 結果セットを戻す Visual Basic のパラメーター・スタイル GENERAL のプロシージャー

この例では、以下について説明します。

- 結果セットを戻す外部 Visual Basic プロシージャーの CREATE PROCEDURE ステートメント
- 結果セットを戻すパラメーター・スタイル GENERAL のプロシージャーの Visual Basic コード

このプロシージャーは、パラメーターとして表の名前を受け入れます。そして、入力パラメーターによって指定されている表の行すべてを含む結果セットを戻します。この処理のために、プロシージャーの戻り時に特定の照会結果セットの DB2DataReader をオープンにしておきます。具体的には、reader.Close() が実行されなければ、結果セットが戻されるということです。

```
CREATE PROCEDURE ReturnResultSet(IN tableName VARCHAR(20))
SPECIFIC ReturnResultSet
DYNAMIC RESULT SETS 1
LANGUAGE CLR
```

```
  PARAMETER STYLE GENERAL
  FENCED
  PROGRAM TYPE SUB
  EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!ReturnResultSet'

Public Shared Sub ReturnResultSet(byVal tableName As String)

      Dim myCommand As DB2Command
      Dim myReader As DB2DataReader

      myCommand = DB2Context.GetCommand()

      ' Set the SQL statement to be executed and execute it.
      myCommand.CommandText = "SELECT * FROM " + tableName
      myReader = myCommand.ExecuteReader()

      ' The DB2DataReader contains the result of the query.
      ' This result set can be returned with the procedure,
      ' by simply NOT closing the DB2DataReader.
      ' Specifically, do NOT execute reader.Close()

    End Sub
```

例 5: dbinfo 構造にアクセスする Visual Basic のパラメーター・スタイル SQL のプロシージャー

この例では、以下について説明します。

- dbinfo 構造にアクセスするプロシージャーの CREATE PROCEDURE ステートメント
- dbinfo 構造にアクセスするパラメーター・スタイル SQL のプロシージャーの Visual Basic コード

dbinfo 構造にアクセスするには、 CREATE PROCEDURE ステートメントに DBINFO 節を指定する必要があります。 CREATE PROCEDURE ステートメントの dbinfo 構造にパラメーターは必要ありませんが、外部ルーチン・コードでそのためのパラメーターを作成する必要があります。このプロシージャーは、 dbinfo 構造の dbname フィールドからの現行データベース名の値だけを戻します。

```
  CREATE PROCEDURE ReturnDbName(OUT dbName VARCHAR(20))
  SPECIFIC ReturnDbName
  LANGUAGE CLR
  PARAMETER STYLE SQL
  DBINFO
  FENCED
  PROGRAM TYPE SUB
  EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!ReturnDbName'

    Public Shared Sub ReturnDbName(byRef dbName As String, _
                                   byRef dbNameNullInd As Int16, _
                                   byRef sqlState As String, _
                                   byVal funcName As String, _
                                   byVal specName As String, _
                                   byRef sqlMessageText As String, _
                                   byVal dbinfo As sqludf_dbinfo)

      ' Retrieve the current database name from the
      ' dbinfo structure and return it.
      dbName = dbinfo.dbname
      dbNameNullInd = 0  ' Return a non-null value

      ' If you want to return a user-defined error in
      ' the SQLCA you can specify a 5 digit user-defined
      ' SQLSTATE and an error message string text.
```

```
    ' For example:
    '
    ' sqlState = "ABCDE"
    ' msg_token = "A user-defined error has occurred"
    '
    ' These will be returned by DB2 in the SQLCA.  It
    ' will appear in the format of a regular DB2 sqlState
    ' error.
  End Sub
```

例 6: PROGRAM TYPE MAIN スタイルの Visual Basic プロシージャー

この例では、以下について説明します。

- メインプログラム・スタイルを使用したプロシージャーの CREATE PROCEDURE ステートメント
- メインプログラム・スタイルを使用した Visual Basic のパラメーター・スタイル GENERAL WITH NULLS のコード

メインプログラム・スタイルでルーチンをインプリメントするには、CREATE PROCEDURE ステートメントの PROGRAM TYPE 節に MAIN という値を指定する必要があります。 CREATE PROCEDURE ステートメントにもパラメーターを指定しますが、コードのインプリメンテーションでは、ルーチンの argc 整数パラメーターと argv パラメーター配列にパラメーターを渡します。

```
CREATE PROCEDURE MainStyle(IN empId CHAR(6),
                           INOUT bonus Decimal(9,2),
                           OUT empName VARCHAR(60))
SPECIFIC mainStyle
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
FENCED
PROGRAM TYPE MAIN
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!Main'
```

```
  Public Shared Sub Main( byVal argc As Int32, _
                          byVal argv As Object())

     Dim myCommand As DB2Command
     Dim myReader As DB2DataReader
     Dim empId As String
     Dim bonus As Decimal
     Dim salary As Decimal
     Dim nullInds As Int16()

     empId = argv(0)  ' argv[0] (IN)    nullInd = argv[3]
     bonus = argv(1)  ' argv[1] (INOUT) nullInd = argv[4]
                      ' argv[2] (OUT)   nullInd = argv[5]
     salary = 0
     nullInds = argv(3)

     If nullInds(0) = -1     ' Check if the empId input is null
        nullInds(1) = -1     ' Return a NULL Bonus value
        argv(1) = ""         ' Set output parameter empName
        nullInds(2) = -1     ' Return a NULL empName value
        Return
     Else
        ' If the employee exists and the current bonus is 0,
        ' calculate a new employee bonus based on the employee's
        ' salary.  Return the employee name and the new bonus
        myCommand = DB2Context.GetCommand()
        myCommand.CommandText = _
                   "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
```

```
                + " FROM EMPLOYEE " _
                + " WHERE EMPNO = '" + empId + "'"

        myReader = myCommand.ExecuteReader()


        If myReader.Read()  ' If employee record is found
           ' Get the employee's full name and salary
           argv(2) = myReader.GetString(0) + " "  _
                   + myReader.GetString(1) + ". " _
                   + myReader.GetString(2)
           nullInds(2) = 0
           salary = myReader.GetDecimal(3)

           If bonus = 0
              If salary > 75000
                 argv(1) = salary * 0.025
                 nullInds(1) = 0  ' Return a non-NULL value
              Else
                 argv(1) = Salary * 0.05
                 nullInds(1) = 0  ' Return a non-NULL value
              End If
           End If
        Else  ' Employee not found
           argv(2) = ""      ' Set output parameter
           nullInds(2) = -1  ' Return a NULL value
        End If

        myReader.Close()
     End If

  End Sub
```

例: C# .NET CLR プロシージャーでの XML および XQuery サポート

プロシージャーの基本、.NET 共通言語ランタイム・ルーチンの本質部分、XQuery および XML を理解したなら、XML フィーチャーを持つ CLR プロシージャーの作成および使用を始めることができます。

次の例は、XML データの更新および照会方法に加えて、タイプ XML のパラメーターを使用する C# .NET CLR プロシージャーを示します。

前提条件

CLR プロシージャーの例を使用した作業を開始する前に、概念について説明している以下のトピックを参照することもできます。

- .NET 共通言語ランタイム (CLR) ルーチン
- DB2 コマンド・ウィンドウから .NET CLR ルーチンを作成する
- ルーチン使用の利点

次の例では、以下のように定義された xmlDataTable という名前の表を使用します。

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
```

```
                                <type>car</type>
                                <make>Pontiac</make>
                                <model>Sunfire</model>
                                </doc>' PRESERVE WHITESPACE)),
(2, XMLPARSE(DOCUMENT '<doc>
                                <type>car</type>
                                <make>Mazda</make>
                                <model>Miata</model>
                                </doc>' PRESERVE WHITESPACE)),
(3, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Mary</name>
                                <town>Vancouver</town>
                                <street>Waterside</street>
                                </doc>' PRESERVE WHITESPACE)),
(4, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Mark</name>
                                <town>Edmonton</town>
                                <street>Oak</street>
                                </doc>' PRESERVE WHITESPACE)),
(5, XMLPARSE(DOCUMENT '<doc>
                                <type>animal</type>
                                <name>dog</name>
                                </doc>' PRESERVE WHITESPACE)),
(6, NULL),
(7, XMLPARSE(DOCUMENT '<doc>
                                <type>car</type>
                                <make>Ford</make>
                                <model>Taurus</model>
                                </doc>' PRESERVE WHITESPACE)),
(8, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Kim</name>
                                <town>Toronto</town>
                                <street>Elm</street>
                                </doc>' PRESERVE WHITESPACE)),
(9, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Bob</name>
                                <town>Toronto</town>
                                <street>Oak</street>
                                </doc>' PRESERVE WHITESPACE)),
(10, XMLPARSE(DOCUMENT '<doc>
                                 <type>animal</type>
                                 <name>bird</name>
                                 </doc>' PRESERVE WHITESPACE))@
```

手順 独自の C# CLR プロシージャーを作成するときには、以下の例を参考にしてください。

- 『C# 外部コード・ファイル』
- 100 ページの『例 1: XML フィーチャーを持つ C# パラメーター・スタイル GENERAL プロシージャー』

C# 外部コード・ファイル

例は、CREATE PROCEDURE ステートメントと、関連アセンブリーのビルド元プロシージャーの外部 C# コード・インプリメンテーションという 2 つの部分から成っています。

以下の例のプロシージャー・インプリメンテーションに含まれる C# ソース・ファイルは、 `gwenProc.cs` という名前であり、以下の形式になっています。

```
using System;
using System.IO;
using System.Data;
using IBM.Data.DB2;
using IBM.Data.DB2Types;

namespace bizLogic
{
   class empOps
   {          ...
     // C# procedures
              ...
   }
}
```

ファイルの先頭には、このファイルに組み込むものを示します。ファイル内のプロシージャーのいずれかに SQL が含まれる場合は、 IBM.Data.DB2 を含める必要があります。ファイル内のプロシージャーのいずれかにタイプ XML のパラメーターまたは変数が含まれる場合は、IBM.Data.DB2Types を含める必要があります。このファイルには、ネーム・スペース宣言を組み込み、プロシージャーを内容とするクラス empOps を組み込みます。ネーム・スペースの使用はオプションです。ネーム・スペースを使用する場合は、 CREATE PROCEDURE ステートメントの EXTERNAL 節に指定するアセンブリー・パス名の中にネーム・スペースを入れなければなりません。

ファイルの名前、ネームスペース、特定のプロシージャー・インプリメンテーションを含むクラスの名前をメモしておくことは重要です。各プロシージャーの CREATE PROCEDURE ステートメントの EXTERNAL 節でその情報を指定して、DB2 データベース・システムがアセンブリーと CLR プロシージャーのクラスを見つけられるようにする必要があるからです。

例 1: XML フィーチャーを持つ C# パラメーター・スタイル GENERAL プロシージャー

この例では、以下について説明します。

- パラメーター・スタイル GENERAL のプロシージャーの CREATE PROCEDURE ステートメント
- XML パラメーターを使用するパラメーター・スタイル GENERAL プロシージャーの C# コード

このプロシージャーは、整数 inNum と inXML という 2 つのパラメーターを取ります。これらの値は表 xmlDataTable に挿入されます。次に、XML 値が XQuery を使用して検索されます。もう 1 つの XML 値が SQL を使用して検索されます。検索された XML 値は 2 つの出力パラメーター、outXML1 と outXML2 に割り当てられます。結果セットは戻されません。

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                               IN inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K)
                             )
LANGUAGE CLR
PARAMETER STYLE GENERAL
DYNAMIC RESULT SETS 0
FENCED
THREADSAFE
DETERMINISTIC
```

```
NO DBINFO
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!xmlProc1' ;

//*************************************************************************
//  Stored Procedure: xmlProc1
//
//  Purpose:   insert XML data into XML column
//
//  Parameters:
//
//   IN:    inNum -- the sequence of XML data to be insert in xmldata table
//          inXML -- XML data to be inserted
//   OUT:   outXML1 -- XML data returned - value retrieved using XQuery
//          outXML2 -- XML data returned - value retrieved using SQL
//*************************************************************************

    public static void xmlProc1 (     int      inNum, DB2Xml   inXML,
                                 out DB2Xml   outXML1, out DB2Xml   outXML2 )
    {
       // Create new command object from connection context
       DB2Parameter parm;
       DB2Command cmd;
       DB2DataReader reader = null;
       outXML1 = DB2Xml.Null;
       outXML2 = DB2Xml.Null;

       // Insert input XML parameter value into a table
       cmd = DB2Context.GetCommand();
       cmd.CommandText = "INSERT INTO "
                       + "xmlDataTable( num , xdata ) "
                       + "VALUES( ?, ? )";

       parm = cmd.Parameters.Add("@num", DB2Type.Integer );
       parm.Direction = ParameterDirection.Input;
       cmd.Parameters["@num"].Value = inNum;
       parm = cmd.Parameters.Add("@data", DB2Type.Xml);
       parm.Direction = ParameterDirection.Input;
       cmd.Parameters["@data"].Value = inXML ;
       cmd.ExecuteNonQuery();
       cmd.Close();

       // Retrieve XML value using XQuery
                     and assign value to an XML output parameter
       cmd = DB2Context.GetCommand();
       cmd.CommandText = "XQUERY for $x " +
                         "in db2-fn:xmlcolumn(¥"xmlDataTable.xdata¥")/doc "+
                         "where $x/make = ¥'Mazda¥' " +
                         "return <carInfo>{$x/make}{$x/model}</carInfo>";
       reader = cmd.ExecuteReader();
       reader.CacheData= true;

       if (reader.Read())
       {  outXML1 = reader.GetDB2Xml(0);  }
       else
       {  outXML1 = DB2Xml.Null;  }

       reader.Close();
       cmd.Close();

       // Retrieve XML value using SQL
                     and assign value to an XML output parameter value
       cmd = DB2Context.GetCommand();
       cmd.CommandText = "SELECT xdata "
                         + "FROM xmlDataTable "
                         + "WHERE num = ?";
```

```
        parm = cmd.Parameters.Add("@num", DB2Type.Integer );
        parm.Direction = ParameterDirection.Input;
        cmd.Parameters["@num"].Value = inNum;
        reader = cmd.ExecuteReader();
        reader.CacheData= true;

        if (reader.Read())
        {  outXML2 = reader.GetDB2Xml(0);  }
        else
        {  outXML = DB2Xml.Null;  }

        reader.Close() ;
        cmd.Close();

        return;
    }
```

例: C プロシージャーでの XML および XQuery サポート

プロシージャーの基本、C ルーチンの本質部分、XQuery および XML を理解したなら、XML 機能を持つ C プロシージャーの作成および使用を始めることができます。

次の例は、XML データの更新および照会方法に加えて、タイプ XML のパラメーターを使用する C プロシージャーを示します。

前提条件

C プロシージャーの例を使用した作業を開始する前に、概念について説明している以下のトピックを参照することもできます。

- ルーチン使用の利点

次の例では、以下のように定義された xmlDataTable という名前の表を使用します。

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                                <type>car</type>
                                <make>Pontiac</make>
                                <model>Sunfire</model>
                                </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                                <type>car</type>
                                <make>Mazda</make>
                                <model>Miata</model>
                                </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Mary</name>
                                <town>Vancouver</town>
                                <street>Waterside</street>
                                </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Mark</name>
                                <town>Edmonton</town>
                                <street>Oak</street>
                                </doc>' PRESERVE WHITESPACE)),
   (5, XMLPARSE(DOCUMENT '<doc>
```

```
                                <type>animal</type>
                                <name>dog</name>
                                </doc>' PRESERVE WHITESPACE)),
(6, NULL),
(7, XMLPARSE(DOCUMENT '<doc>
                                <type>car</type>
                                <make>Ford</make>
                                <model>Taurus</model>
                                </doc>' PRESERVE WHITESPACE)),
(8, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Kim</name>
                                <town>Toronto</town>
                                <street>Elm</street>
                                </doc>' PRESERVE WHITESPACE)),
(9, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Bob</name>
                                <town>Toronto</town>
                                <street>Oak</street>
                                </doc>' PRESERVE WHITESPACE)),
(10, XMLPARSE(DOCUMENT '<doc>
                                 <type>animal</type>
                                 <name>bird</name>
                                 </doc>' PRESERVE WHITESPACE))
```

手順 独自の C プロシージャーを作成するときには、以下の例を参考にしてください。

- 『C 外部コード・ファイル』
- 104 ページの『例 1: XML フィーチャーを持つ C パラメーター・スタイル SQL プロシージャー』

C 外部コード・ファイル

例は、CREATE PROCEDURE ステートメントと、関連アセンブリーのビルド元プロシージャーの外部 C コード・インプリメンテーションという 2 つの部分から成っています。

以下の例のプロシージャー・インプリメンテーションに含まれる C ソース・ファイルは、 gwenProc.SQC という名前であり、以下の形式になっています。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sqlda.h>
#include <sqlca.h>
#include <sqludf.h>
#include <sql.h>
#include <memory.h>

// C procedures
               ...
```

ファイルの先頭には、このファイルに組み込むものを示します。組み込み SQL ルーチンには、XML サポートに必要な余分の組み込みファイルはありません。

ファイルの名前、およびプロシージャー・インプリメンテーションに対応する関数の名前をメモしておくことは重要です。各プロシージャーの CREATE PROCEDURE ステートメントの EXTERNAL 節でその情報を指定して、DB2 データベース・マネージャーがその C プロシージャーに該当するライブラリーとエント

リー・ポイントを見つけられるようにする必要があるからです。

例 1: XML フィーチャーを持つ C パラメーター・スタイル SQL プロシージャー

この例では、以下について説明します。

- パラメーター・スタイル SQL のプロシージャーの CREATE PROCEDURE ステートメント
- XML パラメーターを使用するパラメーター・スタイル SQL プロシージャーの C コード

このプロシージャーは 2 つの入力パラメーターを取ります。最初の入力パラメーターの名前は inNum で、タイプは INTEGER です。 2 番目の入力パラメーターの名前は inXML で、タイプは XML です。入力パラメーターの値を使用して、行を表 xmlDataTable に挿入します。次に、XML 値が SQL ステートメントを使用して検索されます。もう 1 つの XML 値が XQuery 式を使用して検索されます。検索された XML 値はそれぞれ 2 つの出力パラメーター、out1XML と out2XML に割り当てられます。結果セットは戻されません。

```
  CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                               IN inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K)
                             )
  LANGUAGE C
  PARAMETER STYLE SQL
  DYNAMIC RESULT SETS 0
  FENCED
  THREADSAFE
  DETERMINISTIC
  NO DBINFO
  MODIFIES SQL DATA
  PROGRAM TYPE SUB
  EXTERNAL NAME 'gwenProc!xmlProc1' ;

  //****************************************************************************
  //  Stored Procedure: xmlProc1
  //
  //  Purpose:   insert XML data into XML column
  //
  //  Parameters:
  //
  //   IN:    inNum -- the sequence of XML data to be insert in xmldata table
  //          inXML -- XML data to be inserted
  //   OUT:   out1XML -- XML data returned - value retrieved using XQuery
  //          out2XML -- XML data returned - value retrieved using SQL
  //****************************************************************************

#ifdef __cplusplus
extern "C"
#endif
SQL_API_RC SQL_API_FN testSecA1(sqlint32* inNum,
                                SQLUDF_CLOB* inXML,
                                SQLUDF_CLOB* out1XML,
                                SQLUDF_CLOB* out2XML,
                                SQLUDF_NULLIND *inNum_ind,
                                SQLUDF_NULLIND *inXML_ind,
                                SQLUDF_NULLIND *out1XML_ind,
                                SQLUDF_NULLIND *out2XML_ind,
                                SQLUDF_TRAIL_ARGS)
{
  char *str;
  FILE *file;
```

```
  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
       sqlint32 hvNum1;
       SQL TYPE IS XML AS CLOB(200) hvXML1;
       SQL TYPE IS XML AS CLOB(200) hvXML2;
       SQL TYPE IS XML AS CLOB(200) hvXML3;
  EXEC SQL END DECLARE SECTION;

  /* Check null indicators for input parameters */
  if ((*inNum_ind < 0) ||  (*inXML_ind < 0)) {
     strcpy(sqludf_sqlstate, "38100");
     strcpy(sqludf_msgtext, "Received null input");
     return 0;
  }

  /* Copy input parameters to host variables */
  hvNum1 = *inNum;
  hvXML1.length = inXML->length;
  strncpy(hvXML1.data, inXML->data, inXML->length);

  /* Execute SQL statement */
  EXEC SQL
     INSERT INTO xmlDataTable (num, xdata) VALUES (:hvNum1, :hvXML1);

  /* Execute SQL statement */
  EXEC SQL
     SELECT xdata INTO :hvXML2
        FROM xmlDataTable
           WHERE num = :hvNum1;

  sprintf(stmt5, "SELECT XMLQUERY('for $x in $xmldata/doc
                                   return <carInfo>{$x/model}</carInfo>'
                                   passing by ref xmlDataTable.xdata
                                       as ¥"xmldata¥" returning sequence)
                    FROM xmlDataTable WHERE num = ?");

  EXEC SQL PREPARE selstmt5 FROM :stmt5 ;
  EXEC SQL DECLARE c5 CURSOR FOR selstmt5;
  EXEC SQL OPEN c5 using :hvNum1;
  EXEC SQL FETCH c5 INTO :hvXML3;

  exit:

  /* Set output return code */
  *outReturnCode = sqlca.sqlcode;
  *outReturnCode_ind = 0;

  return 0;
}
```

C# .NET CLR 関数の例

ユーザー定義関数 (UDF) の基礎と CLR ルーチンの基本を理解できたら、アプリケーションやデータベース環境の中で CLR UDF をさっそく活用できます。このトピックでは、手始めとして CLR UDF の例をいくつか紹介します。

始める前に

CLR UDF の例を使用した作業を開始する前に、概念について説明している以下のトピックを参照することもできます。

- 53 ページの『第 3 章 .NET 共通言語ランタイム (CLR) ルーチン』

- 65 ページの『DB2 コマンド・ウィンドウから .NET CLR ルーチンを作成する』
- 14 ページの『外部スカラー関数』
- 共通言語ランタイム (CLR) .NET ルーチンのビルド

次の例では、SAMPLE データベースに含まれる EMPLOYEE という名前の表を使用しています。

このタスクについて

C# の CLR プロシージャーの例については、以下を参照してください。

- 77 ページの『C# .NET CLR プロシージャーの例』

手順

独自の C# CLR UDF を作成するときには、以下の例を参考にしてください。

- 106
- 106
- 108

例

C# 外部コード・ファイル

以下の例では、C# UDF のさまざまなインプリメンテーションを示しています。各 UDF ごとに、関連アセンブリーのビルド元になる C# ソース・コードとともに、 CREATE FUNCTION ステートメントを用意します。以下の例で使用している関数宣言に含まれる C# ソース・ファイルは、gwenUDF.cs という名前であり、以下の形式になっています。

```
using System;
using System.IO;
using IBM.Data.DB2;

namespace bizLogic
{
   ...
   // Class definitions that contain UDF declarations
   // and any supporting class definitions
   ...
}
```

