DB2® ユニバーサル・データベース

アプリケーション構築の手引き

バージョン 7

SC88-8515-00
(英文原典：SC09-2948-00)

DB2® ユニバーサル・データベース

アプリケーション構築の手引き

バージョン 7

SC88-8515-00
(英文原典：SC09-2948-00)

ご注意!

本書、および本書がサポートする製品をご使用になる前に、 453ページの『付録E. 特記事項』にある一般的な情報を必ずお読みください。

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

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

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

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

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

(URL は、変更になる場合があります)

原 典:	SC09-2948-00 IBM® DB2® Universal Database Application Building Guide Version 7
発 行:	日本アイ・ビー・エム株式会社
担 当:	ナショナル・ランゲージ・サポート

第1刷 2000.6

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

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

目次

DB2 アプリケーション開発の紹介

このまえがきには、特に DB2 開発者版を使用して DB2 アプリケーション開発を始めるのに必要な情報が記載されています。

『DB2 開発者版』の節では、開発の際の特定の必要に合ったインストール情報を入手する方法について説明します。この情報を活用して、 IBM DB2 ユニバーサル開発者版 バージョン 7.1、または IBM DB2 パーソナル開発者版 バージョン 7.1 のどちらから DB2 をインストールする場合でも、最善の方法を決定することができます。

xiiページの『DB2 アプリケーション開発のための資料』の節では、アプリケーション開発に使用する DB2 ライブラリー内の主な資料について説明します。

xiiiページの『DB2 プログラミング・インターフェース』の節では、 DB2 アプリケーション開発に関係する重要なプログラミング・インターフェースの概念について説明します。

xxiiページの『DB2 機能』の節では、 DB2 のアプリケーション開発で使用できる主な機能について説明します。

DB2 開発者版

DB2 ユニバーサル・データベースには、アプリケーション開発に使用する製品パッケージとして、 DB2 パーソナル開発者版と DB2 ユニバーサル開発者版の 2 つがあります。 パーソナル開発者版には、 OS/2、Linux、および Windows 32 ビット・オペレーティング・システムで実行される、 DB2 ユニバーサル・データベースと DB2 コネクト パーソナル・エディションが入っています。 DB2 ユニバーサル開発者版には、パーソナル開発者版に入っている製品に加えて、 AIX、HP-UX、DYNIX/ptx、Silicon Graphics IRIX、および Solaris** Operating Environment** で実行される DB2 製品が入っています。

これらの製品に付属しているソフトウェアを使用して、実行するオペレーティング・システムは同じであっても、そのオペレーティング・システム上だけでなく、異なるオペレーティング・システム上にあるデータベースにもアクセスできるアプリケーションを開発およびテストすることができます。 たとえば、Windows NT オペレーティング・システムで実行されるものの、 AIX などの UNIX プラットフォーム上のデータベースにアクセスするようなアプリケーシ

ョンを作成できます。開発者版の製品の使用に関する条件については、ライセンスに関する合意事項を参照してください。

開発者版の箱に入っている CD-ROM には、アプリケーションの開発およびテストに必要なすべてのコードが含まれています。箱ごとに、以下のものが入っています。

- 複数のオペレーティング・システム用の DB2 ユニバーサル・データベース製品の CD-ROM。それぞれの CD-ROM には、サポートされているオペレーティング・システム用の DB2 サーバー、アドミニストレーション・クライアント、アプリケーション開発クライアント、ランタイム・クライアントが入っています。 これらの CD-ROM はアプリケーションをテストするために提供されています。貴社の必要に合わせてデータベースをインストールし、使用する場合、その製品を購入して、ユニバーサル・データベースの有効なライセンスを入手する必要があります。 パーソナル開発者版の箱には、OS/2、Linux、および Windows 32 ビット・オペレーティング・システムを含む、パーソナル版製品の CD-ROM が入っています。ユニバーサル開発者版の箱には、 DB2 がサポートしているすべてのオペレーティング・システム用の CD-ROM が入っています。
- 複数のオペレーティング・システム用の Netscape ブラウザーが入っている CD-ROM。ブラウザーがマシンにインストールされていない場合に、この CD を使用して、HTML 情報を表示できます。
- PDF 形式の DB2 資料を収録した DB2 資料 CD-ROM。
- サポートされているプラットフォーム用の DB2 エクステンダー。
- サポートされているプラットフォーム用の OLAP スターター・キット。OLAP スターター・キットをインストールする前に DB2 サーバーをインストールしなければなりません。
- さらに、アプリケーション開発で有用な他のソフトウェアの無料のコピーが入っています。 入っているソフトウェアが変更される場合があります。このような無料のソフトウェアには、使用に関するライセンスの合意事項があるため、必ずそれらの合意事項に従わなければなりません。

ユニバーサル開発者版には、以下のものも含まれています。

- すべてのプラットフォーム用のアドミニストレーション・クライアントが入っている CD-ROM のセット。 これらのクライアントには、コントロール・センターおよびイベント・アナライザーなどのデータベース管理用のツールが含まれています。これらのクライアントにより、どんなシステム上でもアプリケーションを実行できるようになります。

- すべてのプラットフォーム用のアプリケーション開発クライアントが入っている CD-ROM のセット。これらのクライアントには、アプリケーション開発ツール、サンプル・プログラム、およびヘッダー・ファイルが入っています。 各 DB2 AD クライアントには、アプリケーションを開発する際に必要なすべてのものが含まれています。
- すべてのプラットフォーム用のランタイム・クライアントが入っている CD-ROM のセット。任意のシステムにあるランタイム・クライアントからアプリケーションを実行することができます。ランタイム・クライアントには、DB2 のコントロール・センターやイベント・アナライザーなどの、アドミニストレーション・クライアントにはある機能が省略されているため、使用するスペースが少なくて済みます。
- サポートされているプラットフォーム用の Net.Data が入っている CD-ROM。

インストール情報

それぞれの DB2 製品 CD-ROM には、 CD-ROM から直接表示できる HTML 形式のインストール情報が含まれています。サポートされているプラットフォーム用の*概説およびインストール* 資料が提供されており、 DB2 サーバー、アドミニストレーション・クライアント、アプリケーション開発クライアント、およびランタイム・クライアントのインストール方法が説明されています。

追加情報については、*インストールおよび構成 補足* を参照してください。この資料は、クライアント CD-ROM からのみ表示可能です。

必要な資料は、ご使用の言語用のサブディレクトリー内にあります。 CD-ROM の README.TXT ファイルには、 CD-ROM 上の資料ファイルが入っている場所が説明されています。

ブラウザーを実行し、資料サブディレクトリー内の `index.htm` ファイルをクリックしてください。ここに、*概説およびインストール* および *インストールおよび構成 補足* 資料用のサブディレクトリーがあります。

db2i2 *DB2 ユニバーサル・データベース (OS/2 版) 概説およびインストール*

db2ix *DB2 ユニバーサル・データベース (UNIX 版) 概説およびインストール*

db2i6 *DB2 ユニバーサル・データベース (Windows 版) 概説およびインストール*

db2iy *インストールおよび構成 補足*

DB2 資料 CD-ROM には、製品に付属しているすべての DB2 資料用の PDF ファイルが入っています。それらの PDF ファイルは、Adobe Acrobat Reader

を使用して CD-ROM から直接表示できます。ご使用の言語で利用可能な DB2 資料は、該当する言語サブディレクトリー内にあります。インストールに関する情報を入手したい場合には、 *概説およびインストール* およびインストールおよび*構成 補足* 資料にアクセスすることができます。これらのファイル名は、上記にリストされている文字のセットで始まります。

PDF ファイルの印刷については、443ページの『PDF 資料の印刷』を参照してください。

資料ファイルへのアクセス方法の詳細については、 CD-ROM 上の README.TXT ファイルを参照してください。

DB2 アプリケーション開発のための資料

DB2 ライブラリーについては、 431ページの『付録D. DB2 ライブラリーの使用法』で説明されています。 DB2 資料は、 DB2 データベースの管理に関する情報を提供するものと、 DB2 アプリケーション開発に関する情報を提供するものの大きく 2 つに区分できます。中にはその両方の情報を扱っているものもあります。 DB2 アプリケーション開発者であれば、これらの両方の区分に属する DB2 資料を参照していることでしょう。それでも、 DB2 ライブラリーのアプリケーション開発に関するメイン資料を主に参照するはずです。この節ではそれらの資料について扱っています。

アプリケーションのプログラミングに使用するメイン資料は 2 つあります。*コール・レベル・インターフェースの手引きおよび解説書* では DB2 CLI アプリケーションのプログラミングについて扱っており、 *アプリケーション開発の手引き* では DB2 CLI 以外のすべての種類の DB2 プログラミングについて扱っています。また、これらの資料の両方に参照情報が含まれています。

本書、*アプリケーション構築の手引き* には、開発環境のセットアップに関する情報と、アプリケーションのコンパイル、リンク、および実行に関する情報が記載されています。

SQL ステートメントおよび関数の構文については、 *SQL 解説書* を参照してください。

DB2 ライブラリーの中で管理用として区分されるこれらの 2 つの資料は、アプリケーション・プログラミングに関する重要な参照資料でもあります。 *管理 API 解説書* には、 DB2 データベースの管理に使用するすべての管理機能に関する詳細な情報が含まれています。アプリケーションにおいて DB2 API を SQL ステートメントの代わりに使用したり、あるいは SQL ステートメントと

一緒に使用した方が便利な場合もあります。 *コマンド解説書* では、すべての DB2 コマンド (SQL ステートメントを除く) に関する詳細情報が扱われ、DB2 コマンド行プロセッサー (CLP) の使用方法について説明されています。

環境のセットアップ、またはプログラムの開発の際の問題を解決する場合は、*問題判別の手引き* を参照することができます。これは、DB2 顧客サービスを利用して、エラーの原因を判別したり、問題を回復したり、診断ツールを使用したりするのに役立ちます。 *メッセージ解説書* には、 DB2 エラー・メッセージの完全なリストと説明が記載されており、アプリケーションをデバッグする際の参照資料として非常に便利です。

DB2 ライブラリーのこれらの資料および他の資料は、 DB2 環境で見ることのできるオンライン情報と同様に、 DB2 アプリケーションを開発する際に必要な情報を提供しています。厳密には DB2 に関するものではない開発情報については、使用しているコンパイラー、インタープリター、または他の開発ツールの販売元が提供している資料を参照してください。

DB2 プログラミング・インターフェース

DB2 データベースを管理またはアクセスする際に、さまざまなプログラミング・インターフェースを使用することができます。 以下を行うことができます。

1. データベースのバックアップおよび復元などの管理機能を実行する場合は、DB2 API を使用します。
2. アプリケーションに静的および動的 SQL を組み込む。
3. アプリケーションで DB2 コール・レベル・インターフェース (DB2 CLI) 関数をコーディングして、動的 SQL ステートメントを呼び出す。
4. Java データベース・コネクティビティー・アプリケーション・プログラミング・インターフェース (JDBC API) を呼び出す、 Java アプリケーションおよびアプレットを開発する。
5. Data Access Object (DAO) および Remote Data Object (RDO) 仕様に準拠した Microsoft Visual Basic と Visual C++、およびオブジェクトのリンクと埋め込みデータベース (OLE DB) Bridge を使用する ActiveX Data Object (ADO) アプリケーションを開発する。
6. IBM のツール、または Net.Data、Excel、Perl などの他社のツール、および Lotus Approach などのオープン・データベース・コネクティビィティー (ODBC) エンド・ユーザー・ツールと、そのプログラム言語である Lotus Script を使用して、アプリケーションを開発する。

アプリケーションが DB2 データベースにアクセスする方法は、開発するアプリケーションのタイプによって異なります。たとえば、データ入力アプリケーションの場合、アプリケーションに静的 SQL ステートメントを組み込むことができます。ワールド・ワイド・ウェブ (WWW) を介して照会を実行するアプリケーションの場合、 Net.Data、Perl、または Java を選択できます。

組み込み SQL の使用

構造化照会言語 (SQL) は、 DB2 データベース内のデータへのアクセスおよび操作に使用するデータベース・インターフェース言語です。アプリケーションに SQL ステートメントを組み込めば、そのアプリケーションで、 SQL がサポートしている作業 (データの検索または保管など) を行うことができます。 DB2 を使用して、 C/C++、COBOL、FORTRAN、Java (SQLJ)、および REXX プログラム言語で、組み込み SQL アプリケーションをコーディングすることができます。

SQL ステートメントの組み込み先のアプリケーションをホスト・プログラムと呼びます。ホスト・プログラムの作成に使用するプログラム言語をホスト言語と呼びます。このようにプログラムおよび言語を定義するのは、それらが SQL ステートメントに対してホストの役割を果たす、つまりそれらの機能を提供しているためです。

静的 SQL ステートメントの場合、コンパイルを行う前に、ステートメント・タイプおよび表名と列名が決まっています。ただし、ステートメントが検索または更新を行う特定のデータ値は決められていません。それらの値は、ホスト言語変数で指定することができます。静的 SQL ステートメントのプリコンパイル、コンパイル、およびバインドは、アプリケーションを実行する前に行います。静的 SQL は、大幅な変更を行わないデータベースで実行するのに適しています。 それ以外の場合、静的ステートメントは、すぐに最新の情報に則したものではなくなってしまいます。

対照的に、動的 SQL ステートメントとは、実行時にアプリケーションが作成し、実行するステートメントです。エンド・ユーザーに対して SQL ステートメントの重要な部分 (検索する表および列の名前など) を求める対話式アプリケーションが、動的 SQL のよい例です。動的 SQL ステートメントはアプリケーションの実行中に作成され、それからステートメントの処理が実行依頼されます。

静的 SQL ステートメントか動的 SQL ステートメント、あるいはその両方を組み込んで、アプリケーションを作成することができます。

一般的に、静的 SQL ステートメントは、トランザクションが事前定義されている高性能のアプリケーションに向いています。予約システムなどが、そのようなアプリケーションのよい例です。

一般的に、動的 SQL ステートメントは、実行時にトランザクションを指定する必要がある、頻繁に変更が行われるデータベースに対して実行するアプリケーションに向いています。対話式照会インターフェースなどが、そのようなアプリケーションのよい例です。

アプリケーションに SQL ステートメントを組み込む場合、以下のステップに従って、プリコンパイルを行い、アプリケーションをデータベースにバインドしなければなりません。

1. SQL ステートメントを組み込んだプログラムを含むソース・ファイルを作成する。
2. データベースに接続してから、各ソース・ファイルをプリコンパイルする。

   各ソース・ファイル内の SQL ステートメントを、データベース・マネージャーへの DB2 実行時 API 呼び出しに変換します。また、プリコンパイラーはデータベース内にアクセス・パッケージを作成します。また、バインド・ファイルを作成するように指定すれば、任意選択でバインド・ファイルの作成も行います。

   アクセス・パッケージには、アプリケーション内の静的 SQL ステートメント用に、 DB2 最適化プログラムが選択したアクセス・プランが含まれています。アクセス・プランには、最適化プログラムが決定した最も有効な方法で、データベース・マネージャーが静的 SQL ステートメントを実行するのに必要な情報が含まれています。動的 SQL ステートメントの場合、アプリケーションの実行時に、最適化プログラムがアクセス・プランを作成します。

   バインド・ファイルには、アクセス・パッケージを作成するのに必要な、SQL ステートメントと他のデータが含まれています。このバインド・ファイルを使用して、後からアプリケーションを再バインドすることができます。その際に最初にプリコンパイルをする必要はありません。再バインドにより、現在のデータベースの状態に合わせて最適化されたアクセス・プランが作成されます。アプリケーションのプリコンパイルを行ったデータベースとは別のデータベースへアクセスする場合、そのアプリケーションを再バインドする必要があります。最後のバインド以降にデータベース統計を変更した場合、アプリケーションを再バインドするようお勧めします。
3. ホスト言語コンパイラーを使用して、変更したソース・ファイル (および SQL ステートメントを含まない他のファイル) をコンパイルする。

4. オブジェクト・ファイルを DB2 およびホスト言語ライブラリーとリンクさせ、実行可能プログラムを作成する。
5. バインド・ファイルをバインドし、アクセス・パッケージを作成する (プリコンパイル時に行っていなかった場合、あるいは別のデータベースにアクセスする場合)。
6. アプリケーションを実行する。アプリケーションが、パッケージ内のアクセス・プランを使用してデータベースにアクセスします。

Java Embedded SQL (SQLJ)

DB2 Java Embedded SQL (SQLJ) サポートは、 DB2 AD クライアントによって提供されます。 DB2 SQLJ サポートと DB2 JDBC サポートによって、SQLJ アプレット、アプリケーション、およびストアード・プロシージャーを構築し、実行できます。これらには、静的 SQL が含まれ、 DB2 データベースにバインドされた組み込み SQL ステートメントを使用します。

DB2 SQLJ サポートの詳細については、以下のアドレスの Web ページを参照してください。

```
http://www.ibm.com/software/data/db2/java
```

DB2 コール・レベル・インターフェースの使用

DB2 CLI は、 C および C++ アプリケーションが DB2 データベースにアクセスする際に使用できるプログラミング・インターフェースです。 DB2 CLI は、 Microsoft オープン・データベース・コネクティビティー (ODBC) 仕様と、 ISO CLI 規格に基づいています。 DB2 CLI は業界標準に基づいているため、それらのデータベース・インターフェースにすでに精通しているアプリケーション・プログラマーであれば、より短期間で習熟することができます。

DB2 CLI を使用する場合、アプリケーションは動的 SQL ステートメントを関数の引き数としてデータベース・マネージャーに渡し、処理します。そのようにして、 DB2 CLI は組み込み動的 SQL の代替としての役割を果たします。

CLI、ODBC、または JDBC アプリケーション内で静的 SQL として SQL ステートメントを実行することもできます。 CLI/ODBC/JDBC 静的プロファイル作成機能により、アプリケーションのエンド・ユーザーは、多くの場合に、動的 SQL の代わりに静的 SQL を使用することができるようになります。詳細については、以下を参照してください。

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

ODBC ドライバー・マネージャーを使用せずに、単に、UNIX 上の `libdb2` および OS/2 や Windows 32 ビット・オペレーティング・システム上の

db2cli.lib とアプリケーションをリンクすることによっていずれかのプラットフォーム上で DB2 の ODBC ドライバーを使用して、 ODBC アプリケーションを構築することができます。 DB2 CLI サンプル・プログラムは、ODBC アプリケーションの構築方法を例示しています。 DB2 CLI サンプル・プログラムは、 UNIX 上の sqllib/samples/cli および OS/2 や Windows 32 ビット・オペレーティング・システム上の %DB2PATH%¥samples¥cli にあります。

DB2 CLI アプリケーションは DB2 とともに提供されている共通アクセス・パッケージを使用するため、 DB2 CLI アプリケーションのプリコンパイルまたはバインドを行う必要はありません。必要なのは、アプリケーションのコンパイルとリンクだけです。

ただし、 DB2 AD クライアントに付属している DB2 CLI バインド・ファイルを、アクセスする DB2 データベースごとにバインドしてからでなければ、DB2 CLI または ODBC アプリケーションは DB2 データベースにアクセスできません。 これはデータベースへの最初の接続の際に自動的に行われますが、データベース管理者が、各プラットフォームごとに、 DB2 データベースにアクセスするそれぞれのクライアントからバインド・ファイルをバインドするようお勧めします。バインドに関する指示については、 50ページの『バインド』を参照してください。

たとえば、OS/2、AIX、および Windows 95 クライアントがあり、それぞれが 2 つの DB2 データベースにアクセスするとします。管理者は、アクセスするデータベースごとに、 1 つの OS/2 クライアントからバインド・ファイルをバインドしなければなりません。次に、管理者は、アクセスするデータベースごとに、 1 つの AIX クライアントからバインド・ファイルをバインドします。最後に、管理者は Windows 95 クライアントで同様の作業を行います。

DB2 CLI と組み込み動的 SQL の比較

動的アプリケーションを開発する際に、組み込み動的 SQL か DB2 CLI のいずれかを使用できます。どちらの場合でも、 SQL ステートメントは実行時に準備され、処理されます。各方式には、以下にリストしているように、それぞれに固有の利点があります。

DB2 CLI の利点

移行性 DB2 CLI アプリケーションはデータベースに SQL ステートメントを渡す際に、関数の標準セットを使用します。 DB2 CLI アプリケーションを実行する前に必要なのは、コンパイルとリンクだけです。対照的に、組み込み SQL アプリケーションは、プリコンパイルしてからコンパイルし、その後それらのアプリケーションをデータベースにバインド

してからでなければ実行できません。この処理により、アプリケーションを特定のデータベースに効果的に結び付けます。

バインドを行わない

DB2 CLI アプリケーションは、アクセスするデータベースごとに個別にバインドする必要がありません。必要なのは、すべての DB2 CLI アプリケーションに使用できる、 DB2 CLI に付属しているバインド・ファイルをバインドすることだけです。これにより、アプリケーションの管理に費やされる時間が大幅に減ります。

取り出しおよび入力の拡張

DB2 CLI 関数を使用すれば、一度の呼び出しで、データベース内の複数の行を 1 つの配列に取り出すことができます。また、入力変数の配列を使用して、 SQL ステートメントを何回も実行できます。

カタログに対する整合性インターフェース

データベース・システムには、データベースとそのユーザーに関する情報が入っているカタログ表が含まれています。これらのカタログの形式は、システム間で異なる場合があります。 DB2 CLI は、表、列、外部キー、基本キー、およびユーザー特権などの、構成要素に関するカタログ情報を照会する整合性インターフェースを提供します。これにより、データベース・サーバーのリリース間でのカタログの変更、およびデータベース・サーバー間での相違からアプリケーションが保護されます。特定のサーバーまたは製品のバージョンに固有のカタログ照会を作成する必要はありません。

データ変換の拡張

DB2 CLI は、 SQL と C のデータ・タイプ間で自動的にデータを変換します。たとえば、 SQL データ・タイプを C 文字データ・タイプに取り出すと、文字ストリング表示に変換されます。そのため、 DB2 CLI は対話式照会アプリケーションに適しています。

グローバル・データ域がない

一般に、組み込み SQL アプリケーションには、アプリケーションが制御するグローバル・データ域 (SQLDA および SQLCA など) が関連付けられており、それらはしばしば複雑になることがあります。しかし、DB2 CLI にはグローバル・データ域が必要ではありません。その代わりに、 DB2 CLI が、必要なデータ構造を自動的に割り当てて制御し、それらを参照するためのアプリケーションのハンドルを提供します。

ストアード・プロシージャーからの結果セットの取り出し

DB2 CLI アプリケーションは、サーバー上にあるストアード・プロシージャーから生成された複数の行および結果セットを取り出します。

スクロール可能カーソル

DB2 CLI は、サーバー側のスクロール可能カーソルをサポートしています。これは、配列出力と併用することができます。これは、スクロール・ボックス (Page Up、Page Down、Home、および End キーを使用する) 内のデータベース情報を表示する GUI アプリケーションで役に立ちます。カーソルをスクロール可能と宣言すれば、 1 行または複数行ごとに結果セット内を下方または上方に移動できます。また、現在の行からの相対位置や、結果セットの最初または最後からの相対位置、あるいはブックマークが既に付けられている特定の行からの相対位置を指定して、行を取り出すこともできます。

組み込み動的 SQL の利点

すべての DB2 CLI ユーザーは、同じ権限を共有します。組み込み SQL の利点は、パッケージ用の特定のユーザーに実行権限を付与することによってさらにきめ細かなセキュリティーを提供することです。

組み込み SQL は、C および C++ 以外の言語もサポートしています。アプリケーションを別の言語でコーディングする場合、この点が利点と言えるかもしれません。

動的 SQL は一般に静的 SQL よりも整合性があります。すでに静的 SQL のプログラミング方法を知っている場合、動的 SQL に移行する方が DB2 CLI に移行するよりも簡単です。

Java データベース・コネクティビティー (JDBC) の使用

DB2 の Java サポートには JDBC が含まれます。 JDBC はベンダーに依存しない動的 SQL インターフェースで、標準化された Java メソッドを使用した、アプリケーションへのデータ・アクセスを提供します。 JDBC は、 JDBC プログラムのプリコンパイルまたはバインドを必要としないという点で、 DB2 CLI と類似しています。ベンダーに依存しない規格であるため、 JDBC アプリケーションは移行が容易です。 JDBC を使用して作成されたアプリケーションは、動的 SQL だけを使用します。

JDBC は、インターネットを介して DB2 データベースにアクセスする場合に特に便利です。 Java プログラム言語を使用して、ネットワーク接続を使ってリモート DB2 データベース内のデータにアクセスしたり、操作したりする JDBC アプリケーションおよびアプレットを開発できます。また、サーバー上に常駐し、データベース・サーバーにアクセスしたり、ストアード・プロシージャーを呼び出したリモート・クライアント・アプリケーションに情報を戻す、 JDBC ストアード・プロシージャーを作成することもできます。

JDBC API (CLI/ODBC API と類似している) は、 Java コードからデータベースにアクセスする標準的な方法を提供しています。 Java コードは、関数の引き数として SQL ステートメントを DB2 JDBC ドライバーに渡します。ドライバーはクライアントの Java コードから JDBC API 呼び出しを処理します。

Java は移行性が高いため、 DB2 アクセスを複数のプラットフォーム上のクライアントに渡すことができます。しかも、その際に必要なのは Java 対応 Web ブラウザーだけです。

Java アプリケーションでは、 DB2 に接続するために DB2 クライアントが必要です。他のアプリケーションと同様に、デスクトップまたはコマンド行からアプリケーションを開始します。 DB2 JDBC ドライバーは、アプリケーションからの JDBC API 呼び出しを処理し、クライアント接続を使用して要求をサーバーに通信し、その結果を受け取ります。

Java アプレットでは DB2 クライアント接続は必要ありません。通常、アプレットはハイパーテキスト・マークアップ言語 (HTML) の Web ページに組み込みます。

アプレットを実行する際に必要なのは、クライアント・マシンで Java 対応ブラウザーまたはアプレット・ビューアーを使えるようにすることだけです。 HTML ページをロードすると、ブラウザーが Java アプレットをユーザーのマシンにダウンロードし、さらに Java クラス・ファイルおよび DB2 の JDBC ドライバーもダウンロードします。アプレットが DB2 に接続するために JDBC API を呼び出すと、 JDBC ドライバーは、Web サーバーにある JDBC アプレット・サーバーを介して、 DB2 データベースとの個別のネットワーク接続を確立します。

DB2 JDBC サポートの詳細については、以下のアドレスの Web ページを参照してください。

```
http://www.ibm.com/software/data/db2/java
```

DB2 API の使用

アプリケーションでいくつかのデータベース管理作業 (データベースの作成、活動化、バックアップ、または復元など) を実行しなければならない場合があります。 DB2 にはたくさんの API があるため、ご使用のアプリケーション (組み込み SQL および DB2 CLI アプリケーションを含む) から、これらの作業を実行できます。これにより、 xxixページの『DB2 ユニバーサル・データベース・ツール』で説明されている、 DB2 サーバー管理ツールを使用して実行できるのと同じ管理機能を、アプリケーションにプログラミングすることができます。

さらに、 DB2 API を使用してしか実行できない特定の作業を行わなければならない場合があります。たとえば、エラー・メッセージのテキストを取り出し、アプリケーションがエンド・ユーザーに対してそのテキストを表示できるようにしたい場合もあるかもしれません。メッセージを取り出すには、「Get Error Message (エラー・メッセージの入手)」という API を使用しなければなりません。

ActiveX Data Objects (ADO) および Remote Data Objects (RDO) の使用

Data Access Object (DAO) および Remote Data Object (RDO) 仕様に準拠した、 Microsoft Visual Basic および Visual C++ データベース・アプリケーションを開発することができます。さらに、 DB2 では ODBC Bridge に Microsoft OLE DB を使用する ActiveX Data Object (ADO) アプリケーションもサポートしています。

ActiveX Data Object (ADO) を使用すれば、 OLE DB Provider を使用して、データベース・サーバー内のデータにアクセスしたり、操作したりするアプリケーションを作成できます。 ADO の主要な利点は、開発速度が速く、使用が容易で、ディスク・フットプリントが小さいことです。

Remote Data Objects (RDO) は、 ODBC を介してリモート・データ・ソースにアクセスするための情報モデルを提供します。 RDO が提供するオブジェクトのセットを使用すれば、データベースへの接続、照会の実行、ストアード・プロシージャーの実行、結果の操作、変更のサーバーへのコミットが容易になります。これは、リモート ODBC リレーショナル・データ・ソースへアクセスするために特別に設計されたものであり、複雑なアプリケーション・コードを使用せずに ODBC を使用することが容易になっています。

IBM、他社、および ODBC のエンド・ユーザー・ツールの使用

データベースの照会などの基本作業を行う場合、 Net.Data または Perl を使用できます。

Net.Data を使用することにより、 Web アプリケーションを介して、 DB2 データへのインターネットおよびイントラネット・アクセスを行うことができます。これは、共通ゲートウェイ・インターフェース (CGI) アプリケーションよりも高いパフォーマンスを提供する、 Web サーバー・インターフェース (API) を活用します。 Net.Data は、 Java、REXX、Perl、および C++ などの言語を使用したサーバー側での処理の他に、クライアント側の処理もサポートしています。 Net.Data は条件付きの論理および豊富なマクロ言語を提供しています。 Net.Data の Web ページは以下のアドレスにあります。

```
http://www.ibm.com/software/data/net.data/
```

DB2 は、 DBD::DB2 ドライバーを介したデータ・アクセスに使用する、 Perl データベース・インターフェース (DBI) 仕様をサポートしています。 DB2 ユニバーサル・データベース Perl DBI の Web サイトは、以下のアドレスにあります。

```
http://www.ibm.com/software/data/db2/perl/
```

このサイトで、最新の DBD::DB2 ドライバーと関連情報を入手できます。

また、Lotus Approach、Microsoft Access、および Microsoft Visual Basic などの、 ODBC エンド・ユーザー・ツールを使用して、これらの作業を行うアプリケーションを作成できます。 ODBC ツールを使用すれば、高水準プログラム言語を使用する場合よりも簡単にアプリケーションを開発できます。

Lotus Approach には、DB2 データにアクセスする方法が 2 つあります。 1 つ目は、グラフィカル・インターフェースを使用して、照会の実行、レポートの作成、およびデータの分析を行う方法です。もう 1 つは、 LotusScript を使用してアプリケーションを開発する方法です。 LotusScript は完全仕様の、オブジェクト指向プログラム言語で、多岐にわたるオブジェクト、イベント、メソッド、およびプロパティーを備えており、プログラム・エディターが組み込まれています。

DB2 機能

DB2 には、サーバー上で実行するさまざまな機能があります。それらの機能を使用して、アプリケーションを補足したり、拡張したりすることができます。 DB2 機能を使用する場合、同じ作業を行うために独自のコードを作成する必要はありません。また DB2 では、クライアント・アプリケーションに全コードを保持するのではなく、サーバーにコードの一部を保管します。これにより、パフォーマンスが向上し、保守が容易になります。

データの保護、およびデータ間の相互関係を定義する機能があります。さらに、柔軟な拡張アプリケーションを作成する、オブジェクト関連の機能もあります。機能の中には、複数の方法で使用できるものもあります。たとえば、制約はデータの保護と、データ値間の相互関係の定義に使用できます。主な DB2 機能を以下にリストします。

- 制約
- ユーザー定義タイプ (UDT) およびラージ・オブジェクト (LOB)
- ユーザー定義関数 (UDF)
- トリガー
- ストアード・プロシージャー

DB2 機能を使用するかどうか決定する場合、以下の点を考慮してください。

アプリケーションの独立性

アプリケーションを、それが処理するデータから独立させることができます。データベース上で実行する DB2 機能を使用すれば、アプリケーションに影響を与えることなくデータに使用する論理を保守したり、変更したりすることができます。その論理を変更しなければならない場合、変更を行うのはサーバーだけで、データにアクセスする各アプリケーションでは変更を行う必要はありません。

パフォーマンス

サーバー上にアプリケーションの一部を保管し、実行することにより、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。これにより、いくつかの処理を、一般により強力なサーバー・マシンに移し、クライアント・アプリケーションとサーバーの間のネットワーク通信量を減らすことができます。

アプリケーション要件

ご使用のアプリケーションが、他のアプリケーションとは異なる独自の論理を使用している場合があります。たとえば、ご使用のアプリケーションが、他のアプリケーションでは適切ではない特定の順序でデータ入力エラーを処理する場合、そのような状況を処理するために独自のコードを作成する必要があります。

DB2 機能は複数のアプリケーションで使用できるため、ときには、サーバー上で実行する DB2 機能を使用することがあるかもしれません。その一方で、自分のアプリケーションだけが論理を使用するため、それを自分のアプリケーションに保持してしまう場合もあります。

制約

データを保護したり、データ間の相互関係を定義する場合、通常は内部規則を定義します。これらの規則では、表の列で有効なデータ値や、複数の表内の列がお互いにどのように関連付けられているかを定義します。

DB2 が提供する制約は、データベース・システムを使用してこれらの規則を施行するという方法です。データベース・システムを使用して内部規則を施行することにより、アプリケーションでコードを作成し、内部規則を施行する必要がなくなります。ただし、内部規則が 1 つのアプリケーションだけに適用される場合、グローバル・データベース制約を使用するのではなく、アプリケーションでコーディングを行わなければなりません。

DB2 は以下の種類の制約を提供しています。

1. NOT NULL 制約
2. UNIQUE 制約
3. PRIMARY KEY 制約
4. FOREIGN KEY 制約
5. CHECK 制約

SQL ステートメント CREATE TABLE および ALTER TABLE を使用して、制約を定義します。

ユーザー定義タイプ (UDT) およびラージ・オブジェクト (LOB)

データベース内の各データ要素は表の列に保管され、各列はデータ・タイプを持つように定義されます。データ・タイプは列に入れることができる値のタイプと、その値に対して実行できる操作を制限します。たとえば、データ・タイプが整数である列に入れることができるのは、決められた範囲内の数字だけです。 DB2 には、特性と振る舞いが定義されている、組み込みデータ・タイプ (文字ストリング、数値、日時値、ラージ・オブジェクト、ヌル、漢字ストリング、2 進ストリング、およびデータ・リンク) のセットが含まれています。

しかし、組み込みデータ・タイプではアプリケーションの必要を満たさない場合もあります。 DB2 が提供しているユーザー定義タイプ (UDT) を使用すれば、アプリケーションが必要とする特殊なデータ・タイプを定義できます。

UDT は組み込みデータ・タイプに基づいています。 UDT を定義する場合、その UDT に有効な操作も定義します。たとえば、 DECIMAL データ・タイプに基づいて、 MONEY というデータ・タイプを定義することができます。しかし、MONEY データ・タイプで行える操作は加算と減算だけで、乗算と除算の操作は行えません。

ラージ・オブジェクト (LOB) を使用すれば、データベース内の大きくて複雑なオブジェクト (音声、ビデオ、画像、および大きい文書など) を保管したり、操作したりすることができます。

UDT と LOB を組み合わせると、非常に便利です。内部データをモデル化したり、そのデータのセマンティクスを取り込んだりする際に、 DB2 が提供している組み込みデータ・タイプを使用するように制限されることがなくなります。 UDT を使用して、拡張アプリケーションで使用する、大きくて複雑なデータ構造体を定義することができます。

組み込みデータ・タイプの拡張に加えて、 UDT には他に以下の利点があります。

アプリケーション内のオブジェクト指向プログラミングのサポート

類似したオブジェクトを関連データ・タイプにグループ化できます。これらのタイプには、名前、内部表記、および特定の振る舞いを指定できます。 UDT を使用して、新しいタイプの名前、および内部での表記方法を DB2 に知らせることができます。 LOB は新しいタイプの内部表記として使用できるものの 1 つで、大きくて複雑なデータ構造体には最も適しています。

強力なタイプ指定およびカプセル化によるデータ保全性

強力なタイプ指定により、特殊タイプで定義された関数と操作しかそのタイプには適用されません。カプセル化により、 UDT の振る舞いは UDT に適用できる関数と操作によって確実に制限されます。 DB2 では、 UDT の振る舞いをユーザー定義関数 (UDF) の形式で指定できます。 UDF はユーザーの要件に幅広く適応するように作成できます。

データベース・マネージャーへの組み込みによるパフォーマンス

UDT は、組み込みデータ・タイプと同じ方法で内部的に表記されるため、組み込みデータ・タイプと同じ有効なコードを共有して、組み込み関数、比較演算子、索引、および他の関数を実装します。例外は、LOB を使用する UDT です。 LOB は比較演算子および索引と併用することはできません。

ストアード・プロシージャー

多くの場合、アプリケーションはネットワークを介してデータベースにアクセスします。このため、たくさんのデータが戻されると、パフォーマンスが低下します。ストアード・プロシージャーはデータベース・サーバーで実行します。クライアント・アプリケーションはストアード・プロシージャーを呼び出すことができます。ストアード・プロシージャーはデータベースへのアクセスを実行しますが、その際にネットワークを介して必要のないデータを戻すことはありません。クライアント・アプリケーションが必要とする結果だけが、ストアード・プロシージャーによって戻されます。

ストアード・プロシージャーの使用には以下の利点があります。

ネットワーク通信量が削減される

SQL ステートメントをグループ化することにより、ネットワーク通信量を節約できます。典型的なアプリケーションの場合、 SQL ステートメントごとに、ネットワークを介した 2 回のトリップが必要です。 SQL ステートメントをグループ化しておけば、ステートメントのグループごとに、ネットワークを介した 2 回のトリップを行うだけで済みます。これにより、アプリケーションのパフォーマンスが向上します。

サーバー上にしかない機能へのアクセス

ストアード・プロシージャーはサーバー上でしか実行されないコマンド (LIST DATABASE DIRECTORY や LIST NODE DIRECTORY など) にアクセスできます。これには、サーバー・マシンのメモリーおよびディスク・スペースが増加する場合があるという利点があります。さらに、サーバーにインストールされている任意の追加ソフトウェアにアクセスすることもできます。

内部規則の施行

ストアード・プロシージャーを使用して、複数のアプリケーションに共通の内部規則を定義できます。これは、制約およびトリガーを使用する方法に加えて、内部規則を定義する別の方法です。

アプリケーションがストアード・プロシージャーを呼び出すと、ストアード・プロシージャーで定義した規則に従って、整合性が確保される方法でデータが処理されます。規則を変更しなければならない場合、変更は、ストアード・プロシージャーを呼び出すアプリケーションごとに行うのではなく、ストアード・プロシージャー内で一度行うだけで済みます。

ユーザー定義関数 (UDF)

SQL で提供されている組み込み機能では、アプリケーションの必要すべてを満たせない場合があります。それらの機能を拡張するために、 DB2 はユーザー定義関数 (UDF) をサポートしています。 Visual Basic、C/C++、または Java で独自のコードを作成し、単一のスカラー値または表を戻す SQL ステートメント内で操作を実行することができます。

UDF により柔軟性が著しく向上します。 UDF は選択リストの一部としてデータベースから単一のスカラー値を戻すことができます。また、スプレッドシートなどの、非データベース・ソースから表全体を戻すこともできます。

UDF を使用することにより、アプリケーションを標準化することができます。ユーザー定義関数の共通セットを実装することにより、複数のアプリケーションが同じ方法でデータを処理できるので、整合性のある結果を確実に入手できます。

また、ユーザー定義関数は、アプリケーション内のオブジェクト指向プログラミングをサポートしています。 UDF は抽象化を提供しており、これによりデータ・オブジェクトへの操作を実行するのに使用できるメソッドを定義することができます。さらに、 UDF が提供しているカプセル化を使用すれば、オブ

ジェクトの基本データへのアクセスを制御したり、直接操作や起きるかもしれない破壊からオブジェクトを保護することができます。

OLE DB 表関数

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への一様な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。 DB2 ユニバーサル・データベースは、 OLE DB データ・ソースにアクセスする表関数を定義できるようにすることにより、 OLE DB アプリケーションの作成を単純化しています。 GROUP BY、JOIN、および UNION を含む操作を、 OLE DB によってデータを公開しているデータ・ソースに対して実行することができます。たとえば、 Microsoft Access データベースまたは Microsoft Exchange アドレス帳から表を戻す OLE DB 表関数を定義し、その OLE DB 表関数からのデータと、 DB2 データベース内のデータとを完全に結合させることができます。

OLE DB 表関数を使用すると、 OLE DB Provider への組み込みアクセスが提供されるため、アプリケーション開発に費やす労力が軽減されます。データを取り出すために、開発者は、C、Java、および OLE オートメーション表関数の場合は、その表関数を実装し、OLE DB 表関数の場合は、任意の OLE DB Provider との汎用組み込み OLE DB 消費者インターフェースを実装しなければなりません。その際に必要なのは、言語タイプ OLEDB の表関数を登録し、OLE DB Provider と、関係のある行セットをデータ・ソースとして参照することだけです。 OLE DB 表関数を使用するために、なんらかの UDF プログラミングを行う必要はありません。

OLE オートメーション UDF およびストアード・プロシージャー

OLE (オブジェクトのリンクと埋め込み) オートメーションは、 Microsoft Corporation による OLE 2.0 アーキテクチャーの一部です。 OLE を使用すれば、アプリケーションの作成に使用した言語に関係なく、そのアプリケーションで OLE オートメーション・オブジェクトのプロパティーおよびメソッドを公開できます。そうすると、 Lotus Notes や Microsoft Exchange などの他のアプリケーションが、 OLE オートメーションを介してこれらのプロパティーおよびメソッドを利用することにより、これらのオブジェクトを統合することができます。

DB2 (Windows 32 ビット版) は、 UDF およびストアード・プロシージャーを使用する、 OLE オートメーション・オブジェクトへのアクセスを提供しています。 OLE オートメーション・オブジェクトにアクセスし、そのメソッドを呼び出すには、そのオブジェクトのメソッドを UDF またはストアード・プロシージャーとして登録しなければなりません。 その場合、DB2 アプリケーシ

ョンは UDF またはストアード・プロシージャーを呼び出すことによって、メソッドを呼び出します。 UDF はスカラー関数または表関数にできます。

たとえば、 Microsoft Excel などの製品を使用して作成した、スプレッドシート内のデータを照会するアプリケーションを開発できます。このようなアプリケーションを開発する場合、ワークシートからデータを取り出し、DB2 に戻す、 OLE オートメーション表関数を開発します。そうすると、 DB2 はそのデータを処理し、オンライン分析処理 (OLAP) を実行して、照会結果をアプリケーションに戻します。

トリガー

トリガーは、指定した表に対する削除、挿入、または更新操作によって実行されるアクションのセットを定義します。そのような SQL 操作の実行時に、トリガーが活動化されるように指示が出されます。トリガーの活動化は、 SQL 操作の前と後のどちらでも実行することができます。トリガーは SQL ステートメント CREATE TRIGGER を使用して定義します。

更新または挿入の前に実行するトリガーは、以下のいくつかの用途に使用できます。

- データベース内で実際に値の更新または挿入を行う前に、値の検査や変更を行う。これは、ユーザーが見ることのできる方法から何らかの内部データベース形式に、データを変換しなければならない場合に便利です。
- ユーザー定義関数でコーディングされている、他の非データベース操作を実行する。

同様に、更新または挿入の後に実行するトリガーは、以下のいくつかの用途に使用できます。

- 他の表のデータを更新する。これは、データ間の相互関係を保守したり、監査証跡情報を保持したりする際に便利です。
- その表または他の表内の、他のデータに対して検査を行う。これは、参照保全制約が適切でない場合、または表検査制約によって検査が現在の表だけに制限されている場合に、データ保全性を確保するのに役立ちます。
- ユーザー定義関数でコーディングされている、非データベース操作を実行する。これは、アラートを出す場合や、そのデータベース以外の場所にある情報を更新する場合に役立ちます。

トリガーの使用には以下の利点があります。

アプリケーション開発に費やされる時間が短縮される

トリガーはデータベース内に保管され、すべてのアプリケーションに対

して使用できます。このため、アプリケーションごとに同等の関数をコーディングする必要がなくなります。

内部規則のグローバル制約
トリガーをいったん定義すると、そのトリガーによって管理されるデータを使用するすべてのアプリケーションがそのトリガーを使用します。

保守が容易になる
何らかの変更を行う場合、トリガーを使用するアプリケーションごとにではなく、データベース内で一度変更を行うだけで済みます。

DB2 ユニバーサル・データベース・ツール

アプリケーションを開発する際に、さまざまなツールを使用できます。 DB2 ユニバーサル・データベースは、アプリケーションでの SQL ステートメントの作成とテスト、およびパフォーマンスのモニターを行うのに役立つ、以下のツールを提供しています。

注: すべてのツールがすべてのプラットフォーム上で使用可能であるとは限りません。

コントロール・センター

データベース・オブジェクト (データベース、表、およびパッケージなど) とそれらの相互関係を表示するグラフィカル・インターフェース。システムの構成、ディレクトリーの管理、システムのバックアップと回復、ジョブのスケジューリングおよびメディアの管理などの管理用タスクを実行するには、コントロール・センターを使用してください。

コントロール・センターには、以下の機能が組み込まれています。

コマンド・センター
対話式ウィンドウに DB2 コマンドや SQL ステートメントを入力したり、結果ウィンドウに実行結果を表示するのに使用します。結果をスクロールさせたり、出力をファイルに保管することができます。

スクリプト・センター
スクリプトを作成するのに使用します。スクリプトは保管して後に起動することができます。スクリプトには、DB2 コマンド、SQL ステートメント、またはオペレーティング・システム・コマンドを入れることができます。スクリプトが不在実行されるようスケジュールすることもできます。これらのジョブは、一度だけ実行するか、繰り返しスケジュールによって実行されるよう設定することができます。繰り返しスケジュールは、バックアップなどのタスクの場合に特に役立ちます。

ジャーナル
次のタイプの情報を表示するのに使用します。実行を保留しているか、実行中であるか、実行を完了したジョブに関するすべての入手可能な情報、回復活動記録ログ、アラート・ログ、およびメッセージ・ログです。また、ジャーナルを使用して、不在実行されたジョブの結果を検討することもできます。

アラート・センター
潜在的な問題を早期に警告するようにシステムをモニターしたり、問題を訂正するための処置を自動化するのに使用します。

ツール設定
コントロール・センター、アラート・センター、および複製の設定を変更するのに使用します。

パフォーマンス・モニター

コントロール・センターのインストール可能オプション。パフォーマンス・モニターは、包括的なパフォーマンス・データ収集、表示、報告、分析、およびアラート機能を DB2 システムに提供する、グラフィカル・インターフェースです。パフォーマンス・モニターは、パフォーマンスの調整に使用します。

スナップショットかイベントをモニターするよう選択することができます。スナップショット・モニターを使用すれば、指定した間隔で特定時の情報を取り込むことができます。イベント・モニターを使用すれば、接続などのイベント中のパフォーマンス情報を記録することができます。

Visual Explain

コントロール・センターのインストール可能オプション。 Visual Explain は、SQL ステートメントの最適化プログラムが選択したアクセス・プランの表示を含め、 SQL ステートメントの分析と調整を行うことができるグラフィカル・インターフェースです。

ストアード・プロシージャー・ビルダー (SPB)

DB2 ストアード・プロシージャーの迅速な開発をサポートする GUI ベースのツール。 ワークステーションから OS/390 に至る範囲の DB2 ファミリーに、単一の開発環境を提供します。 Windows 32 ビット・オペレーティング・システムでは、一般的なアプリケーション開発ツール、Microsoft Visual Studio、Microsoft Visual Basic、および IBM VisualAge Age for Java から立ち上げるか、または IBM DB2 ユニバーサル・データベース・プログラム・グループか

ら個別のアプリケーションとして立ち上げることができます。 以下のファイルを実行することによって開始することもできます。

```
%DB2PATH%¥bin¥DB2SPB.exe
```

ここで、%DB2PATH% は、 DB2 がインストールされているディレクトリーを示しています。

AIX および Solaris では、ストアード・プロシージャー・ビルダーは、 db2spb コマンドで開始できます。

第1章 概要

本書は、DB2 アプリケーションを開発するための環境をセットアップするのに必要な情報、また開発したアプリケーションをこの環境でコンパイル、リンク、および実行するための手順を段階的に説明しています。本書は、DB2 ユニバーサル・データベース バージョン 7.1 用の DB2 アプリケーション開発 (DB2 AD) クライアントを使用して以下のプラットフォーム用のアプリケーションを作成する方法を説明しています。

- AIX
- HP-UX
- Linux
- OS/2
- DYNIX/ptx
- Silicon Graphics IRIX
- Solaris 実行環境版
- Windows 32 ビット・オペレーティング・システム