C# ファイル内のクラスに関数宣言を組み込む必要があります。ネーム・スペースの使用はオプションです。ネーム・スペースを使用する場合は、CREATE PROCEDURE ステートメントの EXTERNAL 節に指定するアセンブリー・パス名の中にネーム・スペースを入れなければなりません。関数に SQL が含まれる場合は、IBM.Data.DB2. を含める必要があります。

例 1: C# のパラメーター・スタイル SQL の表関数

この例では、以下について説明します。

- パラメーター・スタイル SQL の表関数の CREATE FUNCTION ステートメント
- パラメーター・スタイル SQL の表関数の C# コード

この表関数は、データ配列から作成された従業員データの行を含んだ表を戻します。この例には、2 つの関連クラスがあります。 1 つは従業員を表す

クラス person であり、もう 1 つはクラス person を使用するルーチン表 UDF を含んだクラス empOps です。従業員の給与情報は、入力パラメーターの値に基づいて更新されます。この例のデータ配列は、表関数を最初に呼び出したときに表関数そのものの中に作成されます。そのような配列は、ファイル・システム上のテキスト・ファイルからデータを読み取ることによっても作成できます。表関数のその後の呼び出しで配列データにアクセスするために、データの値がスクラッチパッドに書き込まれます。

表関数を呼び出すたびに、1 つのレコードが配列から読み取られ、 1 つの行が関数によって戻される表の中に生成されます。行を表の中に生成する処理は、表関数の出力パラメーターを対象の行値に設定するという形で実行されます。表関数の最終呼び出しが行われた後、生成された行の表が戻されます。

```
CREATE FUNCTION tableUDF(double)
RETURNS TABLE (name varchar(20),
               job varchar(20),
               salary double)
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!tableUDF'
LANGUAGE CLR
PARAMETER STYLE SQL
NOT DETERMINISTIC
FENCED
THREADSAFE
SCRATCHPAD 10
FINAL CALL
EXECUTION CONTROL SAFE
DISALLOW PARALLEL
NO DBINFO

// The class Person is a supporting class for
// the table function UDF, tableUDF, below.
class Person
{
     private String name;
     private String position;
     private Int32 salary;

     public Person(String newName, String newPosition, Int32
     newSalary)
     {
        this.name = newName;
        this.position = newPosition;
        this.salary = newSalary;
     }

     public String getName()
     {
        return this.name;
     }

     public String getPosition()
     {
        return this.position;
     }

     public Int32 getSalary()
     {
        return this.salary;
     }
}
```

```
class empOps
{

   public static void TableUDF( Double factor, out String name,
                     out String position, out Double salary,
                     Int16 factorNullInd, out Int16 nameNullInd,
                     out Int16 positionNullInd, out Int16 salaryNullInd,
                     ref String sqlState, String funcName,
                     String specName, ref String sqlMessageText,
                     Byte[] scratchPad, Int32 callType)
   {

      Int16 intRow = 0;

      // Create an array of Person type information
      Person[] Staff = new
      Person[3];
      Staff[0] = new Person("Gwen", "Developer", 10000);
      Staff[1] = new Person("Andrew", "Developer", 20000);
      Staff[2] = new Person("Liu", "Team Leader", 30000);

      salary = 0;
      name = position = "";
      nameNullInd = positionNullInd = salaryNullInd = -1;

      switch(callType)
      {
         case (-2):  // Case SQLUDF_TF_FIRST:
           break;

         case (-1):  // Case SQLUDF_TF_OPEN:
           intRow = 1;
           scratchPad[0] = (Byte)intRow;  // Write to scratchpad
           break;
         case (0):   // Case SQLUDF_TF_FETCH:
           intRow = (Int16)scratchPad[0];
           if (intRow > Staff.Length)
           {
              sqlState = "02000";  // Return an error SQLSTATE
           }
           else
           {
              // Generate a row in the output table
              // based on the Staff array data.
              name =
              Staff[intRow-1].getName();
              position = Staff[intRow-1].getPosition();
              salary = (Staff[intRow-1].getSalary()) * factor;
              nameNullInd = 0;
              positionNullInd = 0;
              salaryNullInd = 0;
           }
           intRow++;
           scratchPad[0] = (Byte)intRow;  // Write scratchpad
           break;

         case (1):   // Case SQLUDF_TF_CLOSE:
           break;

         case (2):   // Case SQLUDF_TF_FINAL:
           break;
      }
   }
}
```

例 2: C# のパラメーター・スタイル SQL のスカラー関数

この例では、以下について説明します。

- パラメーター・スタイル SQL のスカラー関数の CREATE FUNCTION ステートメント
- パラメーター・スタイル SQL のスカラー関数の C# コード

このスカラー関数は、操作対象の入力値ごとに 1 つのカウント値を戻します。入力値セットの n 番目の桁にある入力値に対する出力スカラー値は n になります。スカラー関数の各呼び出しでは、行または値の入力セット内のそれぞれの行または値に 1 つの呼び出しが関連付けられており、呼び出しのたびにカウントが 1 つずつ増え、カウントの現行値が戻されます。そのカウントはスクラッチパッドのメモリー・バッファー内に保管されるので、スカラー関数の呼び出しと呼び出しの間でカウントの値が保たれるようになっています。

例えば、表を次のように定義している場合は、このスカラー関数を簡単に呼び出すことができます。

```
CREATE TABLE T (i1 INTEGER);
INSERT INTO T VALUES 12, 45, 16, 99;
```

このスカラー関数の呼び出しには、以下のような簡単な照会を使用できます。

```
   SELECT countUp(i1) as count, i1 FROM T;
```

この照会の出力は次のようになります。

```
COUNT           I1
-----------     ----------
1               12
2               45
3               16
4               99
```

このスカラー UDF は非常に簡単です。スカラー関数を使用するときには、行のカウントだけを戻す代わりに、データの形式を既存の列に合わせることもできます。例えば、住所列の各値にストリングを付加することや、一連の入力ストリングから複雑なストリングを組み立てることや、中間結果の保管先のデータ・セットに対して複雑な数値評価を行うことなども可能です。

```
CREATE FUNCTION countUp(INTEGER)
RETURNS INTEGER
LANGUAGE CLR
PARAMETER STYLE SQL
SCRATCHPAD 10
FINAL CALL
NO SQL
FENCED
THREADSAFE
NOT DETERMINISTIC
EXECUTION CONTROL SAFE
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!CountUp' ;

class empOps
{
   public static void CountUp(     Int32 input,
                                out Int32 outCounter,
                                    Int16 inputNullInd,
                                out Int16 outCounterNullInd,
                                ref String sqlState,
                                    String funcName,
                                    String specName,
                                ref String sqlMessageText,
                                    Byte[] scratchPad,
```

```
                                        Int32 callType)
    {
       Int32 counter = 1;

       switch(callType)
       {
          case -1: // case SQLUDF_FIRST_CALL
            scratchPad[0] = (Byte)counter;
            outCounter = counter;
            outCounterNullInd = 0;
            break;
          case 0:  // case SQLUDF_NORMAL_CALL:
            counter = (Int32)scratchPad[0];
            counter = counter + 1;
            outCounter = counter;
            outCounterNullInd = 0;
            scratchPad[0] =
            (Byte)counter;
            break;
          case 1:  // case SQLUDF_FINAL_CALL:
            counter =
            (Int32)scratchPad[0];
            outCounter = counter;
            outCounterNullInd = 0;
            break;
          default: // Should never enter here
                   // * Required so that at compile time
                   //   out parameter outCounter is always set *
            outCounter = (Int32)(0);
            outCounterNullInd = -1;
            sqlState="ABCDE";
            sqlMessageText = "Should not get here: Default
            case!";
            break;
       }
    }
}
```

第 4 章 IBM Data Server Provider for .NET

IBM Data Server Provider for .NET は、ADO.NET インターフェースのデータ・サーバー・サポートを拡張するものです。プロバイダーは、IBM データ・サーバーへのハイ・パフォーマンスでセキュアなアクセスを提供します。

2 つのプロバイダーは、いずれも IBM Data Server Provider for .NET クライアント・パッケージに含まれています。それらのプロバイダーは、しばしば Common .NET Provider と呼ばれます。

DB2 .NET Provider (IBM.Data.DB2.dll)

DB2 .NET Provider を使用すると、.NET アプリケーションにおいて、以下のデータベース管理システムにアクセスできます。

- DB2 Database for Linux, UNIX, and Windows、バージョン 9.1、バージョン 9.5、バージョン 9.7、バージョン 9.8、およびバージョン 10.1
- DB2 Universal Database™ Version 8 for Windows, UNIX, and Linux
- DB2 for z/OS バージョン 8、バージョン 9、およびバージョン 10 (DB2 Connect™ 経由)
- IBM DB2 for IBM i バージョン 5 リリース 4、バージョン 6 リリース 1 およびバージョン 7 リリース 1、(DB2 Connect 経由) (IBM DB2 バージョン 9.7 フィックスパック 4 以上のバージョン用)
- IBM DB2 for IBM i バージョン 5 リリース 4 およびバージョン 6 リリース 1、 (DB2 Connect 経由) (IBM DB2 バージョン 9.7 フィックスパック 3 以前のバージョン用)
- IBM Informix バージョン 11.10、バージョン 11.50、およびバージョン 11.70

これらのトピックの残りの部分では、Common DB2 .NET Provider について説明します。

Informix データベース・サーバー .NET Provider (IBM.Data.Informix.dll)

Informix データベース・サーバー .NET Provider を使用すると、.NET アプリケーションにおいて、以下のデータベース管理システムにアクセスできます。

- IBM Informix バージョン 11.10 およびバージョン 11.50

このプロバイダーについて詳しくは、IBM Informix Dynamic Server Information Center を参照してください。

Data Server Provider for .NET を使用するアプリケーションを開発および実行するには、.NET Framework が必要です。

IBM Data Server Provider for .NET に加えて、IBM Database Add-In for Visual Studio を使用すると、Microsoft Visual Studio を使用することによって、IBM データ・サーバー用の .NET アプリケーションを短時間で簡単に開発できます。さら

に、Add-In を使用して、データベース・オブジェクト (例えば索引や表) を作成することや、サーバー側オブジェクト (例えばストアード・プロシージャーやユーザー定義関数) を開発することもできます。

DB2 のための IBM Data Server Provider for .NET のデータベース・システム要件

DB2 .NET Provider (IBM.Data.DB2.dll)

DB2 .NET Provider を使用すると、.NET アプリケーションにおいて、以下のデータベース管理システムにアクセスできます。

- DB2 Database for Linux, UNIX, and Windows、バージョン 9.1、バージョン 9.5、バージョン 9.7、バージョン 9.8、およびバージョン 10.1
- DB2 Universal Database Version 8 for Windows, UNIX, and Linux
- DB2 for z/OS バージョン 8、バージョン 9、およびバージョン 10 (DB2 Connect 経由)
- IBM DB2 for IBM i バージョン 5 リリース 4、バージョン 6 リリース 1 およびバージョン 7 リリース 1、(DB2 Connect 経由) (IBM DB2 バージョン 9.7 フィックスパック 4 以上のバージョン用)
- IBM DB2 for IBM i バージョン 5 リリース 4 およびバージョン 6 リリース 1、 (DB2 Connect 経由) (IBM DB2 バージョン 9.7 フィックスパック 3 以前のバージョン用)
- IBM Informix バージョン 11.10、バージョン 11.50、およびバージョン 11.70

これらのトピックの残りの部分では、Common DB2 .NET Provider について説明します。

Informix データベース・サーバー .NET Provider (IBM.Data.Informix.dll)

Informix データベース・サーバー .NET Provider を使用すると、.NET アプリケーションにおいて、以下のデータベース管理システムにアクセスできます。

- IBM Informix バージョン 11.10 およびバージョン 11.50

このプロバイダーについて詳しくは、IBM Informix Dynamic Server Information Center を参照してください。

IBM Data Provider for .NET をインストールする前に、.NET Framework の以下のバージョンのいずれかが必要です。

- .NET Framwork バージョン 2.0
- .NET Framwork バージョン 3.0
- .NET Framwork バージョン 3.5
- .NET Framwork バージョン 4.0

.NET Framework がインストールされていないと、IBM Data Server Client およびドライバー・インストーラーによって IBM Data Server Provider for .NET がインストールされることはありません。Data Provider は手動でインストールする必要があります。

DB2 for i の場合、サーバー上で APAR II13348 の修正を適用する必要があります。

ADO.NET アプリケーションの 32 ビットと 64 ビットのサポート

IBM Data Server Provider for .NET の各バージョンでは、32 ビットおよび 64 ビットの .NET アプリケーションがサポートされます。 IBM Data Server Provider for .NET は、DB2 バージョン 9.5 以降のクライアントとサーバーに付属しています。

以下の表は、IBM Data Server Provider for .NET と Microsoft .NET Framework の組み合わせによって可能になる 32 ビットおよび 64 ビットのサポートを示しています。

表 9. IBM Data Server Provider for .NET と Microsoft .NET Framework によって提供される 32 ビットおよび 64 ビット・サポート

IBM Data Server Provider for .NET と Microsoft .NET Framework のバージョン	**32 ビットのサポート**	**64 ビットのサポート**
IBM Data Server Provider for .NET および Microsoft .NET Framework バージョン 2.0、バージョン 3.0、バージョン 3.5、またはバージョン 4.0	はい	はい

CLR ストアード・プロシージャーおよびユーザー定義関数は、IBM Data Server Provider for .NET の 32 ビット・バージョンと 64 ビット・バージョンでサポートされます。フィックスパックより前では、32 ビット版だけがサポートされていました。

64 ビット ADO.NET アプリケーションでの 32 ビット・サポート

IBM Data Server Package for .NET の各バージョンでは、64 ビット・クライアント・パッケージ内の 32 ビット .NET プロバイダーがサポートされます。 64 ビット IBM Data Server Package をインストールすると、32 ビットおよび 64 ビットの両方のプロバイダーがインストールされて構成されます。

IBM DB2 バージョン 9.7 フィックスパック 2 以降、32 ビットのプロバイダーは 64 ビットのプロバイダーと一緒にパッケージされます。

IBM Data Server Provider for .NET を使用するためのアプリケーションのプログラミング

ADO.NET 共通基本クラスを使用した汎用コーディング

.NET Framework バージョン 2.0、3.0、および 3.5 は、`System.Data.Common` というネーム・スペースを特色としています。これは、任意の .NET データ・プロバイダーが共用できる基本クラスのセットを備えています。これにより、異なる複数のデータベースで共通に使用できる一定のプログラミング・インターフェースを備えた汎用 ADO.NET データベース・アプリケーション開発アプローチが容易になります。

以下の C# は、データベース接続を確立するための汎用の方法を示します。

```
DbProviderFactory factory = DbProviderFactories.GetFactory("IBM.Data.DB2");
DbConnection conn = factory.CreateConnection();
DbConnectionStringBuilder sb = factory.CreateConnectionStringBuilder();

if( sb.ContainsKey( "Database" ) )
{
   sb.Remove( "database" );
   sb.Add( "database", "SAMPLE" );
}

conn.ConnectionString = sb.ConnectionString;

conn.Open();
```

DbProviderFactory オブジェクトは、汎用 ADO.NET アプリケーションの開始点です。このオブジェクトは、.NET データ・プロバイダー・オブジェクトの汎用インスタンス (接続、データ・アダプター、コマンド、およびデータ読取装置など) を作成します。これは、特定のデータベース製品とともに実行します。前述の例において、GetFactory メソッドに渡される "IBM.Data.DB2" ストリングは、IBM Data Server Provider for .NET を固有に識別します。そして、DbProviderFactory インスタンスの初期化が実行され、その結果、IBM Data Server Provider for .NET に固有のデータベース・プロバイダー・オブジェクト・インスタンスが作成されます。

DbConnection オブジェクトは、DB2Connection オブジェクト (実際には DbConnection から継承されたもの) と同じように DB2 familyIDS データベースに接続できます。 DbConnectionStringBuilder クラスを使用して、データ・プロバイダーの接続ストリング・キーワードを判別し、カスタム接続ストリングを生成できます。前述の例のコードは、IBM Data Server Provider for .NET 中に "database" というキーワードが存在するかどうかをチェックし、存在する場合、SAMPLE データベースに接続するための接続ストリングを生成します。

IBM Data Server Provider for .NET を使用してアプリケーションからデータベースに接続する

IBM Data Server Provider for .NET を使用する場合、データベース接続は DB2Connection クラスによって確立されます。

手順

データベースに接続するには、次のようにします。

1. 接続パラメーターを格納するストリングを作成します。典型的な接続ストリングの形式は、以下のとおりです。

   ```
   Server=<ip address/localhost>:<port number>;
   Database=<db name>;
   UID=<userID>;
   PWD=<password>;
   Connect Timeout=<Timeout value>
   ```

   以下は、考えられる接続ストリングの例です。

   例 1:

```
String connectString = "Database=SAMPLE";
// When used, attempts to connect to the SAMPLE database.
```

注: 接続ストリングの中にデータベース名のみを指定する場合、サーバー、ユーザー ID、パスワードなどのその他の情報は db2dsdriver.cfg ファイルの中に含める必要があります。

例 2:

```
String cs = "Server=srv:50000;Database=SAMPLE;UID=db2adm;PWD=ab1d;Connect Timeout=30";
// When used, attempts to connect to the SAMPLE database on the server
// 'srv' through port 50000 using 'db2adm' and 'ab1d' as the user id and
// password respectively. If the connection attempt takes more than thirty seconds,
// the attempt will be terminated and an error will be generated.
```

2. connectString を DB2Connection のコンストラクターに渡します。
   - C# でデータベースに接続する

     ```
     String connectString = "Database=SAMPLE";
     DB2Connection conn = new DB2Connection(connectString);
     conn.Open();
     return conn;
     ```

   - Visual Basic .NET でデータベースに接続する

     ```
     Dim connectString As String = "Database=SAMPLE"
     Dim conn As DB2Connection = new DB2Connection(connectString)
     conn.Open()
     Return conn
     ```

3. DB2Connection オブジェクトの Open メソッドを使用することによって、connectString の中で識別されているデータベースに正式に接続します。

IBM Data Server Provider for .NET での接続プーリング

DB2 データベースとの接続が最初にオープンされると、その時点で接続プールが作成されます。接続がクローズされると、それらはプールに入れられ、同じプロセスの中で接続を必要とする他のアプリケーションにより再利用可能な状態になります。

IBM Data Server Provider for .NET では、接続プール判別のために、一連の接続ストリング属性の正規化されたセットが使用されます。正規化属性を使用することにより、アプリケーションが接続を再利用する可能性が大きくなります。

IBM Data Server Provider for .NET では、接続プールがデフォルトで有効になります。

注: Pooling=false という接続ストリング・キーワード/値の対を使用して、接続プーリングをオフにできます。しかし、接続プールの機能をオフにすると、COM+ アプリケーションは動作しなくなります。

以下の接続ストリング・キーワードを設定して、接続プールの動作を制御できます。

- 最小および最大プール・サイズ (**MinPoolSize** および **MaxPoolSize**)
- 接続がアイドル状態になってからプールに戻されるまでの時間 (**ConnectionLifetimeInPool**)

IBM Data Server Provider for .NET を介したトラステッド接続の作成

バージョン 9.5 フィックスパック 1 以降、.NET アプリケーションで接続ストリング・キーワードを使用してトラステッド・コンテキストをサポートできるようになりました。

接続ストリングでは、以下のキーワードを使用できます。

- TrustedContextSystemUserID または tcsuid。これは、接続で使用されるトラステッド・コンテキスト SYSTEM AUTHID を指定します。
- TrustedContextSystemPassword または tcspwd。これは、接続で使用されるトラステッド・コンテキスト SYSYTEM AUTHID に対応するパスワードを指定します。

TrustedContextSystemUserID キーワード値を使用せずに TrustedContextSystemPassword キーワードを指定した場合、InvalidArgument 例外がスローされます。また、トラステッド・コンテキスト・シナリオでは UserID キーワードも必要です。

IBM Data Server Provider for .NET を介したトラステッド・コンテキストは、現在、以下のバージョンでサポートされています。

- DB2 Database for Linux, UNIX, and Windows バージョン 9.5、バージョン 9.7、およびバージョン 9.8
- DB2 Universal Database for z/OS バージョン 9 およびバージョン 10

例

以下の情報を使用して、トラステッド・コンテキストがサーバー上に確立されたとします。

```
CREATE TRUSTED CONTEXT ctxName1
BASED UPON CONNECTION USING SYSTEM AUTHID masteruser
ATTRIBUTES ( PROTOCOL 'TCPIP',
             ADDRESS '9.26.146.201',
             ENCRYPTION 'NONE' )
ENABLE
WITH USE FOR userapp1 WITH AUTHENTICATION, userapp2 WITH AUTHENTICATION;
```

SYSTEM AUTHID masteruser には、対応するパスワード masterpassword があります。特定のユーザー/アプリケーション userapp1、および userapp2 には、それぞれ対応するパスワード passapp1 および passapp2 があります。

このトラステッド・コンテキストを使用するために、例えば以下のような接続ストリングがアプリケーションによって発行されます。

- アプリケーション 1

  ```
  database=db;server=server1:446;
  UserID=userapp1;Password=passapp1;
  TrustedContextSystemUserID=masteruser;TrustedContextSystemPassword=masterpassword
  ```

- アプリケーション 2

  ```
  database=db;server=server1:446;
  UserID=userapp2;Password=passapp2;
  TrustedContextSystemUserID=masteruser;TrustedContextSystemPassword=masterpassword
  ```

注: トラステッド・コンテキスト状態では、標準的なアプリケーションの場合と同じように、キーワード UserID が接続のエンド・ユーザーに対応します。

次の .NET プログラムは、接続のオープンとクローズを行います。

```
[C#]
DB2Connection conn = new DB2Connection();

conn.ConnectionString = "database=db;server=server1:446;"
     + "UserID=userapp1;Password=passapp1;"
     + "TrustedContextSystemUserID=masteruser;"
     + "TrustedContextSystemPassword=masterpassword;"

conn.Open();

// Do processing as userapp1, such as querying tables

conn.Close();

conn.ConnectionString = "database=db;server=server1:446;UserID=userapp2;"
     + "Password=passapp2;TrustedContextSystemUserID=masteruser;"
     + "TrustedContextSystemPassword=masterpassword;"

conn.Open();

// Do processing as userapp2

conn.Close();
```

サーバー上にトラステッド・コンテキストが設定されなかったためにトラステッド・コンテキスト処理が失敗した場合、あるいはサーバーでトラステッド・コンテキストがサポートされない場合には、SQLCODE CLI0197E とともにエラーがスローされます。 TrustedContextSystemUserID キーワード値が無効である場合 (例えば長すぎる場合) には、SQLCODE CLI0124E とともにエラーがスローされます。サーバーは、固有エラー・コード -20361 を伴う SQLCODE SQL1046N、SQL30082N、または SQL0969N とともにエラーを報告する場合があります。これらのどのエラーが発生した場合も、Open() は失敗します。

注: トラステッド・コンテキスト処理は、サーバーとの次回の通信で行われます。

ADO.NET データベース・アプリケーションでの SQL データ・タイプ表記

ADO.NET データベース・アプリケーションでは、SQL ステートメントの実行の一環で使われるパラメーター値および変数として、DB2 SQL データ・タイプ値を参照できます。ただし、値のアクセス時または取得時にデータの切り捨てや損失が発生しないよう、適切な IBM Data Server Provider for .NET データ・タイプ値と .NET Framework データ・タイプ値を使用する必要があります。