注:

1. DB2 (NUMA-Q 版) は、DYNIX/ptx オペレーティング・システムをサポートします。
2. Windows 32 ビット・オペレーティング・システムとは、 Windows NT、Windows 95、Windows 98、および Windows 2000 のことです。 本書

で Windows 32 ビット・オペレーティング・システムに言及している場合は必ず、システム・ネットワーク体系 (SNA) サポートおよび REXX サポートの場合を除き、これら 4 つを暗黙に示しています。 これらは Windows NT および Windows 2000 上だけでサポートされます。

アプリケーションを開発するには、次のプログラミング・インターフェースを使用できます。

DB2 アプリケーション・プログラミング・インターフェース (DB2 API)
DB2 を管理するための管理機能を提供します。

DB2 コール・レベル・インターフェース (DB2 CLI)
これは、X/Open CLI 仕様に基づく呼び出し可能 SQL インターフェースであり、 Microsoft Corporation の ODBC インターフェースと互換性があります。

組み込み SQL
実行時関数呼び出しに変換するためにプリコンパイルする必要のあるプログラムで、直接コーディングされる SQL ステートメントを使用します。

組み込み SQL for Java (SQLJ)
順番にデータベース・マネージャーにインターフェースを提供する実行時関数呼び出しに、プリコンパイルおよびカスタマイズされる生成済みプロファイルで SQL ステートメントを使用します。

Java データベース・コネクティビティー (JDBC)
Java 用の動的 SQL API です。 JDBC API は、サポートされるプラットフォームで入手可能な Java 開発者キットに入っています。

各プログラミング・インターフェースの詳細については、次を参照してください。

- *アプリケーション開発の手引き*。組み込み SQL、 および Java データベース・コネクティビティー (JDBC) を使用して、 DB2 ファミリー・サーバー機能にアクセスするアプリケーション・プログラムをコーディングおよび設計する方法について解説しています。 ユーザー定義関数 (UDF) についても説明しています。
- *コール・レベル・インターフェースの手引きおよび解説書*。 DB2 コール・レベル・インターフェースおよび ODBC を使用するアプリケーション・プログラムをコーディングおよび設計する方法について解説したものです。
- *管理 API 解説書*。 DB2 アプリケーション・プログラミング・インターフェースを使用するアプリケーション・プログラムをコーディングおよび設計する方法について解説したものです。

次の資料には、製品のインストールおよびセットアップなどについての、役立つ関連情報が記載してあります。

- *DB2 ユニバーサル・データベース (OS/2 版) 概説およびインストール*。データベース・マネージャーおよび DB2 アプリケーション開発クライアントを、 OS/2 サーバーおよびクライアントのワークステーションにインストールする方法を説明します。
- *DB2 ユニバーサル・データベース (UNIX 版) 概説およびインストール*。データベース・マネージャーおよび DB2 アプリケーション開発クライアントを、 UNIX サーバーおよびクライアントのワークステーションにインストールする方法を説明します。
- *DB2 ユニバーサル・データベース (Windows 版) 概説およびインストール*。データベース・マネージャーおよび DB2 アプリケーション開発クライアントを、 Windows 32 ビット・オペレーティング・システム用のサーバーおよびクライアントのワークステーションにインストールする方法を説明します。
- *コマンド解説書*。DB2 コマンド行プロセッサー (CLP)、およびすべての非 SQL DB2 コマンドの使用法を説明しています。
- *問題判別の手引き*。DB2 クライアントおよびサーバーに関係したアプリケーション開発上の問題や、データベース管理および接続性での関連タスクの問題を解決するのに役立ちます。

DB2 資料ライブラリーの完全なリストは、 431ページの『付録D. DB2 ライブラリーの使用法』を参照してください。

注: 本書に記載されている例は、保証はまったくありませんが、「そのまま」使用できます。ユーザーおよび IBM 以外の方は、品質、パフォーマンス、何らかの欠陥の訂正のすべてのリスクをご承知いただきます。

本書の対象読者

本書は、現在 DB2 ユニバーサル・データベース バージョン 7.1 でサポートされているプラットフォームの 1 つで、プログラムを開発したい場合に使用します。 本書は、ご使用のプログラムが DB2 API を使用して DB2 データベースを管理する方法、および DB2 CLI、組み込み SQL、SQLJ、および JDBC を使用して DB2 データベースにアクセスする方法を説明します。

本書を使用するために、使用しているプラットフォームでサポートされるプログラム言語を 1 つ以上知っている必要があります。それらの言語は、8ページの『各プラットフォームでサポートされるソフトウェア』にリストされています。

本書の使用法

本書は、アプリケーション開発のために必要な情報を容易に入手できるように設計されています。章は、以下のようにグループ分けされます。

- 第 1 章～第 3 章: それぞれの章には、すべてのプラットフォームに関する一般的な概説情報が含まれています。
- 第 4 章および第 5 章: それぞれの章には、すべてのプラットフォームに関する特定のプログラミング情報が含まれています。
- 第 6 章～第 13 章: それぞれの章には、 1 つのプラットフォームに特有のプログラミング情報が含まれています。

すべてのアプリケーション開発者は、最初の 3 つの章をお読みになってから、使用するオペレーティング・システムおよびプログラミング言語によって、必要な特定のプログラミング情報を含む『アプリケーションの構築』の章をお読みください。

付録は、さまざまなトピックに関する追加情報を提供します。

強調表示の規則

本書では、以下の規則を使用しています。

イタリック (Italics)

イタリックは、以下のいずれかを示します。

- 新しい用語の紹介
- ユーザーによって指定される名前または値
- 別の情報源への参照
- 一般的な強調

大文字 (UPPERCASE)

大文字は、以下のいずれかを示します。

- データベース・マネージャーのデータ・タイプ
- フィールド名
- キーワード
- SQL ステートメント

例文 (Example text)

この書体のサンプル・テキストは、以下のいずれかを示します。

- コーディング例またはコードの一部
- コマンド
- 出力例、システムによって表示されるものと類似
- 特定のデータ値の例

- システム・メッセージの例
- ファイルおよびディレクトリー名
- ユーザーが入力する情報

太字 (Bold)
要点の強調を示します。

DB2 アプリケーション開発クライアントについて

注: アプリケーション開発クライアントは、 DB2 の旧バージョンでは、DB2 ソフトウェア開発者キット (DB2 SDK) クライアントとして知られていました。

DB2 アプリケーション開発 (DB2 AD) クライアントは、分散関係データベース体系 (DRDA) を実現する DB2 サーバーおよびアプリケーション・サーバーにアクセスするアプリケーションを開発するのに必要なツールおよび環境を提供します。

DB2 AD クライアントをインストールすれば、DB2 アプリケーションを構築し、実行することができます。次の DB2 クライアントで DB2 アプリケーションを実行することもできます。

- DB2 ランタイム・クライアント
- DB2 アドミニストレーション・クライアント

プログラミング環境のセットアップに関する情報は、 39ページの『第2章 セットアップ』を参照してください。

本書で記述しているプラットフォーム用の DB2 AD クライアントには、次のものが含まれます。

- **C/C++、Java、COBOL、および Fortran プリコンパイラー** (その言語が、使用するプラットフォームでサポートされている場合。詳細は、8ページの『各プラットフォームでサポートされるソフトウェア』を参照)。
- プログラミング・ライブラリー、インクルード・ファイル、およびコード・サンプルを含む、**組み込み SQL アプリケーション・サポート**。
- ODBC SDK への移植や ODBC SDK とのコンパイルが容易に行えるアプリケーションを開発するためのプログラミング・ライブラリー、インクルード・ファイル、およびコード例を含む、**DB2 コール・レベル・インターフェース (DB2 CLI) アプリケーション・サポート**。 ODBC SDK は、Windows 32 ビット・オペレーティング・システムの場合は Microsoft、および他のサポートされるプラットフォームの場合はさまざまなベンダーから入手可能で

す。 Windows 32 ビット・オペレーティング・システムの場合、 DB2 クライアントには Microsoft ODBC ソフトウェア開発者キットで開発されたアプリケーションをサポートする、 ODBC ドライバーが含まれています。他のすべてのプラットフォームについては、そのプラットフォーム用の ODBC SDK があれば、それを使用して開発されたアプリケーションをサポートする、任意でインストールされた ODBC ドライバーが DB2 クライアントに含まれます。 Windows 32 ビット・オペレーティング・システム用の DB2 クライアントだけに ODBC ドライバー・マネージャーが含まれています。

- Java アプリケーションおよびアプレットを開発する DB2 Java データベース・コネクティビティー (DB2 JDBC) サポート、および Java Embedded SQL アプリケーションとアプレットを開発する DB2 組み込み SQL for Java (DB2 SQLJ) サポートを含む、 **DB2 Java Enablement**。
- DB2 (AIX 版) および DB2 (Windows 32 ビット・オペレーティング・システム版) とともにインストールされ、 DB2 (OS/2 版) に付属している、IBM の **Java 開発者キット (JDK) 1.1.8 および Java Runtime Environment (JRE) 1.1.8**。
- AIX (32 ビット・アプリケーションのみ)、OS/2、および Windows 32 ビット・オペレーティング・システム上では、 **REXX 言語サポート**。このサポートは、DB2 バージョン 5.2 以上では更新されていません。
- Windows 32 ビット・オペレーティング・システムでは、 Microsoft Visual Basic および Microsoft Visual C++ で実装された、 **ActiveX Data Objects (ADO) およびオブジェクトのリンクと埋め込み (OLE) オートメーション UDF およびストアード・プロシージャー**。また、Microsoft Visual Basic で実装された Remote Data Object (RDO) を持つコード・サンプル。
- Windows 32 ビット・オペレーティング・システムでは、**オブジェクトのリンクと埋め込みデータベース (OLE DB) 表関数**。
- AIX、Solaris、および Windows 32 ビット・オペレーティング・システム上で使用可能な、 **DB2 ストアード・プロシージャー・ビルダー (SPB)**。これは、DB2 ストアード・プロシージャーの迅速な開発をサポートする GUI ベースのツールです。ワークステーションから OS/390 に至る範囲の DB2 ファミリーに、単一の開発環境を提供します。 Windows では、一般的なアプリケーション開発ツール、 Microsoft Visual Studio、Microsoft Visual Basic、および IBM VisualAge for Java から立ち上げるか、または IBM DB2 ユニバーサル・データベース・プログラム・グループから個別のアプリケーションとして立ち上げることができます。 AIX および Solaris では、 `db2spb` コマンドで開始できます。

- SQL ステートメントのプロトタイピングまたはデータベースの随時照会の実行のための、コマンド・センターまたはコマンド行プロセッサー (CLP) を介する**対話式 SQL**。
- 他のアプリケーション開発ツールの製品が、それらの製品内で直接に DB2 用のプリコンパイラー・サポートを実現するための**文書化された API のセット**。 たとえば、AIX および OS/2 上で、IBM COBOL はこのインターフェースを使用します。プリコンパイラー・サービス API のセットに関する情報は、匿名 FTP サイトの ftp://ftp.software.ibm.com から入手できます。prepapi.psbin と呼ばれるポストスクリプト・ファイルは、ディレクトリー /ps/products/db2/info にあります。このファイルは、バイナリー形式です。この電子フォーラムにアクセスしないでこの文書のコピーを入手したい場合は、サービス情報の用紙に記載されている IBM サービスと連絡を取ってください。
- ISO/ANSI SQL92 Entry Level 基準に適合しないか、または DB2 (OS/390 版) がサポートしないアプリケーション内の組み込み SQL ステートメントを識別する、 **SQL92 および MVS 適合標識機能**。ワークステーション上で開発したアプリケーションを他のプラットフォームに移行する場合には、標識機能によって構文の非互換性が示されるため、時間が節約されます。PRECOMPILE PROGRAM コマンドの SQLFLAG オプションについては、*コマンド解説書* を参照してください。

サポートされるサーバー

DB2 AD クライアントを使用して、特定のプラットフォームで実行するアプリケーションを開発します。 ただし、アプリケーションでは、次のプラットフォーム・サーバー上のリモート・データベースにアクセスすることができます。

- DB2 (AIX 版)
- DB2 (HP-UX 版)
- DB2 (Linux 版)
- DB2 (OS/2 版)
- DB2 (NUMA-Q 版)
- DB2 (SCO UnixWare 7 版)
- DB2 (Solaris 版)
- DB2 (Windows NT 版)
- 分散関係データベース体系 (DRDA) に準拠するアプリケーション・サーバー。たとえば、以下のものがあります。
  - DB2 (OS/390 版)
  - DB2 (AS/400 版)
  - DB2 (VSE および VM 版) (以前は、SQL/DS (VM および VSE))

– IBM 以外のデータベース・ベンダーからの、DRDA に準拠するアプリケーション・サーバー

注:

1. DB2 (NUMA-Q 版) は、DYNIX/ptx オペレーティング・システムをサポートします。
2. DB2 (SCO UnixWare 7 版) は、DB2 バージョン 5.2 のみで使用可能です。

各プラットフォームでサポートされるソフトウェア

この節では、本書に記載されているプラットフォーム用の DB2 によってサポートされるコンパイラーおよびその関連ソフトウェアをリストします。このコンパイラー情報は、ユーザーがそのプラットフォームに DB2 プリコンパイラーを使用しており、リストされたコンパイラーの 1 つに組み込まれている可能性があるプリコンパイラー・サポートを使用していないことを前提としています。サポートされる通信製品の情報については、ご使用のオペレーティング・システム用の*概説およびインストール* を参照してください。

最新の DB2 コンパイラー情報および関連ソフトウェアの更新については、次の DB2 アプリケーション開発の Web ページを参照してください。

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

注:

1. **DB2 リリース情報**には、そのプラットフォームでサポートされているコンパイラーについての更新情報およびオペレーティング・システム情報が含まれています。リリース情報は、製品 CD-ROM 上の次のパスで、フラット・テキストおよび HTML 形式でご覧になれます。ここで、`<language_directory>` は使用している言語のディレクトリーで、`index.htm` は主な HTML ファイルです。

   フラット・テキスト・ファイル

   doc/<language_directory>/release.txt (UNIX)

   Doc¥<language_directory>¥release.txt (OS/2 および Windows)

   HTML ファイル:

   doc/<language_directory>/db2ir/index.htm (UNIX)

   Doc¥<language_directory>¥db2ir¥index.htm (OS/2 および Windows)

2. **Fortran および REXX。** DB2 は、Fortran および REXX については、DB2 ユニバーサル・データベース バージョン 5.2 がこれらの言語をサポートするレベルを超えて、機能を拡張することはありません。

3. **Fortran。** Fortran サンプル・プログラムは、DB2 バージョン 7.1 では提供されていません。 DB2 バージョン 6.1 用の Fortran サンプルの入手については、上記の DB2 アプリケーション開発の Web ページを参照してください。
4. **HP-UX。** HP-UX バージョン 10 または HP-UX バージョン 11 以前から DB2 を移行している場合、 DB2 プログラムは HP-UX バージョン 11 (組み込み SQL がある場合) 上の DB2 で再びプリコンパイルして、再コンパイルしなければなりません。これには、すべての DB2 アプリケーション、ストアード・プロシージャー、ユーザー定義関数、およびユーザー出口プログラムが含まれます。さらに、HP-UX バージョン 11 上でコンパイルされた DB2 プログラムは、 HP-UX バージョン 10 以前の上では実行できない可能性があります。 HP-UX バージョン 10 上でコンパイルされ実行される DB2 プログラムは、 HP-UX バージョン 11 サーバーにリモートで接続することができます。
5. **Micro Focus COBOL。** DB2 バージョン 2.1.1 またはそれ以前を使ってプリコンパイルし、 Micro Focus COBOL を使ってコンパイルした既存アプリケーションは、現行バージョンの DB2 を使って再プリコンパイルした後、 Micro Focus COBOL を使って再コンパイルする必要があります。 IBM プリコンパイラーの旧バージョンを使って構築したアプリケーションを再プリコンパイルしないと、異常終了時にデータベースが破壊される恐れがあります。
6. **Perl。** Perl Database Interface (Perl DBI) 用の DB2 UDB ドライバー (DBD::DB2) のリリース 0.71 は、 AIX、HP-UX、Linux、Solaris および Windows NT 版で使用可能です。このドライバーは、次の Web サイトからダウンロードすることができます。

   `http://www.ibm.com/software/data/db2/perl`
7. **REXX**. IBM Object REXX (Windows NT/95 版) は、DB2 には付属していません。 Object REXX の入手については、次の Web サイトを参照してください。

   `http://www.ibm.com/software/ad/obj-rexx/`

AIX

DB2 (AIX 版) は、以下のオペレーティング・システムをサポートします。

AIX/6000

バージョン 4.2.1 以降

(64 ビットの場合、バージョン 4.3.3 以降)

DB2 (AIX 版) は、以下のプログラム言語およびコンパイラーをサポートします。

C IBM C for AIX バージョン 3.6.6 (64 ビットの場合、バージョン 3.6.6.3)

C++ IBM C Set++ for AIX バージョン 3.6.6 (64 ビットの場合、バージョン 3.6.6.3)

IBM VisualAge C++ バージョン 4.0

COBOL

IBM COBOL Set for AIX バージョン 1.1

Micro Focus COBOL バージョン 4.0.20 (PRN 12.03 以上)

Micro Focus COBOL バージョン 4.1.10 (PRN 13.04 以上)

Fortran

IBM XL Fortran for AIX バージョン 4.1 (32 ビットの場合) および 5.1.0 (32 ビットおよび 64 ビットの場合)

Java Java 開発者キット (JDK) バージョン 1.1.8 および Java Runtime Environment (JRE) バージョン 1.1.8 (AIX 版) (IBM 社提供で、DB2 に付属)

Java 開発者キット (JDK) バージョン 1.2.2 および Java Runtime Environment (JRE) バージョン 1.2.2 (AIX 版) (IBM 社提供)

Perl Perl Database Interface (Perl DBI) 用の DB2 UDB ドライバー (DBD::DB2) のリリース 0.71 (上記の注を参照)。

REXX IBM AIX REXX/6000 AISPO 製品番号: 5764-057

IBM Object REXX (AIX 版) バージョン 1.1

REXXSAA 4.00

注: REXX サポートは、32 ビット用のみです。

HP-UX

DB2 (HP-UX 版) は、以下のオペレーティング・システムをサポートします。

HP-UX

HP-UX Core OS Year 2000 Patch Bundle Version B.11.00.A1214 (Y2K-1100) 付きのバージョン 11.0 以降のパッチ・バンドル。

DB2 (HP-UX 版) は、以下のプログラム言語およびコンパイラーをサポートします。

C HP C コンパイラーのバージョン A.11.00.03

C++ HP-UX C++ バージョン A.12.00

COBOL

Micro Focus COBOL バージョン 4.1

Fortran

HP Fortran/9000 バージョン 10.0

HP-UX F77 B.11.00.01

Java HP-UX Developer's Kit for Java リリース 1.1.8 (Hewlett-Packard 社提供)

Perl Perl Database Interface (Perl DBI) 用の DB2 UDB ドライバー (DBD::DB2) のリリース 0.71 (上記の注を参照)。

Linux

DB2 (Linux 版) は、以下のオペレーティング・システムをサポートします。

Linux カーネル バージョン 2.2.12 以降、 glibc バージョン 2.1.2 以降、 libstdc++ バージョン 2.9.0, rpm (インストールの必要あり)、 pdksh パッケージ (DB2 コマンド行プロセッサーを実行するのに必要)

DB2 (Linux 版) では、以下のプログラム言語およびコンパイラーをサポートします。

C GNU/Linux gcc バージョン egcs-2.91.66 (egcs-1.1.2 リリース)

C++ GNU/Linux g++ バージョン egcs-2.91.66 (egcs-1.1.2 リリース)

Java IBM 開発者キットおよび Runtime Environment (Linux 版) バージョン 1.1.8

Perl Perl Database Interface (Perl DBI) 用の DB2 UDB ドライバー (DBD::DB2) のリリース 0.71 (上記の注を参照)。

OS/2

DB2 (OS/2 版) は、以下のオペレーティング・システムをサポートします。

OS/2 WARP 3.0、WARP 4.0、および WARP 4.5

DB2 (OS/2 版) は、以下のプログラム言語をサポートします。

C/C++ IBM VisualAge C++ for OS/2 バージョン 3 および 4.0

COBOL

IBM VisualAge COBOL for OS/2 バージョン 2.0

Micro Focus COBOL バージョン 4.0.20

FORTRAN

WATCOM FORTRAN 77 32 バージョン 10.5

Java Java 開発者キット (JDK) バージョン 1.1.8 および Java Runtime Environment (JRE) バージョン 1.1.8 (OS/2 版) (IBM 社提供で、DB2 に付属)

REXX IBM プロシージャー言語 2/REXX (OS/2 の一部として提供される)

DYNIX/ptx

DB2 (NUMA-Q 版) は、以下のオペレーティング・システムをサポートします。

DYNIX/ptx

バージョン 4.5

DB2 (NUMA-Q 版) は、以下のプログラム言語およびコンパイラーをサポートします。

C ptx/C バージョン 4.5

C++ ptx/C++ バージョン 5.2

Java ptx/JSE バージョン 3.0

Silicon Graphics IRIX

DB2 for Silicon Graphics IRIX では、以下のオペレーティング・システムをサポートします。

Silicon Graphics IRIX

バージョン 6.2 以降

DB2 for Silicon Graphics IRIX では、以下のプログラム言語とコンパイラーをサポートします。

C MIPSpro C Compiler 7.2

C++ MIPSpro C++ 7.2

Fortran

MIPSpro Fortran-77 7.2

Java Java2 Software Development Kit バージョン 1.2.1 (JDK 1.2.1) (Silicon Graphics 社提供)

Solaris

DB2 (Solaris 版) は、以下のオペレーティング・システムをサポートします。

Solaris
バージョン 2.6、Solaris 7、および Solaris 8

DB2 (Solaris 版) は、以下のプログラム言語およびコンパイラーをサポートします。

C SPARCompiler C バージョン 4.2 (32 ビットの場合) および 5.0 (32 ビットおよび 64 ビットの場合)

C++ SPARCompiler C++ バージョン 4.2 (32 ビットの場合) および 5.0 (32 ビットおよび 64 ビットの場合)

COBOL
Micro Focus COBOL バージョン 4.0

Fortran
SPARCompiler Fortran バージョン 4.2 および 5.0

Java Java 開発者キット (JDK) バージョン 1.1.8 および 1.2 (Solaris 版) (Sun Microsystems 社提供)

Perl Perl Database Interface (Perl DBI) 用の DB2 UDB ドライバー (DBD::DB2) のリリース 0.71 (上記の注を参照)。

Windows 32 ビット・オペレーティング・システム

DB2 (Windows 32 ビット・オペレーティング・システム版) は、以下をサポートします。

Microsoft Windows NT
Service Pack 4 付きのバージョン 4.0 以降。

Microsoft Windows 2000

Microsoft Windows 98

Microsoft Windows 95
バージョン 4.00.950 またはそれ以降

DB2 (Windows 32 ビット・オペレーティング・システム版) は、以下のプログラム言語をサポートします。

Basic Microsoft Visual Basic バージョン 4.2 およびバージョン 5.0 (この言語の DB2 プリコンパイラーは提供されていない)

C/C++ Microsoft Visual C++ バージョン 5.0 および 6.0

IBM VisualAge C++ for Windows バージョン 4.2 およびバージョン 5.0

COBOL

Micro Focus COBOL バージョン 4.0.20

IBM VisualAge COBOL バージョン 2.0

REXX IBM Object REXX for Windows NT/95 バージョン 1.1 (上記の注を参照)

Java Java 開発者キット (JDK) 1.1.8 および Java Runtime Environment (JRE) 1.1.8 (Win32 版) (IBM 社提供で、DB2 に付属)

Java 開発者キット (JDK) 1.2 (Win32 版) (Sun Microsystems 提供)

Microsoft Software Developer's Kit for Java、バージョン 3.1

Perl Perl Database Interface (Perl DBI) 用の DB2 UDB ドライバー (DBD::DB2) のリリース 0.71 (Windows NT 上で使用可能。上記の注を参照。)

サンプル・プログラム

注:

1. この節では、DB2 がサポートするすべてのプラットフォーム用のプログラム言語のサンプル・プログラムを記載しています。以下に示す例のすべてがサポートされるプログラミング言語に移植されているわけではありません。
2. DB2 サンプル・プログラムは、保証はまったくありませんが、「そのまま」使用できます。ユーザーおよび IBM 以外の方は、品質、パフォーマンス、何らかの欠陥の訂正のすべてのリスクをご承知いただきます。

サンプル・プログラムは、DB2 アプリケーション開発 (DB2 AD) クライアントに付属しています。サンプル・プログラムをテンプレートとして使用して、独自のアプリケーションを作成することができます。

サンプル・プログラムのファイル拡張子は、サポートされる各言語ごとに異なり、各言語内でも、組み込み SQL プログラムと非組み込み SQL プログラムとでは異なります。さらには、各言語内のプログラム・グループごとにも異なっています。これらのサンプル・ファイル拡張子を分類したのが、次の表です。

言語別のサンプル・ファイル拡張子
16ページの表1.

プログラム・グループ別のサンプル・ファイル拡張子
17ページの表2.

次の表は、サンプル・プログラムをタイプで分類しています。

組み込み SQL なしの DB2 API サンプル・プログラム
19ページの表3.

DB2 API 組み込み SQL サンプル・プログラム
22ページの表4.

DB2 API なしの組み込み SQL サンプル・プログラム
25ページの表5.

ユーザー定義関数サンプル・プログラム
26ページの表6

DB2 CLI サンプル・プログラム
27ページの表7.

Java JDBC サンプル・プログラム
28ページの表8.

Java SQLJ サンプル・プログラム
29ページの表9.

SQL プロシージャー・サンプル・プログラム
31ページの表10.

ActiveX Data Object、Remote Data Objects、および Microsoft Transaction Server サンプル・プログラム
33ページの表11.

オブジェクトのリンクと埋め込み (OLE) オートメーションのサンプル・プログラム 34ページの表12.

オブジェクトのリンクと埋め込みデータベース (OLE DB) 表関数
35ページの表13.

コマンド行プロセッサー・プロセッサー (CLP) サンプル・プログラム
35ページの表14.

ログ管理ユーザー出口プログラム
36ページの表15.

注:

1. 22ページの表4 には、 DB2 API と組み込み SQL ステートメントの両方を持つプログラムが入っています。すべての DB2 API サンプル・プログラムについては、 19ページの表3 および 22ページの表4 の両方を参照してください。 すべての組み込み SQL サンプル・プログラム (Java SQLJ を除く) については、 22ページの表4 および 25ページの表5 の両方を参照してください。
2. UDF サンプル・プログラムの 26ページの表6 には、 DB2 CLI UDF プログラムが含まれていません。それらについては、27ページの表7 を参照してください。

表 1. 言語別のサンプル・ファイル拡張子

言語	ディレクトリー	組み込み SQL を含むプログラム	組み込み SQL を含まないプログラム
C	samples/c samples/cli (CLI プログラム)	.sqc	.c
C++	samples/cpp	.sqC (UNIX) .sqx (Windows および OS/2)	.C (UNIX) .cxx (Windows および OS/2)
COBOL	samples/cobol samples/cobol_mf	.sqb	.cbl
JAVA	samples/java	.sqlj	.java
REXX	samples/rexx	.cmd	.cmd

表 2. プログラム・グループ別のサンプル・ファイル拡張子

サンプル・グループ	ディレクトリー	ファイル拡張子
ADO, RDO, MTS	samples¥ADO¥VB (Visual Basic) samples¥ADO¥VC (Visual C++) samples¥RDO samples¥MTS	.bas .frm .vbp (Visual Basic) .cpp .dsp .dsw (Visual C++)
CLP	samples/clp	.db2
OLE	samples¥ole¥msvb (Visual Basic) samples¥ole¥msvc (Visual C++)	.bas .vbp (Visual Basic) .cpp (Visual C++)
OLE DB	samples¥oledb	.db2
SQL プロシージャー	samples/sqlproc	.db2 .c .sqc (クライアント・アプリケーション)
ユーザー出口	samples/c	.cad (OS/2) .cadsm (UNIX および Windows) .cdisk (UNIX および Windows) .ctape (UNIX)

注:

ディレクトリー区切り文字

UNIX では /。 OS/2 および Windows プラットフォームでは ¥。表の中では、Windows または OS/2 あるいはその両方でのみ使用可能なディレクトリー以外は、 UNIX の区切り文字が使用されます。

ファイル拡張子

拡張子が 1 つしか存在しない表にあるサンプルに提供されます。

組み込み SQL を含むプログラム

このプログラムは、プリコンパイルが必要です。 REXX 組み込み SQL プログラムは、プログラムの実行時に組み込み SQL ステートメントが解釈されるので例外になります。

IBM COBOL サンプル

AIX、OS/2、および Windows 32 ビットのオペレーティング・システムだけで、 cobol サブディレクトリーに提供されます。

Micro Focus Cobol サンプル

AIX、HP-UX、OS/2、Solaris Operating Environment、および Windows 32 ビットのオペレーティング・システムだけで、 cobol_mf サブディレクトリーに提供されます。

Java サンプル

Java UDF に加えて、 Java データベース・コネクティビティー (JDBC) アプレット、アプリケーション、およびストアード・プロシージャー、Java Embedded SQL (SQLJ) アプレット、アプリケーション、およびストアード・プロシージャー。 Java サンプルは、サポートされるすべての DB2 プラットフォーム上で使用可能です。

REXX サンプル

AIX、OS/2、および Windows NT オペレーティング・システムだけで提供されます。

CLP サンプル

SQL ステートメントを実行するコマンド行プロセッサーのスクリプトです。

OLE サンプル

Microsoft Visual Basic および Microsoft Visual C++ のオブジェクトのリンクと埋め込み (OLE) のためのサンプルで、 Windows 32 ビット・オペレーティング・システム上でのみ提供されます。

ADO、RDO、および MTS サンプル

Microsoft Visual Basic および Microsoft Visual C++ の ActiveX Data Object サンプル、および Microsoft Visual Basic の Remote Data Objects と Microsoft Transaction Server サンプルで、 Windows 32 ビット・オペレーティング・システム上でのみ提供されます。

ユーザー出口サンプル

データベース・ログ・ファイルを保存し検索するのに使用する、ログ管理ユーザー出口プログラムです。ファイルは、.c 拡張子を付けて名前変更し、 C 言語プログラムとしてコンパイルしなければなりません。

サンプル・プログラムは、DB2 がインストールされているディレクトリーの samples サブディレクトリーに入っています。サポートされている言語ごとにサブディレクトリーが 1 つずつ作成されます。以下の例では、サポートされている各プラットフォーム用に C または C++ で作成されたサンプルがある場所を探す方法を示しています。

- **UNIX プラットフォームの場合:**

  組み込み SQL および DB2 API プログラムの C ソース・コードは、データベース・インスタンス・ディレクトリーの下にある sqllib/samples/c にあります。 DB2 CLI プログラムの C ソース・コードは、

sqllib/samples/cli にあります。サンプル表にあるプログラムの追加情報については、 DB2 インスタンスの下の該当する samples サブディレクトリーの README ファイルを参照してください。 README ファイルには、本書でリストされていない追加のサンプルが含まれることがあります。

- **OS/2 および Windows 32 ビット・オペレーティング・システムの場合：**

  組み込み SQL と DB2 API プログラムの C ソース・コードは、 DB2 インストール・ディレクトリーの下にある %DB2PATH%¥samples¥c にあります。 DB2 CLI プログラムの C ソース・コードは、 %DB2PATH%¥samples¥cli にあります。 変数 %DB2PATH% で、 DB2 のインストール先を判別できます。 DB2 がインストールされているドライブによっては、 %DB2PATH% は *drive*:¥sqllib を指します。サンプル表にあるプログラムの追加情報については、該当する %DB2PATH%¥samples サブディレクトリーの README ファイルを参照してください。 README ファイルには、本書でリストされていない追加のサンプルが含まれることがあります。

サンプル・プログラム・ディレクトリーは、たいていのプラットフォームでは一般に読み取り専用です。サンプル・プログラムは、変更または作成する前に、ユーザーの作業ディレクトリーにコピーしてください。

組み込み SQL なしの DB2 API サンプル

表 3. 組み込み SQL なしの DB2 API サンプル・プログラム

サンプル・プログラム	組み込み API
backrest	• sqlbftcq - 表スペース・コンテナー照会の取り出し • sqlbstsc - 表スペース・コンテナーの設定 • sqlfudb - データベース構成の更新 • sqlubkp - データベースのバックアップ • sqluroll - データベースのロールフォワード • sqlurst - データベースの復元
checkerr	• sqlaintp - エラー・メッセージの入手 • sqlogstt - SQLSTATE メッセージの入手
cli_info	• sqleqryi - クライアント情報の照会 • sqleseti - クライアント情報の設定
client	• sqleqryc - クライアントの照会 • sqlesetc - クライアントの設定

表 3. 組み込み SQL なしの DB2 API サンプル・プログラム (続き)

サンプル・プログラム	組み込み API
d_dbconf	• sqleatin - 接続 • sqledtin - 切り離し • sqlfddb - データベース構成デフォルト値の入手
d_dbmcon	• sqleatin - 接続 • sqledtin - 切り離し • sqlfdsys - データベース・マネージャー構成デフォルト値の入手
db_udcs	• sqleatin - 接続 • sqlecrea - データベースの作成 • sqledrpd - データベースの除去
db2mon	• sqleatin - 接続 • sqlmon - モニター・スイッチの入手 / 更新 • sqlmonss - スナップショットの入手 • sqlmonsz - sqlmonss() 出力バッファーに必要なサイズの見積もり • sqlmrset - モニターのリセット
dbcat	• sqlecadb - データベースのカタログ化 • sqledcls - データベース・ディレクトリー・スキャンのクローズ • sqledgne - 次のデータベース・ディレクトリー・エントリーの入手 • sqledosd - データベース・ディレクトリー・スキャンのオープン • sqleuncd - データベースのアンカタログ
dbcmt	• sqledcgd - データベースのコメントの変更 • sqledcls - データベース・ディレクトリー・スキャンのクローズ • sqledgne - 次のデータベース・ディレクトリー・エントリーの入手 • sqledosd - データベース・ディレクトリー・スキャンのオープン • sqleisig - シグナル・ハンドラーのインストール

表 3. 組み込み SQL なしの DB2 API サンプル・プログラム (続き)

サンプル・プログラム	組み込み API
dbconf	• sqleatin - 接続 • sqlecrea - データベースの作成 • sqledrpd - データベースの除去 • sqlfrdb - データベース構成のリセット • sqlfudb - データベース構成の更新 • sqlfxdb - データベース構成の入手
dbinst	• sqleatcp - 接続およびパスワードの変更 • sqleatin - 接続 • sqledtin - 切り離し • sqlegins - インスタンス
dbmconf	• sqleatin - 接続 • sqledtin - 切り離し • sqlfrsys - データベース・マネージャー構成のリセット • sqlfusys - データベース・マネージャー構成の更新 • sqlfxsys - データベース・マネージャー構成の入手
dbsnap	• sqleatin - 接続 • sqlmonss - スナップショットの入手
dbstart	• sqlepstart - データベース・マネージャーの開始
dbstop	• sqlefrce - アプリケーションの強制 • sqlepstp - データベース・マネージャーの停止
dcscat	• sqlegdad - DCS データベースのカタログ化 • sqlegdcl - DCS ディレクトリー・スキャンのクローズ • sqlegdel - DCS データベースのアンカタログ • sqlegdge - データベースの DCS ディレクトリー・エントリーの入手 • sqlegdgt - DCS ディレクトリー・エントリーの入手 • sqlegdsc - DCS ディレクトリー・スキャンのオープン
dmscont	• sqleatin - 接続 • sqlecrea - データベースの作成 • sqledrpd - データベースの除去

表 3. 組み込み SQL なしの DB2 API サンプル・プログラム (続き)

サンプル・プログラム	組み込み API
ebcdicdb	• sqleatin - 接続 • sqlecrea - データベースの作成 • sqledrpd - データベースの除去
migrate	• sqlemgdb - データベースの移行
monreset	• sqleatin - 接続 • sqlmrset - モニターのリセット
monsz	• sqleatin - 接続 • sqlmonss - スナップショットの入手 • sqlmonsz - sqlmonss() 出力バッファーに必要なサイズの見積もり
nodecat	• sqlectnd - ノードのカタログ化 • sqlencls - ノード・ディレクトリー・スキャンのクローズ • sqlengne - 次のノード・ディレクトリー・エントリーの入手 • sqlenops - ノード・ディレクトリー・スキャンのオープン • sqleuncn - ノードのアンカタログ
restart	• sqlerstd - データベースの再始動
setact	• sqlesact - 会計ストリングの設定
setrundg	• sqlesdeg - 実行時間の程度の設定
sws	• sqleatin - 接続 • sqlmon - モニター・スイッチの入手 / 更新
utilapi	• sqlaintp - エラー・メッセージの入手 • sqlogstt - SQLSTATE メッセージの入手

DB2 API 組み込み SQL サンプル

表 4. DB2 API 組み込み SQL サンプル・プログラム

サンプル・プログラム	組み込み API
asynrlog	• sqlurlog - 非同期読み取りログ

表 4. DB2 API 組み込み SQL サンプル・プログラム (続き)

サンプル・プログラム	組み込み API
autocfg	• db2AutoConfig -- 自動構成 • db2AutoConfigMemory -- 自動構成の空きメモリー • sqlfudb -- データベース構成の更新 • sqlfusys -- データベース・マネージャー構成の更新 • sqlesetc -- クライアントの設定 • sqlaintp -- SQLCA メッセージ
dbauth	• sqluadau - 許可の入手
dbstat	• sqlureot - 表の再編成 • sqlustat - Runstats
expsamp	• sqluexpr - エクスポート • sqluimpr - インポート
impexp	• sqluexpr - エクスポート • sqluimpr - インポート
loadqry	• db2LoadQuery - 照会のロード
makeapi	• sqlabndx - バインド • sqlaprep - プログラムのプリコンパイル • sqlepstp - データベース・マネージャーの停止 • sqlepstr - データベース・マネージャーの開始
rebind	• sqlarbnd - 再バインド
rechist	• sqlubkp - データベースのバックアップ • sqluhcls - 回復活動記録ファイル・スキャンのクローズ • sqluhgne - 次の回復活動記録ファイル・エントリーの入手 • sqluhops - 回復活動記録ファイル・スキャンのオープン • sqluhprn - 回復活動記録ファイルの枝取り • sqluhupd - 回復活動記録ファイルの更新
tabscont	• sqlbctcq - 表スペース・コンテナー照会のクローズ • sqlbftcq - 表スペース・コンテナー照会の取り出し • sqlbotcq - 表スペース・コンテナー照会のオープン • sqlbtcq - 表スペース・コンテナー照会 • sqlefmem - 空きメモリー

表 4. DB2 API 組み込み SQL サンプル・プログラム (続き)

サンプル・プログラム	組み込み API
tabspace	• sqlbctsq - 表スペース照会のクローズ • sqlbftpq - 表スペース照会の取り出し • sqlbgtss - 表スペース統計の入手 • sqlbmtsq - 表スペース照会 • sqlbotsq - 表スペース照会のオープン • sqlbstpq - 単一表スペース照会 • sqlefmem - 空きメモリー
tload	• sqluexpr - エクスポート • sqluload - ロード • sqluvqdp - 表の表スペースの静止
tspace	• sqlbctcq - 表スペース・コンテナー照会のクローズ • sqlbctsq - 表スペース照会のクローズ • sqlbftcq - 表スペース・コンテナー照会の取り出し • sqlbftpq - 表スペース照会の取り出し • sqlbgtss - 表スペース統計の入手 • sqlbmtsq - 表スペース照会 • sqlbotcq - 表スペース・コンテナー照会のオープン • sqlbotsq - 表スペース照会のオープン • sqlbstpq - 単一表スペース照会 • sqlbstsc - 表スペース・コンテナーの設定 • sqlbtcq - 表スペース・コンテナー照会 • sqlefmem - 空きメモリー
utilemb	• sqlaintp - エラー・メッセージの入手 • sqlogstt - SQLSTATE メッセージの入手

DB2 API なしの組み込み SQL サンプル

表 5. DB2 API なしの組み込み SQL サンプル・プログラム

サンプル・プログラム名	プログラムの説明
adhoc	動的 SQL および SQLDA 構造を使用して SQL コマンドを対話式に処理する方法を示します。 SQL コマンドはユーザーによって入力され、SQL コマンドに応じた出力が返されます。
advsql	CASE、CAST、およびスカラー全選択などの拡張 SQL 式の使用例を示します。
blobfile	2 進ラージ・オブジェクト (BLOB) の操作例を、 BLOB 値をサンプル・データベースから読み取ってそれをファイル内に置くことにより示します。この内容は、外部ビューアーを使用して表示できます。
columns	動的 SQL を使用して処理されるカーソルの使用例を示します。このプログラムは、指定のスキーマ名の下にある SYSCAT.COLUMNS から結果セットをリストします。
cursor	静的 SQL を使用するカーソルの使用例を示します。
delet	データベースから項目を削除する静的 SQL の使用例を示します。
dynamic	動的 SQL を使用するカーソルの使用例を示します。
joinsql	拡張 SQL 結合式の使用例を示します。
largevol	区分化された環境で並列照会処理を行う例、および結果セットの組み合わせを自動化するための NFS ファイル・システムの使用例を示します。 AIX でのみ使用可能です。
lobeval	LOB ロケーターの使用例を示し、実際の LOB データの評価を遅らせます。
lobfile	LOB ファイル・ハンドルの使用例を示します。
lobloc	LOB ロケーターの使用例を示します。
lobval	LOB の使用例を示します。
openftch	静的 SQL を使用した行の取り出し、更新、および削除について示します。
recursql	拡張 SQL 再帰的照会の使用例を示します。
sampudf	表エントリーを修正するための、ユーザー定義タイプ (UDT) およびユーザー定義関数 (UDF) の使用例を示します。このプログラムで宣言される UDF は、ソース UDF です。
spclient	spserver 共用ライブラリー内のストアード・プロシージャーを呼び出すクライアント・アプリケーション。
spcreate.db2	spserver プログラムによって作成されたストアード・プロシージャーを登録するための CREATE PROCEDURE ステートメントを含む CLP スクリプト。
spdrop.db2	spserver プログラムによって作成されたストアード・プロシージャーの登録を解除するために必要な DROP PROCEDURE ステートメントを含む CLP スクリプト。

表 5. DB2 API なしの組み込み SQL サンプル・プログラム (続き)

サンプル・プログラム名	プログラムの説明
spserver	ストアード・プロシージャーのデモを示すサーバー・プログラム。クライアント・プログラムは spclient です。
static	情報を検索する静的 SQL を示します。
tabsql	拡張 SQL 表式の使用例を示します。
tbdefine	表の作成方法および除去方法を示します。
thdsrver	スレッドの作成と管理用の POSIX スレッド API の使用例を示します。プログラムは、コンテキストのプールを保守します。 generate_work 関数がメインから実行され、作業スレッドが実行する動的 SQL ステートメントを作成します。コンテキストが使用可能になったとき、スレッドが作成され、指定された作業を行うためにディスパッチされます。生成された作業は、 sample データベースの STAFF 表または EMPLOYEE 表のどちらかからの項目を削除するステートメントで成っています。このプログラムは、UNIX プラットフォームでのみ使用可能です。
trigsql	拡張 SQL トリガーおよび制約の使用例を示します。
udfcli	udfsrv プログラムによって作成されるユーザー定義関数 (UDF) の呼び出しを例示し、 sample データベースの表にアクセスするためにサーバー上に保管されます。
updat	データベースを更新する静的 SQL の使用例を示します。
varinp	パラメーター・マーカーを使用した組み込み動的 SQL ステートメント呼び出しへの変数入力を示します。

ユーザー定義関数のサンプル

表 6. ユーザー定義関数のサンプル・プログラム

サンプル・プログラム名	プログラムの説明
DB2Udf.java	整数除算、文字ラージ・オブジェクト (CLOB) の操作、および Java インスタンス変数の使用を含む複数のタスクの実例を示す Java UDF。
udfsrv.c	ユーザー定義関数 ScalarUDF でライブラリーを作成し、サンプル・データベース表にアクセスします。
UDFsrv.java	Java ユーザー定義関数 (UDF) の使用例を示します。

DB2 コール・レベル・インターフェースのサンプル

表 7. DB2 ユニバーサル・データベースのサンプル CLI プログラム

サンプル・プログラム名	プログラムの説明
共通ユーティリティー・ファイル	
utilcli.c	CLI サンプルで使用されるユーティリティー関数。
utilapi.c	DB2 API を呼び出すユーティリティー関数。
アプリケーション・レベル - DB2 と CLI のアプリケーション・レベルを扱うサンプル。	
apinfo.c	アプリケーション・レベル情報の入手および設定方法。
aphndls.c	ハンドルの割り当ておよび解放方法。
apsqlca.c	SQLCA データの処理方法。
インストール・イメージ・レベル - DB2 と CLI のインストール・イメージ・レベルを扱うサンプル。	
ilinfo.c	インストール・レベル情報 (CLI ドライバーのバージョンなど) の入手および設定方法。
インスタンス・レベル - DB2 と CLI のインスタンス・レベルを扱うサンプル。	
ininfo.c	インスタンス・レベル情報の入手および設定方法。
データベース・レベル - DB2 内のデータベース・オブジェクトを扱うサンプル。	
dbconn.c	データベースからの接続および切断方法。
dbinfo.c	データベース・レベルの情報の入手および設定方法。
dbmconn.c	複数のデータベースからの接続および切断方法 (DB2 API を使用して 2 番目のデータベースを作成および除去する)。
dbmuse.c	複数のデータベースとのトランザクションの実行方法 (DB2 API を使用して 2 番目のデータベースを作成および除去する)。
dbnative.c	ODBC エスケープ文節を含むステートメントを、データ・ソース特有の形式に変換する方法。
dbuse.c	データベース・オブジェクトの処理方法。
dbusemx.sqc	組み込み SQL とともに単一データベースを使用する方法。
表レベル - DB2 内の表オブジェクトを扱うサンプル。	
tbconstr.c	表の制約の処理方法。
tbconstr.c	表の作成、更新、および除去方法。
tbinfo.c	表レベルの情報の入手および設定方法。
tbmod.c	表内の情報の修正方法。
tbread.c	表内の情報の読み取り方法。
データ・タイプ・レベル - データ・タイプを扱うサンプル。	
dtinfo.c	データ・タイプに関する情報の入手方法。

表 7. DB2 ユニバーサル・データベースのサンプル CLI プログラム (続き)

サンプル・プログラム名	プログラムの説明
dtlob.c	LOB データの読み取りおよび書き込み方法。
dtudt.c	ユーザー定義特殊タイプの作成、使用、および除去方法。
UDF レベル - ユーザー定義関数を示すサンプル。	
udfcli.c	udfsrv.c 内のユーザー定義関数を呼び出すクライアント・アプリケーション。
udfsrv.c	udfcli.c サンプルによって呼び出されるユーザー定義関数 ScalarUDF。
ストアード・プロシージャー・レベル - CLI 内のストアード・プロシージャーを示すサンプル。	
spcreate.db2	CREATE PROCEDURE ステートメントを発行するための CLP スクリプト。
spdrop.db2	カタログからストアード・プロシージャーを除去するための CLP スクリプト。
spclient.c	spserver.c 内で宣言されるサーバー関数を呼び出すために使用されるクライアント・プログラム。
spserver.c	サーバー上で構築および実行されるストアード・プロシージャー関数。
spcall.c	ストアード・プロシージャーを呼び出すためのプログラム。

注: samples/cli ディレクトリー内には、次のファイルもあります。

- README - すべてのサンプル・ファイルのリスト
- makefile - すべてのファイルの makefile
- アプリケーションおよびストアード・プロシージャー用のビルド・ファイル

Java サンプル

表 8. Java データベース・コネクティビティー (JDBC) サンプル・プログラム

サンプル・プログラム名	プログラムの説明
DB2Appl.java	呼び出しユーザー特権を使用してサンプル・データベースを照会する、JDBC アプリケーション。
DB2Applt.java	JDBC アプレット・ドライバーを使用してデータベースを照会する JDBC アプレット。 DB2Applt.html で指定されるユーザー名、パスワード、サーバー、およびポート番号パラメーターを使用します。
DB2Applt.html	DB2Applt アプレット・サンプル・プログラムを組み込む HTML ファイル。このファイルは、サーバーおよびユーザー情報でカスタマイズする必要があります。
DB2UdCli.java	Java ユーザー定義関数、 DB2Udf を呼び出す Java クライアント・アプリケーション。
Dynamic.java	動的 SQL を使うカーソルの使用例を示します。