SQL ステートメントの実行で使用されるパラメーター値を指定するには、IBM Data Server Provider for .NET オブジェクトを使用する必要があります。SQL ステートメントを表す `DB2Command` オブジェクトに追加するパラメーターを表すために、`DB2Parameter` オブジェクトが使用されます。パラメーターのデータ・タイプ値を指定する際には、`IBM.Data.DB2Types` 名前空間で使用可能な IBM Data Server Provider for .NET データ・タイプ値を使用する必要があります。
`IBM.Data.DB2Types` 名前空間には、サポートされる DB2 SQL データ・タイプのそれぞれを表すクラスおよび構造体が提供されています。

SQL データ・タイプ値を一時的に保持する可能性のあるローカル変数に関しては、IBM.Data.DB2Types 名前空間で定義されている該当する IBM Data Server Provider for .NET データ・タイプを使用する必要があります。

次の表は、DB2Type データ・タイプ、DB2 データ・タイプ、 Informix データ・タイプ、Microsoft .NET Framework 型、および DB2Types クラスと構造体の対応を示しています。

表 10. データ・タイプ、クラスおよび構造体の間のマッピング

分類	DB2Types クラスと構造体	DB2Type データ・タイプ	DB2 データ・タイプ	Informix データ・タイプ	.NET データ・タイプ
バイナリー・データ	DB2Binary	Binary	CHAR FOR BIT DATA		Byte[]
	DB2Binary	Binary[7]	BINARY		Byte[]
	DB2Binary	VarBinary[7]	VARBINARY		Byte[]
	DB2Binary	LongVarBinary[5]	LONG VARCHAR FOR BIT DATA		Byte[]
文字データ	DB2String	Char	CHAR	CHAR	String
	DB2String	VarChar	VARCHAR	VARCHAR	String
	DB2String	LongVarChar[5]	LONG VARCHAR	LVARCHAR	String
グラフィック・データ	DB2String	Graphic	GRAPHIC		String
	DB2String	VarGraphic	VARGRAPHIC		String
	DB2String	LongVarGraphic[5]	LONG VARGRAPHIC		String
LOB データ	DB2Clob	Clob	CLOB	CLOB, TEXT	String
	DB2Blob	Blob	BLOB	BLOB, BYTE	Byte[]
	DB2Clob	DbClob	DBCLOB		String

5. これらのデータ・タイプは、DB2 .NET 共通言語ランタイム・ルーチンではパラメーターとしてサポートされていません。

6. DB2 の DB2Type.Xml 型の ParameterClass.ParameterName プロパティーには、String 型、byte[] 型、DB2Xml 型、および XmlReader 型の変数を入れることができます。

7. これらのデータ・タイプを適用できるのは、DB2 for z/OS だけです。

8. このデータ・タイプは、DB2 for z/OS バージョン 9 以降のリリースおよび DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降のリリースでのみサポートされています。

9. 日付および時刻のオブジェクトには、タイム・スタンプのストリング・リテラルが使用できます。タイム・スタンプ・オブジェクトとしては、日付ストリング・リテラルが可能です。

表 10. データ・タイプ、クラスおよび構造体の間のマッピング (続き)

分類	DB2Types クラスと構造体	DB2Type データ・タイプ	DB2 データ・タイプ	Informix データ・タイプ	.NET データ・タイプ
数値データ	DB2Int16	SmallInt	SMALLINT	BOOLEAN, SMALLINT	Int16
	DB2Int32	Integer	INT	INTEGER、 INT、 SERIAL	Int32
	DB2Int64	BigInt, BigSerial	BIGINT	BIGINT, BIGSERIAL, INT8, SERIAL8	Int64
	DB2Real, DB2Real370	Real	REAL	REAL, SMALLFLOAT	Single
	DB2Double	Double	DOUBLE PRECISION	DECIMAL (≤ 29), DOUBLE PRECISION	Double
	DB2Double	Float	FLOAT	DECIMAL (32), FLOAT	Double
	DB2Decimal	Decimal	DECIMAL	MONEY	Decimal
	DB2DecimalFloat	DecimalFloat	DECFLOAT(16|34)[58]		Decimal
	DB2Decimal	Numeric	DECIMAL	DECIMAL (≤ 29), NUMERIC	Decimal
日付/時刻データ	DB2Date	Date	DATE	DATETIME (日付精度)	DateTime String[9]
	DB2Time	Time	TIME	DATETIME (時刻精度)	TimeSpan String[9]
	DB2TimeStamp	Timestamp	TIMESTAMP	DATETIME (時刻と日付の精度)	DateTime String[9]
	DB2TimeStamp Offset	TimestampWith TimeZone	TIMESTAMP WITH TIME ZONE	なし	DateTimeOffset String[9]
行 ID データ	DB2RowId	RowId	ROWID		Byte[]
XML データ	DB2Xml	Xml[6]	XML		Byte[]

IBM Data Server Provider for .NET を使用したアプリケーションからの SQL ステートメントの実行

IBM Data Server Provider for .NET を使用する場合、SQL ステートメントは、DB2Command クラスのメソッド ExecuteReader() および ExecuteNonQuery()、およびそのプロパティー CommandText、CommandType、および Transaction を使用して実行します。

このタスクについて

出力を生成する SQL ステートメントの場合は ExecuteReader() メソッドを使用する必要があり、その結果は DB2DataReader オブジェクトから取り出すことができます。他のすべての SQL ステートメントの場合は、 ExecuteNonQuery() メソッドを使用する必要があります。 DB2Command オブジェクトの Transaction プロパティー

は、DB2Transaction オブジェクトに初期化してください。 DB2Transaction オブジェクトは、データベース・トランザクションのロールバックとコミットを担当します。

C# で UPDATE ステートメントを実行する

```
// assume a DB2Connection conn
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "                  FROM staff " +
    "                  WHERE id >= 310) " +
                  "  WHERE id = 310";

cmd.ExecuteNonQuery();
```

Visual Basic .NET で UPDATE ステートメントを実行する

```
' assume a DB2Connection conn
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "                  FROM staff " +
    "                  WHERE id >= 310) " +
                  "  WHERE id = 310";
cmd.ExecuteNonQuery();
```

C# で SELECT ステートメントを実行する

```
// assume a DB2Connection conn
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "SELECT deptnumb, location " +
                  "  FROM org " +
                  "  WHERE deptnumb < 25";

DB2DataReader reader = cmd.ExecuteReader();
```

Visual Basic .NET で SELECT ステートメントを実行する

```
' assume a DB2Connection conn
Dim cmd As DB2Command = conn.CreateCommand()
Dim trans As DB2Transaction = conn.BeginTransaction()
cmd.Transaction = trans
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "                  FROM staff " +
    "                  WHERE id >= 310) " +
                  "  WHERE id = 310";

cmd.ExecuteNonQuery()
```

アプリケーションでデータベース・トランザクションを実行後、それをロールバックするか、またはコミットする必要があります。そのためには、DB2Transaction オブジェクトの Commit() メソッドおよび Rollback() メソッドを使用します。

C# でトランザクションをロールバックまたはコミットする

```
// assume a DB2Transaction object conn
trans.Rollback();
...
trans.Commit();
```

Visual Basic.NET でトランザクションをロールバックまたはコミットする

```
' assume a DB2Transaction object conn
trans.Rollback()
...
trans.Commit()
```

IBM Data Server Provider for .NET を使用したアプリケーションからの結果セットの読み取り

IBM Data Server Provider for .NET を使用する場合、結果セットを読み取るには、DB2DataReader オブジェクトを使用します。結果セットの中で次の行に進むには、DB2DataReader の Read() メソッドを使用します。

このタスクについて

出力に含まれる個々の列からデータを抽出するには、メソッド GetString()、GetInt32()、GetDecimal()、およびその他の使用可能なすべてのデータ・タイプ用のメソッドを使用します。出力の読み取りが終了したら、DB2DataReader オブジェクトを必ずクローズしなければなりません。そのためには、DB2DataReader の Close() メソッドを使用します。

C# で結果セットを読み取る

```
// assume a DB2DataReader reader
Int16 deptnum = 0;
String location="";

// Output the results of the query
while(reader.Read())
{
  deptnum = reader.GetInt16(0);
  location = reader.GetString(1);
  Console.WriteLine("    " + deptnum + " " + location);
}
reader.Close();
```

Visual Basic .NET で結果セットを読み取る

```
' assume a DB2DataReader reader
Dim deptnum As Int16 = 0
Dim location As String ""

' Output the results of the query
Do While (reader.Read())
  deptnum = reader.GetInt16(0)
  location = reader.GetString(1)
  Console.WriteLine("    " & deptnum & " " & location)
Loop
reader.Close();
```

IBM Data Server Provider for .NET を使用した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

IBM Data Server Provider for .NET を使用する場合、DB2Command オブジェクトを使用することによってストアード・プロシージャーを呼び出すことができます。

このタスクについて

CommandType プロパティーのデフォルト値は、CommandType.Text です。これは SQL ステートメントに適した値であり、ストアード・プロシージャーを呼び出すためにも使用できます。ただし、CommandType に CommandType.StoredProcedure を設定したほうが、ストアード・プロシージャーの呼び出しはより容易になります。この場合は、ストアード・プロシージャー名とパラメーターのみを指定する必要があります。

ストアード・プロシージャーの作業を実行する場合、ホスト変数、名前付きパラメーター、または位置によるパラメーターを使用することにより、パラメーターを渡すことができます。しかし、同じ SQL ステートメントの中でそれらを組み合わせて使用することはできません。

以下に示す C# および Visual Basic の例は、INOUT_PARAM というストアード・プロシージャーを呼び出す方法を示します。その際、CommandType プロパティーを CommandType.StoredProcedure または CommandType.Text のいずれかに設定します。

手順

- C# で DB2Command の CommandType プロパティーを CommandType.Text に設定することによって、ストアード・プロシージャーを呼び出します。

```
// assume a DB2Connection conn
DB2Transaction trans = conn.BeginTransaction();
DB2Command cmd = conn.CreateCommand();
String procName = "INOUT_PARAM";
String procCall = "CALL INOUT_PARAM (@param1, @param2, @param3)";
cmd.Transaction = trans;
cmd.CommandType = CommandType.Text;
cmd.CommandText = procCall;

// Register input-output and output parameters for the DB2Command
cmd.Parameters.Add( new DB2Parameter("@param1", "Value1");
cmd.Parameters.Add( new DB2Parameter("@param2", "Value2");
DB2Parameter param3 = new DB2Parameter("@param3", IfxType.Integer);
param3.Direction = ParameterDirection.Output;
cmd.Parameters.Add( param3 );

// Call the stored procedure
Console.WriteLine("  Call stored procedure named " + procName);
cmd.ExecuteNonQuery();
```

- Visual Basic で DB2Command の CommandType プロパティーを CommandType.Text に設定することによって、ストアード・プロシージャーを呼び出します。

```
' assume a DB2Connection conn
Dim trans As DB2Transaction = conn.BeginTransaction()
Dim cmd As DB2Command = conn.CreateCommand()
Dim procName As String = "INOUT_PARAM"
Dim procCall As String = "CALL INOUT_PARAM (?, ?, ?)"
cmd.Transaction = trans
cmd.CommandType = CommandType.Text
cmd.CommandText = procCall

' Register input-output and output parameters for the DB2Command
...

' Call the stored procedure
Console.WriteLine("  Call stored procedure named " & procName)
cmd.ExecuteNonQuery()
```

注: CALL と EXECUTE PROCEDURE は、いずれもサポートされています。

- C# で DB2Command の CommandType プロパティーを CommandType.StoredProcedure に設定することによって、ストアード・プロシージャーを呼び出します。この方法を使用する場合、名前付きパラメーターはサポートされません。

```
// assume a DB2Connection conn
DB2Transaction trans = conn.BeginTransaction();
DB2Command cmd = conn.CreateCommand();
String procName = "INOUT_PARAM";
cmd.Transaction = trans;
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = procName;

// Register input-output and output parameters for the DB2Command
...

// Call the stored procedure
Console.WriteLine("  Call stored procedure named " + procName);
cmd.ExecuteNonQuery();
```

- Visual Basic で DB2Command の CommandType プロパティーを CommandType.StoredProcedure に設定することによって、ストアード・プロシージャーを呼び出します。

```
' assume a DB2Connection conn
Dim trans As DB2Transaction = conn.BeginTransaction()
Dim cmd As DB2Command = conn.CreateCommand()
Dim procName As String = "INOUT_PARAM"
cmd.Transaction = trans
cmd.CommandType = CommandType.StoredProcedure
cmd.CommandText = procName

' Register input-output and output parameters for the DB2Command
...

' Call the stored procedure
Console.WriteLine("  Call stored procedure named " & procName)
cmd.ExecuteNonQuery()
```

- パラメーター名で修飾する場合、複数のパラメーターを任意の順序でストアード・プロシージャーに渡すことができます。この名前付き引数フィーチャーは、DB2 for Linux, UNIX, and Windows データ・サーバーでのみサポートされます。例えば、以下の SQL ステートメントでは、ストアード・プロシージャーを定義した後、定義とは異なる順序でパラメーターを指定してそれを呼び出しています。

```
CREATE PROCEDURE schema.my_proc ( IN var1 int, INOUT var2 int )
 LANGUAGE SQL
 BEGIN
  -- (プロシージャーのコード)
 END

CALL my_proc (var2=>@param2, var1=>@param1)
```

CURSOR タイプの出力パラメーターによって戻された結果セットへの同時アクセス

IBM Data Server Provider for .NET を使用している場合、DB2Type.Cursor は出力パラメーターのすべてのカーソルに同時にアクセスするように指定されます。

このタスクについて

複数の CURSOR タイプ出力パラメーターのあるストアード・プロシージャーの場合、 DB2TYPE.Cursor を出力パラメーター・オブジェクトにバインドすると、出力パラメーターに含まれるすべてのカーソルに同時にアクセスできます。

例えば、OrderDetails ストアード・プロシージャーで 3 つのカーソルを宣言し、それぞれのカーソルが製品とその売り上げについての関連情報を提供するとします。

```
CREATE OR REPLACE TYPE cur AS CURSOR
CREATE PROCEDURE OrderDetails (p_startDate TIMESTAMP, p_endDate TIMESTAMP,
  OUT prodDetails cur, OUT prodOrderDetails cur, OUT custOrderDetails cur)
LANGUAGE SQL
BEGIN
  SET prodDetails = CURSOR WITH HOLD FOR
    SELECT p.pid, price, quantity FROM products p, inventory i
      WHERE p.pid = i.pid AND p.pid IN (SELECT DISTINCT pid FROM orders) ORDER BY pid;
  SET prodOrderDetails = CURSOR WITH HOLD FOR
    SELECT pid, COUNT(*), SUM (quantity) FROM orders
      WHERE date >= p_startDate AND date <= p_endDate GROUP BY pid ORDER BY pid;
  SET custOrderDetails = CURSOR WITH HOLD FOR
    SELECT pid, custID, COUNT(*), SUM(quantity) FROM orders
      WHERE date >= p_startDate AND date <= p_endDate
      GROUP BY pid, custID ORDER by pid, custID;
  OPEN prodDetails;
  OPEN prodOrderDetails;
  OPEN custOrderDetails;
END;
```

それぞれのカーソルから特定の製品についての関連情報を収集して割引を計算できるように、呼び出し元はこれらのカーソルに同時にアクセスする必要があります。カーソルへの同時アクセスを可能にするために、ストアード・プロシージャーはこれらのカーソルを出力パラメーターとして戻します。 CURSOR タイプの出力パラメーターをバインドして同時アクセスが行われるようにするときは、アプリケーションで DB2Type を DB2Type.Cursor に設定する必要があります。

```
//C# Code sample
cmd.CommandText = "CALL OrderDetails(
      @p_startDate, @p_endDate, @prodDetails, @prodOrderDetails, @custOrderDetails)";
cmd.Parameters.Add("@p_startDate", DateTime.Parse("1/1/2010"));
cmd.Parameters.Add("@p_endDate", DateTime.Parse("12/31/2010"));
cmd.Parameters.Add("@prodDetails", DB2Type.Cursor);
cmd.Parameters["@prodDetails"].Direction = ParameterDirection.Output;
cmd.Parameters.Add("@prodOrderDetails", DB2Type.Cursor);
cmd.Parameters["@prodOrderDetails"].Direction = ParameterDirection.Output;
cmd.Parameters.Add("@custOrderDetails", DB2Type.Cursor);
cmd.Parameters["@custOrderDetails"].Direction = ParameterDirection.Output;
cmd.ExecuteNonQuery();
DB2DataReader prodDetailsDR =
    (DB2DataReader)cmd.Parameters["@prodDetails"].Value;
DB2DataReader prodOrderDetailsDR =
    (DB2DataReader)cmd.Parameters["@prodOrderDetails"].Value;
DB2DataReader custOrderDetailsDR =
    (DB2DataReader)cmd.Parameters["@custOrderDetails"].Value;

while (prodOrderDetailsDR.Read())
{
    pid = prodOrderDetailsDR.GetInt32(0);
    numOrders = prodOrderDetailsDR.GetInt32(1);
    totalOrderQuantity = prodOrderDetailsDR.GetInt32(2);
    prodDetailsDR.Read();
    price = prodDetailsDR.GetDecimal(1);
    currentInventory = prodDetailsDR.GetInt32(2);
    int totalCustOrders = 0;
    while (custOrderDetailsDR.Read())
    {
        custID = custOrderDetailsDR.GetInt32(1);
        numOrdersByCust = custOrderDetailsDR.GetInt32(2);
        totalCustOrders += numOrdersByCust;
        totalOrderQuantityByCust = custOrderDetailsDR.GetInt32(3);
        //Calculate discount based on numOrders, numOrdersByCust,
        //  totalOrderQuantity, totalOrderQuantityByCust, price and currentInventory
        if (totalCustOrders == numOrders) //done with this pid
            break;
    }
}
prodDetailsDR.Close();
prodOrderDetailsDR .Close();
custOrderDetailsDR .Close();
```

CURSOR タイプの出力パラメーターからデータを読み取るプログラムへは、ExecuteNonQuery メソッドを起動した後でのみ Value プロパティーからアクセスできます。

ExecuteReader メソッドか ExecuteResultSet メソッドのいずれかを使用してコマンドが実行されると、結果セットは DB2DataReader オブジェクトまたは DB2ResultSet オブジェクトで戻されます。それ以降の結果セットへは、NextResult メソッドを呼び出すことにより順にアクセスしなければなりません。出力パラメーターはバインドされているものの、出力パラメーター Value プロパティーにアクセスすると、InvalidOperation 例外が発生します。これは、照会が ExecuteNonQuery メソッドを使用して実行されなかったことによるものです。

カーソルを同時に操作しているとき、アプリケーションはカーソルの読み取りを続行する前に、操作をコミットしようとすることがあります。オープン・カーソルを破壊することなくアプリケーションがコミットを発行するためには、ストアード・プロシージャーの中でカーソルを保留可能なものとして宣言しておく必要があります。

pureQuery を使用した、.NET アプリケーションでの照会の最適化

.NET クライアント・ドライバーは、pureQuery テクノロジーに備わって要るフィーチャーを活用することができます。これらのフィーチャーを使用して、既存の .NET アプリケーション照会を静的 SQL として実行することができます。静的照会では、実行時に特定のステートメントを準備する必要がなくなります。 これが、アプリケーションでのセキュリティーおよびパフォーマンスの向上につながる場合もあります。

始める前に

pureQuery テクノロジー・フィーチャーを活用するには、IBM InfoSphere® Optim™ pureQuery Runtime 2.1 (またはそれ以降) とともに、IBM Data Server Provider for .NET バージョン 9.5.3 (またはそれ以降) 用の pureQuery を使用可能にする必要があります。

このタスクについて

このタスクは、.NET アプリケーションで静的照会を発見および使用するための基本的なステップです。

手順

1. 次のようにして、可能性のあるステートメントをキャプチャーするように .NET アプリケーションをセットアップします。
   a. データベースへの接続を管理するコード部分を見つけ出します。
   b. captureMode キーワードを `on` に設定します。
   c. executionMode キーワードが設定されている場合、その値が `dynamic` に設定されていることを確認します。
   d. collection を、パッケージ名 (collection.rootPkgName) のコレクション名に設定します。
   e. rootPkgName を、パッケージ名 (collection.rootPkgName) のルート・パッケージに設定します。
   f. pureQueryXML キーワードを、キャプチャーしたステートメントの保管先のパスおよびファイル名に設定します。

g. アプリケーションを実行し、指定した pureQueryXML ファイル内のステートメントをキャプチャーします。

h. コマンド・プロンプトで **db2cap** ユーティリティーを実行し、キャプチャー・ファイルをデータベースにバインドします。 **db2cap** コマンドには、渡す必要のあるいくつかのパラメーターがあります。

2. 次のようにして、キャプチャーしたステートメントを静的に実行するように .NET アプリケーションを変更します。

a. データベースへの接続を管理するコード部分を見つけ出します。

b. captureMode キーワードを除去するか、または `off` に設定します。

c. executionMode キーワードを `static` に設定します。

タスクの結果

これで .NET アプリケーションは、キャプチャーしたステートメントを静的に実行するようになりました。キャプチャーされなかったステートメントは、引き続き動的に実行されます。

例

実行されたステートメントをキャプチャーするための接続ストリングのセットアップの例を以下のコードに示します。

```
[C#]
string myConnectionString =
    "Database=Sample;captureMode=ON;pureQueryXML=c:¥temp¥capfile.xml";
DB2Connection myConn = new DB2Connection(myConnectionString);
string myInsertQuery = "INSERT INTO STAFF (ID, NAME) Values(...)";
DB2Command myDB2Command = new DB2Command(myInsertQuery);
myDB2Command.Connection = myConn;
myConn.Open();
myDB2Command.ExecuteNonQuery();
myConn.Close();

[Visual Basic]
Dim myConnectionString As String = _
    "Database=Sample;captureMode=ON;pureQueryXML=c:¥temp¥capfile.xml"
Dim myConn As New DB2Connection(myConnectionString)
Dim myInsertQuery As String = "INSERT INTO STAFF (ID, NAME) Values(...)"
Dim myDB2Command As New DB2Command(myInsertQuery)
myDB2Command.Connection = myConn
myConn.Open()
myDB2Command.ExecuteNonQuery()
myConn.Close()
```

次の例は、キャプチャーしたステートメントを静的に実行するために接続ストリングを変更します。

```
[C#]
string myConnectionString =
    "Database=Sample;executionMode=STATIC;pureQueryXML=c:¥temp¥capfile.xml";
DB2Connection myConn = new DB2Connection(myConnectionString);
string myInsertQuery = "INSERT INTO STAFF (ID, NAME) Values(...)";
DB2Command myDB2Command = new DB2Command(myInsertQuery);
myDB2Command.Connection = myConn;
myConn.Open();
myDB2Command.ExecuteNonQuery();
myConn.Close();

[Visual Basic]
```

```
Dim myConnectionString As String = _
    "Database=Sample;captureMode=ON;pureQueryXML=c:¥temp¥capfile.xml"
Dim myConn As New DB2Connection(myConnectionString)
Dim myInsertQuery As String = "INSERT INTO STAFF (ID, NAME) Values(...)"
Dim myDB2Command As New DB2Command(myInsertQuery)
myDB2Command.Connection = myConn
myConn.Open()
myDB2Command.ExecuteNonQuery()
myConn.Close()
```

.NET アプリケーションの pureQuery の使用可能化

Optim pureQuery は、購入の必要のあるライセンス付きフィーチャーです。IBM InfoSphere Optim pureQuery Runtime 製品を購入すると、pureQuery フィーチャーを入手することができます。

始める前に

pureQuery を使用可能にする前に、IBM InfoSphere Optim pureQuery Runtime 2.1 以降をインストールしておく必要があります。また、IBM Data Server Provider for .NET が含まれる以下の製品のうち少なくとも 1 つをインストールしておく必要もあります。

- IBM Data Server Driver for ODBC, CLI, and .NET バージョン 9.5.3、または IBM Data Server Driver Package バージョン 9.5.4 (またはそれ以降)
- IBM Data Server Runtime Client バージョン 9.5.3 (またはそれ以降)
- IBM Data Server Client バージョン 9.5.3 (またはそれ以降)
- DB2 for Linux, UNIX, and Windows バージョン 9.5 フィックスパック 3 (またはそれ以降)

このタスクについて

このタスクは、.NET アプリケーション用の pureQuery を使用可能にします。このタスクの完了後、.NET アプリケーションでの pureQuery テクノロジーのフィーチャーの使用が可能になります。

手順

1. .NET 用の pureQuery ライセンス・ファイルを活動化します。
   - IBM Data Server Driver Package の場合:
     a. pureQuery インストール・ディレクトリーで、正しいバージョンのライセンス・ファイルを見つけます。例: `C:¥Program Files¥IBM¥purequery¥license¥clientv97¥`

        注: ドライバー・パッケージにある **db2level** コマンドを使用することで、ドライバー・パッケージのバージョンを判別できます。
     b. ライセンス・ファイル `dspq_rt.lic` を IBM Data Server Driver Package のライセンス・ディレクトリーにコピーします。ライセンス・ディレクトリーは、 IBM Data Server Driver Package のインストール先のディレクトリーで見つけ出すことができます。例えば、`C:¥Program Files¥IBM¥IBM DATA SERVER DRIVER¥license¥` などです。
   - その他の DB2 クライアントまたはデータ・サーバーの場合:

– コマンド・プロンプトで、コマンド `db2licm -a C:¥pqRuntime21¥pureQuery¥dspq_rt.lic` を実行します。ここで `C:¥pqRuntime21¥` は、pureQuery ランタイムのインストール先のディレクトリーです。

2. Microsoft Windows Vista または Windows 7 オペレーティング・システムの場合、pureQuery ライセンス・ファイルのアクティベーションを完了するには、以下の手順のいずれかを実行する必要があります。これらの手順の実行は、一度だけ必要です。これらの手順を実行しないで Windows Vista または Windows 7 オペレーティング・システム上で pureQuery のフィーチャーを使用しようとすると、有効なライセンス・キーが見つからないというエラーになります。
   - 管理者アカウントで最初のキャプチャーを実行します。管理者アカウントを使用して最初のキャプチャーを実行して正常に終了すると、pureQuery のフィーチャーがそのシステム上のユーザーから使用可能になります。
   - IBM Data Server Driver Package ライセンス・ディレクトリーを構成することにより、pureQuery フィーチャーを使用するユーザー・アカウントに対して書き込みアクセスを許可します。

タスクの結果

pureQuery ライセンス・ファイルが、ご使用のドライバーに対して活動化されます。これで .NET アプリケーションは、クライアント最適化フィーチャーの利点を活用できるようになりました。

次のタスク

pureQuery ライセンス・ファイルを活動化した後、pureQuery を使用して、.NET アプリケーションのデータ・プロバイダーの最適化を開始することができます。

Microsoft Entity Framework のプロバイダー・サポート

IBM Data Server Provider for .NET を使用することにより、IBM データ・サーバーで Microsoft ADO.NET Entity Framework を活用できます。サポートされているサーバー・バージョンを使用して、EDM スキーマを生成したり、Entities アプリケーションのために Entity SQL および LINQ ステートメントを作成および実行したりできます。

システム要件

IBM Data Server Provider for .NET は、以下の IBM データ・サーバーと共に機能します。

- DB2 Database for Linux, UNIX, and Windows バージョン 8 (バージョン 9.8 経由)
- IBM Data Server Client バージョン 9.5.3 以降
- IBM Data Server Runtime Client バージョン 9.5.3 以降
- IBM Data Server Driver for ODBC, CLI, and .NET バージョン 9.5.3、または IBM Data Server Driver Package バージョン 9.5.4、またはそれ以降

- DB2 Universal Database for AS/400® バージョン 5 リリース 4、バージョン 6 リリース 1 およびバージョン 7 リリース 1、(DB2 Connect 経由) (IBM DB2 バージョン 9.7 フィックスパック 4 以上のバージョン用)
- DB2 Universal Database for AS/400 バージョン 5 リリース 4 および DB2 Universal Database for iSeries® バージョン 6 リリース 1 (DB2 Connect 経由) (IBM DB2 バージョン 9.7 フィックスパック 3 以前のバージョン用)
- DB2 for z/OS バージョン 8 (バージョン 10 経由)
- IBM Informix バージョン 11.170 以降

Microsoft .NET Framework 3.5 SP1 以降および Microsoft ADO.NET Entity Framework が必要です。Microsoft Entity Data Model ウィザードまたは ADO.NET Entity Designer を使用してエンティティー・データ・モデルを操作するには、Microsoft Visual Studio 2008 以降も必要です。 .NET Framework 4.0 および Visual Studio 2010 のサポートは、DB2 for Linux, UNIX, and Windows バージョン 9.7 フィックスパック 4 以降に導入されています。

以下の表は、IBM エンティティー・プロバイダーでサポートされる正規関数をリストしています。正規関数は、データ・プロバイダーによって、対応するデータ・ソース関数に変換されます。

表 11. IBM エンティティー・プロバイダーでの正規関数サポート

正規関数のタイプ	LINQ 関数	DB2 for Linux, UNIX, and Windows	DB2 for z/OS	DB2 for IBM i	Informix
Aggregate	Average	Y	Y	Y	Y
	BigCount	Y	Y	Y	Y
	Count	Y	Y	Y	Y
	Maximum	Y	Y	Y	Y
	Minimum	Y	Y	Y	Y
	NewGuid[1]	Y*	Y*	Y*	Y*
	StDev	Y	Y	Y	Y
	StDevP	Y	Y	Y	Y
	Sum	Y	Y	Y	Y
	Var	Y	Y	Y	Y
	VarP	Y	Y	Y	Y
Bitwise	BitWiseAnd[1]	Y	Y*	Y*	Y
	BitWiseNot[1]	Y	Y*	Y*	Y
	BitWiseOr[1]	Y	Y*	Y*	Y
	BitWiseXor[1]	Y	Y*	Y*	Y
Math	Abs	Y	Y	Y	Y
	Ceiling	Y	Y	Y	Y
	Floor	Y	Y	Y	Y
	Power	Y	Y	Y	Y
	Round (value,digits)	Y	Y	Y	Y
	Truncate (value,digits)	Y	Y	Y	Y

表 11. *IBM* エンティティー・プロバイダーでの正規関数サポート *(続き)*

正規関数のタイプ	LINQ 関数	DB2 for Linux, UNIX, and Windows	DB2 for z/OS	DB2 for IBM i	Informix
String	Concat	Y	Y	Y	Y
	Contains[1]	Y	Y	Y	Y*
	EndsWith	Y	Y	Y	Y
	IndexOf[1]	Y	Y	Y	Y*
	Left	Y	Y	Y	Y
	Length	Y	Y	Y	Y
	LTrim	Y	Y	Y	Y
	Replace	Y	Y	Y	Y
	Right	Y	Y	Y	Y
	RTrim	Y	Y	Y	Y
	StartsWith	Y	Y	Y	Y
	Substring	Y	Y	Y	Y
	ToLower	Y	Y	Y	Y
	ToUpper	Y	Y	Y	Y
	Trim	Y	Y	Y	Y
Datetime	AddNanoseconds	Y	Y	Y	Y
	AddMicroseconds	Y	Y	Y	Y
	AddMilliseconds	Y	Y	Y	Y
	AddSeconds	Y	Y	Y	Y
	AddMinutes	Y	Y	Y	Y
	AddHours	Y	Y	Y	Y
	AddDays	Y	Y	Y	Y
	AddMonths	Y	Y	Y	Y
	AddYears	Y	Y	Y	Y
	CreateDateTime	Y	Y	Y	Y
	CreateDateTimeOffset		Y		
	CurrentDateTimeOffset[1]		Y		
	CreateTime	Y	Y	Y	Y
	CurrentDateTime	Y	Y	Y	Y
	CurrentUtcDateTime	Y	Y	Y	
	Day	Y	Y	Y	Y
	DayOfYear	Y	Y	Y	Y
	DiffNanoseconds[1]	Y	Y	Y	Y*
	DiffMicroseconds[1]	Y	Y	Y	Y*
	DiffMilliseconds[1]	Y	Y	Y	Y*
	DiffSeconds[1]	Y	Y	Y	Y*
	DiffMinutes[1]	Y	Y	Y	Y*
	DiffHours[1]	Y	Y	Y	Y*
	DiffDays[1]	Y	Y	Y	Y*
	DiffMonths[1]	Y	Y	Y	Y*
	DiffYears[1]	Y	Y	Y	Y*
	GetTotalOffsetMinutes[1]		Y		
	Hour	Y	Y	Y	Y
	Millisecond	Y	Y	Y	Y
	Minute	Y	Y	Y	Y
	Month	Y	Y	Y	Y
	Second	Y	Y	Y	Y
	Truncate (datetime exp)	Y	Y	Y	Y
	Year	Y	Y	Y	Y

既知の制約

注:
正規関数の中には、サーバーにより提供される機能的なサポートに完全に依存しているものがあります。SQL0440N* エラーが発生した場合、エラー・メッセージで示されている機能をご使用のサーバーがサポートしていないことを示しています。サーバーがサポートしている機能に関する詳細情報については、IBM 技術サポートにお問い合わせください。

1 - この正規関数は、バージョン 9.7 フィックスパック 4 以降でのみサポートされています。

一般:

- データベースが最初のシナリオのみがサポートされます。つまり、Entity Framework が使用される前に、すべてのデータベース・オブジェクトが存在している必要があります。
- ストア固有の関数の呼び出しはサポートされていません。
- サーバー・エクスプローラーの「接続の追加」ダイアログで設定されたトラステッド・コンテキスト接続プロパティーは、Entity Framework 接続に渡されません。これは、サーバー・エクスプローラーに関する既知の制約です。

DB2 for z/OS:

- データ・タイプ REAL はサポートされていません。アプリケーションは、表のスキーマで FLOAT を使用するか、サーバーでの実際のタイプが REAL であっても、クライアント・スキーマ (EDM) でタイプを FLOAT として指定する必要があります。
- バージョン 8/バージョン 7 固有: Take/Top/First/Intersect/Except 式を含む照会で、ある種の構文エラーを示す例外が生成される場合があります。これらの式を含む照会の結果は未定義です。

  例:

```
1) var query = from o in context.Orders
where o.ShipCity == "Seattle"
select o;
var result = query.First();

2) var mexico =
context.OrderDetails.Where(od => od.Order.ShipCountry
   == "Mexico").Select(od => od.Product);
   var canada =
context.OrderDetails.Where(od => od.Order.ShipCountry
   == "Canada").Select(od => od.Product);
   var query = mexico.Intersect(canada);

3) var query =
context.Customers.Select(e => e).Except(context.Orders.Where
   (o => o.ShipCountry == "Mexico").Select(o => o.Customer));

4) var query = context.Orders.Include("OrderDetails").Top("1");

5) var query = context.Orders.Include("OrderDetails").
   Include("OrderDetails.Product").Take(3).Select(o => o);
```

IBM Informix サーバー:

- 相関副照会を持つ特定の照会で、「未実装」または「構文エラー」と類似の例外が生成される場合があります。典型的なシナリオには、次のようなものがあります。
  1. ページ送りを含む相関副照会。
  2. 相関副照会経由、またはナビゲーションにより生成されるコレクション経由のすべての要素、要素セレクターを受け入れるグループ化メソッドを使用するエンティティー・フレームワーク照会 (以下の例を参照)。
  3. REF 構成を経由する DEREF 構成を含む照会。

例:

```
var query = from p in context.Products
group p by p.Category.CategoryID into g
select new
{
g.Key,
ExpensiveProducts = from p2 in g where p2.UnitPrice >
g.Average(p3 => p3.UnitPrice)
select p2
};
```

注:
正規関数の中には、サーバーにより提供される機能的なサポートに完全に依存しているものがあります。SQL0440N* エラーが発生した場合、エラー・メッセージで示されている機能をご使用のサーバーがサポートしていないことを示しています。サーバーがサポートしている機能に関する詳細情報については、IBM 技術サポートにお問い合わせください。

Enterprise Library データ・アクセス・モジュールの使用

Enterprise Library は、開発者が開発作業で直面する一般的な問題に対処するのに役立つ、アプリケーション・ブロックの集まりです。アプリケーション・ブロックはソース・コードの形で提供され、そのまま使用することも開発プロジェクトに応じて修正することも可能です。

IBM データ・サーバーのための Enterprise Library データ・アクセス・モジュールは、その他のモジュールと共に http://codeplex.com/entlibcontrib/SourceControl/PatchList.aspx で入手できます。

IBM データ・サーバー (DB2、IDS、および U2) と共に Enterprise Library データ・アクセス・モジュールをインストールして使用する方法については、ダウンロードした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。

リソース

以下に、データ・アクセス・モジュールの使用方法を説明しているオンライン・リソースを紹介します。

- EntLib Contrib Project ホーム・ページ: http://www.codeplex.com/entlibcontrib
- patterns & practices for Enterprise Library: http://www.codeplex.com/entlib
- Microsoft Enterprise Library ホーム・ページ: http://msdn.microsoft.com/en-us/library/cc467894.aspx

- IBM DB2 for .NET: http://www.ibm.com/software/data/db2/windows/dotnet.html

.NET アプリケーションの構築

Visual Basic .NET アプリケーションの構築

DB2 製品には、DB2 Visual Basic .NET アプリケーションのコンパイルおよびリンク用の bldapp.bat バッチ・ファイルが用意されています。このバッチ・ファイルは、このファイルを使用して構築できるサンプル・プログラムと一緒に sqllib¥samples¥.NET¥vb ディレクトリーに置かれています。このバッチ・ファイルは 1 つのパラメーター %1 をとります。このパラメーターは、コンパイルするソース・ファイルの名前を (.vb 拡張を付けずに) 指定します。

このタスクについて

このタスクは、DbAuth サンプル・ファイルを使用し bldapp.bat を介して Visual Basic .NET アプリケーションを構築する基本ステップをたどります。

手順

ソース・ファイル DbAuth.vb からプログラム DbAuth を構築するには、次のように入力します。

```
bldapp DbAuth
```

実行可能ファイルを実行する際に必要なパラメーターを確実に指定するようにするには、以下のように入力する情報の数に応じてさまざまなパラメーターの組み合わせを指定できます。

1. パラメーターなし。次のように、プログラム名のみ入力します。

   ```
   DbAuth
   ```

2. 1 つのパラメーター。次のように、プログラム名に加えてデータベース別名を入力します。

   ```
   DbAuth <db_alias>
   ```

3. 2 つのパラメーター。次のように、プログラム名に加えてユーザー ID とパスワードを入力します。

   ```
   DbAuth <userid> <passwd>
   ```

4. 3 つのパラメーター。次のように、プログラム名に加えてデータベース別名、ユーザー ID、およびパスワードを入力します。

   ```
   DbAuth <db_alias> <userid> <passwd>
   ```

5. 4 つのパラメーター。次のように、プログラム名に加えてサーバー名、ポート番号、ユーザー ID、およびパスワードを入力します。

   ```
   DbAuth <server> <portnum> <userid> <passwd>
   ```

6. 5 つのパラメーター。次のように、プログラム名に加えてデータベース別名、サーバー名、ポート番号、ユーザー ID、およびパスワードを入力します。

   ```
   DbAuth <db_alias> <server> <portnum> <userid> <passwd>
   ```

次のタスク

LCTrans サンプル・プログラムを構築して実行するには、ソース・ファイル LCTrans.vb に示されている詳細な指示に従う必要があります。

C# .NET アプリケーションの構築

DB2 製品には、DB2 C# .NET アプリケーションのコンパイルおよびリンク用の bldapp.bat バッチ・ファイルが用意されています。このバッチ・ファイルは、sqllib\samples\.NET\cs ディレクトリーに入っています。そこには、このファイルでビルド可能なサンプル・プログラムも含まれています。

このバッチ・ファイルには 1 個のパラメーター %1 があります。それは、コンパイル対象のソース・ファイルの名前 (.cs 拡張子を除いたもの) を指定します。

このタスクについて

このタスクは、DbAuth サンプル・ファイルを使用し bldapp.bat を介して C# .NET アプリケーションを構築する基本ステップをたどります。

手順

ソース・ファイル DbAuth.cs からプログラム DbAuth を構築するには、次のように入力します。

```
bldapp DbAuth
```

実行可能ファイルを実行する際に必要なパラメーターを確実に指定するようにするには、以下のように入力する情報の数に応じてさまざまなパラメーターの組み合わせを指定できます。

1. パラメーターなし。次のように、プログラム名のみ入力します。

   ```
   DbAuth
   ```

2. 1 つのパラメーター。次のように、プログラム名に加えてデータベース別名を入力します。

   ```
   DbAuth <db_alias>
   ```

3. 2 つのパラメーター。次のように、プログラム名に加えてユーザー ID とパスワードを入力します。

   ```
   DbAuth <userid> <passwd>
   ```

4. 3 つのパラメーター。次のように、プログラム名に加えてデータベース別名、ユーザー ID、およびパスワードを入力します。

   ```
   DbAuth <db_alias> <userid> <passwd>
   ```

5. 4 つのパラメーター。次のように、プログラム名に加えてサーバー名、ポート番号、ユーザー ID、およびパスワードを入力します。

   ```
   DbAuth <server> <portnum> <userid> <passwd>
   ```

6. 5 つのパラメーター。次のように、プログラム名に加えてデータベース別名、サーバー名、ポート番号、ユーザー ID、およびパスワードを入力します。

   ```
   DbAuth <db_alias> <server> <portnum> <userid> <passwd>
   ```

次のタスク

LCTrans サンプル・プログラムを構築して実行するには、ソース・ファイル LCTrans.cs に示されている詳細な指示に従う必要があります。

Visual Basic .NET アプリケーションのコンパイルとリンクのオプション

このトピックは、Visual Basic .NET アプリケーションのコンパイルおよびリンク時に利用できるさまざまなオプションを説明します。

以下に示すコンパイルとリンクのオプションは、Windows 上で Microsoft Visual Basic .NET コンパイラーを使用して、Visual Basic .NET アプリケーションを構築するために使用できます。これらは、bldapp.bat バッチ・ファイルの中で示されているものです。

注: .NET Framework バージョン 1.1 は、.NET Provider バージョン 9.5 以前でのみサポートされます。

スタンドアロン VB .NET アプリケーションのコンパイルおよびリンクのオプション (bldapp を使用)

スタンドアロン VB .NET アプリケーションのコンパイルおよびリンクのオプション

%BLDCOMP%

コンパイラー用の変数です。デフォルトは vbc (Microsoft Visual Basic .NET コンパイラー) です。

/r:"%DB2PATH%"¥bin¥%VERSION%¥IBM.Data.DB2.dll

使用している .NET Framework のバージョンに対応する DB2 ダイナミック・リンク・ライブラリーを参照します。

%DB2PATH%

%DB2PATH% 変数はインストール済みの DB2 環境のルート・パスを表します。 %DB2PATH% 変数は、IBM Data Server Driver for ODBC and CLI または Data Server Driver Package インストール済み環境にはありません。 IBM IBM Data Server Driver for ODBC and CLI または Data Server Driver Package を使用している場合、ドライバー製品がインストールされているパスで %DB2PATH% を置き換えます。

%VERSION%

アプリケーション用にサポートされている .NET Framework のバージョンはいくつかあります。 DB2 には、各バージョンに対応するダイナミック・リンク・ライブラリーがあります。 .NET Framework バージョン 2.0、3.0、および 3.5 の場合、%VERSION% は netf20¥ サブディレクトリーを示します。

疎結合サンプル・プログラム LCTrans のコンパイルおよびリンク・オプション (bldapp を使用)

%BLDCOMP%
コンパイラー用の変数です。デフォルトは vbc (Microsoft Visual Basic .NET コンパイラー) です。

/out:RootCOM.dll
LCTrans アプリケーションが使用する RootCOM ダイナミック・リンク・ライブラリーを、RootCOM.vb ソース・ファイルから出力します。

/out:SubCOM.dll
LCTrans アプリケーションが使用する SubCOM ダイナミック・リンク・ライブラリーを、SubCOM.vb ソース・ファイルから出力します。

/target:library %1.cs
入力ソース・ファイル (RootCOM.vb または SubCOM.vb) からダイナミック・リンク・ライブラリーを作成します。

/r:System.EnterpriseServices.dll
Microsoft Windows の System EnterpriseServices データ・リンク・ライブラリーを参照します。

/r:"%DB2PATH%"¥bin¥%VERSION%¥IBM.Data.DB2.dll
使用している .NET Framework のバージョンに対応する DB2 ダイナミック・リンク・ライブラリーを参照します。

%DB2PATH%
%DB2PATH% 変数は DB2 製品のインストール済み環境のルート・パスを表します。 %DB2PATH% 変数は、IBM Data Server Driver for ODBC and CLI または Data Server Driver Package インストール済み環境にはありません。 IBM IBM Data Server Driver for ODBC and CLI または Data Server Driver Package を使用している場合、ドライバー製品がインストールされているパスで %DB2PATH% を置き換えます。

%VERSION%
アプリケーション用にサポートされている .NET Framework のバージョンはいくつかあります。 DB2 には、各バージョンに対応するダイナミック・リンク・ライブラリーがあります。 .NET Framework バージョン 2.0 および 3.0 の場合、%VERSION% は netf20¥ サブディレクトリーを示します。

/r:System.Data.dll
Microsoft Windows の System Data ダイナミック・リンク・ライブラリーを参照します。

/r:System.dll
Microsoft Windows の System ダイナミック・リンク・ライブラリーを参照します。

/r:System.Xml.dll
Microsoft Windows の System XML ダイナミック・リンク・ライブラリーを参照します (SubCOM.vb 用)。