表 8. *Java* データベース・コネクティビティー *(JDBC)* サンプル・プログラム *(続き)*

サンプル・プログラム名	プログラムの説明
MRSPcli.java	これは、サーバー・プログラム MRSPsrv を呼び出すクライアント・プログラムです。このプログラムは、Java ストアード・プロシージャーから戻される複数の結果セットを示します。
MRSPsrv.java	これは、クライアント・プログラム MRSPcli が呼び出すサーバー・プログラムです。このプログラムは、Java ストアード・プロシージャーから戻される複数の結果セットを示します。
Outcli.java	SQLJ ストアード・プロシージャー、 Outsrv を呼び出す Java クライアント・アプリケーション。
PluginEx.java	新しいメニュー項目およびツールバー・ボタンを、 DB2 Web コントロール・センターに追加する Java プログラム。
Spclient.java	Spserver ストアード・プロシージャー・クラス内の PARAMETER STYLE JAVA ストアード・プロシージャーを呼び出す JDBC クライアント・アプリケーション。
Spcreate.db2	ストアード・プロシージャーとして Spserver クラス内に含まれているメソッドを登録するための CREATE PROCEDURE ステートメントを含む CLP スクリプト。
Spdrop.db2	Spserver クラス内に含まれているストアード・プロシージャーの登録を解除するために必要な DROP PROCEDURE ステートメントを含む CLP スクリプト。
Spserver.java	PARAMETER STYLE JAVA ストアード・プロシージャーのデモを示す JDBC プログラム。クライアント・プログラムは Spclient.java です。
UDFcli.java	Java ユーザー定義関数ライブラリー UDFsrv にある関数を呼び出す、 JDBC クライアント・アプリケーション。
UseThrds.java	SQL ステートメントを非同期で実行するためにスレッドを使用する方法を示します (CLI サンプル async.c の JDBC バージョン)。
V5SpCli.java	DB2GENERAL ストアード・プロシージャー、V5Stp.java を呼び出す Java クライアント・アプリケーション。
V5Stp.java	サーバー上の EMPLOYEE 表を更新し、クライアントに新しい給与および給与計算情報を戻す DB2GENERAL ストアード・プロシージャーのデモを示します。クライアント・プログラムは V5SpCli.java です。
Varinp.java	パラメーター・マーカーを使用した組み込み動的 SQL ステートメント呼び出しへの変数入力を示します。

表 9. *Java Embedded SQL (SQLJ)* サンプル・プログラム

サンプル・プログラム名	プログラムの説明
App.sqlj	サンプル・データベースの EMPLOYEE 表からデータを検索し更新するために、静的 SQL を使用します。

表 9. Java Embedded SQL (SQLJ) サンプル・プログラム (続き)

サンプル・プログラム名	プログラムの説明
Applt.sqlj	JDBC アプレット・ドライバーを使用してデータベースを照会するアプレット。Applt.html で指定されるユーザー名、パスワード、サーバー、およびポート番号パラメーターを使用します。
Applt.html	Applt アプレット・サンプル・プログラムを組み込む HTML ファイル。このファイルは、サーバーおよびユーザー情報でカスタマイズする必要があります。
Cursor.sqlj	静的 SQL を使用するイテレーターを示します。
OpF_Curs.sqlj	Openftch プログラム用のクラス・ファイル。
Openftch.sqlj	静的 SQL を使用した行の取り出し、更新、および削除について示します。
Outsrv.sqlj	SQLDA 構造を使用するストアード・プロシージャーを示します。このプログラムは、sample データベースの STAFF 表にある従業員の給与の中央値を SQLDA に記入します。データベースの処理 (中央値を求める) 後、ストアード・プロシージャーは、値を記入した SQLDA と SQLCA 状況とを JDBC クライアント・アプリケーション、 Outcli に戻します。
Stclient.sqlj	SQLJ ストアード・プロシージャー・プログラム Stserver によって作成された PARAMETER STYLE JAVA ストアード・プロシージャーを呼び出す SQLJ クライアント・アプリケーション。
Stcreate.db2	ストアード・プロシージャーとして Stserver クラス内に含まれているメソッドを登録するための CREATE PROCEDURE ステートメントを含む CLP スクリプト。
Stdrop.db2	Stserver クラス内に含まれているストアード・プロシージャーの登録を解除するために必要な DROP PROCEDURE ステートメントを含む CLP スクリプト。
Stserver.sqlj	PARAMETER STYLE JAVA ストアード・プロシージャーのデモを示す SQLJ プログラム。クライアント・プログラムは Stclient.sqlj です。
Static.sqlj	静的 SQL を使用して情報を検索します。
Stp.sqlj	サーバー上の EMPLOYEE 表を更新し、 JDBC クライアント・プログラム StpCli に新しい給料および給与計算情報を返すストアード・プロシージャー。
UDFclie.sqlj	Java ユーザー定義関数ライブラリー UDFsrv から関数を呼び出す、クライアント・アプリケーション。
Updat.sqlj	静的 SQL を使用してデータベースを更新します。

SQL プロシージャーのサンプル

表 10. SQL プロシージャーのサンプル・プログラム

サンプル・プログラム名	プログラムの説明
basecase.db2	UPDATE_SALARY プロシージャーは、 ″sample″ データベースの ″staff″ 表内の ″empno″ の IN パラメーターによって識別される従業員の給料を上昇させます。このプロシージャーは、 ″rating″ の IN パラメーターを使用する CASE ステートメントによって上昇率を判別します。
basecase.sqc	UPDATE_SALARY プロシージャーを呼び出します。
baseif.db2	UPDATE_SALARY_IF プロシージャーは、 ″sample″ データベースの ″staff″ 表内の ″empno″ の IN パラメーターによって識別される従業員の給料を上昇させます。 このプロシージャーは、 ″rating″ の IN パラメーターを使用する IF ステートメントによって上昇率を判別します。
baseif.sqc	UPDATE_SALARY_IF プロシージャーを呼び出します。
dynamic.db2	CREATE_DEPT_TABLE プロシージャーは、動的 DDL を使用して新しい表を作成します。表の名前は、このプロシージャーの IN パラメーターの値に基づいています。
dynamic.sqc	CREATE_DEPT_TABLE プロシージャーを呼び出します。
iterate.db2	ITERATOR プロシージャーは、FETCH ループを使用して、 ″department″ 表からデータを検索します。 ″deptno″ 列の値が 'D11' でない場合、修正されたデータが ″department″ 表の中に挿入されます。 ″deptno″ 列が 'D11' である場合、 ITERATE ステートメントは、LOOP ステートメントの冒頭に制御の流れを渡します。
iterate.sqc	ITERATOR プロシージャーを呼び出します。
leave.db2	LEAVE_LOOP プロシージャーは、 ″not_found″ 条件ハンドラーが LEAVE ステートメントを呼び出す前に LOOP ステートメント内で実行される FETCH 操作の数をカウントします。 LEAVE ステートメントは、制御の流れがループから出て、ストアード・プロシージャーを完了するようにします。
leave.sqc	LEAVE_LOOP プロシージャーを呼び出します。
loop.db2	LOOP_UNTIL_SPACE プロシージャーは、カーソルが ″midinit″ 列のスペース (' ') 値を持つ行を検索するまで、 LOOP ステートメント内で実行される FETCH 操作の数をカウントします。 LOOP ステートメントは、制御の流れがループから出て、ストアード・プロシージャーを完了するようにします。
loop.sqc	LOOP_UNTIL_SPACE プロシージャーを呼び出します。
nestcase.db2	BUMP_SALARY プロシージャーは、ネストされた CASE ステートメントを使用して、 ″sample″ データベースの ″staff″ 表から部署の IN パラメーターによって識別される部署内の従業員の給料を上昇させます。
nestcase.sqc	BUMP_SALARY プロシージャーを呼び出します。

表 10. SQL プロシージャーのサンプル・プログラム (続き)

サンプル・プログラム名	プログラムの説明
nestif.db2	BUMP_SALARY_IF プロシージャーは、ネストされた IF ステートメントを使用して、 ″sample″ データベースの ″staff″ 表から部署の IN パラメーターによって識別される部署内の従業員の給料を上昇させます。
nestif.sqc	BUMP_SALARY_IF プロシージャーを呼び出します。
repeat.db2	REPEAT_STMT プロシージャーは、カーソルが行を検索できなくなるまで、繰り返しステートメント内で実行される FETCH 操作の数をカウントします。条件ハンドラーは、制御の流れが繰り返しループから出て、ストアード・プロシージャーを完了するようにします。
repeat.sqc	REPEAT_STMT プロシージャーを呼び出します。
rsultset.c	MEDIAN_RESULT_SET プロシージャーを呼び出し、給与の中央値を表示し、SQL プロシージャーによって生成された結果セットを表示します。このクライアントは、結果セットを扱える CLI API で書き出されます。
rsultset.db2	MEDIAN_RESULT_SET プロシージャーは、 ″sample″ データベースの ″staff″ 表から ″dept″ の IN パラメーターによって識別される部署内の従業員の給与の中央値を入手します。給与の中央値は、給料の OUT パラメーターに割り当てられ、 ″rsultset″ クライアントに戻されます。次に、このプロシージャーは、 WITH RETURN カーソルをオープンし、中央値よりも高い給料の従業員の結果セットを戻します。このプロシージャーは、クライアントに結果セットを戻します。
spserver.db2	この CLP スクリプト内の SQL プロシージャーは、基本的なエラー処理、ネストされたストアード・プロシージャーの呼び出し、クライアント・アプリケーションまたは呼び出し側アプリケーションに対して結果セットを戻すことについてのデモを示します。 CLI サンプル・ディレクトリーで、″spcall″ アプリケーションを使用してプロシージャーを呼び出すことができます。 C および CPP サンプル・ディレクトリーで、 ″spclient″ アプリケーションを使用して、結果セットを戻さないプロシージャーを呼び出すこともできます。
whiles.db2	DEPT_MEDIAN プロシージャーは、 ″sample″ データベースの ″staff″ 表から ″dept″ の IN パラメーターによって識別される部署内の従業員の給与の中央値を入手します。給与の中央値は、給料の OUT パラメーターに割り当てられ、 ″whiles″ クライアントに戻されます。次に、whiles クライアントは、給与の中央値を印刷します。
whiles.sqc	DEPT_MEDIAN プロシージャーを呼び出します。

ADO、RDO、および MTS サンプル

表 11. ADO、RDO、および MTS サンプル

サンプル・プログラム名	プログラムの説明
Bank.vbp	顧客アカウントでトランザクションを実行する機能を使用して、銀行の支店に関するデータを作成し、保守する RDO プログラム。このプログラムには、アプリケーションがデータを格納するために必要な表を作成する DDL が入っているので、ユーザーが指定する任意のデータベースに使用できます。
Blob.vbp	この ADO プログラムは、BLOB データの検索について示します。 sample データベースの emp_photo 表から、ピクチャーを検索し、表示します。また、このプログラムは emp_photo 表のイメージを、ローカル・ファイルからのイメージに置き換えることができます。
BLOBAccess.dsw	このサンプルは、Microsoft Visual C++ を使用した ADO/Blob 強調表示アクセスを示します。これは Visual Basic サンプル、Blob.vbp と類似しています。 BLOB サンプルには次の 2 つのメイン関数があります。 1. サンプル・データベースから BLOB を読み取り、画面に表示する 2. ファイルから BLOB を読み取り、データベースに挿入する (インポート)
Connect.vbp	この ADO プログラムは、sample データベースに対して接続オブジェクトを作成し、接続を確立します。いったん完了すると、プログラムは切断され、終了します。
Commit.vbp	このアプリケーションは、 ADO の自動確定 / 手動確定機能の使用例を示します。プログラムは、従業員番号および名前を、 sample データベースの EMPLOYEE 表で照会します。ユーザーは、自動確定または手動確定モードのどちらでデータベースに接続するか、オプションで選択できます。自動確定モードでは、ユーザーがレコード上で行うすべての変更は、データベースで自動的に更新されます。手動確定モードでは、ユーザーはトランザクションを開始してから、変更を行う必要があります。トランザクションが開始してから行われた変更は、ロールバックを実行すれば取り消すことができます。トランザクションをコミットすることによって、変更を永続的に保管できます。プログラムを自動的に終了すると、変更がロールバックされます。

表 11. ADO、RDO、および MTS サンプル (続き)

サンプル・プログラム名	プログラムの説明
db2com.vbp	この Visual Basic プロジェクトは、Microsoft Transaction Server を使用した、データベースの更新例を示します。また、クライアント・プログラム db2mts.vbp が使用するサーバー DLL を作成します。 Visual Basic プロジェクトには、以下の 4 つのクラス・モジュールがあります。 • UpdateNumberColumn.cls • UpdateRow.cls • UpdateStringColumn.cls • VerifyUpdate.cls このプログラムの場合、一時表 DB2MTS が sample データベースに作成されます。
db2mts.vbp	これは、Microsoft Transaction Server を使用して、 db2com.vbp から作成されたサーバー DLL を呼び出す、クライアント・プログラム用の Visual Basic プロジェクトです。
Select-Update.vbp	この ADO プログラムは、Connect.vbp と同じ機能を実行しますが、 GUI インターフェースも提供しています。このインターフェースを使用して、ユーザーは sample データベースの ORG 表に保管されたデータを表示、更新、および削除することができます。
Sample.vbp	この Visual Basic プロジェクトは Keyset カーソルを使用し、ADO を経由して、sample データベース中のすべてのデータにグラフィカル・ユーザー・インターフェースを提供します。
VarCHAR.dsp	ADO を使用してテキスト・フィールドとして VarChar にアクセスする、 Visual C++ プログラム。グラフィカル・ユーザー・インターフェースを提供するので、ユーザーは sample データベースの ORG 表のデータを表示および更新できます。

オブジェクトのリンクと埋め込みのサンプル

表 12. オブジェクトのリンクと埋め込み (OLE) サンプル・プログラム

サンプル・プログラム名	プログラムの説明
sales	Microsoft Excel スプレッドシート上でのロールアップ照会を示します (Visual Basic で実現)。
names	ロータス ノーツ・アドレス・ブックを照会します (Visual Basic で実現)。
inbox	OLE/メッセージ機能を使用する Microsoft Exchange インボックス E メール・メッセージを照会します (Visual Basic で実現)。
invoice	Microsoft Word インボイス文書を E メール接続として送信する OLE オートメーション・ユーザー定義関数 (Visual Basic で実現)。

表 12. オブジェクトのリンクと埋め込み (OLE) サンプル・プログラム (続き)

サンプル・プログラム名	プログラムの説明
bcounter	インスタンス変数を使用したスクラッチパッドのデモを示す OLE オートメーション・ユーザー定義関数 (Visual Basic で実現)。
ccounter	カウンター OLE オートメーション・ユーザー定義関数 (Visual C++ で実現)。
salarysrv	sample データベースの STAFF 表の給与の中央値を計算する OLE オートメーション・ストアード・プロシージャー (Visual Basic で実現)。
salarycltvc	Visual Basic ストアード・プロシージャー salarysrv を呼び出す Visual C++ 組み込み SQL サンプル。
salarycltvb	Visual Basic ストアード・プロシージャー salarysrv を呼び出す Visual Basic DB2 CLI サンプル。
testcli	ストアード・プロシージャー tstsrv を呼び出す OLE オートメーション組み込み SQL クライアント・アプリケーション (Visual C++ で実現)。
tstsrv	クライアントとストアード・プロシージャーの間での各種の受け渡しのデモを示す OLE オートメーション・ストアード・プロシージャー (Visual C++ で実現)。

表 13. オブジェクトのリンクと埋め込みデータベース (OLE DB) 表関数

サンプル・プログラム名	プログラムの説明
jet.db2	Microsoft.Jet.OLEDB.3.51 Provider
mapi.db2	INTERSOLV Connect OLE DB for MAPI
msdaora.db2	Microsoft OLE DB Provider (Oracle 用)
msdasql.db2	Microsoft OLE DB Provider (ODBC ドライバー用)
msidxs.db2	Microsoft OLE DB Index Server Provider
notes.db2	INTERSOLV Connect OLE DB (ノーツ用)
sampprov.db2	Microsoft OLE DB Sample Provider
sqloledb.db2	Microsoft OLE DB Provider (SQL サーバー用)

コマンド行プロセッサーのサンプル

表 14. コマンド行プロセッサー (CLP) サンプル・プログラム

サンプル・ファイル名	ファイルの説明
const.db2	CHECK CONSTRAINT 節がある表を作成します。
cte.db2	共通表式を示します。この拡張 SQL ステートメントの例を示す同等のサンプル・プログラムは、 tabsql です。

表 14. コマンド行プロセッサー (CLP) サンプル・プログラム (続き)

サンプル・ファイル名	ファイルの説明
flt.db2	再帰的照会を示します。この拡張 SQL ステートメントの例を示す同等のサンプル・プログラムは、 recursql です。
join.db2	表の外部結合を示します。この拡張 SQL ステートメントの例を示す同等のサンプル・プログラムは、 joinsql です。
stock.db2	トリガーの使用例を示します。この拡張 SQL ステートメントの例を示す同等のサンプル・プログラムは、 trigsql です。
testdata.db2	ランダムに生成されるテスト・データを表に入れるための RAND() および TRANSLATE() などの DB2 組み込み関数を使用します。
thaisort.db2	このスクリプトは、タイ語を使用するユーザー専用です。タイ語の発音順のソートでは、主な母音と子音の事前ソートとスワップが必要になります。同じように、正確なソート順でデータを表示するために事後ソートも必要になります。ファイルでタイ語のソートを実現するには、事前ソートと事後ソートを行う UDF 関数を作成し、表を作成します。次いで、表データをソートするためにその表に対して関数を呼び出します。このプログラムを実行するには、最初に C ソース・ファイル udf.c からユーザー定義関数プログラム udf を作成しなければなりません。

ログ管理ユーザー出口サンプル

表 15. ログ管理ユーザー出口サンプル・プログラム

サンプル・ファイル名	ファイルの説明
db2uext2.cadsm	これは、データベース・ログ・ファイルを保存し検索するために、 ADSTAR DSM (ADSM) API を使用するサンプル・ユーザー出口です。このサンプルは、タイム・スタンプと受け取ったパラメーターを含む呼び出しの監査証跡 (オプションごとに別々のファイルに保存) を提供します。 また、問題判別のために、エラーのタイム・スタンプとエラー分離ストリングを含む呼び出しのエラー証跡も提供します。これらのオプションは使用不能であることがあります。 このファイルは、db2uext2.c に名前変更し、C プログラムとしてコンパイルしなければなりません。 UNIX および Windows 32 ビット・オペレーティング・システムで使用可能です。 OS/2 版は、db2uexit.cad です。
db2uexit.cad	これは、db2uext2.cadsm の OS/2 版です。このファイルは、db2uexit.c に名前変更し、C プログラムとしてコンパイルしなければなりません。

表 15. ログ管理ユーザー出口サンプル・プログラム (続き)

サンプル・ファイル名	ファイルの説明
db2uext2.cdisk	これは、出荷時の特定のプラットフォームに合わせた、システム・コピー・コマンドを使用するサンプル・ユーザー出口です。このプログラムは、タイム・スタンプと受け取ったパラメーターを含む呼び出しの監査証跡 (オプションごとに別々のファイルに保存) を提供します。また、問題判別のために、エラーのタイム・スタンプとエラー分離ストリングを含む呼び出しのエラー証跡も提供します。 これらのオプションは使用不能であることがあります。このファイルは、db2uext2.c に名前変更し、C プログラムとしてコンパイルしなければなりません。 UNIX および Windows 32 ビット・オペレーティング・システムで使用可能です。
db2uext2.ctape	これは、出荷時の特定の UNIX プラットフォームに合わせた、システム・テープ・コマンドを使用するサンプル・ユーザー出口です。プログラムは、データベース・ログ・ファイルをアーカイブおよび検索します。システム・テープ・コマンドの制限すべてが、このユーザー出口の制限になります。このサンプルは、タイム・スタンプと受け取ったパラメーターを含む呼び出しの監査証跡 (オプションごとに別々のファイルに保存) を提供します。 また、問題判別のために、エラーのタイム・スタンプとエラー分離ストリングを含む呼び出しのエラー証跡も提供します。 これらのオプションは使用不能であることがあります。 このファイルは、db2uext2.c に名前変更し、C プログラムとしてコンパイルしなければなりません。 UNIX プラットフォームでのみ使用可能です。

第2章 セットアップ

DB2 アプリケーションの構築および実行に使用するための適切な環境を作成するには、以下のものを正しくセットアップする必要があります。

1. コンパイラーまたはインタープリター
2. DB2 (データベース・マネージャー、DB2 AD クライアント、およびクライアント接続)
3. オペレーティング・システム環境
4. DB2 サンプル・データベース (任意)

コンパイラー環境またはインタープリター環境の検査

DB2 プログラムを開発するには、ご使用のオペレーティング・システムがサポートしているいずれかのプログラミング言語 (8ページの『各プラットフォームでサポートされるソフトウェア』のリストを参照) のコンパイラーまたはインタープリターを使う必要があります。まず最初に非 DB2 アプリケーションを作成することによって、既存のコンパイラー環境またはインタープリター環境が正しくセットアップされていることを確認なさるようお勧めします。 その後、問題が発生した場合、ご使用のコンパイラーまたはインタープリターに付属している資料を参照してください。

DB2 環境のセットアップ

DB2 環境をセットアップするには、次のものをインストールして作動させる必要があります。

- ご使用の環境のデータベース・インスタンスを使う、サーバー上のデータベース・マネージャー。データベース・インスタンスに関する情報が必要であれば、 417ページの『付録A. データベース・マネージャー・インスタンスについて』を参照してください。
- アプリケーションを開発するクライアントまたはサーバー・ワークステーション上にインストールされた DB2 AD クライアント。

- クライアント・ワークステーション上で開発している場合は、そのリモート・サーバーとの接続。

データベース・マネージャー構成ファイルの更新

このファイルには、アプリケーション開発のための重要な設定が含まれています。次のように入力することによってこれらの設定を変更することができます。

```
db2 update dbm cfg using <keyword> <value>
```

次のように入力することによってその設定を表示することができます。

```
db2 get dbm cfg
```

これらのコマンドの使用に関する詳細については、 *コマンド解説書* を参照してください。

- ストアード・プロシージャーの場合、キーワード KEEPDARI は、yes という省略時値を持っています。これは、ストアード・プロシージャーのプロセスを活動状態に保ちます。ストアード・プロシージャーを開発している場合、何度も同じストアード・プロシージャー・ライブラリーのロードをテストしたいこともあるでしょう。この省略時設定は、ライブラリーの再ロードを妨害することがあります。ストアード・プロシージャーを開発している間は、このキーワードの値を no に変更し、ストアード・プロシージャーの最終バージョンをロードする準備ができたら、このキーワード値を yes に戻すことが最善です。
- Java の場合には、JDK11_PATH キーワードを更新します。詳細については、77ページの『環境のセットアップ』を参照してください。

インストールとセットアップの詳細については、ご使用のオペレーティング・システムの*概説およびインストール* を参照してください。

すべてのものをインストールして作動させたなら、以下に示すいずれかの節の手順に従って、ご使用のオペレーティング・システム環境のセットアップを行うことができます。

- 41ページの『OS/2 環境の設定』
- 42ページの『UNIX 環境の設定』
- 43ページの『Windows 32 ビット・オペレーティング・システム環境の設定』

オペレーティング・システム環境のセットアップ後、サンプル・データベースを作成することができます。このデータベースは本書の例で使用します。この

データベースを作成する方法は、 47ページの『サンプル・データベースの作成、カタログ、およびバインド』を参照してください。

OS/2 環境の設定

ほとんどの OS/2 コンパイラーは、環境変数を使用してさまざまなオプションを制御します。 CONFIG.SYS ファイル内でそれらの変数を設定するか、またはコマンド・ファイルを作成してそれらの変数を設定することができます。

CONFIG.SYS

CONFIG.SYS ファイルに環境変数を設定する利点は、一度それらを正確に指定すれば、コンピューターを始動 (ブート) するたびに設定されることです。

コマンド・ファイル

コマンド・ファイル内に環境変数を設定する利点は、設定するパスの長さが短くて済み、また複数のコンパイラーを使用する上での柔軟性が増すということです。欠点は、各プログラミング・セッションの開始時に、コマンド・ファイルを最初に実行しなければならないことです。