/r:SubCOM.dll
SubCOM ダイナミック・リンク・ライブラリーを参照します (RootCOM.vb および LCTrans.vb 用)。

/r:RootCOM.dll
RootCOM ダイナミック・リンク・ライブラリーを参照します (LCTrans.vb 用)。

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。

C# .NET アプリケーションのコンパイルとリンクのオプション

このトピックでは、C# .NET アプリケーションをコンパイルおよびリンクする際に利用できるさまざまなオプションについて説明します。

Windows 上で Microsoft C# コンパイラーを使用して C# アプリケーションを構築する場合に DB2 で使用可能なコンパイルとリンクのオプション。これらは、bldapp.bat バッチ・ファイルの中で示されているものです。

注: .NET Framework バージョン 1.1 は、.NET Provider バージョン 9.5 以前でのみサポートされます。

スタンドアロン C# アプリケーションのコンパイルおよびリンクのオプション (bldapp を使用)

スタンドアロン C# アプリケーションのコンパイルおよびリンクのオプション

%BLDCOMP%
コンパイラー用の変数です。デフォルトは、csc (Microsoft C# コンパイラー) です。

/r:"%DB2PATH%"¥bin¥%VERSION%IBM.Data.DB2.dll
使用している .NET Framework のバージョンに対応する DB2 ダイナミック・リンク・ライブラリーを参照します。

%VERSION%
アプリケーション用にサポートされている .NET Framework のバージョンはいくつかあります。 DB2 には、その各バージョンに対応するダイナミック・リンク・ライブラリーがあります。 .NET Framework バージョン 2.0、3.0、および 3.5 の場合、%VERSION% は netf20¥ サブディレクトリーを示します。

疎結合サンプル・プログラム LCTrans のコンパイルおよびリンク・オプション (bldapp を使用)

%BLDCOMP%
コンパイラー用の変数です。デフォルトは、csc (Microsoft C# コンパイラー) です。

/out:RootCOM.dll
LCTrans アプリケーションが使用する RootCOM ダイナミック・リンク・ライブラリーを、RootCOM.cs ソース・ファイルから出力します。

/out:SubCOM.dll
LCTrans アプリケーションが使用する SubCOM ダイナミック・リンク・ライブラリーを、SubCOM.cs ソース・ファイルから出力します。

/target:library %1.cs
入力ソース・ファイル (RootCOM.cs または SubCOM.cs) からダイナミック・リンク・ライブラリーを作成します。

/r:System.EnterpriseServices.dll
Microsoft Windows の System EnterpriseServices データ・リンク・ライブラリーを参照します。

/r:"%DB2PATH%"¥bin¥%VERSION%IBM.Data.DB2.dll
使用している .NET Framework のバージョンに対応する DB2 ダイナミック・リンク・ライブラリーを参照します。

%VERSION%
アプリケーション用にサポートされている .NET Framework のバージョンはいくつかあります。 DB2 には、各バージョンに対応するダイナミック・リンク・ライブラリーがあります。 .NET Framework バージョン 2.0、3.0、および 3.5 の場合、%VERSION% は netf20¥ サブディレクトリーを示します。

/r:System.Data.dll
Microsoft Windows の System Data ダイナミック・リンク・ライブラリーを参照します。

/r:System.dll
Microsoft Windows の System ダイナミック・リンク・ライブラリーを参照します。

/r:System.Xml.dll
Microsoft Windows の System XML ダイナミック・リンク・ライブラリーを参照します (SubCOM.cs 用)。

/r:SubCOM.dll
SubCOM ダイナミック・リンク・ライブラリーを参照します (RootCOM.cs および LCTrans.cs 用)。

/r:RootCOM.dll
RootCOM ダイナミック・リンク・ライブラリーを参照します (LCTrans.cs 用)。

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。

第 5 章 IBM OLE DB Provider for DB2

IBM OLE DB Provider for DB2 を使用すれば、DB2 は OLE DB Provider のリソース・マネージャーとして機能できます。このサポートにより、OLE DB2 ベースのアプリケーションは、OLE インターフェースを使用して DB2 データの抽出や照会が可能です。

Microsoft OLE DB は、さまざまな情報ソースに保管されているデータに対し、単一アクセスをアプリケーションに提供する、OLE/COM インターフェースのセットです。OLE DB のアーキテクチャーでは、 OLE DB Consumer と OLE DB Provider を定義しています。 OLE DB Consumer は、 OLE DB インターフェースを使用するシステムまたはアプリケーションで、 OLE DB Provider は、OLE DB インターフェースを公開するコンポーネントです。

IBM OLE DB Provider for DB2 (Provider 名は IBMDADB2) を使用すれば、OLE DB Consumer は DB2 データベース・サーバー上のデータにアクセスできます。DB2 Connect がインストールされていれば、これらの OLE DB consumer は、DB2 for z/OS、DB2 Server for VM and VSE、または DB2 Universal Database for AS/400 などのホスト DBMS 上のデータにもアクセスすることができます。

IBM OLE DB Provider for DB2 には以下の機能が備わっています。

- OLE DB Provider 仕様のサポート・レベル 0。いくつかの付加的なレベル 1 インターフェースが含まれます。
- フリー・スレッド Provider のインプリメンテーション。アプリケーションは、1 つのスレッドで作成したコンポーネントを他の任意のスレッドで使用できます。
- DB2 エラー・メッセージを戻すエラー検索サービス。

IBM OLE DB Provider はクライアントに存在し、OLE DB 表関数 (これも DB2 データベース・システムでサポートされる) とは異なるものであることに注意してください。

本書の以下の節では、IBM OLE DB Provider for DB2 固有のインプリメンテーションについて説明します。Microsoft OLE DB 2.0 仕様の詳細については Microsoft Press から出版されている「Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK」を参照してください。

準拠バージョン

IBM OLE DB Provider for DB2 は、Microsoft OLE DB 仕様のバージョン 2.7 以降に準拠しています。

システム要件

サポートされている Windows オペレーティング・システムについては、IBM OLE DB Provider for DB2 データ・サーバーに関するアナウンス・レターを参照してください。

IBM OLE DB Provider for DB2 をインストールするには、上記のサポートされているオペレーティング・システムのいずれかをまず実行している必要があります。さらに DB2 製品全体、IBM Data Server Driver for ODBC and CLI、または IBM Data Server Driver Package をインストールする必要があります。

IBM OLE DB Provider for DB2 でサポートされているアプリケーション・タイプ

IBM OLE DB Provider for DB2 を使用して、以下のタイプのアプリケーションを作成できます。

- ADO アプリケーション。以下のものが含まれます。
  - Microsoft Visual Studio C++ アプリケーション
  - Microsoft Visual Basic アプリケーション
- OLE DB .NET Data Provider を使用する ADO.NET アプリケーション
- OLE DB インターフェースを使用して IBMDADB2 に直接アクセスする C/C++ アプリケーション。 Data Access Consumer Object が ATL COM AppWizard によって生成される ATL アプリケーションが含まれます。

OLE DB サービス

IBM OLE DB Provider でサポートされているスレッド・モデル

IBM OLE DB Provider for DB2 では、フリー・スレッド・モデルがサポートされています。このモデルでは、アプリケーションが、1 つのスレッドで作成したコンポーネントを他の任意のスレッドで使用できます。

IBM OLE DB Provider によるラージ・オブジェクトの操作

IBMDADB2 プロバイダーでデータをストレージ・オブジェクト (DBTYPE_IUNKNOWN) として取得したり設定したりするには、以下のように ISequentialStream インターフェースを使用します。

- ストレージ・オブジェクトをパラメーターにバインドするには、DBBINDING 構造の DBOBJECT で、dwFlag フィールドに値 STGM_READ だけを含めることができます。IBMDADB2 では、バインド済みオブジェクトの ISequentialStream インターフェースの Read メソッドが実行されます。
- ストレージ・オブジェクトからデータを取得するには、アプリケーションがそのストレージ・オブジェクトの ISequentialStream インターフェースで Read メソッドを実行する必要があります。
- データを取得したとき、長さ部分の値は実データの長さであり、IUnknown ポインターの長さではありません。

IBM OLE DB Provider でサポートされているスキーマ行セット

以下の表は、IDBSchemaRowset によってサポートされているスキーマ行セットを示しています。行セットのうちサポートされていない列は NULL に設定されます。

表 12. *IBM OLE DB Provider for DB2* でサポートされているスキーマ行セット

サポートされている GUID	サポートされている制限	サポートされている列	注
DBSCHEMA _COLUMN_PRIVILEGES	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	COLUMN_NAME GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_COLUMNS	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH COLUMN_DEFAULT COLUMN_FLAGS COLUMN_HASDEFAULT COLUMN_NAME DATA_TYPE DESCRIPTION IS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_FOREIGN_KEYS	FK_TABLE_NAME FK_TABLE_SCHEMA PK_TABLE_NAME PK_TABLE_SCHEMA	DEFERRABILITY DELETE_RULE FK_COLUMN_NAME FK_NAME FK_TABLE_NAME FK_TABLE_SCHEMA ORDINAL PK_COLUMN_NAME PK_NAME PK_TABLE_NAME PK_TABLE_SCHEMA UPDATE_RULE	PK_TABLE_NAME または FK_TABLE_NAME の制限のうち少なくとも 1 つを指定する必要があります。 “%” ワイルドカードは使用できません。
DBSCHEMA_INDEXES	TABLE_NAME TABLE_SCHEMA	CARDINALITY CLUSTERED COLLATION COLUMN_NAME INDEX_NAME INDEX_SCHEMA ORDINAL_POSITION PAGES TABLE_NAME TABLE_SCHEMA TYPE UNIQUE	ソート順序はサポートされていません。指定されているソート順序は無視されます。

表 12. *IBM OLE DB Provider for DB2* でサポートされているスキーマ行セット *(続き)*

サポートされている GUID	サポートされている制限	サポートされている列	注
DBSCHEMA_PRIMARY_KEYS	TABLE_NAME TABLE_SCHEMA	COLUMN_NAME ORDINAL PK_NAME TABLE_NAME TABLE_SCHEMA	少なくとも TABLE_NAME の制限を指定する必要があります。 “%” ワイルドカードは使用できません。
DBSCHEMA _PROCEDURE_PARAMETERS	PARAMETER_NAME PROCEDURE_NAME PROCEDURE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH DATA_TYPE DESCRIPTION IS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION PARAMETER_DEFAULT PARAMETER_HASDEFAULT PARAMETER_NAME PARAMETER_TYPE PROCEDURE_NAME PROCEDURE_SCHEMA TYPE_NAME	
DBSCHEMA_PROCEDURES	PROCEDURE_NAME PROCEDURE_SCHEMA	DESCRIPTION PROCEDURE_NAME PROCEDURE_SCHEMA PROCEDURE_TYPE	
DBSCHEMA_PROVIDER_TYPES	DATA_TYPE BEST_MATCH	AUTO_UNIQUE_VALUE BEST_MATCH CASE_SENSITIVE CREATE_PARAMS COLUMN_SIZE DATA_TYPE FIXED_PREC_SCALE IS_FIXEDLENGTH IS_LONG IS_NULLABLE LITERAL_PREFIX LITERAL_SUFFIX LOCAL_TYPE_NAME MINIMUM_SCALE MAXIMUM_SCALE SEARCHABLE TYPE_NAME UNSIGNED_ATTRIBUTE	
DBSCHEMA_STATISTICS	TABLE_NAME TABLE_SCHEMA	CARDINALITY TABLE_NAME TABLE_SCHEMA	ソート順序はサポートされていません。指定されているソート順序は無視されます。

表 12. IBM OLE DB Provider for DB2 でサポートされているスキーマ行セット (続き)

サポートされている GUID	サポートされている制限	サポートされている列	注
DBSCHEMA _TABLE_PRIVILEGES	TABLE_NAME TABLE_SCHEMA	GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_TABLES	TABLE_NAME TABLE_SCHEMA TABLE_TYPE	DESCRIPTION TABLE_NAME TABLE_SCHEMA TABLE_TYPE	

IBM OLE DB Provider で自動的に使用可能になる OLE DB サービス

デフォルトでは、IBM OLE DB Provider for DB2 は、プロバイダーのクラス ID (CLSID) の下に DWORD 値が 0xFFFFFFFF であるレジストリー項目 OLEDB_SERVICES を追加することにより、すべての OLE DB サービスを自動的に有効にします。この値の意味は以下のとおりです。

表 13. OLE DB サービス

使用可能になっているサービス	DWORD 値
すべてのサービス (デフォルト)	0xFFFFFFFF
プールおよび AutoEnlistment を除くすべて	0xFFFFFFFC
クライアント・カーソルを除くすべて	0xFFFFFFFB
プール、エンリスト、およびカーソルを除くすべて	0xFFFFFFF8
サービスなし	0x000000000

データ・サービス

IBM OLE DB Provider でサポートされているカーソル・モード

IBM OLE DB Provider for DB2 では、読み取り専用カーソル、前方スクロール・カーソル、読み取り専用の両方向スクロール・カーソル、更新可能な両方向スクロール・カーソルがネイティブでサポートされています。

DB2 と OLE DB の間のデータ・タイプ・マッピング

IBM OLE DB Provider for DB2 では、DB2 データ・タイプと OLE DB データ・タイプの間のデータ・タイプ・マッピングがサポートされています。

以下の表は、サポートされているマッピングと使用可能な名前の完全なリストを示すことにより、列およびパラメーターのデータ・タイプを示しています。

表 14. DB2 データ・タイプと OLE DB データ・タイプの間のデータ・タイプ・マッピング

DB2 データ・タイプ	OLE DB データ・タイプ標識	OLE DB 標準タイプ名	DB2 固有の名前
SMALLINT	DBTYPE_I2	“DBTYPE_I2”	“SMALLINT”
INTEGER	DBTYPE_I4	“DBTYPE_I4”	“INTEGER”または “INT”
BIGINT	DBTYPE_I8	“DBTYPE_I8”	“BIGINT”
REAL	DBTYPE_R4	“DBTYPE_R4”	“REAL”
FLOAT	DBTYPE_R8	“DBTYPE_R8”	“FLOAT”
DOUBLE	DBTYPE_R8	“DBTYPE_R8"	“DOUBLE”または “DOUBLE PRECISION”
DECIMAL	DBTYPE_NUMERIC	“DBTYPE_NUMERIC”	“DEC”または “DECIMAL”
NUMERIC	DBTYPE_NUMERIC	“DBTYPE_NUMERIC”	“NUM” または “NUMERIC”
DATE	DBTYPE_DBDATE	“DBTYPE_DBDATE”	“DATE”
TIME	DBTYPE_DBTIME	“DBTYPE_DBTIME”	“TIME”
TIMESTAMP	DBTYPE_DBTIMESTAMP	“DBTYPE_DBTIMESTAMP”	“TIMESTAMP”
CHAR	DBTYPE_STR	“DBTYPE_CHAR”	“CHAR” または “CHARACTER”
VARCHAR	DBTYPE_STR	“DBTYPE_VARCHAR”	“VARCHAR”
LONG VARCHAR	DBTYPE_STR	“DBTYPE_LONGVARCHAR”	“LONG VARCHAR”
CLOB	DBTYPE_STR および DBCOLUMNFLAGS_ISLONG または DBPARAMFLAGS_ISLONG	“DBTYPE_CHAR” “DBTYPE_VARCHAR” “DBTYPE_LONGVARCHAR” および DBCOLUMNFLAGS_ISLONG または DBPARAMFLAGS_ISLONG	“CLOB”
GRAPHIC	DBTYPE_WSTR	“DBTYPE_WCHAR”	“GRAPHIC”
VARGRAPHIC	DBTYPE_WSTR	“DBTYPE_WVARCHAR”	“VARGRAPHIC”
LONG VARGRAPHIC	DBTYPE_WSTR	“DBTYPE_WLONGVARCHAR”	“LONG VARGRAPHIC”
DBCLOB	DBTYPE_WSTR および DBCOLUMNFLAGS_ISLONG または DBPARAMFLAGS_ISLONG	“DBTYPE_WCHAR” “DBTYPE_WVARCHAR” “DBTYPE_WLONGVARCHAR” および DBCOLUMNFLAGS_ISLONG または DBPARAMFLAGS_ISLONG	“DBCLOB”
CHAR(n) FOR BIT DATA	DBTYPE_BYTES	“DBTYPE_BINARY”	
VARCHAR(n) FOR BIT DATA	DBTYPE_BYTES	“DBTYPE_VARBINARY”	

表 14. DB2 データ・タイプと OLE DB データ・タイプの間のデータ・タイプ・マッピング (続き)

DB2 データ・タイプ	OLE DB データ・タイプ標識	OLE DB 標準タイプ名	DB2 固有の名前
LONG VARCHAR FOR BIT DATA	DBTYPE_BYTES	“DBTYPE_LONGVARBINARY”	
BLOB	DBTYPE_BYTES および DBCOLUMNFLAGS_ISLONG または DBPARAMFLAGS_ISLONG	“DBTYPE_BINARY” “DBTYPE_VARBINARY” “DBTYPE_LONGVARBINARY” および DBCOLUMNFLAGS_ISLONG または DBPARAMFLAGS_ISLONG	“BLOB”

OLE DB タイプから DB2 タイプにデータを設定するためのデータ変換

IBM OLE DB Provider for DB2 では、OLE DB タイプから DB2 タイプにデータを設定するためのデータ変換がサポートされています。

OLE DB タイプから DB2 タイプへのサポートされているデータ変換

以下の表に、OLE DB タイプから DB2 タイプへのデータ変換を示します。データのタイプや値に応じて、一部のケースではデータの切り捨てが起こる場合があることに注意してください。

表 15. OLE DB タイプから DB2 タイプへのデータ変換

	DB2 データ・タイプ																					
																		ビット・データの場合				
OLE DB タイプ標識	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_EMPTY																						
DBTYPE_NULL																						
DBTYPE_RESERVED																						

表 15. OLE DB タイプから DB2 タイプへのデータ変換 (続き)

OLE DB タイプ標識	DB2 データ・タイプ																					
																		ビット・データの場合				
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_I1	X	X	X	X	X	X				X	X											
DBTYPE_I2	X	X	X	X	X	X				X	X											
DBTYPE_I4	X	X	X	X	X	X				X	X											
DBTYPE_I8	X	X	X	X	X	X				X	X											
DBTYPE_UI1	X	X	X	X	X	X				X	X											
DBTYPE_UI2	X	X	X	X	X	X				X	X											
DBTYPE_UI4	X	X	X	X	X	X				X	X											
DBTYPE_UI8	X	X	X	X	X	X				X	X											
DBTYPE_R4	X	X	X	X	X	X				X	X											
DBTYPE_R8	X	X	X	X	X	X				X	X											
DBTYPE_CY																						
DBTYPE_DECIMAL	X	X	X	X	X	X				X	X											
DBTYPE_NUMERIC	X	X	X	X	X	X				X	X											
DBTYPE_DATE																						
DBTYPE_BOOL	X	X	X	X	X	X				X	X											
DBTYPE_BYTES			X			X				X	X	X				X		X	X	X		

表 15. OLE DB タイプから DB2 タイプへのデータ変換 (続き)

OLE DB タイプ標識	DB2 データ・タイプ																					
																		ビット・データの場合				
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_BSTR - 未定																						
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_WSTR														X	X	X						
DBTYPE_VARIANT - 未定																						
DBTYPE_IDISPATCH																						
DBTYPE_IUNKNOWN										X	X	X	X	X	X	X	X	X	X	X		
DBTYPE_GUID																						
DBTYPE_ERROR																						
DBTYPE_BYREF																						
DBTYPE_ARRAY																						
DBTYPE_VECTOR																						
DBTYPE_UDT																						
DBTYPE_DBDATE							X		X	X	X											
DBTYPE_DBTIME								X	X	X	X											
DBTYPE_DBTIMESTAMP							X	X	X	X	X											

表 15. OLE DB タイプから DB2 タイプへのデータ変換 (続き)

OLE DB タイプ標識	DB2 データ・タイプ																					
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	ビット・データの場合			BLOB	DATALINK
																		CHAR	VARCHAR	LONG VARCHAR		
DBTYPE_FILETIME																						
DBTYPE_PROP_VARIANT																						
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

DB2 タイプから OLE DB タイプにデータを設定するためのデータ変換

データを取得するため、IBM OLE DB Provider では DB2 タイプから OLE DB タイプへのデータ変換を行うことができます。

DB2 タイプから OLE DB タイプへのサポートされているデータ変換

以下の表は、DB2 タイプから OLE DB タイプへのサポートされるデータ変換を示します。データのタイプや値に応じて、一部のケースではデータの切り捨てが起こる場合があることに注意してください。

表 16. DB2 タイプから OLE DB タイプへのデータ変換

OLE DB タイプ標識	DB2 データ・タイプ																					
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	ビット・データの場合			BLOB	DATALINK
																		CHAR	VARCHAR	LONG VARCHAR		
DBTYPE_EMPTY																						
DBTYPE_NULL																						
DBTYPE_RESERVED																						
DBTYPE_I1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R8	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_CY	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_DECIMAL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_NUMERIC	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X

表 16. DB2 タイプから OLE DB タイプへのデータ変換 (続き)

OLE DB タイプ標識	DB2 データ・タイプ																				
																		ビット・データの場合			
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_DATE	X	X		X	X		X	X	X	X	X	X		X	X	X						X
DBTYPE_BOOL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_BYTES	X	X		X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_BSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_WSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_VARIANT	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_IDISPATCH																						
DBTYPE_IUNKNOWN	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
DBTYPE_GUID										X	X	X		X	X	X		X	X	X		X
DBTYPE_ERROR																						
DBTYPE_BYREF																						
DBTYPE_ARRAY																						
DBTYPE_VECTOR																						
DBTYPE_UDT																						
DBTYPE_DBDATE							X	X	X	X	X	X		X	X	X		X	X	X		X

表 16. DB2 タイプから OLE DB タイプへのデータ変換 (続き)

OLE DB タイプ標識	DB2 データ・タイプ																					
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	ビット・データの場合 CHAR	ビット・データの場合 VARCHAR	ビット・データの場合 LONG VARCHAR	BLOB	DATALINK
DBTYPE_DBTIME							X	X	X	X	X	X		X	X	X						X
DBTYPE_DBTIMESTAMP							X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_FILETIME			X				X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_PROP_VARIANT	X	X	X	X	X					X	X	X		X	X	X		X	X	X		X
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

注: アプリケーションが ISequentialStream::Read を実行してストレージ・オブジェクトからデータを取得するとき、戻されるデータのフォーマットは列のデータ・タイプによって決まります。

- 文字でないデータおよびバイナリー・データ・タイプの場合、列のデータはそのオペレーティング・システムでその値を表すバイトのシーケンスとして表されます。
- 文字データ・タイプの場合、データは最初に DBTYPE_STR に変換されます。
- DBCLOB の場合、データは最初に DBTYPE_WCHAR に変換されます。

IBM OLE DB Provider の制約事項

IBM OLE DB Provider の制限は、以下のとおりです。

- IBMDADB2 では、ITransactionLocal インターフェースにより、自動コミットおよびユーザー制御のトランザクション・スコープがサポートされています。デフォルトのスコープは自動コミット・トランザクション・スコープです。ネストされたトランザクションはサポートされていません。
- RestartPosition は、コマンド・テキストにパラメーターが含まれている場合はサポートされません。

- IBMDADB2 では、DBID パラメーター (IOpenRowset インターフェースで使用されるパラメーター) を介して渡される表名が引用符で囲まれません。その代わりに、引用符が必要な場合に OLE DB Consumer で表名に引用符を追加する必要があります。

IBM OLE DB Provider での OLE DB コンポーネントおよびインターフェースのサポート

以下の表は、IBM OLE DB Provider for DB2 および Microsoft OLE DB Provider for ODBC でサポートされている OLE DB コンポーネントおよびインターフェースを示しています。

表 17. BLOB

インターフェース	DB2	ODBC Provider
ISequentialStream	はい	はい

表 18. コマンド

インターフェース	DB2	ODBC Provider
IAccessor	はい	はい
ICommand	はい	はい
ICommandPersist	いいえ	いいえ
ICommandPrepare	はい	はい
ICommandProperties	はい	はい
ICommandText	はい	はい
ICommandWithParameters	はい	はい
IColumnsInfo	はい	はい
IColumnsRowset	はい	はい
IConvertType	はい	はい
ISupportErrorInfo	はい	はい

表 19. データ・ソース

インターフェース	DB2	ODBC Provider
IConnectionPoint	いいえ	はい
IDBAsynchNotify (Consumer)	いいえ	いいえ
IDBAsynchStatus	いいえ	いいえ
IDBConnectionPointContainer	いいえ	はい
IDBCreateSession	はい	はい
IDBDataSourceAdmin	いいえ	いいえ
IDBInfo	はい	はい
IDBInitialize	はい	はい
IDBProperties	はい	はい
IPersist	はい	いいえ
IPersistFile	はい	はい

表 19. データ・ソース (続き)

インターフェース	DB2	ODBC Provider
ISupportErrorInfo	はい	はい

表 20. 列挙子

インターフェース	DB2	ODBC Provider
IDBInitialize	はい	はい
IDBProperties	はい	はい
IParseDisplayName	はい	いいえ
ISourcesRowset	はい	はい
ISupportErrorInfo	はい	はい

表 21. エラー検索サービス

インターフェース	DB2	ODBC Provider
IErrorLookUp	はい	はい

表 22. エラー・オブジェクト

インターフェース	DB2	ODBC Provider
IErrorInfo	はい	いいえ
ISQLErrorInfo (カスタム)	はい	いいえ

表 23. 複数結果

インターフェース	DB2	ODBC Provider
IMultipleResults	はい	はい
ISupportErrorInfo	はい	はい

表 24. 行セット

インターフェース	DB2	ODBC Provider
IAccessor	はい	はい
IColumnsRowset	はい	はい
IColumnsInfo	はい	はい
IConvertType	はい	はい
IChapteredRowset	いいえ	いいえ
IConnectionPointContainer	はい	はい
IDBAsynchStatus	いいえ	いいえ
IParentRowset	いいえ	いいえ
IRowset	はい	はい
IRowsetChange	はい	はい
IRowsetChapterMember	いいえ	いいえ
IRowsetFind	いいえ	いいえ
IRowsetIdentity	はい	はい
IRowsetIndex	いいえ	いいえ

表 24. 行セット (続き)

インターフェース	DB2	ODBC Provider
IRowsetInfo	はい	はい
IRowsetLocate	はい	はい
IRowsetNotify (Consumer)	はい	いいえ
IRowsetRefresh	Cursor Service Component	はい
IRowsetResynch	Cursor Service Component	はい
IRowsetScroll	はい[1]	はい
IRowsetUpdate	Cursor Service Component	はい
IRowsetView	いいえ	いいえ
ISupportErrorInfo	はい	はい
注:		
1. 戻される値は近似値です。削除された行はスキップされません。		

表 25. セッション

インターフェース	DB2	ODBC Provider
IAlterIndex	いいえ	いいえ
IAlterTable	いいえ	いいえ
IDBCreateCommand	はい	はい
IDBSchemaRowset	はい	はい
IGetDataSource	はい	はい
IIndexDefinition	いいえ	いいえ
IOpenRowset	はい	はい
ISessionProperties	はい	はい
ISupportErrorInfo	はい	はい
ITableDefinition	いいえ	いいえ
ITableDefinitionWithConstraints	いいえ	いいえ
ITransaction	はい	はい
ITransactionJoin	はい	はい
ITransactionLocal	はい	はい
ITransactionObject	いいえ	いいえ
ITransactionOptions	いいえ	はい

表 26. ビュー・オブジェクト

インターフェース	DB2	ODBC Provider
IViewChapter	いいえ	いいえ
IViewFilter	いいえ	いいえ
IViewRowset	いいえ	いいえ
IViewSort	いいえ	いいえ

IBM OLE DB Provider での OLE DB プロパティーのサポート

以下の表は、IBM OLE DB Provider for DB2 でサポートされている OLE DB プロパティーを示しています。

表 27. IBM OLE DB Provider for DB2 でサポートされているプロパティー: データ・ソース (DBPROPSET_DATASOURCE)

プロパティー	デフォルト値	R/W
DBPROP_MULTIPLECONNECTIONS	VARIANT_FALSE	R
DBPROP_RESETDATASOURCE	DBPROPVAL_RD_RESETALL	R/W

表 28. IBM OLE DB Provider for DB2 でサポートされているプロパティー: DB2 データ・ソース (DBPROPSET_DB2DATASOURCE)

プロパティー	デフォルト値	R/W
DB2PROP_REPORTISLONGFORLONGTYPES	VARIANT_FALSE	R/W
DB2PROP_RETURNCHARASWCHAR	VARIANT_TRUE	R/W
DB2PROP_SORTBYORDINAL	VARIANT_FALSE	R/W

表 29. IBM OLE DB Provider for DB2 でサポートされているプロパティー: データ・ソース情報 (DBPROPSET_DATASOURCEINFO)

プロパティー	デフォルト値	R/W
DBPROP_ACTIVESESSIONS	0	R
DBPROP_ASYNCTXNABORT	VARIANT_FALSE	R
DBPROP_ASYNCTXNCOMMIT	VARIANT_FALSE	R
DBPROP_BYREFACCESSORS	VARIANT_FALSE	R
DBPROP_COLUMNDEFINITION	DBPROPVAL_CD_NOTNULL	R
DBPROP_CONCATNULLBEHAVIOR	DBPROPVAL_CB_NULL	R
DBPROP_CONNECTIONSTATUS	DBPROPVAL_CS_INITIALIZED	R
DBPROP_DATASOURCENAME	なし	R
DBPROP_DATASOURCEREADONLY	VARIANT_FALSE	R
DBPROP_DBMSNAME	なし	R
DBPROP_DBMSVER	なし	R
DBPROP_DSOTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R
DBPROP_GROUPBY	DBPROPVAL_GB_CONTAINS_SELECT	R
DBPROP_IDENTIFIERCASE	DBPROPVAL_IC_UPPER	R
DBPROP_MAXINDEXSIZE	0	R
DBPROP_MAXROWSIZE	0	R
DBPROP_MAXROWSIZEINCLUDESBLOB	VARIANT_TRUE	R
DBPROP_MAXTABLEINSELECT	0	R
DBPROP_MULTIPLEPARAMSETS	VARIANT_FALSE	R
DBPROP_MULTIPLERESULTS	DBPROPVAL_MR_SUPPORTED	R
DBPROP_MULTIPLESTORAGEOBJECTS	VARIANT_TRUE	R
DBPROP_MULTITABLEUPDATE	VARIANT_FALSE	R

表 29. IBM OLE DB Provider for DB2 でサポートされているプロパティー: データ・ソース情報 (DBPROPSET_DATASOURCEINFO) (続き)

プロパティー	デフォルト値	R/W
DBPROP_NULLCOLLATION	DBPROPVAL_NC_LOW	R
DBPROP_OLEOBJECTS	DBPROPVAL_OO_BLOB	R
DBPROP_ORDERBYCOLUMNSINSELECT	VARIANT_FALSE	R
DBPROP _OUTPUTPARAMETERAVAILABILITY	DBPROPVAL_OA_ATEXECUTE	R
DBPROP_PERSISTENTIDTYPE	DBPROPVAL_PT_NAME	R
DBPROP_PREPAREABORTBEHAVIOR	DBPROPVAL_CB_DELETE	R
DBPROP_PROCEDURETERM	"STORED PROCEDURE"	R
DBPROP_PROVIDERFRIENDLYNAME	"IBM OLE DB Provider for DB2"	R
DBPROP_PROVIDERNAME	"IBMDADB2.DLL"	R
DBPROP_PROVIDEROLEDBVER	"02.7"	R
DBPROP_PROVIDERVER	なし	R
DBPROP_QUOTEIDENTIFIERCASE	DBPROPVAL_IC_SENSITIVE	R
DBPROP _ROWSETCONVERSIONSONCOMMAND	VARIANT_TRUE	R
DBPROP_SCHEMATERM	"SCHEMA"	R
DBPROP_SCHEMAUSAGE	DBPROPVAL_SU_DML_STATEMENTS | DBPROPVAL_SU_TABLE_DEFINITION | DBPROPVAL_SU_INDEX_DEFINITION | DBPROPVAL_SU_PRIVILEGE_DEFINITION	R
DBPROP_SQLSUPPORT	DBPROPVAL_SQL_ODBC_EXTENDED | DBPROPVAL_SQL_ESCAPECLAUSES | DBPROPVAL_SQL_ANSI92_ENTRY	R
DBPROP_SERVERNAME	なし	R
DBPROP_STRUCTUREDSTORAGE	DBPROPVAL_SS_ISEQUENTIALSTREAM	R
DBPROP_SUBQUERIES	DBPROPVAL_SQ_CORRELATEDSUBQUERIES | DBPROPVAL_SQ_COMPARISON | DBPROPVAL_SQ_EXISTS | DBPROPVAL_SQ_IN | DBPROPVAL_SQ_QUANTIFIED |	R
DBPROP_SUPPORTEDTXNDDL	DBPROPVAL_TC_ALL	R
DBPROP_SUPPORTEDTXNISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY | DBPROPVAL_TI_READCOMMITTED | DBPROPVAL_TI_READUNCOMMITTED | DBPROPVAL_TI_SERIALIZABLE |	R
DBPROP_SUPPORTEDTXNISORETAIN	DBPROPVAL_TR_COMMIT_DC | DBPROPVAL_TR_ABORT_NO |	R

表 29. IBM OLE DB Provider for DB2 でサポートされているプロパティー: データ・ソース情報 (DBPROPSET_DATASOURCEINFO) (続き)

プロパティー	デフォルト値	R/W
DBPROP_TABLETERM	"TABLE"	R
DBPROP_USERNAME	なし	R

表 30. IBM OLE DB Provider for DB2 でサポートされているプロパティー: 初期化 (DBPROPSET_DBINIT)

プロパティー	デフォルト値	R/W
DBPROP_AUTH_PASSWORD	なし	R/W
DBPROP_INIT_TIMEOUT (1)	0	R/W
DBPROP_AUTH_PERSIST _SENSITIVE_AUTHINFO	VARIANT_FALSE	R/W
DBPROP_AUTH_USERID	なし	R/W
DBPROP_INIT_DATASOURCE	なし	R/W
DBPROP_INIT_HWND	なし	R/W
DBPROP_INIT_MODE	DB_MODE_READWRITE	R/W
DBPROP_INIT_OLEDBSERVICES	0xFFFFFFFF	R/W
DBPROP_INIT_PROMPT	DBPROMPT_NOPROMPT	R/W
DBPROP_INIT_PROVIDERSTRING	なし	R/W

表 31. IBM OLE DB Provider for DB2 でサポートされているプロパティー: 行セット (DBPROPSET_ROWSET)

プロパティー	デフォルト値	R/W
DBPROP_ABORTPRESERVE	VARIANT_FALSE	R
DBPROP_ACCESSORDER	DBPROPVAL_AO_RANDOM	R
DBPROP_BLOCKINGSTORAGEOBJECTS	VARIANT_FALSE	R
DBPROP_BOOKMARKS	VARIANT_FALSE	R/W
DBPROP_BOOKMARKSKIPPED	VARIANT_FALSE	R
DBPROP_BOOKMARKTYPE	DBPROPVAL_BMK_NUMERIC	R
DBPROP_CACHEDEFERRED	VARIANT_FALSE	R/W
DBPROP_CANFETCHBACKWARDS	VARIANT_FALSE	R/W
DBPROP_CANHOLDROWS	VARIANT_FALSE	R
DBPROP_CANSCROLLBACKWARDS	VARIANT_FALSE	R/W
DBPROP_CHANGEINSERTEDROWS	VARIANT_FALSE	R
DBPROP_COMMITPRESERVE	VARIANT_TRUE	R/W
DBPROP_COMMANDTIMEOUT	0	R/W
DBPROP_DEFERRED	VARIANT_FALSE	R
DBPROP_IAccessor	VARIANT_TRUE	R
DBPROP_IColumnsInfo	VARIANT_TRUE	R
DBPROP_IColumnsRowset	VARIANT_TRUE	R/W
DBPROP_IConvertType	VARIANT_TRUE	R
DBPROP_IMultipleResults	VARIANT_FALSE	R/W
DBPROP_IRowset	VARIANT_TRUE	R

表 31. IBM OLE DB Provider for DB2 でサポートされているプロパティー: 行セット (DBPROPSET_ROWSET) (続き)

プロパティー	デフォルト値	R/W
DBPROP_IRowChange	VARIANT_FALSE	R/W
DBPROP_IRowsetFind	VARIANT_FALSE	R
DBPROP_IRowsetIdentity	VARIANT_TRUE	R
DBPROP_IRowsetInfo	VARIANT_TRUE	R
DBPROP_IRowsetLocate	VARIANT_FALSE	R/W
DBPROP_IRowsetScroll	VARIANT_FALSE	R/W
DBPROP_IRowsetUpdate	VARIANT_FALSE	R
DBPROP_ISequentialStream	VARIANT_TRUE	R
DBPROP_ISupportErrorInfo	VARIANT_TRUE	R
DBPROP_LITERALBOOKMARKS	VARIANT_FALSE	R
DBPROP_LITERALIDENTITY	VARIANT_TRUE	R
DBPROP_LOCKMODE	DBPROPVAL_LM_SINGLEROW	R/W
DBPROP_MAXOPENROWS	32767	R
DBPROP_MAXROWS	0	R/W
DBPROP_NOTIFICATIONGRANULARITY	DBPROPVAL_NT_SINGLEROW	R/W
DBPROP_NOTIFICATION PHASES	DBPROPVAL_NP_OKTODO DBPROPBAL_NP_ABOUTTODO DBPROPVAL_NP_SYNCHAFTER DBPROPVAL_NP_FAILEDTODO DBPROPVAL_NP_DIDEVENT	R
DBPROP_NOTIFYROWSETRELEASE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP _NOTIFYROWSETFETCHPOSITIONCHANGE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP_NOTIFYCOLUMNSET	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP_NOTIFYROWDELETE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP_NOTIFYROWINSERT	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP_ORDEREDBOOKMARKS	VARIANT_FALSE	R
DBPROP_OTHERINSERT	VARIANT_FALSE	R
DBPROP_OTHERUPDATEDELETE	VARIANT_FALSE	R/W
DBPROP_OWNINSERT	VARIANT_FALSE	R
DBPROP_OWNUPDATEDELETE	VARIANT_FALSE	R
DBPROP_QUICKRESTART	VARIANT_FALSE	R/W
DBPROP_REMOVEDELETED	VARIANT_FALSE	R/W
DBPROP_ROWTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R

表 31. IBM OLE DB Provider for DB2 でサポートされているプロパティー: 行セット (DBPROPSET_ROWSET) (続き)

プロパティー	デフォルト値	R/W
DBPROP_SERVERCURSOR	VARIANT_TRUE	R
DBPROP_SERVERDATAONINSERT	VARIANT_FALSE	R
DBPROP_UNIQUEROWS	VARIANT_FALSE	R/W
DBPROP_UPDATABILITY	0	R/W

表 32. IBM OLE DB Provider for DB2 でサポートされているプロパティー: DB2 行セット (DBPROPSET_DB2ROWSET)

プロパティー	デフォルト値	R/W
DBPROP_ISLONGMINLENGTH	32000	R/W

表 33. IBM OLE DB Provider for DB2 でサポートされているプロパティー: セッション (DBPROPSET_SESSION)

プロパティー	デフォルト値	R/W
DBPROP_SESS_AUTOCOMMITISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY	R/W

注:

1. TCP/IP プロトコルを使用してサーバーに接続する場合にのみ、タイムアウトを適用できます。タイムアウトは TCP/IP sock 接続時にのみ強制されます。指定されたタイムアウトの有効期限が切れる前に sock 接続が完了する場合、初期化プロセスの残りの部分に対しては、タイムアウトが強制されなくなります。クライアント・リルート機能が使用される場合、タイムアウトは二倍になります。一般に、クライアント・リルートが有効になっている場合、接続タイムアウト動作はクライアント・リルートによって指示されます。

IBM OLE DB Provider によるデータ・ソースへの接続

以下の例は、IBM OLE DB Provider for DB2 を使用して DB2 データ・ソースに接続する方法を示しています。

例 1: ADO を使用する Visual Basic アプリケーション

```
Dim db As ADODB.Connection
Set db = New ADODB.Connection
db.Provider = "IBMDADB2"
db.CursorLocation = adUseClient
...
```

例 2 IDataInitialize と Service Component を使用する C/C++ アプリケーション

```
hr = CoCreateInstance (
     CLSID_MSDAINITIALIZE,
     NULL,
     CLSCTX_INPROC_SERVER,
     IID_IDataInitialize,
     (void**)&pIDataInitialize);

hr = pIDataInitialize->CreateDBInstance(
     CLSID_IBMDADB2, // ClassID of IBMDADB2
     NULL,
```

```
CLSCTX_INPROC_SERVER,
NULL,
IID_IDBInitialize,
(IUnknown**)&pIDBInitialize);
```

ADO アプリケーション

ADO 接続ストリング・キーワード

ADO (ActiveX Data Objects) 接続ストリング・キーワードを指定するには、Provider (接続) ストリングで `keyword=value` のフォーマットを使用してキーワードを指定します。複数のキーワードを指定する場合はセミコロン (;) で区切ります。

以下の表は、IBM OLE DB Provider for DB2 でサポートされているキーワードを示しています。

表 34. IBM OLE DB Provider for DB2 でサポートされているキーワード

キーワード	値	意味
DSN	データベース別名の名前	データベース・ディレクトリーで使用される DB2 データベース別名。
UID	ユーザー ID	DB2 サーバーへの接続に使用するユーザー ID。
PWD	UID のパスワード	DB2 サーバーへの接続に使用するユーザー ID のパスワード。

他の CLI 構成キーワードも、IBM OLE DB Provider の動作に影響します。

Visual Basic ADO アプリケーションによるデータ・ソースへの接続

IBM OLE DB Provider for DB2 を使用して DB2 データ・ソースに接続するには、IBMDADB2 Provider 名を指定します。

ADO アプリケーションにおける更新可能な両方向スクロール・カーソル

IBM OLE DB Provider for DB2 では、読み取り専用カーソル、前方スクロール・カーソル、読み取り専用の両方向スクロール・カーソル、更新可能な両方向スクロール・カーソルがネイティブでサポートされています。更新可能な両方向スクロール・カーソルにアクセスする ADO アプリケーションはカーソル位置を `adUseClient` または `adUseServer` に設定する必要があります。カーソル位置を `adUseServer` に設定すると、カーソルがサーバー上で実体化されます。

ADO アプリケーションの制限

ADO アプリケーションの制限は以下のとおりです。

- ストアード・プロシージャーを呼び出す ADO アプリケーションでは、事前にパラメーターが作成されて明示的にバインドされていなければなりません。パラメーターを自動生成するための `Parameters.Refresh` 方式は、 DB2 Server for VSE & VM ではサポートされていません。

- デフォルトのパラメーター値はサポートされていません。
- サーバー・サイドの両方向スクロール・カーソルを使用して新しい行を挿入する場合、AddNew() 方式を Fieldlist および Values 引数値を指定して使用します。これは、各列に対して Update() 呼び出しに続いて引数なしの AddNew() を呼び出すより効率的です。 AddNew() および Update() の各呼び出しはサーバーに対する別個の要求です。したがって、これは AddNew() を単独で呼び出すより効率が悪くなります。
- 新規に挿入された行は、サーバー・サイドの両方向スクロール・カーソルでは更新できません。
- サーバー・サイドの両方向スクロール・カーソルを使用している場合には、LONG データ、または LOB データを持つ表は更新できません。

IBM OLE DB Provider での ADO メソッドおよびプロパティーのサポート

IBM OLE DB Provider は、以下の ADO メソッドおよびプロパティーをサポートしています。

表 35. コマンド・メソッド

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
Cancel	ICommand	はい
CreateParameter		はい
Execute		はい

表 36. コマンド・プロパティー

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
ActiveConnection	(ADO 固有)	
Command Text	ICommandText	はい
Command Timeout	ICommandProperties::SetProperties DBPROP_COMMANDTIMEOUT	はい
CommandType	(ADO 固有)	
Prepared	ICommandPrepare	はい
State	(ADO 固有)	

表 37. コマンド・コレクション

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
パラメーター	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	はい

表 37. コマンド・コレクション (続き)

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
Properties	ICommandProperties IDBProperties	はい

表 38. 接続メソッド

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
BeginTrans CommitTrans RollbackTrans	ITransactionLocal	はい (ただし、非ネスト) はい (ただし、非ネスト) はい (ただし、非ネスト)
Execute	ICommand IOpenRowset	はい
Open	IDBCreateSession IDBInitialize	はい
OpenSchema adSchemaColumnPrivileges adSchemaColumns adSchemaForeignKeys adSchemaIndexes adSchemaPrimaryKeys adSchemaProcedureParam adSchemaProcedures adSchemaProviderType adSchemaStatistics adSchemaTablePrivileges adSchemaTables	IDBSchemaRowset	はい はい はい はい はい はい はい はい はい はい はい はい
Cancel		はい

表 39. 接続プロパティー

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
Attributes adXactCommitRetaining adXactRollbackRetaining	ITransactionLocal	はい はい
CommandTimeout	ICommandProperties DBPROP_COMMAND_TIMEOUT	はい
ConnectionString	(ADO 固有)	
ConnectionTimeout	IDBProperties DBPROP_INIT_TIMEOUT	いいえ

表 39. 接続プロパティー (続き)

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
CursorLocation: adUseClient adUseNone adUseServer	(OLE DB Cursor Service を使用) (使用されず)	はい いいえ はい
DefaultDataBase	IDBProperties DBPROP_CURRENTCATALOG	いいえ
IsolationLevel	ITransactionLocal DBPROP_SESS _AUTOCOMMITISOLEVELS	はい
Mode adModeRead adModeReadWrite adModeShareDenyNone adModeShareDenyRead adModeShareDenyWrite adModeShareExclusive adModeUnknown adModeWrite	IDBProperties DBPROP_INIT_MODE	いいえ はい いいえ いいえ いいえ いいえ いいえ いいえ
Provider	ISourceRowset::GetSourceRowset	はい
State	(ADO 固有)	
バージョン	(ADO 固有)	

表 40. 接続コレクション

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
エラー	IErrorRecords	はい
Properties	IDBProperties	はい