コマンド・ファイルを実行することによって環境変数を設定する場合は、アプリケーションの作成も、環境変数を設定したその同じウィンドウで行う必要があります。別のウィンドウでアプリケーションを作成すると、最初のウィンドウで設定した同じオプションを使用しないことになる場合があります。

DB2 AD クライアントをインストールすると、以下に示すステートメントが CONFIG.SYS ファイルに置かれます。

```
set LIB=%DB2PATH%¥lib;%LIB%
```

本書のコマンド・ファイルは、このステートメントが存在していることを前提にしています。 DB2 AD クライアントのインストール後に CONFIG.SYS ファイルを編集する場合は、このステートメントを除去しないでください。

さらに、以下に示す環境変数が DB2 によって自動的に更新されます。

- PATH。ディレクトリー %DB2PATH%¥bin が組み込まれます。
- LIBPATH。ディレクトリー %DB2PATH%¥dll が組み込まれます。

DB2 が更新する Java 環境変数については、81ページの『OS/2』を参照してください。

さらに、以下に示すプログラミング言語のいずれかを使用する場合は、CONFIG.SYS ファイルに以下の該当するステートメントが入っていなければなりません。

C/C++ `set INCLUDE=%DB2PATH%¥include;%INCLUDE%`

FORTAN

```
set FINCLUDE=%DB2PATH%¥include;%FINCLUDE%
```

IBM COBOL

```
set SYSLIB=%SYSLIB%;%DB2PATH%¥include¥cobol_a
```

Micro Focus COBOL

```
set COBCPY=%DB2PATH%¥include¥cobol_mf;%COBCPY%
```

OS/2 上では、 DB2PATH および DB2INSTPROF 以外に CONFIG.SYS に DB2 環境変数を定義するべきではありません。 すべての DB2 変数は、グローバル・レベル、インスタンス・レベル、またはインスタンス・ノード・レベル (パラレル・エディション) のいずれかで、 DB2 インスタンス・プロファイル・レジストリーに定義する必要があります。 `db2set.exe` コマンドを使用して、これらの変数を設定、修正、およびリストしてください。

注: DB2INSTDEF 登録変数を設定する場合、 DB2INSTANCE は必須ではありません。 DB2INSTDEF は、DB2INSTANCE が設定されない場合に使用される省略時インスタンス名を定義します。

UNIX 環境の設定

データベース・マネージャーのインストール時に作成したデータベース・インスタンスにアクセスできるように、環境変数を設定する必要があります。 *DB2 ユニバーサル・データベース (UNIX 版) 概説およびインストール* には、環境変数の設定に関する一般情報が記載されています。この節では、データベース・インスタンスにアクセスするために環境変数を設定するための特定の指示を説明します。

それぞれのデータベース・マネージャー・インスタンスには 2 つのファイル、`db2profile` および `db2cshrc` があります。これらには、そのインスタンス用の環境変数を設定するためのスクリプトが含まれています。使用するシェルによって、以下を入力してスクリプトを実行してください。

bash または Korn シェルの場合

```
. $HOME/sqllib/db2profile
```

C シェルの場合

```
source $HOME/sqllib/db2cshrc
```

ここで `$HOME` は、インスタンス所有者のホーム・ディレクトリーです。

このコマンドを .profile または .login ファイルに組み込めば、ログオン時に自動的に実行されるので便利です。

使用している UNIX プラットフォームに応じて、以下の環境変数が DB2 インスタンスの際に自動的に更新されます。

AIX:

- PATH。sqllib/bin を含むいくつかの DB2 ディレクトリーが組み込まれます。
- LIBPATH。ディレクトリー sqllib/lib が組み込まれます。

HP-UX:

- PATH。sqllib/bin を含むいくつかの DB2 ディレクトリーが組み込まれます。
- SHLIB_PATH。ディレクトリー sqllib/lib が組み込まれます。

Linux、DYNIX/ptx、および Solaris:

- PATH。sqllib/bin を含むいくつかの DB2 ディレクトリーが組み込まれます。
- LD_LIBRARY_PATH。ディレクトリー sqllib/lib が組み込まれます。

Silicon Graphics IRIX:

- PATH。sqllib/bin を含むいくつかの DB2 ディレクトリーが組み込まれます。
- LD_LIBRARY_PATH。ディレクトリー sqllib/lib が組み込まれます (o32 オブジェクト・タイプ・アプリケーションに必要)。
- LD_LIBRARYN32_PATH。ディレクトリー sqllib/lib32 が組み込まれます (n32 オブジェクト・タイプ・アプリケーションに必要)。

DB2 が更新する Java 環境変数については、77ページの『環境のセットアップ』を参照してください。

Windows 32 ビット・オペレーティング・システム環境の設定

Windows NT または Windows 2000 上に DB2 AD クライアントをインストールすると、インストール・プログラムは、構成レジストリーにある環境変数 INCLUDE、LIB、PATH、DB2PATH、および DB2INSTANCE を更新します。デフォルトのインスタンスは DB2 です。

DB2 が更新する Java 環境変数については、87ページの『Windows 32 ビット・オペレーティング・システム』を参照してください。

Windows 98 または Windows 95 上に DB2 AD クライアントをインストールすると、インストール・プログラムは、autoexec.bat ファイルを更新します。

これらの環境変数を一時変更して、マシンまたは現在ログオンしているユーザーの値を設定することができます。これらの値を一時変更するには、以下のいずれかを使用してください。

- Windows NT のコントロール・パネル
- Windows 2000 のコントロール・パネル
- Windows 95 または Windows 98 のコマンド・ウィンドウ
- Windows 95 または Windows 98 の autoexec.bat ファイル

注:

1. これらの環境変数の変更は、注意深く行ってください。 DB2PATH 環境変数は変更**しないでください**。
2. コマンド内で変数 %DB2PATH% を使用するとき、set LIB="%DB2PATH%¥lib";%LIB% のように、全パスを引用符で囲んでください。 DB2 バージョン 7.1 では、この変数の省略時インストール値は ¥Program Files¥sqllib で、スペースが入っており、引用符を使用していないので、エラーが起こる場合があります。

これらの環境変数を更新することにより、 Windows 32 ビット・オペレーティング・システム上でほとんどのプログラムを動作させることができます。さらに、以下に示す特定のステップに従って、 DB2 アプリケーションを実行させなければなりません。

- C または C++ プログラムを作成するときは、必ず INCLUDE 環境変数に %DB2PATH%¥INCLUDE が最初のディレクトリーとして含まれていなければなりません。

  たとえば、 Microsoft Visual C++ コンパイラー環境セットアップ・ファイル Vc¥bin¥vcvars32.bat には、以下のコマンドがあります。

  ```
  set INCLUDE=%MSVCDir%¥INCLUDE;%MSVCDir%¥MFC¥INCLUDE;%MSVCDir%¥ATL¥INCLUDE;%INCLUDE%
  ```

  このファイルを DB2 で使用するには、まず、%DB2PATH%¥INCLUDE パスを設定する %INCLUDE% を以下のようにリストの末尾から先頭に移動します。

  ```
  set INCLUDE=%INCLUDE%;%MSVCDir%¥INCLUDE;%MSVCDir%¥MFC¥INCLUDE;%MSVCDir%¥ATL¥INCLUDE;
  ```

- Micro Focus COBOL プログラムを作成するときは、 COBCPY 環境変数を %DB2PATH%¥INCLUDE¥cobol_mf を指すように設定してください。

- IBM COBOL プログラムを作成するときは、 SYSLIB 環境変数を %DB2PATH%¥INCLUDE¥cobol_a を指すように設定してください。
- 以下を使用して、必ず LIB 環境変数が %DB2PATH%¥lib を指すようにしてください。

  ```
  set LIB="%DB2PATH%¥lib";%LIB%
  ```

- DB2COMM 環境変数を、必ずリモート・データベースのサーバーで設定してください。
- 機密保護サービスが、SERVER 認証用のサーバー、および CLIENT 認証のレベルに応じてクライアントで開始されているようにしてください。 機密保護サービスを開始するには、 NET START DB2NTSECSERVER コマンドを使用します。

注:

1. すべての DB2 環境変数は、ユーザーの環境で定義するか、またはレジストリー変数としてセットアップすることができます。 レジストリー変数については、*管理の手引き* を参照してください。 db2set コマンドについては、*コマンド解説書* を参照してください。
2. DB2INSTANCE は、ユーザー環境レベルだけで定義する必要があります。 DB2INSTANCE が設定されていないときに使用するデフォルトのインスタンス名が定義してある DB2INSTDEF レジストリー変数を使用する場合、DB2INSTANCE は必要ありません。
3. Windows NT または Windows 2000 環境におけるデータベース・マネージャーは NT サービスとして実装されているため、このサービスが正常に開始されていれば、他の問題が生じていてもエラーや警告は戻されません。 つまり、db2start または NET START コマンドを実行した場合、いずれかの通信サブシステムが開始できなかったとしても警告が戻されないということです。そのため、ユーザーはこれらのコマンドの実行中にエラーが発生していないかを確認するため、常に Windows NT、Windows 2000 イベント・ログ、または DB2DIAG.LOG を調べなければなりません。

サーバー上の通信を使用可能にする

この節では、DB2 ユニバーサル・データベース・サーバーに接続する方法について説明します。

sample データベースのインストール、カタログおよびバインドを開始する前に、サーバーが操作可能であり、カタログされるプロトコルをサポートするように構成する必要があります。 サーバーで以下の手順を実行してください。

1. db2comm 環境変数が設定されているようにします。たとえば、TCP/IP が使用されている場合には、次のように入力してください。

   ```
   db2set DB2COMM=tcpip
   ```

   TCP/IP サポート用のプロトコルが構成されているようにします。

   TCP/IP 設定をサービス・ファイルに追加する際の指示に関しては、ご使用のプラットフォームの*概説およびインストール* を参照してください。
2. 以下のように入力して、データベース・インスタンスを開始します。

   ```
   db2start
   ```

サンプル・データベースにユーティリティーをバインドする作業は、クライアントから行われなければなりません。詳細については、50ページの『バインド』を参照してください。

Windows NT および Windows 2000

DB2 (Windows NT 版) または DB2 (Windows 2000 版) の実行用システムでは、データベース・インスタンスをサービスとして開始する必要があります。このステップは以下のとおりです。

- 通信プロトコルを使用している場合、 db2comm 環境変数が Windows NT または Windows 2000 コントロール・パネルのシステム環境変数セクションで設定されているようにします。
- セキュリティー・サービスを開始します。これは自動的に実行することもできますし (下記の注を参照)、または、以下のコマンドを使用してこのサービスを手動で開始することもできます。

  ```
  NET START DB2NTSECSERVER
  ```
- 以下のように入力して、インスタンスを開始します。

  ```
  db2start
  ```

機密保護サービスの自動開始。 通常、機密保護サービスを自動的に開始したいのは、ワークステーションが、クライアント認証用に構成されたサーバーと接続する DB2 クライアントとして動作している場合だけです。機密保護サービスを自動的に開始させるためには、以下のことを実行してください。

Windows NT

1. 「スタート」ボタンをクリックします。
2. 「設定」をクリックします。
3. 「コントロール パネル」をクリックします。
4. 「コントロール パネル」で、「サービス」をクリックします。

5. 「サービス」ウィンドウで、「DB2 セキュリティー・サーバー (DB2 Security Server)」を強調表示します。
6. 設定値として「開始」と「自動」が表示されない場合は、「スタートアップ」をクリックします。
7. 「自動」をクリックします。
8. 「OK」をクリックします。
9. 設定値を有効にするために、マシンをリブートします。

Windows 2000

1. 「スタート」ボタンをクリックします。
2. 「設定」をクリックします。
3. 「コントロール パネル」をクリックします。
4. 「管理ツール」をクリックします。
5. 「サービス」をクリックします。
6. 「サービス」ウィンドウで、「DB2 セキュリティー・サーバー (DB2 Security Server)」を強調表示します。
7. 設定値として「開始」と「自動」が表示されない場合は、トップ・メニューから「アクション (Action)」をクリックします。
8. 「プロパティー (Properties)」をクリックします。
9. 「一般 (General)」タブにいることを確認します。
10. 「スタートアップの種類」ドロップダウン・メニューから「自動」を選択します。
11. 「OK」をクリックします。
12. 設定値を有効にするために、マシンをリブートします。

サンプル・データベースの作成、カタログ、およびバインド

DB2 に付属しているサンプル・プログラムを使用するには、サーバー・ワークステーション上に sample データベースを作成する必要があります。 sample データベースの内容のリストは、 *SQL 解説書* を参照してください。

リモート・クライアントを使ってサーバー上にある sample データベースにアクセスする予定であれば、その sample データベースをクライアント・ワークステーション上でカタログする必要があります。

さらに、別バージョンの DB2 を実行しているサーバーや、別のオペレーティング・システム上で稼働しているサーバーにある sample データベースに、リ

モート・クライアントを使ってアクセスする予定であれば、 DB2 CLI を含むデータベース・ユーティリティーを sample データベースにバインドする必要があります。

作成

sample データベースを作成するには、SYSADM 権限が必要です。 SYSADM 権限についての詳しい情報が必要な場合は、ご使用のオペレーティング・システムの*概説およびインストール* を参照してください。

データベースを作成するには、サーバーで以下のことを行ってください。

1. db2sampl (sample データベースを作成するプログラム) が必ず、ご使用のパスにあるようにします。ファイル db2profile または db2cshrc は、ご使用のパスの db2sampl に置かれます。変更しない限り、そのパスがファイルのある場所です。
   - UNIX サーバーでは、db2sampl は次の場所にあります。

     ```
     $HOME/sqllib/bin
     ```

     ここで $HOME は、DB2 インスタンス所有者のホーム・ディレクトリーです。
   - OS/2 および Windows では、db2sampl は次の場所にあります。

     ```
     %DB2PATH%¥bin
     ```

     ここで %DB2PATH% は、 DB2 がインストールされているパスです。
2. DB2INSTANCE 環境変数が、 sample データベースを作成するインスタンスの名前に設定されていることを確認してください。この変数が設定されていない場合は、以下に示すコマンドで設定することが可能です。
   - UNIX プラットフォームの場合

     bash または Korn シェルでは次のように入力します。

     ```
     DB2INSTANCE=instance_name
     export DB2INSTANCE
     ```

     C シェルでは次のように入力します。

     ```
     setenv DB2INSTANCE instance_name
     ```
   - OS/2 および Windows の場合は、次のように入力します。

     ```
     set DB2INSTANCE=instance_name
     ```

   ここで *instance_name* は、データベース・インスタンスの名前です。

3. db2sampl と、それに続いて、サンプル・データベースを作成したい場所を入力することによって、 sample データベースを作成します。 UNIX プラットフォームでは、その場所のことをパス といい、次のように入力します。

   ```
   db2sampl path
   ```

   OS/2 および Windows では、その場所のことをドライブ といい、次のように入力します。

   ```
   db2sampl drive
   ```

   パスまたはドライブを指定しないと、インストール・プログラムは、データベース・マネージャー構成ファイルの DFTDBPATH パラメーターによって指定されているデフォルトのパスまたはドライブに、サンプル表をインストールします。構成ファイルに関する情報が必要な場合には、 *管理の手引き* を参照してください。

   データベースの認証タイプは、データベースが作成されるインスタンスの認証タイプと同じです。データベース・インスタンスの作成時の認証の指定の詳細については、 *概説およびインストール* を参照してください。

ホストまたは AS/400 サーバー上での作成

DB2 (OS/390 版) などのホスト・サーバー、あるいは AS/400 サーバーに対してサンプル・プログラムを実行したい場合は、 *SQL 解説書* に記述されているサンプル表を含んだデータベースを作成する必要があります。 サンプル・プログラム expsamp を参照することができます。これは、STAFF および ORG 表を使用しており、 API を使用して表および表データを DB2 コネクト・データベースとの間でインポート / エクスポートする方法を例示しています。

データベースを作成するには、次の手順で行います。

1. db2sampl を使用して、 DB2 共通サーバー・インスタンスに sample データベースを作成します。
2. sample データベースに接続します。
3. サンプル表をファイルにエクスポートします。
4. DB2 コネクト・データベースに接続します。
5. サンプル表を作成します。
6. サンプル表をインポートします。

ファイルのエクスポートおよびインポートに関する情報が必要な場合は、 データ移動ユーティリティー 手引きおよび解説書 を参照してください。データベースの接続および表の作成については、*SQL 解説書* を参照してください。

カタログ

サーバー上の sample データベースにリモート・クライアントからアクセスするには、クライアント・ワークステーションで sample データベースをカタログする必要があります。

サーバー・ワークステーションでは、 sample データベースをカタログする必要がありません。データベースの作成時にカタログされているからです。

カタログを作成すると、クライアント・ワークステーションのデータベース・ディレクトリーが、クライアント・アプリケーションのアクセスしたいデータベース名に更新されます。 クライアントの要求を処理するときに、データベース・マネージャーはカタログされた名前を使用して、データベースを見つけて接続します。

概説およびインストール には、データベースのカタログ作成に関する一般情報が記載されています。この節では、sample データベースをカタログするための具体的な説明を提供します。

リモート・クライアント・ワークステーションからサンプル・データベースをカタログするには、次のように入力します。

```
db2 catalog database sample as sample at node nodename
```

ここで *nodename* は、サーバー・ノードの名前です。

概説およびインストール は、通信プロトコルのセットアップの一部としてノードをカタログする方法を説明しています。リモート・ノードもカタログしてからでないと、データベースには接続できません。

バインド

また、DB2 の異なるバージョンを実行しているか、または別のオペレーティング・システム上で実行しているリモート・クライアントから、サーバー上の sample データベースにアクセスする予定である場合には、 DB2 CLI を含むデータベース・ユーティリティーを、 sample データベースにバインドする必要があります。

バインドでは、アプリケーションの実行時にデータベースにアクセスするために、データベース・マネージャーが必要とするパッケージを作成します。 バイ

ンドは、プリコンパイル時に作成されるバインド・ファイルに対して BIND ファイルを指定することによって明示的に実行できます。

コマンド解説書 は、データベース・ユーティリティーをバインドすることに関する一般的な情報を提供しています。 この節では、データベース・ユーティリティーを sample データベースにバインドするための具体的な手順を説明します。

使用するクライアント・ワークステーションのプラットフォームに応じて、データベース・ユーティリティーは異なる仕方でバインドします。

OS/2 クライアント・ワークステーションの場合

1. 次のように入力して、sample データベースに接続します。

   ```
   db2 connect to sample user userid using password
   ```

   ここで、*userid* と *password* は、 sample データベースが置かれているインスタンスのユーザー ID とパスワードを表します。

   ユーティリティーは、 DB2 によってこのコマンドを使用してデータベースに自動的にバインドされるので、ユーザーはこれらを明示的にバインドする必要はありません。
2. コマンド行プロセッサーを終了し、バインド・メッセージ・ファイル bind.msg を調べて、バインドが成功したかを確認します。

UNIX クライアント・ワークステーションの場合

1. 次のように入力して、sample データベースに接続します。

   ```
   db2 connect to sample user userid using password
   ```

   ここで、*userid* と *password* は、 sample データベースが置かれているインスタンスのユーザー ID とパスワードを表します。
2. 次のように入力して、データベースにユーティリティーをバインドします。

   ```
   db2 bind BNDPATH/@db2ubind.lst blocking all sqlerror continue \
   messages bind.msg
   db2 bind BNDPATH/@db2cli.lst blocking all sqlerror continue \
   messages cli.msg
   ```

   ここで、*BNDPATH* はバインド・ファイルが置かれているパスです。たとえば、$HOME/sqllib/bnd ($HOME は DB2 インスタンス所有者のホーム・ディレクトリー)。
3. バインドが成功したかどうかを、バインド・メッセージ・ファイル bind.msg および cli.msg を調べて確認します。

Windows 32 ビット・オペレーティング・システムが稼働しているクライアント・ワークステーションの場合

1. 「スタート」メニューから「プログラム」を選択します。
2. 「プログラム」メニューから「IBM DB2」を選択します。
3. 「IBM DB2」メニューから、「DB2 コマンド・ウィンドウ (DB2 Command Window)」を選択します。

   コマンド・ウィンドウが表示されます。
4. 次のように入力して、sample データベースに接続します。

   ```
   db2 connect to sample user userid using password
   ```

   ここで、*userid* と *password* は、 sample データベースが置かれているインスタンスのユーザー ID とパスワードを表します。
5. 次のように入力して、データベースにユーティリティーをバインドします。

   ```
   db2 bind "%DB2PATH%¥bnd¥@db2ubind.lst" blocking all
   sqlerror continue messages bind.msg
   ```

   ここで %DB2PATH% は、DB2 のインストール先を示すパスです。
6. コマンド・ウィンドウを終了し、バインド・メッセージ・ファイル bind.msg を調べて、バインドが成功したかを確認します。

すべてのプラットフォーム

DRDA に従っているアプリケーション・サーバー上に sample データベースを作成した場合、 db2ubind.lst の代わりに、次の .lst ファイルのいずれかを指定してください。

ddcsmvs.lst
: DB2 (OS/390 版) の場合

ddcsvm.lst
: DB2 (VM 版) の場合

ddcsvse.lst
: DB2 (VSE 版) の場合

ddcs400.lst
: DB2 (AS/400 版) の場合

ご使用のプラットフォームの*概説およびインストール* は、データベース・ユーティリティーのバインドに関する一般的な情報を提供しています。

次に行うこと

環境をセットアップしたなら、DB2 アプリケーションを作成する準備ができました。これ以降の各章では、サンプル・プログラムについて説明し、またそれらをコンパイル、リンク、および実行する方法を示しています。最初に 55ページの『第3章 DB2 アプリケーションの構築についての一般情報』 を読んだ後、構築しているアプリケーションごとに特定の章に進んでください。

Java でプログラミングをしている場合には、 75ページの『第4章 Java アプレットおよびアプリケーションの構築』を参照してください。

SQL プロシージャーでプログラミングしている場合には、 105ページの『第5章 SQL プロシージャーの構築』を参照してください。

DB2 API、DB2 CLI、組み込み SQL を利用したプログラミングについては、ご使用のプラットフォームに該当する『アプリケーションの構築』の章を参照してください。

詳細については、以下の資料を参照してください。

- 組み込み SQL、JDBC、および SQLJ を利用したアプリケーションおよびユーザー定義関数 (UDF) については、 *アプリケーション開発の手引き* を参照してください。
- DB2 CLI または ODBC を使用するアプリケーションについては、 *コール・レベル・インターフェースの手引きおよび解説書* を参照してください。
- DB2 API アプリケーションについては、*管理 API 解説書* を参照してください。

第3章 DB2 アプリケーションの構築についての一般情報

この章の情報は、複数のオペレーティング・システムに適用されます。取り上げられているトピックの多くは、 DB2 がサポートする大部分のプラットフォームに適用されます。

DB2 アプリケーション開発の最新の更新事項については、次の Web ページを参照してください。

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

DB2 プログラムを構築し、実行するための一般事項

1. アプリケーション環境
   - OS/2 で CONFIG.SYS ファイルの代わりにコマンド・ファイルで環境変数を設定した場合、アプリケーションを同一のウィンドウで構築しなければなりません。
   - UNIX では、DB2 アプリケーションは、環境変数を設定したシェルから構築して実行しなければなりません。 使用するシェルによっては、`db2profile` または `db2cshrc` を実行することで、これを行うことができます。
   - Windows 32 ビット・オペレーティング・システムでは、アプリケーションを DB2 コマンド・ウィンドウで構築しなければなりません。 詳細については、39ページの『第2章 セットアップ』 を参照してください。
2. 組み込み SQL を含む DB2 プログラムを作成し、何らかの DB2 プログラムを実行するには、サーバー上のデータベース・マネージャーを開始する必要があります。データベース・マネージャーを開始するには SYSADM (システム管理) 権限が必要です。 SYSADM 権限の詳細については、*概説およびインストール* を参照してください。