表 41. エラー・プロパティー

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
	IErrorRecords	
Description NativeError Number Source SQLState		はい はい はい はい はい
HelpContext HelpFile		いいえ いいえ

表 42. フィールド・メソッド

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
AppendChunk GetChunk	ISequentialStream	はい はい

表 43. フィールド・プロパティー

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
Actual Size	IAccessor IRowset	はい
Attributes DataFormat DefinedSize Name NumericScale Precision タイプ	IColumnInfo	はい はい はい はい はい はい
OriginalValue	IRowsetUpdate	はい (カーソル・サービス)
UnderlyingValue	IRowsetRefresh IRowsetResynch	はい (カーソル・サービス) はい (カーソル・サービス)
値	IAccessor IRowset	はい

表 44. フィールド・コレクション

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
Properties	IDBProperties IRowsetInfo	はい

表 45. パラメーター・メソッド

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
AppendChunk	ISequentialStream	はい

表 45. パラメーター・メソッド (続き)

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
Attributes Direction Name NumericScale Precision Scale Size タイプ	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	はい いいえ はい はい はい はい はい
値	IAccessor ICommand	はい

表 46. パラメーター・コレクション

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
Properties		はい

表 47. レコード・セット・メソッド

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
AddNew	IRowsetChange	はい
Cancel		はい
CancelBatch	IRowsetUpdate::Undo	はい (カーソル・サービス)
CancelUpdate		はい (カーソル・サービス)
Clone	IRowsetLocate	はい
Close	IAccessor IRowset	はい
CompareBookmarks		いいえ
Delete	IRowsetChange	はい
GetRows	IAccessor IRowset	はい
Move	IRowset IRowsetLocate	はい
MoveFirst	IRowset IRowsetLocate	はい
MoveNext	IRowset IRowsetLocate	はい
MoveLast	IRowsetLocate	はい

表 47. レコード・セット・メソッド (続き)

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
MovePrevious	IRowsetLocate	はい
NextRecordSet	IMultipleResults	はい
Open	ICommand IOpenRowset	はい
Requery	ICommand IOpenRowset	はい
Resync	IRowsetRefresh	はい (カーソル・サービス)
Supports	IRowsetInfo	はい
更新 UpdateBatch	IRowsetChange IRowsetUpdate	はい はい (カーソル・サービス)

表 48. レコード・セット・プロパティー

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
AbsolutePage	IRowsetLocate IRowsetScroll	はい はい[1]
AbsolutePosition	IRowsetLocate IRowsetScroll	はい はい[1]
ActiveConnection	IDBCreateSession IDBInitialize	はい
BOF	(ADO 固有)	
Bookmark	IAccessor IRowsetLocate	はい
CacheSize	IRowsetLocate 内の cRows IRowset	はい
CursorType adOpenDynamic adOpenForwardOnly adOpenKeySet adOpenStatic	ICommandProperties	いいえ はい はい はい
EditMode	IRowsetUpdate	はい (カーソル・サービス)
EOF	(ADO 固有)	

表 48. レコード・セット・プロパティー (続き)

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
Filter	IRowsetLocate IRowsetView IRowsetUpdate IViewChapter IViewFilter	いいえ
LockType	ICommandProperties	はい
MarshallOption		いいえ
MaxRecords	ICommandProperties IOpenRowset	はい
PageCount	IRowsetScroll	はい[1]
PageSize	(ADO 固有)	
Sort	(ADO 固有)	
Source	(ADO 固有)	
State	(ADO 固有)	
Status	IRowsetUpdate	はい (カーソル・サービス)
注: 1. 戻される値は近似値です。削除された行はスキップされません。		

表 49. レコード・セット・コレクション

メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB サポート
Fields	IColumnInfo	はい
Properties	IDBProperties IRowsetInfo::GetProperties	はい

C/C++ アプリケーションのコンパイルおよびリンクと IBM OLE DB Provider

定数 CLSID_IBMDADB2 を使用する C/C++ アプリケーションには、SQLLIB¥include ディレクトリーにある ibmdadb2.h ファイルを組み込む必要があります。これらのアプリケーションでは、include ステートメントの前に DBINITCONSTANTS を定義する必要があります。以下の例は、C/C++ プリプロセッサー・ディレクティブの正しい順序を示しています。

```
#define DBINITCONSTANTS
#include "ibmdadb2.h"
```

IBM OLE DB Provider による、C/C++ アプリケーションでのデータ・ソースへの接続

C/C++ アプリケーションで IBM OLE DB Provider for DB2 を使用して DB2 データ・ソースに接続するには、2 つの OLE DB コア・インターフェース

(IDBPromptInitialize および IDataInitialize) のいずれかを使用するか、COM API CoCreateInstance を呼び出すことができます。 IDataInitialize インターフェースは OLE DB Service Component によって提供され、IDBPromptInitialize は Data Links Component によって提供されます。

COM+ 分散トランザクションのサポートと IBM OLE DB Provider

Windows 2000 または XP 上の Microsoft Component Services (COM+) 環境で実行されている OLE DB アプリケーションは、ITransactionJoin インターフェースを使用することにより、複数の DB2 Database for Linux, UNIX, and Windows、ホスト、および System i® データベース・サーバーや COM+ 仕様に準拠する他のリソース・マネージャーとの間で行われる分散トランザクションに参加することができます。

前提条件

IBM OLE DB Provider for DB2 に備わっている COM+ 分散トランザクション・サポートを使用するには、ご使用のサーバーが以下の前提条件を満たしていることを確かめてください。

注: これらの要件は、DB2 クライアントがインストールされている Windows ベースのコンピューターにのみ適用されます。

- Windows 2000 (Service Pack 3 以降)
- Windows XP

C/C++ データベース・アプリケーションでの COM+ サポートの使用可能化

C または C++ アプリケーションを COM+ トランザクション・モードで実行するには、CoCreateInstance を使用して IBMDADB2 データ・ソース・インスタンスを作成し、セッション・オブジェクトを入手して、JoinTransaction を使用できます。詳しくは、C または C++ アプリケーションをデータ・ソースに接続する方法に関する説明を参照してください。

ADO アプリケーションを COM+ トランザクション・モードで実行するには、C または C++ アプリケーションをデータ・ソースに接続する方法に関する説明を参照してください。

COM+ パッケージ内のコンポーネントをトランザクション・モードで使用するには、そのコンポーネントの Transactions プロパティーを以下の値のいずれかに設定します。

- 『Required』
- 『Required New』
- 『Supported』

これらの値については、COM+ の資料を参照してください。

第 6 章 OLE DB .NET Data Provider

OLE DB .NET Data Provider は、ConnectionString オブジェクト内では IBMDADB2 として示される IBM DB2 OLE DB Driver を使用します。

OLE DB .NET Data Provider でサポートされている接続ストリング・キーワードは、IBM OLE DB Provider for DB2 でサポートされている接続ストリング・キーワードと同じです。今後このプロバイダーはテストされません。 IBM Data Server Provider for .NET を使用することをお勧めします。

また、OLE DB .NET Data Provider には、IBM DB2 OLE DB Provider と同じ制約事項があります。OLE DB .NET Data Provider に対しては追加の制約事項があり、それについては「*ADO.NET および OLE DB アプリケーションの開発*」の『OLE DB .NET Data Provider の制約事項』のトピックで説明されています。

OLE DB .NET Data Provider を使用するには、.NET Framework バージョン 2.0、3.0、または 3.5 のいずれかをインストールする必要があります。

DB2 Universal Database for AS/400 R520、 R530 および R540、の場合、サーバー上で APAR ii13348 の修正を適用する必要があります。

OLE DB .NET Data Provider でサポートされているすべての接続キーワードを表 1 に示します。

表 50. 有用な、OLE DB .NET Data Provider の **ConnectionString** キーワード

キーワード	値	意味
PROVIDER	IBMDADB2	IBM OLE DB Provider for DB2 を指定します (必須)
DSN または **データ・ソース**	データベース別名	データベース・ディレクトリーにカタログされた DB2 データベース別名。
UID	user ID	DB2 データ・サーバーへの接続に使用するユーザー ID
PWD	password	DB2 データ・サーバーへの接続に使用するユーザー ID のパスワード

注: ConnectionString キーワードの完全なリストは、Microsoft 資料を参照してください。

以下に、OleDbConnection を作成して SAMPLE データベースに接続する例を示します。

```
[Visual Basic .NET]
Dim con As New OleDbConnection("Provider=IBMDADB2;" +
    "Data Source=sample;UID=userid;PWD=password;")
con.Open()
```

```
[C#]
OleDbConnection con = new OleDbConnection("Provider=IBMDADB2;" +
                "Data Source=sample;UID=userid;PWD=password;" );
con.Open()
```

OLE DB .NET Data Provider の制約事項

今後 OLE DB .NET Data Provider はテストされません。 IBM Data Server Provider for .NET を使用することをお勧めします。

以下の表は、OLE DB .NET Data Provider の使用に関係した制約事項を示しています。

表 51. OLE DB .NET Data Provider の制約事項

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
ASCII 文字ストリーム	DbType.AnsiString または DbType.AnsiStringFixedLength を使用している場合、OleDbParameters で ASCII 文字ストリームを使用することはできません。 OLE DB .NET Data Provider が次の例外をスローします。 "Specified cast is not valid" **回避策:** DbType.AnsiString または DbType.AnsiStringFixedLength を使用する代わりに DbType.Binary を使用します。	すべて
ADORecord	ADORecord はサポートされていません。	すべて
ADORecordSet および Timestamp	MSDN で説明されているように、ADORecordSet の変化時間単位は、1 秒として解決されます。そのため、DB2 Timestamp 列が ADORecordSet に保管されるとき、小数点未満の秒数はすべて失われます。同様に、DataSet に ADORecordSet からデータを取り込んだ後は、 DataSet の Timestamp 列に、小数点未満の秒数は含まれません。 **回避策:** この回避策は DB2 Universal Database for Linux, UNIX, and Windows バージョン 8.1 フィックスパック 4 以降でのみ有効です。小数点未満の秒数が失われるのを回避するため、次の CLI キーワードを設定することができます。 MAPTIMESTAMPDESCRIBE = 2 このキーワードは、Timestamp を WCHAR(26) として記述します。キーワードを設定するには、DB2 コマンド・ウィンドウから次のコマンドを実行してください。 db2 update cli cfg for section common using MAPTIMESTAMPDESCRIBE 2	すべて
Chapters	Chapters はサポートされていません。	すべて
キー情報	OLE DB .NET Data Provider は、IDataReader を開くのと同時にキー情報を検索することはできません。	DB2 for VM/VSE

表 51. OLE DB .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
ストアード・プロシージャーのキー情報	OLE DB .NET Data Provider は、DB2 Database for Linux, UNIX, and Windows からのみ、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関するキー情報を検索できます。なぜなら、Linux、UNIX、および Windows 以外のプラットフォーム用の DB2 サーバーは、ストアード・プロシージャーで開かれた結果セットの拡張記述情報を戻さないからです。 DB2 Database for Linux, UNIX, and Windows 上で、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のキー情報を検索するには、DB2 サーバーで、以下のレジストリー変数を設定する必要があります。 `db2set DB2_APM_PERFORMANCE=8` このサーバー・サイドの DB2 レジストリー変数を設定すると、サーバー上で結果セットのメタデータがより長い期間使用可能になり、 OLE DB が正常にキー情報を検索できるようになります。しかし、サーバーのワークロードによっては、 OLE DB Provider が情報を照会するのに十分な期間メタデータが使用可能にならないことがあります。したがって、ストアード・プロシージャーから戻された結果セットに関しては、キー情報を常に入手できるという保障はありません。 CALL ステートメントに関するキー情報を検索するためには、アプリケーションが CALL ステートメントを実行しなければなりません。 `OleDbDataAdapter.FillSchema()` または `OleDbCommand.ExecuteReader (CommandBehavior.SchemaOnly | CommandBehavior.KeyInfo)` を呼び出しても、ストアード・プロシージャー呼び出しは実際には実行されません。したがって、ストアード・プロシージャーによって戻される結果セットのキー情報は検索されません。	すべて

バッチ SQL ステートメントからのキー情報	複数の結果を戻すバッチ SQL ステートメントを使用している場合、`FillSchema()` メソッドは、バッチ SQL ステートメント・リストの中の最初の SQL ステートメントについてのみ、スキーマ情報の検索を試行します。このステートメントが結果セットを戻さなければ、表は作成されません。以下に例を示します。
	`[C#]`
	`cmd.CommandText =`
	`"INSERT INTO ORG(C1) VALUES(1000); SELECT C1 FROM ORG;";`
	`da = new OleDbDataAdapter(cmd);`
	`da.FillSchema(ds, SchemaType.Source);`
	
	バッチ SQL ステートメントの最初のステートメントが、結果セットを戻さない INSERT ステートメントなので、データ・セット内に表は作成されません。
		すべて

表 51. OLE DB .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
`OleDbCommandBuilder`	SELECT ステートメントに以下のデータ・タイプの列が含まれている場合、`OleDbCommandBuilder` によって自動的に生成された UPDATE、DELETE、および INSERT ステートメントは正しくありません。 • CLOB • BLOB • DBCLOB • LONG VARCHAR • LONG VARCHAR FOR BIT DATA • LONG VARGRAPHIC DB2 Database for Linux, UNIX, and Windows 以外の DB2 サーバーに接続している場合は、以下のデータ・タイプの列も問題を引き起こします。 • VARCHAR[1] • VARCHAR FOR BIT DATA[1] • VARGRAPHIC[1] • REAL • FLOAT または DOUBLE • TIMESTAMP **注:** 1. これらのデータ・タイプの列は、254 バイトより大きい VARCHAR 値、 254 バイトより大きい VARCHAR 値 FOR BIT DATA、または 127 バイトより大きい VARGRAPHIC として定義されている場合は適切です。この条件は、DB2 Database for Linux, UNIX, and Windows 以外の DB2 サーバーに接続している場合にのみ有効です。 `OleDbCommandBuilder` は、 WHERE 節の等価比較で選択されたすべての列を使用する SQL ステートメントを生成しますが、以前にリストされたデータ・タイプを等価比較に使用することはできません。 **注:** この制限は、`OleDbCommandBuilder` を利用して UPDATE、 DELETE、および INSERT ステートメントを自動生成する `IDbDataAdapter.Update()` メソッドに影響することにご注意ください。生成されたステートメントに、前にリストされたデータ・タイプがいずれか 1 つ含まれていると、 UPDATE 操作は失敗します。 **回避策:** 生成された SQL ステートメントの WHERE 節から、上にリストされたデータ・タイプのすべての列を明示的に除去する必要があります。独自に UPDATE、DELETE および INSERT ステートメントをコーディングすることをお勧めします。	すべて
`OleDbCommandBuilder.DeriveParameters`	`DeriveParameters()` を使用する際は、大文字小文字の区別が重要です。`OleDbCommand.CommandText` で指定されたストアード・プロシージャー名は、 DB2 システム・カタログ表に保管されている名前と同じケースでなければなりません。ストアード・プロシージャー名がどのように保管されているかを確認するには、プロシージャー名制限なしで OpenSchema(OleDbSchemaGuid.Procedures) を呼び出します。これは、すべてのストアード・プロシージャー名を戻します。デフォルトで、DB2 はストアード・プロシージャー名を大文字で保管します。したがって、ほとんどの場合、ストアード・プロシージャー名は大文字で指定する必要があります。	すべて

表 51. OLE DB .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
OleDbCommandBuilder.DeriveParameters	OleDbCommandBuilder.DeriveParameters() メソッドは、生成される OleDbParameterCollection 内に ReturnValue パラメーターを含めません。SqlClient および IBM Data Server Provider for .NET はデフォルトで、生成される ParameterCollection に、ParameterDirection.ReturnValue と共にパラメーターを追加します。	すべて
OleDbCommandBuilder.DeriveParameters	多重定義されたストアード・プロシージャーの場合、OleDbCommandBuilder.DeriveParameters() メソッドは失敗します。 MYPROC という名前の複数のストアード・プロシージャーがあり、それぞれが異なる数または異なるタイプのパラメーターを取る場合、 OleDbCommandBuilder.DeriveParameters() では、多重定義されているすべてのストアード・プロシージャーのすべてのパラメーターが検索されます。	すべて
OleDbCommandBuilder.DeriveParameters	アプリケーションがストアード・プロシージャーをスキーマで修飾していない場合、 DeriveParameters() では、そのプロシージャー名のすべてのパラメーターが戻されます。したがって、同一のプロシージャー名に対して複数のスキーマが存在する場合、 DeriveParameters() では、同じ名前を持つすべてのプロシージャーのすべてのパラメーターが戻されます。	すべて
OleDbConnection.ChangeDatabase	OleDbConnection.ChangeDatabase() メソッドはサポートされていません。	すべて
OleDbConnection.ConnectionString	接続ストリングの中で ¥b、¥a、または ¥O などの印刷不可文字を使用すると、例外が出されます。 以下のキーワードには、制約があります。 **Data Source** データ・ソースは、サーバーではなく、データベースの名前です。SERVER キーワードは指定できますが、IBMDADB2 プロバイダーからは無視されます。 **Initial Catalog および Connect Timeout** これらのキーワードはサポートされていません。通常、OLE DB .NET Data Provider は、認識されない非サポート・キーワードはすべて無視します。しかし、これらのキーワードを指定すると、次の例外が発生します。 `Multiple-step OLE DB operation generated errors. Check each OLE DB status value, if available. No work was done.` **ConnectionTimeout** ConnectionTimeout は読み取り専用です。	すべて

表 51. OLE DB .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー

`OleDbConnection.GetOleDbSchemaTable`	制限値は、大文字小文字の区別があり、システム・カタログ表に保管されているデータベース・オブジェクトとケースが一致している必要があります。デフォルトではこれは大文字です。
	
	例えば、次のようにして表を作成したとします。
	`CREATE TABLE abc(c1 SMALLINT)`
	
	DB2 は表の名前を大文字 ("ABC") でシステム・カタログに保管します。そのため、"ABC" を制限値として使用する必要があります。例えば、次のようにします。
	`schemaTable = con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, new object[] { null, null, "ABC", "TABLE" });`
	
	回避策: データ定義で大/小文字の区別またはスペースが必要な場合、引用符で囲む必要があります。以下に例を示します。
	`cmd.CommandText = "create table \"Case Sensitive\"(c1 int)";`
	`cmd.ExecuteNonQuery();`
	`tablename = "¥"Case Sensitive¥"";`
	`schemaTable = con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, new object[] { null, null, tablename, "TABLE" });`
		すべて

`OleDbDataAdapter` および `DataColumnMapping`	ソース列名には、大文字小文字の区別があります。これは、DB2 カタログに保管されるケースと一致している必要があり、それはデフォルトで大文字です。 以下に例を示します。 `colMap = new DataColumnMapping("EMPNO", "Employee ID");`	すべて
`OleDbDataReader.GetSchemaTable`	OLE DB .NET Data Provider は、拡張記述情報を戻さないサーバーからは、拡張記述情報を検索できません。拡張記述をサポートしていないサーバー (影響を受けるサーバー) に接続している場合、 `IDataReader.GetSchemaTable()` から戻されたメタデータ表の中の以下の列は無効となります。 • `IsReadOnly` • `IsUnique` • `IsAutoIncrement` • `BaseSchemaName` • `BaseCatalogName`	DB2 for OS/390®, V7 以前 DB2 for OS/400 DB2 for VM/VSE
ストアード・プロシージャー: 結果セットの列名はなし	DB2 for OS/390 バージョン 6.1 サーバーは、ストアード・プロシージャーから戻された結果セットの列名を戻しません。 OLE DB .NET Data Provider は、これらの無名列を、その順序位置 (例えば、"1"、"2"、"3") にマップします。これは、MSDN で説明されているマッピング (`"Column1", "Column2", "Column3"`) とは異なります。	DB2 for OS/390 バージョン 6.1

ヒント

OLE DB .NET Data Provider アプリケーションでの接続プーリング

OLE DB .NET Data Provider は、OLE DB セッション・プーリングを使用して、自動的に接続をプールします。プーリングを含む OLE DB サービスを使用可能または使用不可にするには、接続ストリング引数を使用します。例えば、次の接続ストリングは、 OLE DB セッション・プーリングおよび自動トランザクション参加を使用不可にします。

```
Provider=IBMDADB2;OLE DB Services=-4;Data Source=SAMPLE;
```

次の表では、OLE DB サービスを設定するために使用できる、 ADO 接続ストリング属性について説明しています。

表 52. ADO 接続ストリング属性を使用した OLE DB サービスの設定

使用可能サービス	接続ストリングの値
すべてのサービス (デフォルト)	"OLE DB Services = -1;"
プーリングを除くすべてのサービス	"OLE DB Services = -2;"
プーリングおよび自動的参加を除くすべてのサービス	"OLE DB Services = -4;"
クライアント・カーソルを除くすべてのサービス	"OLE DB Services = -5;"
クライアント・カーソルおよびプーリングを除くすべてのサービス	"OLE DB Services = -6;"
サービスなし	"OLE DB Services = 0;"

OLE DB セッション・プーリングまたはリソース・プーリングについて、および OLE DB プロバイダー・サービスのデフォルトをオーバーライドしてプーリングを無効にする方法についての詳細は、以下のアドレスの MSDN ライブラリーの「OLE DB Programmer's Reference」を参照してください。

```
http://msdn.microsoft.com/library
```

OLE DB .NET Data Provider アプリケーションの時刻列

以下のセクションでは、OLE DB .NET Data Provider アプリケーションに、時刻列をインプリメントする方法について説明します。

パラメーター・マーカーを使用した挿入

次のように、時刻列に時刻値を挿入しようとしています。

```
command.CommandText = "insert into mytable(c1) values( ? )";
```

列 c1 は時刻列です。ここでは、時刻値をパラメーター・マーカーにバインドする 2 つの方法があります。

`OleDbParameter.OleDbType = OleDbType.DBTime` を使用する

OleDbType.DBTime は TimeSpan オブジェクトにマップするので、 TimeSpan オブジェクトをパラメーター値として指定する必要があります。パラメーター値は、String または DateTime オブジェクトにはできません。これは TimeSpan オブジェクトでなければなりません。以下に例を示します。

```
p1.OleDbType = OleDbType.DBTime;
p1.Value = TimeSpan.Parse("0.11:20:30");
rowsAffected = cmd.ExecuteNonQuery();
```

MSDN の資料で説明されているとおり、 TimeSpan の形式は、[-]d.hh:mm:ss.ff の形式のストリングで表記されます。

`OleDbParameter.OleDbType = OleDbType.DateTime` を使用する

この場合、OLE DB .NET Data Provider は、パラメーター値を TimeSpan オブジェクトではなく、DateTime オブジェクトに変換します。したがって、パラメーター値は、 DateTime オブジェクトに変換できる任意の有効なストリング/オブジェクトにできます。つまり、11:20:30 などの値を用いることができます。値を DateTime オブジェクトにすることもできます。値を TimeSpan オブジェクトにすることはできません。なぜなら、TimeSpan オブジェクトは DateTime オブジェクトに変換できないからです。 TimeSpan は IConvertible をインプリメントしていません。

以下に例を示します。

```
p1.OleDbType = OleDbType.DBTimeStamp;
p1.Value = "11:20:30";
rowsAffected = cmd.ExecuteNonQuery();
```

検索

時刻列を検索するには、 IDataRecord.GetValue() メソッドまたは OleDbDataReader.GetTimeSpan() メソッドを使用する必要があります。

以下に例を示します。

```
TimeSpan ts1 = ((OleDbDataReader)reader).GetTimeSpan( 0 );
TimeSpan ts2 = (TimeSpan) reader.GetValue( 0 );
```

OLE DB .NET Data Provider アプリケーションの ADORecordset オブジェクト

ADORecordset オブジェクトの使用に関する考慮事項を以下に示します。

- ADO タイプ adDBTime クラスは、 .NET Framework の DateTime クラスにマップされます。 OleDbType.DBTime は、TimeSpan オブジェクトに対応します。
- TimeSpan オブジェクトを ADORecordset オブジェクトの Time フィールドに割り当てることはできません。なぜなら、ADORecordset オブジェクトの Time フィールドでは、 DateTime オブジェクトが期待されているからです。 TimeSpan オブジェクトを ADORecordset オブジェクトに割り当てると、次のメッセージが表示されます。

  ```
  Method's type signature is not Interop compatible.
  ```

  Time フィールドに入れることができるのは、DateTime オブジェクトか、DateTime オブジェクトに解析できる String のみです。
- OleDbDataAdapter を使用して DataSet に ADORecordset を取り込むとき、ADORecordset の Time フィールドは、 DataSet の TimeSpan 列に変換されます。
- Recordsets には主キーや制約は保管されません。したがって、MissingSchemaAction.AddWithKey を使用して DataSet に Recordset からデータを取り込む際は、キー情報は追加されません。

第 7 章 ODBC .NET Data Provider

ODBC .NET Data Provider は、CLI ドライバーを使用して、 DB2 データ・ソースに対して ODBC 呼び出しを行います。したがって、ODBC .NET Data Provider がサポートする接続ストリング・キーワードは、CLI ドライバーがサポートする接続ストリング・キーワードと同じです。今後このプロバイダーはテストされません。IBM Data Server Provider for .NET を使用することをお勧めします。

また、ODBC .NET Data Provider には、CLI ドライバーと同じ制約事項があります。ODBC .NET Data Provider に対しては追加の制約事項があり、それについては「*ADO.NET および OLE DB アプリケーションの開発*」の『ODBC .NET Data Provider の制約事項』のトピックで説明されています。

ODBC .NET Data Provider を使用するには、.NET Framework バージョン 2.0、3.0、または 3.5 のいずれかをインストールする必要があります。DB2 Universal Database for AS/400 V5R4 以前の場合、サーバー上で APAR II13348 の修正を適用する必要があります。

ODBC .NET Data Provider でサポートされている接続キーワードを表 1 に示します。

表 53. 有用な、ODBC .NET Data Provider の **ConnectionString** *キーワード*

キーワード	値	意味
DSN	データベース別名	データベース・ディレクトリーにカタログされた DB2 データベース別名。
UID	user ID	DB2 サーバーへの接続に使用するユーザー ID
PWD	password	DB2 サーバーへの接続に使用するユーザー ID のパスワード

注: ConnectionString キーワードの完全なリストは、Microsoft 資料を参照してください。

以下のコードに、OdbcConnection を作成して SAMPLE データベースに接続する例を示します。

```
[Visual Basic .NET]
Dim con As New OdbcConnection("DSN=sample;UID=userid;PWD=password;")
con.Open()

[C#]
OdbcConnection con = new OdbcConnection("DSN=sample;UID=userid;PWD=password;");
con.Open()
```

ODBC .NET Data Provider の制約事項

今後 ODBC .NET Data Provider はテストされません。 IBM Data Server Provider for .NET を使用することをお勧めします。

以下の表は、ODBC .NET Data Provider の使用に関係した制約事項を示しています。

表 54. ODBC .NET Data Provider の制約事項

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
ASCII 文字ストリーム	DbType.AnsiString または DbType.AnsiStringFixedLength を使用している場合、OdbcParameters で ASCII 文字ストリームを使用することはできません。 ODBC .NET Data Provider が次の例外をスローします。 "Specified cast is not valid" **回避策:** DbType.AnsiString または DbType.AnsiStringFixedLength を使用する代わりに DbType.Binary を使用します。	すべて

Command.Prepare	最後の準備の後に CommandText が変更されている場合は、コマンド (Command.ExecuteNonQuery または Command.ExecuteReader) を実行する前に、OdbcCommand.Prepare() を明示的に実行する必要があります。OdbcCommand.Prepare() を再び呼び出さないと、 ODBC .NET Data Provider は以前に準備された CommandText を実行します。
	
	以下に例を示します。
	[C#]
	command.CommandText="select CLOB('ABC') from table1";
	command.Prepare();
	command.ExecuteReader();
	command.CommandText="select CLOB('XYZ') from table2";
	
	// This ends up re-executing the first statement
	command.ExecuteReader();
		すべて

表 54. ODBC .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー

CommandBehavior.SequentialAccess	CommandBehavior.SequentialAccess で作成されたリーダーから、IDataReader.GetChars() を使用して読み取る場合は、列全体を保持するのに十分な大きさのバッファーを割り振る必要があります。そうしないと、次の例外が発生します。
	`Requested range extends past the end of the array.`
	`  at System.Runtime.InteropServices.Marshal.Copy(Int32 source,`
	`  Char[] destination, Int32 startIndex, Int32 length)`
	`  at System.Data.Odbc.OdbcDataReader.GetChars(Int32 i,`
	`  Int64 dataIndex, Char[] buffer, Int32 bufferIndex, Int32 length)`
	`  at OleRestrict.TestGetCharsAndBufferSize(IDbConnection con)`
	次の例で、十分なバッファーを割り振る方法を示します。
	`CREATE TABLE myTable(c0 int, c1 CLOB(10K))`
	`SELECT c1 FROM myTable;`
	`[C#]`
	`cmd.CommandText = "SELECT c1 from myTable";`
	`IDataReader reader =`
	`cmd.ExecuteReader(CommandBehavior.SequentialAccess);`
	`Int32 iChunkSize = 10;`
	`Int32 iBufferSize = 10;`
	`Int32 iFieldOffset = 0;`
	`Char[] buffer = new Char[ iBufferSize ];`
	`reader.Read();`
	`reader.GetChars(0, iFieldOffset, buffer, 0, iChunkSize);`
	GetChars() を呼び出すと、次の例外が出されます。
	`"Requested range extends past the end of the array"`
	GetChars() によって前述の例外が出されないようにするためには、次のようにして、BufferSize に列のサイズを設定する必要があります。
	`Int32 iBufferSize = 10000;`
	iBufferSize の値 10,000 は、 CLOB 列 c1 に割り振られている値 10K と対応します。
		すべて

CommandBehavior.SequentialAccess	ODBC .NET Data Provider は、 OdbcDataReader.GetChars() を使用しているときに読み取るデータがなくなると、次の例外を出します。
	`NO_DATA - no error information available`
	`  at System.Data.Odbc.OdbcConnection.HandleError(`
	`  HandleRef hrHandle, SQL_HANDLE hType, RETCODE retcode)`
	`  at System.Data.Odbc.OdbcDataReader.GetData(`
	`  Int32 i, SQL_C sqlctype, Int32 cb)`
	`  at System.Data.Odbc.OdbcDataReader.GetChars(`
	`  Int32 i, Int64 dataIndex, Char[] buffer,`
	`  Int32 bufferIndex, Int32 length)`
		すべて

CommandBehavior.SequentialAccess	OdbcDataReader.GetChars() を使用するときは、値 5000 などの大きなチャンク・サイズは使用できません。大きなチャンク・サイズを使用しようとすると、ODBC .NET Data Provider が次の例外をスローします。
	`Object reference not set to an instance of an object.`
	`  at System.Runtime.InteropServices.Marshal.Copy(Int32 source,`
	`  Char[] destination, Int32 startIndex, Int32 length)`
	`  at System.Data.Odbc.OdbcDataReader.GetChars(`
	`  Int32 i, Int64 dataIndex, Char[] buffer,`
	`  Int32 bufferIndex, Int32 length)`
	`  at OleRestrict.TestGetCharsAndBufferSize(IDbConnection con)`
		すべて

表 54. ODBC .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
接続プール	ODBC .NET Data Provider は接続プーリングを制御しません。接続プーリングは、ODBC Driver Manager によって取り扱われます。接続プーリングについて詳しくは、MSDN ライブラリーの「ODBC Programmer's Reference」を参照してください。MSDN ライブラリーの URL は以下のとおりです。 `http://msdn.microsoft.com/library`	すべて
DataColumnMapping	ソース列名の大文字小文字は、システム・カタログ表で使用されている大文字小文字と一致している必要があります。これは、デフォルトで大文字です。	すべて

10 進列	10 進列では、パラメーター・マーカーはサポートされません。
	
	通常、ターゲット `SQLType` が `Decimal` 列の場合は、 `OdbcParameter` に `OdbcType.Decimal` を使用します。しかし、ODBC .NET Data Provider は `OdbcType.Decimal` を見付けると、 `SQL_C_WCHAR` の C タイプと `SQL_VARCHAR` の `SQLType` を使用してパラメーターをバインドし、それは無効となります。
	
	以下に例を示します。
	`[C#]`
	`cmd.CommandText = "SELECT dec_col FROM MYTABLE WHERE dec_col > ? ";`
	`OdbcParameter p1 = cmd.CreateParameter();`
	`p1.DbType = DbType.Decimal;`
	`p1.Value = 10.0;`
	`cmd.Parameters.Add(p1);`
	`IDataReader rdr = cmd.ExecuteReader();`
	
	次の例外が戻されます。
	`ERROR [07006] [IBM][CLI Driver][SQLDS/VM]`
	`SQL0301N The value of input host variable or parameter`
	`number "" cannot be used because of its data type.`
	`SQLSTATE=07006`
	
	回避策: `OdbcParameter` 値を使用するのではなく、リテラルのみを使用してください。
		DB2 for VM/VSE

キー情報	表名を修飾するために使用されるスキーマ名 (例えば、`MYSCHEMA.MYTABLE`) は、接続ユーザー ID と一致していなければなりません。 ODBC .NET Data Provider は、指定されたスキーマが接続ユーザー ID と異なるキー情報は検索できません。
	
	以下に例を示します。
	`CREATE TABLE USERID2.TABLE1(c1 INT NOT NULL PRIMARY KEY);`
	
	`[C#]`
	`// Connect as user bob`
	`odbcCon = new OdbcConnection("DSN=sample;UID=bob;PWD=mypassword");`
	
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	
	`// Select from table with schema USERID2`
	`cmd.CommandText="SELECT * FROM USERID2.TABLE1";`
	
	`// Fails - No key info retrieved`
	`da.FillSchema(ds, SchemaType.Source);`
	
	`// Fails - SchemaTable has no primary key`
	`cmd.ExecuteReader(CommandBehavior.KeyInfo)`
	
	`// Throws exception because no primary key`
	`cbuilder.GetUpdateCommand();`
		すべて

表 54. ODBC .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー

キー情報	ODBC .NET Data Provider は、 IDataReader を開くのと同時にキー情報を検索することはできません。 ODBC .NET Data Provider が IDataReader を開くと、サーバー上でカーソルが開きます。キー情報が要求された場合、これは SQLPrimaryKeys() または SQLStatistic() を呼び出してキー情報を取得しますが、これらのスキーマ関数は他のカーソルを開きます。 DB2 for VM/VSE はカーソル保留をサポートしていないため、最初のカーソルはクローズされます。その結果、IDataReader.Read() が IDataReader を呼び出すと、次の例外が発生します。
	`System.Data.Odbc.OdbcException: ERROR [HY010] [IBM][CLI Driver] CLI0125E Function sequence error. SQLSTATE=HY010`
	回避策: 最初にキー情報を取得してからデータを取得する必要があります。以下に例を示します。
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`OdbcDataAdapter da = new OdbcDataAdapter(cmd);`
	`cmd.CommandText = "SELECT * FROM MYTABLE";`
	`// Use FillSchema to retrieve just the schema information`
	`da.FillSchema(ds, SchemaType.Source);`
	`// Use FillSchema to retrieve just the schema information`
	`da.Fill(ds);`
		DB2 for VM/VSE

キー情報	SQL ステートメントの中では、データベース・オブジェクトは、データベース・オブジェクトをシステム・カタログ表に保管するのに用いられているケース (大文字小文字) と同じケースを使用して参照する必要があります。デフォルトでは、データベース・オブジェクトは大文字でシステム・カタログ表に保管されるので、ほとんどの場合は、大文字を使用する必要があります。 ODBC .NET Data Provider は、SQL ステートメントをスキャンしてデータベース・オブジェクト名を検索し、それらを、システム・カタログ表内のこれらのオブジェクトについての照会を発行する SQLPrimaryKeys および SQLStatistics などのスキーマ関数に渡します。データベース・オブジェクトの参照は、システム・カタログ表にそれらが保管されている状態と完全に一致していなければなりません。そうでないと、空の結果セットが戻されます。	DB2 for OS/390 DB2 for OS/400 DB2 for VM/VSE
バッチの非選択 SQL ステートメントのキー情報	ODBC .NET Data Provider は、 SELECT で始まっていないバッチ・ステートメントのキー情報は検索できません。	DB2 for OS/390 DB2 for OS/400 DB2 for VM/VSE

表 54. ODBC .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー

LOB 列	ODBC .NET Data Provider は LOB データ・タイプをサポートしていません。そのため、DB2 サーバーが SQL_CLOB (-99)、SQL_BLOB (-98)、または SQL_DBCLOB (-350) を戻すと、 ODBC .NET Data Provider は次の例外を出します。
	`"Unknown SQL type - -98"` (Blob 列の場合)
	`"Unknown SQL type - -99"` (Clob 列の場合)
	`"Unknown SQL type - -350"` (DbClob 列の場合)
	
	直接または間接的に LOB 列にアクセスするメソッドはどれも失敗します。
	
	回避策: CLI/ODBC `LongDataCompat` キーワードを 1 に設定します。そのようにすると、CLI ドライバーは、ODBC .NET Data Provider によって認識されるデータ・タイプへの、次のデータ・タイプ・マッピングを行います。
	• SQL_CLOB から SQL_LONGVARCHAR へ
	• SQL_BLOB から SQL_LONGVARBINARY へ
	• SQL_DBCLOB から SQL_WLONGVARCHAR へ
	
	`LongDataCompat` キーワードを設定するには、クライアント・マシンの DB2 コマンド・ウィンドウから次の DB2 コマンドを実行してください。
	`db2 update cli cfg for section common using longdatacompat 1`
	
	次のように接続ストリングを使用して、このキーワードをアプリケーションに設定することもできます。
	`[C#]`
	`OdbcConnection con =`
	`  new OdbcConnection("DSN=SAMPLE;UID=uid;PWD=mypwd;LONGDATACOMPAT=1;");`
	
	すべての CLI/ODBC キーワードのリストについては、「DB2 CLI ガイドおよびリファレンス」の『UID CLI/ODBC 構成キーワード』を参照してください。
		すべて

`OdbcCommand.Cancel`	`OdbcCommand.Cancel` の実行後にステートメントを実行すると、次の例外が発生することがあります。
	`"ERROR [24000] [Microsoft][ODBC Driver Manager]`
	`    Invalid cursor state"`
		すべて

`OdbcCommandBuilder`	`OdbcCommandBuilder` は、エスケープ文字をサポートしないサーバーに対するコマンドの生成に失敗します。`OdbcCommandBuilder` は、コマンドを生成するときに、まず `SQLGetInfo` を呼び出して、`SQL_SEARCH_PATTERN_ESCAPE` 属性を要求します。サーバーがエスケープ文字をサポートしていない場合は空ストリングが戻され、ODBC .NET Data Provider は次の例外をスローします。
	`Index was outside the bounds of the array.`
	`   at System.Data.Odbc.OdbcConnection.get_EscapeChar()`
	`   at System.Data.Odbc.OdbcDataReader.GetTableNameFromCommandText()`
	`   at System.Data.Odbc.OdbcDataReader.BuildMetaDataInfo()`
	`   at System.Data.Odbc.OdbcDataReader.GetSchemaTable()`
	`   at System.Data.Common.CommandBuilder.BuildCache(`
	`    Boolean closeConnection)`
	`   at System.Data.Odbc.OdbcCommandBuilder.GetUpdateCommand()`
		DB2 for OS/390、DBCS サーバーのみ; DB2 for VM/VSE、DBCS サーバーのみ

表 54. ODBC .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー

OdbcCommandBuilder	OdbcCommandBuilder を使用して UPDATE、DELETE、および INSERT ステートメントを自動生成する場合、大文字小文字の区別は重要です。システム・カタログ表が、(作成時にデータベース・オブジェクトの前後に引用符を追加することによって) 明示的に大文字小文字の区別付きで作成されているのでない限り、デフォルトで DB2 はスキーマ情報 (表名や列名など) を大文字でシステム・カタログ表に保管します。したがって、SQL ステートメントは、カタログに保管されているケース (デフォルトでは大文字) と一致している必要があります。
	
	例えば、次のステートメントを使用して表を作成したとします。
	`"db2 create table mytable (c1 int) "`
	
	この場合、DB2 は表名 mytable をシステム・カタログ表に MYTABLE として保管します。
	
	次のコード例は、OdbcCommandBuilder クラスの正しい使用法を例示しています。
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`cmd.CommandText  = "SELECT * FROM MYTABLE";`
	`OdbcDataAdapter da = new OdbcDataAdapter(cmd);`
	`OdbcCommandBuilder cb = new OdbcCommandBuilder(da);`
	`OdbcCommand updateCmd = cb.GetUpdateCommand();`
	
	この例では、表名を大文字で参照しないと、次の例外を受け取ります。
	`"Dynamic SQL generation for the UpdateCommand is not supported against a SelectCommand that does not return any key column information."`
		すべて

OdbcCommandBuilder	SELECT ステートメントに以下の列データ・タイプが含まれている場合、OdbcCommandBuilder によって生成されたコマンドは正しくありません。
	`REAL`
	`FLOAT または DOUBLE`
	`TIMESTAMP`
	
	これらのデータ・タイプは、SELECT ステートメントの WHERE 節では使用できません。
		DB2 for OS/390
		DB2 for OS/400
		DB2 for VM/VSE

OdbcCommandBuilder.DeriveParameters	DeriveParameters() メソッドは、SQLProcedureColumns にマップされ、ストアード・プロシージャーの名前に CommandText プロパティーを使用します。CommandText には、 (完全な ODBC 呼び出し構文を使用する) ストアード・プロシージャーの名前は含まれないので、 SQLProcedureColumns は、 ODBC 呼び出し構文に基づいて識別されたプロシージャー名で呼び出されます。以下に例を示します。 `"{ CALL myProc(?) }"` この結果は、空の結果セットとなり、ここからはプロシージャーの列は見付かりません。	すべて
OdbcCommandBuilder.DeriveParameters	DeriveParameters() を使用するには、 CommandText にストアード・プロシージャーの名前を指定します (例えば、cmd.CommandText = "MYPROC")。プロシージャー名は、システム・カタログ表に保管されているケースと一致していなければなりません。 DeriveParameters() では、システム・カタログ表内で見付かる、そのプロシージャー名のすべてのパラメーターが戻されます。ステートメントを実行する前に、 CommandText を元の完全な ODBC 呼び出し構文に戻すのを忘れないでください。	すべて
OdbcCommandBuilder.DeriveParameters	ODBC .NET Data Provider に関しては、ReturnValue パラメーターは戻されません。	すべて

表 54. ODBC .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
OdbcCommandBuilder.DeriveParameters	DeriveParameters() は、完全修飾ストアード・プロシージャー名をサポートしていません。例えば、CommandText = "MYSCHEMA.MYPROC" に対して DeriveParameters() を呼び出すと失敗します。このとき、パラメーターは戻されません。	すべて
OdbcCommandBuilder.DeriveParameters	DeriveParameters() は、多重定義のストアード・プロシージャーに対しては機能しません。 SQLProcedureColumns は、ストアード・プロシージャーのすべてのバージョンのすべてのパラメーターを戻します。	すべて
OdbcConnection.ChangeDatabase	OdbcConnection.ChangeDatabase() メソッドはサポートされていません。	すべて
OdbcConnection.ConnectionString	• Server キーワードは無視されます。 • Connect Timeout キーワードは無視されます。 CLI は接続タイムアウトをサポートしていないので、このプロパティーを設定しても、ドライバーに影響はありません。 • 接続プーリング・キーワードは無視されます。具体的には、これは Pooling、Min Pool Size、Max Pool Size、 Connection Lifetime、および Connection Reset というキーワードに影響します。	すべて
OdbcDataReader.GetSchemaTable	ODBC .NET Data Provider は、拡張記述情報を戻さないサーバーからは、拡張記述情報を検索できません。そのため、拡張記述をサポートしていないサーバー (影響を受けるサーバー) に接続している場合、 IDataReader.GetSchemaTable() から戻されたメタデータ表の中の以下の列は無効となります。 • IsReadOnly • IsUnique • IsAutoIncrement • BaseSchemaName • BaseCatalogName	DB2 for OS/390, バージョン 7 以降 DB2 for OS/400 DB2 for VM/VSE

ストアード・プロシージャー	ストアード・プロシージャーを呼び出すには、完全な ODBC 呼び出し構文を指定する必要があります。
	
	例えば、VARCHAR(10) をパラメーターとして取るストアード・プロシージャー MYPROC を呼び出すには、次のようにします。
	[C#]
	OdbcCommand cmd = odbcCon.CreateCommand();
	cmd.CommandType = CommandType.Text;
	cmd.CommandText = "{ CALL MYPROC(?) }"
	OdbcParameter p1 = cmd.CreateParameter();
	p1.Value = "Joe";
	p1.OdbcType = OdbcType.NVarChar;
	cmd.Parameters.Add(p1);
	cmd.ExecuteNonQuery();
	
	注: CommandType.StoredProcedure を使用している場合であっても、完全な ODBC 呼び出し構文を使用する必要があることに注意してください。 これは、MSDN の中の OdbcCommand.CommandText プロパティーの項で説明されています。
		すべて

ストアード・プロシージャー: 結果セットの列名はなし	DB2 for OS/390 バージョン 6.1 サーバーは、ストアード・プロシージャーから戻された結果セットの列名を戻しません。 ODBC .NET Data Provider は、これらの無名列を、その順序位置 (例えば、"1"、"2"、"3") にマップします。これは、MSDN で説明されているマッピング ("Column1", "Column2", "Column3") とは異なります。	DB2 for OS/390 バージョン 6.1
ユニーク索引の主キーへのプロモーション	ODBC .NET Data Provider は、NULL 可能なユニーク索引を主キーにプロモートします。これは MSDN の説明と異なります。 MSDN では、NULL 可能なユニーク索引は主キーにプロモートすべきでないと説明されています。	すべて

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 55. 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 55. 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 55. 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 56. 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]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[数字]

A

C

D

E

I

J

M

N

O

R

S

T

U

V

X

[特殊文字]

Printed in Japan

SA88-4665-00

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

ADO.NET および OLE DB アプリケーションの開発