データベース・マネージャーを開始する (まだ実行中でない場合) には、サーバー上で以下に示すコマンドを入力します。

```
db2start
```

3. 実動用のアプリケーションを構築しているとき、実行可能ファイルに組み込む DB2 ランタイム・パスは、インストール・パスにすべきであり、アプリケーションを開発しているローカル DB2 インスタンスのパスにすべきではありません。本書は、開発環境でアプリケーションを構築する方法を示すことを目的としているので、 UNIX 上の sqllib/include および sqllib/lib のインスタンス・コピー、 OS/2 や Windows 32 ビット・オペレーティング・システム上の %DB2PATH%¥include および %DB2PATH%¥lib のインスタンス・コピーを提供しています。
4. サンプル・プログラムの変更や構築を行うときは、それらの言語のサンプルのうち使用する予定のものを、 UNIX の場合には sqllib/samples から、OS/2 または Windows 32 ビット・オペレーティング・システムの場合には %DB2PATH%¥samples から、あらかじめ自分の作業ディレクトリーにコピーすることをお勧めします。こうすることで、元のサンプルを参照する必要が将来生じたときに備えて、それらのサンプルを保存しておくことができます。

ビルド・ファイル、makefile、およびエラー検査ユーティリティー

DB2 には、プログラム開発における必要に応えるための一群の構築ツールが用意されています。これらのツールを利用することにより、 DB2 の幅広い機能性を示している、提供されたサンプル・プログラムを簡単に構築することができます。また、これらのツールを使って独自のデータベース・プログラムを構築することも可能です。 DB2 には、サポートしているコンパイラーごとにビルド・ファイル、makefile、エラー検査ユーティリティーなどが用意されています。これらのものはサンプル・プログラムとともにサンプル・ディレクトリーにあります。この節では、これらのツールの使い方を説明します。

ビルド・ファイル

以下の各章で使われているファイルには、サポートしているプラットフォームのコンパイラーでプログラムを構築するための compile および link コマンドが含まれています。これらのファイルは、OS/2 ではコマンド・ファイル、UNIX ではスクリプト・ファイル、 Windows 32 ビット・オペレーティング・システムではバッチ・ファイルと呼ばれています。本書では、これらのファイルを総称してビルド・ファイルと呼ぶことにします。

DB2 では、ビルド・ファイルが、各言語のサンプル・プログラムと同じディレクトリーの中で、構築されるプログラムを使用できるプラットフォームの各言

語ごとに用意されています。 表16 では、 Windows 32 ビット・オペレーティング・システムを除く、サポートされているすべてのプラットフォーム上のすべての言語のビルド・ファイルをリストします。ファイル名の拡張子は、省略されています。 OS/2 の場合、拡張子は .cmd で、 Windows 32 ビット・オペレーティング・システムの場合、拡張子は .bat です。 UNIX プラットフォームの場合には、拡張子はありません。

Windows 32 ビット・オペレーティング・システムの場合、サポートされている 2 つの C++ コンパイラー (Microsoft Visual C++ および IBM VisualAge C++) があります。これらのコンパイラーの場合、bldclisp を除き、それぞれ、 ″m″ または ″v″ が各ビルド・ファイル名の ″bld″ の後に挿入されており、 bldmclis または bldvclis のいずれかになります。これらのビルド・ファイルは、58ページの表17にリストされています。 .bat という拡張子は省略されています。

表 16. DB2 ビルド・ファイル

ビルド・ファイル	構築されるプログラムのタイプ
bldsqlj	Java Embedded SQLJ アプリケーション。
bldsqljs	Java Embedded SQLJ ストアード・プロシージャー。
bldcli	DB2 CLI アプリケーション (組み込み SQL を含むものと含まないもの)。
bldapi	DB2 CLI アプリケーション (組み込み SQL を含むものと含まないもの)。データベースを作成および除去するための DB2 API を含む、 utilapi ユーティリティー・ファイル内のリンクが必要です。
bldclisp	DB2 CLI ストアード・プロシージャー (組み込み SQL なし)。
bldapp	アプリケーション・プログラム (組み込み SQL を含むものと含まないもの)。
bldsrv	組み込み SQL ストアード・プロシージャー。
bldudf	ユーザー定義関数 (UDF)。
bldmt	マルチスレッド組み込み SQL アプリケーション (サポートされている UNIX プラットフォーム上の C/C++ にのみ使用可能)。
bldevm	イベント・モニター・サンプル・プログラム evm (AIX および OS/2 上でのみ使用可能)。

表 17. C/C++ ビルド・ファイル (Windows 32 ビット・オペレーティング・システム用)

ビルド・ファイル	構築されるプログラムのタイプ
bldmcli	Microsoft Visual C++ DB2 CLI アプリケーション (組み込み SQL を含むものと含まないもの)。
bldvcli	VisualAge C++ DB2 CLI アプリケーション (組み込み SQL を含むものと含まないもの)。
bldmapi	Microsoft Visual C++ DB2 CLI アプリケーション (組み込み SQL を含むものと含まないもの)。データベースを作成および除去するための DB2 API を含む、 `utilapi` ユーティリティー・ファイル内のリンクが必要です。
bldvapi	VisualAge C++ DB2 CLI アプリケーション (組み込み SQL を含むものと含まないもの)。データベースを作成および除去するための DB2 API を含む、 `utilapi` ユーティリティー・ファイル内のリンクが必要です。
bldmclis	Microsoft Visual C++ DB2 CLI ストアード・プロシージャー (組み込み SQL なし)。
bldvclis	VisualAge C++ DB2 CLI ストアード・プロシージャー (組み込み SQL なし)。
bldmapp	Microsoft Visual C++ アプリケーション・プログラム (組み込み SQL を含むものと含まないもの)。
bldvapp	VisualAge C++ アプリケーション・プログラム (組み込み SQL を含むものと含まないもの)。
bldmsrv	Microsoft Visual C++ 組み込み SQL ストアード・プロシージャー。
bldvsrv	VisualAge C++ 組み込み SQL ストアード・プロシージャー。
bldmudf	Microsoft Visual C++ ユーザー定義関数 (UDF)。
bldvudf	VisualAge C++ ユーザー定義関数 (UDF)。

本書にはビルド・ファイルが記載されていますが、それはこれらのファイルに、サポートしているコンパイラーで各種のプログラムを構築するときに推奨される DB2 のコンパイルおよびリンクのオプションの使い方が非常に分かりやすく示されているからです。通常はこれらの他にも使用できるコンパイルとリンクのオプションは多数あり、ユーザーはそれらを自由に試すことができます。用意されているコンパイルとリンクのすべてのオプションについて知りたい場合は、ご使用のコンパイラーのマニュアルを参照してください。開発者はそれらのビルド・ファイルを使ってサンプル・プログラムを構築するだけでな

く、自分のプログラムを構築することも可能です。サンプル・プログラムをユーザーが変更できるテンプレートとして利用することにより、プログラム開発に役立てることができます。

これらのビルド・ファイルの便利な点として、コンパイラーで使用可能なファイル名が付いていれば、どのソース・ファイルでも構築するように設計されています。これは、プログラム名がファイル中にハードコーディングされる makefile とは異なります。ビルド・ファイルでは、UNIX の場合には $1 変数、また OS/2 や Windows 32 ビット・オペレーティング・システムの場合には %1 変数を使って、プログラム名を内部的に置き換えます。同様にして名前が付けられた他の変数は、必要とされる他の引き数を置き換えます。各ビルド・ファイルは特定のプログラム構築、たとえば DB2 API、 DB2 CLI、組み込み SQL、ストアード・プロシージャー、ユーザー定義関数など、それぞれに適したものになっているため、ビルド・ファイルは短時間で、かつ簡単に試してみることが可能です。それぞれのタイプのビルド・ファイルは、そのビルド・ファイルが想定している特定の種類のプログラムをコンパイラーがサポートしている場合に用意されています。

ビルド・ファイルが作成するオブジェクト・ファイルや実行可能ファイルは、ソース・ファイルが修正されない場合でさえ、プログラムが構築されるたびに自動的に上書きされます。これは makefile の場合とは異なっています。つまり、開発者は以前のオブジェクト・ファイルや実行可能ファイルを削除したり、またはソースを修正したりすることなく、既存のプログラムを再構築することができます。

ビルド・ファイルには、サンプル・データベース用のデフォルト設定が組み込まれています。ユーザーが別のデータベースにアクセスする場合は、別のパラメーターを指定してデフォルトの指定を変更するだけで済みます。その別のデータベースを一貫して使用する予定であれば、ビルド・ファイルの中にある sample を置き換えて、このデータベースの名前をハードコーディングすることができます。

組み込み SQL プログラムに使用されるビルド・ファイルは別のファイル embprep を呼び出し、これらの組み込み SQL プログラムが使用するプリコンパイルとバインドのステップがその中に入っています。これらのステップでは、組み込み SQL プログラムをどこに組み込むかによって、任意指定のパラメーターであるユーザー ID とパスワードが必要になる場合があります。

データベースがあるサーバー・インスタンス上で開発者がプログラムを構築する場合はユーザー ID とパスワードが共通であるため、ビルド・ファイルに入力パラメーターとして指定する必要がないからです。 その一方で、開発者が別

のインスタンスで作業する場合、たとえばサーバー・データベースへリモート・アクセスするクライアント・マシン上で作業する場合などは、ビルド・ファイルを実行するときにこれらのパラメーターを指定しておくことが必要です。 embprep ファイルは、makefile によっても使用されます。このファイルの詳細については、『makefile』を参照してください。

最後の点として、ビルド・ファイルは開発者が自分の都合に合わせて修正することが可能です。開発者は (前述のように) ビルド・ファイル中のデータベース名を変更できるだけでなく、他のパラメーターをファイル内にハードコーディングしたり、コンパイルとリンクのオプションを変更したり、デフォルトの DB2 インスタンス・パスを変更したりすることが簡単に行えます。ビルド・ファイルはその性質上、簡単で分かりやすく、具体的であるため、自分の必要に応じてそれらのファイルに手を加えるのが容易です。

makefile

サポートしているコンパイラーごとに用意されているサンプル・ディレクトリーの中には、提供されているサンプル・プログラムを構築するための makefile が入っています。 makefile は、コンパイラーに付属している大部分の DB2 サンプル・プログラムを構築します。また、サンプル・プログラムのコンパイルごとに使われる共通要素の多くに対して変数を利用します。 makefile の構文、およびそれらのコマンドからの出力は、提供されているビルド・ファイルのものとはいくつかの重要な点で異なっています。以下に示すように、make コマンドの使用は簡単でありながら強力です。

make <program_name>
: 指定したプログラムのコンパイルとリンクを実行します。

make all
: makefile に記述されているすべてのプログラムのコンパイルとリンクを実行します。

make clean
: makefile に記述されているすべてのプログラムの中間ファイル (オブジェクト・ファイルなど) を削除します。

make cleanall
: makefile に記述されているすべてのプログラムの中間ファイルと実行可能ファイルをすべて削除します。

ビルド・ファイルとは異なり、 makefile はその中に記述されているプログラムの既存の中間ファイルと実行可能ファイルを上書きしません。 make all コマンドを使用した場合、他のファイルに実行可能ファイルがすでにあると

make all はそれらのファイルを単に無視するだけなので、いくつかのファイルの実行可能ファイルを作成するのであれば makefile による処理の方が高速です。ただし、既存のオブジェクト・ファイルと実行可能ファイルが不要な場合には、 make clean および make cleanall コマンドを使ってそれらのファイルを除去しなければなりません。

makefile はプログラム開発に使用することができます。ビルド・ファイルに比べると使い勝手が良くありませんが、 make コマンドの持つ強力な機能と利便性を利用したい場合には一考の価値があります。新しいプログラムを既存の makefile に組み込むには、既存のサンプル・プログラムで似ているものをテンプレートとして利用し、構文に従ってプログラム項目をコーディングします。構文はストアード・プロシージャーと UDF の場合と同様に、 DB2 API、DB2 CLI、組み込み SQL の各プログラムで異なる点に注意してください。

ここでは、提供されている makefile の使用例を示します。使用する makefile は AIX の samples/cli ディレクトリーにあるもので、これは AIX で提供されているすべての DB2 CLI サンプルを IBM C コンパイラーで構築します。この例ではストアード・プロシージャー spserver を構築し、クライアント・アプリケーションが呼び出すことのできる共用ライブラリー spclient にする方法を示しています。

makefile を使用する前に、このファイル中の以下の変数を編集しなければならない場合があります。

DB 使用しているデータベース。デフォルトでは sample に設定されています。

UID 使用しているユーザー ID。デフォルトでは値が設定されていません。

PWD UID ユーザー ID のパスワード。デフォルトでは値が設定されていません。

AIX の場合、DB2 CLI の makefile には変数 DB2PATH、CC、COPY、ERASE、CFLAGS、および LIBS が以下のように定義されています。

```
# Set DB2PATH to where DB2 will be accessed.
# The default is the instance path.
DB2PATH=$(HOME)/sqllib
CC = cc
COPY = cp
ERASE = rm -f
# The required compiler flags
CFLAGS= -I$(DB2PATH)/include
# The required libraries
LIBS= -L$(DB2PATH)/lib -Wl,-rpath,$(DB2PATH)/lib -ldb2
```

makefile はこれらの変数をストアード・プロシージャー spserver をコンパイルするときに使用し、共用ライブラリーを function サブディレクトリーにコピーします。

```
spserver : utilcli.o
        $(CC) -o spserver spserver.c utilcli.o $(CFLAGS) $(LIBS) \
        -H512 -T512 -bE:spserver.exp -e outlanguage
        $(ERASE) $(DB2PATH)/function/spserver
        $(COPY) spserver $(DB2PATH)/function/spserver
```

組み込み SQL プログラムの場合、 makefile は embprep ファイルを呼び出し、これらの組み込み SQL プログラムが使用するプリコンパイルとバインドのステップがその中に入っています。これを組み込み SQL プログラムごとに呼び出される別個のファイルにすることにより、 makefile そのものの本文中でこれらのステップが繰り返し出現すことを避けられます。 このファイルには、サーバー上のデータベースへ接続するためのユーザー ID、パスワード、データベース名を指定しなければなりません。 makefile は呼び出しの際に、これらの値を embprep に渡します。

データベース変数 DB はデフォルトで sample データベースにハードコーディングされており、別のデータベースを使用する場合、ユーザーはこれを変更することが可能です。 ユーザー ID 変数とパスワード変数、UID および PWD には、デフォルトでは値が設定されていません。これらの任意指定のパラメーターは、ユーザーがすでにサーバー・データベースと同じインスタンスで作業している場合には使用する必要がありません。それ以外の場合、たとえばユーザーがクライアント・マシンからサーバーにリモート接続している場合などは、makefile を変更して UID 変数と PWD 変数に適切な値を指定すると、それらの値が embprep プリコンパイルおよびバインドのファイルに自動的に渡されるようになります。 以下に示してあるのは embprep ファイルの例で、 Windows NT 上の Micro Focus COBOL makefile がこれを呼び出して、組み込み SQL アプリケーション updat を構築します。

```
updat.cbl : updat.sqb
        embprep updat $(DB) $(UID) $(PWD)
updat.obj : updat.cbl
        $(CC) updat.cbl;
updat : updat.obj checkerr.obj
        $(LINK) updat.obj checkerr.obj $(LIBS)
```

エラー検査ユーティリティー

DB2 AD クライアントは、いくつかのユーティリティー・ファイルを提供します。これらのファイルには、エラー検査とエラー情報の印刷出力を行う関数があります。例外は、CLI ユーティリティー・ファイル utilapi.c で、 DB2 API を呼び出して、データベースを除去します。ユーティリティー・ファイル

は、サンプル・ディレクトリーの中に、言語ごとに別々のバージョンが用意されています。このユーティリティーはアプリケーション・プログラムで使用するときに有用なエラー情報を提供し、 DB2 プログラムのデバッグの労力を大幅に軽減します。 エラー検査ユーティリティーのほとんどは、プログラム実行中に検出した問題に直接関連した SQLSTATE および SQLCA 情報を取得するのに、 DB2 API GET SQLSTATE MESSAGE および GETERROR MESSAGE を使います。 DB2 CLI ユーティリティーである utilcli は、これらの DB2 API を使用する代わりにそれらと同じ働きをする DB2 CLI ステートメントを使用します。どのエラー検査ユーティリティーを使用した場合でも詳細なエラー・メッセージが印刷出力されるため、開発者は短時間で問題を把握することができます。

ストアード・プロシージャーやユーザー定義関数など、 DB2 プログラムによっては DB2 関連の問題でもこれらのユーティリティーを使用する必要はありません。これは、例外が発生すると SQLException オブジェクトが破棄されるので不要です。

以下に示すのは、DB2 がサポートしているコンパイラーが使用する、プログラム言語別のエラー検査ユーティリティー・ファイルです。

checkerr.cbl
COBOL プログラム用

utilcli.c
CLI プログラム用

utilapi.c
C 組み込み SQL を含まないプログラム用

utilemb.sqc
C 組み込み SQL プログラム用

utilapi.C
C++ 組み込み SQL を含まないプログラム用

utilemb.sqC
C++ 組み込み SQL プログラム用

ユーティリティー関数を使用するには、まず最初にユーティリティー・ファイルをコンパイルした後、ターゲット・プログラムの実行可能ファイルの作成中にそのオブジェクト・ファイルをリンクしなければなりません。 samples ディレクトリー中の makefile とビルド・ファイルは両方とも、エラー検査ユーティリティーを必要とするプログラムで使用することによってこの処理を行います。

以下の例は、エラー検査ユーティリティーを DB2 プログラム中でどのように使用するかを示しています。 utilemb.h ヘッダー・ファイルは、関数 SqlInfoPrint() および TransRollback() の代わりに使用される EMB_SQL_CHECK マクロを定義します。

```
#define EMB_SQL_CHECK( MSG_STR )          \
   if( SqlInfoPrint( MSG_STR, &sqlca, __LINE__, __FILE__)  != 0 ) TransRollback( );
```

SqlInfoPrint() は SQLCODE フラグを検査します。この関数は、このフラグが示している特定のエラーに関連した、入手可能なすべての情報を印刷します。 また、この関数は、ソース・コード内のどこでエラーが発生したかを示します。 TransRollback() により、エラーが発生した場所にユーティリティー・ファイルがトランザクションを安全にロールバックできるようになります。データベースに接続し、ロールバックを実行するための組み込み SQL ステートメントが必要です。以下に、 C++ プログラム cursor が、マクロを使用し、SqlInfoPrint() 関数の MSG_STR パラメーターに値 "DECLARE CURSOR" を提供することによって、ユーティリティー関数を呼び出す方法の例を示します。

```
Cursor::Fetch () {
   EXEC SQL DECLARE c1 CURSOR FOR
            SELECT name, dept FROM staff WHERE job='Mgr'
            FOR UPDATE OF job;
   EMB_SQL_CHECK("DECLARE CURSOR") ;
```

EMB_SQL_CHECK マクロは、 DECLARE ステートメントが失敗すると、トランザクションが安全にロールバックし、該当するエラー・メッセージが確実に印刷されるようにします。

開発者の方々には DB2 プログラムの作成時に、これらのエラー検査ユーティリティーを使って構築することをお勧めします。

Java アプレットおよびアプリケーション

Java アプレットおよびアプリケーションの構築はどのプラットフォームでも同じ手順で行うため、これに関する情報は 75ページの『第4章 Java アプレットおよびアプリケーションの構築』という 1 つの章にまとめられています。プラットフォームごとに必要とされる具体的なセットアップ情報もありますが、それはこの章の別々の節で取り上げています。このセットアップ情報は、 39ページの『第2章 セットアップ』で取り上げている DB2 セットアップ情報への追加となっています。

Java に関する章では、JDBC ドライバーを利用する JDBC プログラム、および JDBC ドライバーとともに Java サポート用の組み込み SQL を利用する SQLJ プログラムを構築する方法を説明します。また、JDBC および SQLJ に

よるアプレット、アプリケーション、そしてストアード・プロシージャーを構築する方法を説明します。さらに、Java のユーザー定義関数を構築する方法についても説明しますが、これには JDBC または SQLJ ステートメントを含めることはできません。

samples ディレクトリーには、 SQLJ プログラムで使用する Java の makefile とビルド・ファイルが入っています。 JDBC プログラムはコマンド行で構築するのが簡単なため、ビルド・ファイルは特に提供されていません。

DB2 API アプリケーション

DB2 AD クライアントにはそれぞれ、 DB2 API を呼び出すサンプル・プログラムが付属しています。本書の後の部分にある「アプリケーションの構築」というタイトルの各章では、該当するプラットフォーム用の DB2 AD クライアントに付属している DB2 アプリケーションのビルド・ファイルを使って、サポートされるコンパイラーのサンプル・プログラムを構築する方法を説明しています。 また、提供される makefile も使用することができます。 makefile とビルド・ファイルのどちらにも、利用可能なコンパイラー・オプションが示されています。 このオプションに関しては、サポートされているコンパイラーのプラットフォームごとに、該当する章で定義されています。ご使用の環境に合わせてオプションを修正することが必要な場合もあります。

以下のサンプル・プログラムは、サポートされるプログラム言語を使って DB2 API アプリケーションの構築と実行を行うステップの具体例を示すために、本書で使われているものです。使用するステップは、環境によって異なる可能性があります。

client これは、SET CLIENT および QUERY CLIENT という API の使い方を示します。

DB2 API のすべてのサンプル・プログラムに関する詳細な説明が必要な場合は、 15ページの『サンプル・プログラム表』を参照してください。

これらの (サポートされている) サンプル・プログラムのソース・ファイルは、 sqllib/samples (UNIX)、および %DB2PATH%¥samples (OS/2 と Windows 32 ビット・オペレーティング・システム) の該当するプログラム言語用のサブディレクトリーにあります。

サンプル・プログラムを作成した後、アプリケーションを構築するためのテンプレートとしてそれらを使用できます。 DB2 API プログラムは、makefile や提供されたビルド・ファイルのいずれかを使って構築できます。

注: API アプリケーションを作成する場合、明示的に設定されない構造要素が 0 に初期設定されるように、すべての API 入力構造を memset で 0 に設定するとよいでしょう。これは、下位レベル・バージョンの API でコーディングされているアプリケーションを再コンパイルする場合に重要です。再コンパイルする場合、構造の新しい定義が使用されますが、すべての要素が初期設定されるわけではありません。 memset への呼び出しを行うと、すべての要素が確実に初期設定されます。以下は、入力データ構造 pLoadInStruct が memset を 0 に設定することを示す例です。

```
memset( pLoadInStruct, 0, sizeof(pLoadInStruct) );
```

DB2 コール・レベル・インターフェース (CLI) アプリケーション

DB2 AD クライアントには、DB2 コール・レベル・インターフェース (DB2 CLI) 関数呼び出しを使用するサンプル・プログラムが付属しています。 サンプルを研究して、アプリケーションでこれらの関数を使用して DB2 データベースにアクセスする方法を学ぶことができます。

ODBC に準拠する DB2 CLI アプリケーションは、 ODBC SDK (DB2 には含まれない) を使用するアプリケーションを再コンパイルする場合、およびアプリケーション・プラットフォームで ODBC ドライバー・マネージャーが使用できるなら、 ODBC 下で作動するように移植することができます。

サンプル・プログラム、ビルド・ファイル、および makefile は、 UNIX の場合にはディレクトリー sqllib/samples/cli、 OS/2 や Windows 32 ビット・オペレーティング・システムの場合には %DB2PATH%¥samples¥cli の中にそれぞれあります。ビルド・ファイルおよび makefile にあるコンパイラー・オプションは、ご使用の環境に合わせて修正する必要があるかもしれません。

以下に示すサンプル・プログラムは、 DB2 CLI アプリケーションの構築と実行のためのステップを具体的に示すために本書で使われているものです (実際のステップは、ご使用の環境によって異なります)。

tbinfo 表レベルの情報の入手および設定方法を示しています。

dbusemx
組み込み SQL とともに単一データベースを使用する方法を示しています。

dbmconn
複数のデータベースに接続したり、複数のデータベースから切断したりする方法を示しています。

spclient
これは、クライアント / サーバー例のクライアント・プログラムです。サーバー・プログラムは spserver です。

spserver
これは、クライアント / サーバー例のサーバー・プログラムです。クライアント・プログラムは spclient です。

udfcli これは、ユーザー定義関数プログラムが作成した関数、 udfsrv を使用します。

DB2 CLI のすべてのサンプル・プログラムに関するさらに詳細な説明が必要な場合は、27ページの表7 を参照してください。 *コール・レベル・インターフェースの手引きおよび解説書* では、DB2 CLI を使用しているサンプルがどのように動作するかを説明しています。

組み込み SQL アプリケーション

注: Java Embedded SQL (SQLJ) についてここでは取り上げていませんが、 75ページの『第4章 Java アプレットおよびアプリケーションの構築』で詳しく説明しています。

DB2 AD クライアントには、 SQL ステートメントを組み込むサンプル・プログラムが付属しています。本書の後の部分にある『アプリケーションの構築』というタイトルの各章では、該当するプラットフォーム用の DB2 AD クライアントに付属しているビルド・ファイルを使って、サポートされるコンパイラーのサンプル・プログラムを構築する方法を説明しています。また、提供される makefile も使用することができます。 makefile とビルド・ファイルのどちらにも、利用可能なコンパイラー・オプションが示されています。 このオプションに関しては、サポートされているコンパイラーのプラットフォームごとに、該当する章で定義されています。 ご使用の環境に合わせてオプションを修正することが必要な場合もあります。

ビルド・ファイルを実行して、組み込み SQL を含むサンプル・プログラムを作成するときには、ビルド・ファイルは次のステップを実行します。
- データベースに接続します。
- ソース・ファイルをプリコンパイルします。
- バインド・ファイルをデータベースにバインドします。
- データベースから切断します。
- ソース・ファイルをコンパイルおよびリンクします。

以下のサンプル・プログラムは、サポートされるプログラム言語を使って組み込み SQL アプリケーションの構築と実行を行うステップの具体例を示すために本書で使われているものです。 使用するステップは、環境によって異なる可能性があります。

updat 静的 SQL を使用してデータベースを更新します。

以下のサンプルは、 C および C++ を使って、ストアード・プロシージャーおよびユーザー定義関数 (UDF) 用の組み込み SQL クライアント・アプリケーションを示すために使用されます。

spclient
これは、ストアード・プロシージャーの呼び出しを示すクライアント・プログラムです。サーバー・プログラムは spserver です。

spserver
これは、ストアード・プロシージャーを示すサーバー・プログラムです。クライアント・プログラムは spclient です。

udfcli これは、ユーザー定義関数ライブラリー udfsrv 内の ScalarUDF 関数を使用します。

以下のサンプルは、 COBOL を使って、ストアード・プロシージャーおよびユーザー定義関数 (UDF) を示すために使用されます。

outcli これは、ストアード・プロシージャーの呼び出しを示すクライアント・プログラムです。サーバー・プログラムは outsrv です。

outsrv これは、ストアード・プロシージャーを示すサーバー・プログラムです。クライアント・プログラムは outcli です。

calludf
これは、ユーザー定義関数ライブラリー udf 内の関数を呼び出します。

これらのサンプル・プログラムの詳細は、 15ページの『サンプル・プログラム表』を参照してください。

これらの (サポートされている) サンプル・プログラムのソース・ファイルは、 UNIX の場合は sqllib/samples、 OS/2 と Windows 32 ビット・オペレーティング・システムの場合は %DB2PATH%¥samples の該当するプログラム言語用のサブディレクトリーにあります。

サンプル・プログラムを作成した後、アプリケーションを構築するためのテンプレートとしてそれらを使用できます。独自の SQL ステートメントを使って

サンプル・プログラムを修正することができます。プログラムは、提供されている makefile またはビルド・ファイルのいずれかを使って構築できます。

14ページの『サンプル・プログラム』に、すべてのサンプル・プログラムをリストしています。 *アプリケーション開発の手引き* では、組み込み SQL が含まれているサンプルがどのように動作するかを説明しています。

ストアード・プロシージャー

ストアード・プロシージャーは、サーバー上で構築され、格納されます。ストアード・プロシージャーは、クライアント・アプリケーションによってリモートからアクセスできます。次に、ストアード・プロシージャー関数は、サーバー・データベース上でローカルに処理を実行し、クライアントに結果を送り返します。これは、ネットワーク通信量を削減し、パフォーマンス全体を改善します。

ストアード・プロシージャーは、分離形式でも非分離形式でも実行できます。非分離ストアード・プロシージャーは、データベース・マネージャーと同じアドレス空間で実行するので、データベース・マネージャーとは分離したアドレス空間で実行する分離ストアード・プロシージャーと比較すると、パフォーマンスが向上します。非分離ストアード・プロシージャーを使う場合、ユーザーのコードがデータベース制御構造を損傷する可能性があります。したがって、非分離ストアード・プロシージャーは、最高のパフォーマンスを得たいときにのみ実行しなければなりません。 これらのプログラムは、完全にテストしてから、分離されていないものとして稼働してください。詳細については、*アプリケーション開発の手引き* を参照してください。

本書で示されているストアード・プロシージャーは、サーバーのパス sqllib/function に格納されています。 DB2DARI パラメーター・スタイルのストアード・プロシージャーにおいて、呼び出されたプロシージャーと共用ライブラリー名が同じになっている場合、ストアード・プロシージャーが分離されていることをこのロケーションが示します。このタイプのストアード・プロシージャーを分離したくない場合は、それを sqllib/function/unfenced ディレクトリーに移す必要があります。これ以外のタイプの DB2 ストアード・プロシージャーの場合は、呼び出し側プログラムで CREATE FUNCTION ステートメントを使用することによって、そのストアード・プロシージャーが分離されているかどうかを示します。 異なるタイプの DB2 ストアード・プロシージャーの作成および使用の詳細については、 *アプリケーション開発の手引き* の『ストアード・プロシージャー』という章を参照してください。

以下に示すサンプル・プログラムは、 SQL プロシージャーを使ってサーバー上でストアード・プロシージャー・ライブラリーの構築と格納を行うステップの具体例を示すために本書で使われているものです。

spserver.db2
これは、サーバー上で共用ライブラリーを作成するために使用される SQL プロシージャーを含む CLP スクリプトです。これは、CLP call コマンド、または C、C++、および CLI ディレクトリー内の spclient アプリケーションによって呼び出すことができます。

以下に示すサンプル・プログラムは、 C および C++ を使ってサーバー上でストアード・プロシージャー・ライブラリーの構築と格納を行うステップの具体例を示すために本書で使われているものです。

spserver
これは、クライアント / サーバー例のサーバー・プログラムです。クライアント・プログラムは spclient です。

以下に示すサンプル・プログラムは、 Java を使ってサーバー上でストアード・プロシージャー・ライブラリーの構築と格納を行うステップの具体例を示すために本書で使われているものです。

Spserver
これは、クライアント / サーバー例のサーバー・プログラムです。クライアント・プログラムは Spclient です。

以下に示すサンプル・プログラムは、 COBOL を使ってサーバー上でストアード・プロシージャー・ライブラリーの構築と格納を行う具体例を示すために本書で使われているものです。

outsrv これは、クライアント / サーバー例のサーバー・プログラムです。クライアント・プログラムは outcli です。

ユーザー定義関数 (UDF)

ユーザー定義関数を使用すれば、自分の必要に応じた SQL の拡張機能を独自に作成することができます。ストアード・プロシージャーと同様、ユーザー定義関数はクライアント・アプリケーションがアクセスできるようにサーバーに格納されます。 UDF には組み込み SQL ステートメントは含まれていません。

以下に示すサンプル・プログラムは、サーバー上で UDF ライブラリーの構築と格納を行うステップの具体例を示すために本書で使われているものです。

udfsrv ユーザー定義関数 (UDF) のライブラリーを作成します。これは、C および C++ でのみ使用可能です。クライアント・アプリケーション udfcli は、これらの関数を呼び出します。

udf ユーザー定義関数 (UDF) のライブラリーを作成します。これは、COBOL でのみ使用可能です。クライアント・アプリケーション calludf は、これらの関数を呼び出します。

UDFsrv ユーザー定義関数 (UDF) のライブラリーを作成します。これは、Java でのみ使用可能です。 UDFcli および UDFclie は、それぞれ JDBC および SQLJ クライアント・アプリケーションで、これらの関数を呼び出します。

マルチスレッド・アプリケーション

DB2 は、サポートされる UNIX プラットフォーム上の C および C++ マルチスレッド・アプリケーションをサポートしています。ユーザーはこれらのアプリケーションを使用することで、複数のプロセスを同時に動作させたり、非同期イベントを処理したり、ポーリング方式に依存せずにイベント・ドリブンのアプリケーションを作成したりすることができます。 以下に示すサンプル・プログラムは、 DB2 マルチスレッド・アプリケーションを構築する具体的な方法を示すのに使われています。

thdsrver
スレッドの作成と管理の方法を示します。

UDF およびストアード・プロシージャーに関する C++ 考慮事項

関数名は C++ では「多重定義」することが可能です。同じ名前を持つ 2 つの関数は、引き数が違っていれば以下のような形で共存させることができます。

```
int func( int i )
```

および

```
int func( char c )
```

C++ コンパイラーはデフォルトの場合、関数名を「型装飾」する、または「マングル」します。これはつまり、関数名の後に引き数の型名を追加してそれらを解決する、ということです。前述の例では、func__Fi および func__Fc となります。マングルした名前はプラットフォームによって異なるため、マングルした名前を明示的にコード中で使用しているとそのコードの移植性は失われます。

OS/2 および Windows 32 ビット・オペレーティング・システムでは、型修飾した関数名は .obj (オブジェクト) ファイルから判別できます。

OS/2 および Windows 上の VisualAge C++ コンパイラーでは、以下のように、cppfilt コマンドを使って、 .obj (オブジェクト) ファイルから型修飾した関数名を判別することができます。

```
cppfilt -b /p myprog.obj
```

ここで、myprog.obj はプログラムのオブジェクト・ファイルです。

Windows 上の Microsoft Visual C++ コンパイラーでは、以下のように、dumpbin コマンドを使って、 .obj (オブジェクト) ファイルから型修飾した関数名を判別することができます。

```
dumpbin /symbols myprog.obj
```

ここで、myprog.obj はプログラムのオブジェクト・ファイルです。

UNIX プラットフォームでは、型修飾した関数名は、nm コマンドを使って、.o (オブジェクト) ファイルまたは共用ライブラリーから判別することができます。このコマンドを実行するとかなりの出力量になるため、以下のような方法でその出力を grep にパイプ処理して、目的の行を探し出すことをお勧めします。

```
nm myprog.o | grep myfunc
```

ここで myprog.o はプログラムのオブジェクト・ファイル、また myfunc はプログラムのソース・ファイル中の関数です。

これらのコマンドで生成した出力には、マングルした関数名を指定した行が含まれています。たとえば、UNIX では、この行は以下のようになります。

```
myfunc__FPlT1PsT3PcN35|     3792|unamex|          | ...
```

いったん上記のコマンドの 1 つからマングルした関数名を入手すると、該当するコマンドでその関数名を使用することができます。上記の UNIX 例から入手したマングルした関数名の使用法を下記に示します。 OS/2 または Windows 上で入手したマングルした関数名は、同じ方法で使用されます。

UDF を CREATE FUNCTION、EXTERNAL NAME 文節を使って登録するときには、以下に示すように、マングルした関数名を指定する必要があります。

```
CREATE FUNCTION myfunco(...) RETURNS...
       ...
       EXTERNAL NAME '/whatever/path/myprog!myfunc__FPlT1PsT3PcN35'
       ...
```

同様に、ストアード・プロシージャーの呼び出し時には、 CALL 関数はマングルした関数名も指定する必要があります。

```
CALL 'myprog!myfunc__FPlT1PsT3PcN35' ( ... )
```

使用するストアード・プロシージャーまたは UDF ライブラリーに、多重定義した C++ 関数名が含まれていない場合は、extern "C" を使用するオプションがあり、これを使って、関数名を型修飾しないようコンパイラーに強制します。 (UDF に指定した SQL 関数名は、常に多重定義することができます。というのは、DB2 は、名前およびパラメーターに基づいて、呼び出されるライブラリー関数を解決するからです。)

```
#include <string.h>
#include <stdlib.h>
#include "sqludf.h"

/*------------------------------------------------------------------------*/
/* function fold: output = input string is folded at point indicated      */
/*                         by the second argument.                        */
/*         inputs: CLOB,                  input string                    */
/*                 LONG                   position to fold on             */
/*         output: CLOB                   folded string                   */
/*------------------------------------------------------------------------*/
extern "C" void fold(
      SQLUDF_CLOB       *in1,                    /* input CLOB to fold      */
   ...
   ...
}
/* end of UDF: fold */

/*------------------------------------------------------------------------*/
/* function find_vowel:                                                   */
/*             returns the position of the first vowel.                   */
/*             returns error if no vowel.                                 */
/*             defined as NOT NULL CALL                                   */
/*         inputs: VARCHAR(500)                                           */
/*         output: INTEGER                                                */
/*------------------------------------------------------------------------*/
extern "C" void findvwl(
      SQLUDF_VARCHAR    *in,                     /* input smallint          */
   ...
   ...
}
/* end of UDF: findvwl */
```

この例では、UDF fold および findvwl は、コンパイラーによって型修飾されません。そのままの名前を使用して、 CREATE FUNCTION ステートメントで登録すべきです。同様に、C++ ストアード・プロシージャーを extern "C" を使ってコーディングする場合、 CALL ステートメントでは修飾しない関数名を使用します。

第4章 Java アプレットおよびアプリケーションの構築

DB2 データベースにアクセスする Java プログラムは、ご使用のプラットフォームに対応した Java 開発者キット (JDK) を使って開発できます。 JDK には、Java 用の動的 SQL API である Java データベース・コネクティビティー (JDBC) が含まれています。

DB2 JDBC サポートは、DB2 クライアントおよびサーバー上の Java Enablement オプションの一部として提供されます。このサポートによって、JDBC アプリケーションとアプレットを構築し、実行できます。これらには動的 SQL だけが含まれ、 Java 呼び出しインターフェースを使って SQL ステートメントを DB2 に渡します。

DB2 Java Embedded SQL (SQLJ) サポートは、 DB2 AD クライアントの一部として提供されます。 DB2 JDBC サポートとともに DB2 SQLJ サポートを利用することで、 SQLJ アプレットおよびアプリケーションの構築と実行が可能になります。これらには、静的 SQL が含まれ、 DB2 データベースにバインドされた組み込み SQL ステートメントを使用します。

DB2 AD クライアントによって提供される SQLJ サポートには、次のものが含まれます。

- DB2 SQLJ 変換プログラム sqlj。これは、SQLJ プログラム中の組み込み SQL ステートメントを Java ソース・ステートメントで置き換え、 SQLJ プログラム中に検出される SQL 操作に関する情報を含む、順番に並べられたプロファイルを生成します。
- DB2 SQLJ プロファイル・カスタマイザー db2profc。これは逐次化プロファイルに保管された SQL ステートメントをプリコンパイルし、それらを実行時関数呼び出しにカスタマイズし、そして DB2 データベース中にパッケージを生成します。 db2profc コマンドの詳細は、*コマンド解説書* を参照してください。
- DB2 SQLJ プロファイル・プリンター db2profp。これはカスタマイズしたバージョンの DB2 プロファイルの内容をプレーン・テキスト形式で印刷します。

DB2 Java アプリケーションを実行するには、ネイティブ・スレッド・サポートを提供する Java 仮想マシン (JVM) をインストールして呼び出さなければなりません。ネイティブ・スレッドを使って Java アプリケーションを実行するには、コマンド内の -native オプションを使用することができます。たとえば、Java サンプル・アプリケーション App.class を実行するには、次のコマンドを使うことができます。

```
java -native App
```

いくつかの Java 仮想マシン用のデフォルトのスレッド・サポートとしてネイティブ・スレッドを指定することができます。この章にある情報は、ネイティブ・スレッド・サポートがデフォルトであることを前提としています。システム上でネイティブ・スレッドをデフォルトにする方法については、 JVM 資料を参照してください。

DB2 Java アプレットを実行するには、ネイティブ・スレッド・サポートまたはグリーン・スレッド・サポートのいずれかを提供する Java 仮想マシンを呼び出すことができます。

Java による DB2 プログラミングについて詳しくは、 *アプリケーション開発の手引き* の中の『Java でのプログラミング』という章を参照してください。

最新の更新された DB2 Java 情報について知りたい場合は、以下の Web ページを参照してください。

```
http://www.ibm.com/software/data/db2/java
```

環境のセットアップ

AIX

AIX 上で DB2 JDBC サポートを利用して Java アプリケーションを構築するには、開発マシンで次のものをインストールし、構成する必要があります。

1. 以下のいずれか
   - Java 開発者キット (JDK) バージョン 1.1.8 および Java Runtime Environment (JRE) バージョン 1.1.8 (AIX 版) (IBM 社提供)。これらは、DB2 とともにインストールされます。
     (http://www.ibm.com/software/data/db2/java を参照してください。)
   - Java 開発者キット (JDK) バージョン 1.2.2 および Java Runtime Environment (JRE) バージョン 1.2.2 (AIX 版) (IBM 社提供)。
     (http://www.ibm.com/software/data/db2/java を参照してください。)
2. DB2 Java Enablement。これは AIX クライアントおよびサーバーに対応した DB2 ユニバーサル・データベース・バージョン 7.1 で提供しています。

JDBC 1.22 ドライバーは、AIX を含むすべてのオペレーティング・システム上のデフォルトのドライバーです。 JDBC 2.0 の新しい機能を使用するには、ご使用のプラットフォーム用の JDBC 2.0 ドライバーと JDK 1.2 サポートの両方をインストールしなければなりません。 AIX 用の JDBC 2.0 をインストールするには、 sqllib/java12 ディレクトリーから usejdbc2 コマンドを入力してください。 このコマンドは、以下のタスクを実行します。

- 1.22 ドライバー・ファイル用の sqllib/java11 ディレクトリーを作成します。
- JDBC 1.22 ドライバー・ファイルを sqllib/java11 ディレクトリーにバックアップします。
- sqllib/java12 ディレクトリーから該当するディレクトリーに JDBC 2.0 ドライバー・ファイルをコピーします。

JDBC 1.22 ドライバーに切り替えるには、 sqllib/java12 ディレクトリーから usejdbc1 コマンドを実行してください。

DB2 Java ストアード・プロシージャーまたは UDF を実行するには、そのマシンにインストールされている JDK のパスを含むように、サーバー上の DB2 データベース・マネージャー構成を更新する必要があります。 サーバーのコマンド行で、次のように入力することによってこれを行えます。

```
db2 update dbm cfg using JDK11_PATH /home/db2inst/jdk11
```

/home/db2inst/jdk11 は、JDK がインストールされているパスです。

次のコマンドをサーバーで入力して、 DB2 データベース・マネージャー構成をチェックし、 JDK11_PATH フィールドの値が正しいことを確認できます。

```
db2 get dbm cfg
```

出力をファイルにリダイレクトすれば、一層容易に表示できます。 JDK11_PATH フィールドは、出力の先頭近くに表示されます。これらのコマンドの詳細は、*コマンド解説書* を参照してください。

DB2 JDBC サポートを利用して AIX 上で JDBC プログラムや SQLJ プログラムを実行するために、 AIX の Java 環境を更新するコマンドが、データベース・マネージャー・ファイル db2profile と db2cshrc に含まれています。 DB2 インスタンスの作成時には、 .profile や .cshrc の内容を変更して以下のものを CLASSPATH に含めます。

- "." (現行ディレクトリー)
- ファイル sqllib/java/db2java.zip

SQLJ プログラムを作成するには、次のファイルを組み込むように CLASSPATH を更新します。

```
sqllib/java/sqlj.zip
```

SQLJ プログラムを実行するには、次のファイルを組み込むように CLASSPATH を更新します。

```
sqllib/java/runtime.zip
```

HP-UX

HP-UX 上で DB2 JDBC サポートを利用して Java アプリケーションを構築するには、開発マシンで次のものをインストールし、構成する必要があります。

1. Hewlett-Packard 社提供の HP-UX Developer's Kit for Java リリース 1.1.8 以降 (http://www.ibm.com/software/data/db2/java を参照)。
2. DB2 Java Enablement。これは HP-UX クライアントおよびサーバーに対応した DB2 ユニバーサル・データベース バージョン 7.1 で提供しています。

DB2 Java ストアード・プロシージャーまたは UDF を実行するには、そのマシンにインストールされている JDK のパスを含むように、サーバー上の DB2 データベース・マネージャー構成を更新する必要もあります。 サーバーのコマンド行で、次のように入力することによってこれを行えます。

```
db2 update dbm cfg using JDK11_PATH /home/db2inst/jdk11
```

/home/db2inst/jdk11 は、JDK がインストールされているパスです。

次のコマンドをサーバーで入力して、 DB2 データベース・マネージャー構成をチェックし、 JDK11_PATH フィールドの値が正しいことを確認できます。

```
db2 get dbm cfg
```

出力をファイルにリダイレクトすれば、一層容易に表示できます。 JDK11_PATH フィールドは、出力の先頭近くに表示されます。これらのコマンドの詳細は、*コマンド解説書* を参照してください。

DB2 JDBC サポートを利用して HP-UX 上で JDBC プログラムや SQLJ プログラムを実行するために、 HP-UX の Java 環境を更新するコマンドが、データベース・マネージャー・ファイル db2profile と db2cshrc に含まれています。 DB2 インスタンスの作成時には、 .profile または .cshrc (あるいはその両方) の内容を以下のように変更します。

1. THREADS_FLAG を ″native″ に設定します。
2. CLASSPATH に次のものを含めます。
   - ″.″ (現行ディレクトリー)
   - ファイル sqllib/java/db2java.zip

SQLJ プログラムを作成するには、次のファイルを組み込むように CLASSPATH を更新します。

```
sqllib/java/sqlj.zip
```

SQLJ プログラムを実行するには、次のファイルを組み込むように CLASSPATH を更新します。

```
sqllib/java/runtime.zip
```

Linux

Linux 上で DB2 JDBC サポートを利用して Java アプリケーションを構築するには、開発マシンで次のものをインストールし、構成する必要があります。

1. IBM 開発者キットおよび Runtime Environment (Linux 版) バージョン 1.1.8。 (http://www.ibm.com/software/data/db2/java を参照)。
2. DB2 Java Enablement。これは Linux クライアントおよびサーバーに対応した DB2 ユニバーサル・データベース バージョン 7.1 で提供しています。

DB2 Java ストアード・プロシージャーまたは UDF を実行するには、そのマシンにインストールされている JDK のパスを含むように、サーバー上の DB2

データベース・マネージャー構成を更新する必要もあります。 サーバーのコマンド行で、次のように入力することによってこれを行えます。

```
db2 update dbm cfg using JDK11_PATH /usr/local/jdk118
```

/usr/local/jdk118 は、 JDK がインストールされているパスです。

次のコマンドをサーバーで入力して、 DB2 データベース・マネージャー構成をチェックし、 JDK11_PATH フィールドの値が正しいことを確認できます。

```
db2 get dbm cfg
```

出力をファイルにリダイレクトすれば、一層容易に表示できます。 JDK11_PATH フィールドは、出力の先頭近くに表示されます。これらのコマンドの詳細は、*コマンド解説書* を参照してください。

注: Linux の場合、その Java 仮想マシン・インプリメンテーションが、″setuid″ 環境で実行するプログラムをうまく動作しないことがあります。 Java インタープリター libjava.so が入っている共用ライブラリーが、ロードに失敗する可能性があります。これを回避する方法として、以下に示すコマンドと同様のコマンド (これは、ご使用のマシンで Java がインストールされている場所によって異なります) を使って、 /usr/lib にある JVM 共用ライブラリーへの記号リンクを作成できます。

```
ln -s /usr/local/jdk118/lib/linux/native_threads/libjava.so /usr/lib
```

これに関する詳細、および問題を回避するその他の方法については、以下に示すサイトを参照してください。

```
http://www.ibm.com/software/data/db2/java/faq.html
```

DB2 JDBC サポートを利用して Linux 上で JDBC プログラムや SQLJ プログラムを実行するために、 Linux の Java 環境を更新するコマンドが、データベース・マネージャー・ファイル db2profile と db2cshrc に含まれています。 DB2 インスタンスの作成時には、 .bashrc、.profile、.cshrc のいずれか (あるいはその全部) の内容を以下のように変更します。

1. THREADS_FLAG を ″native″ に設定します。
2. CLASSPATH に次のものを含めます。
   - ″.″ (現行ディレクトリー)
   - ファイル sqllib/java/db2java.zip

SQLJ プログラムを作成するには、次のファイルを組み込むように CLASSPATH を更新します。

```
sqllib/java/sqlj.zip
```

SQLJ プログラムを実行するには、次のファイルを組み込むように CLASSPATH を更新します。

```
sqllib/java/runtime.zip
```

OS/2

OS/2 上で DB2 JDBC サポートを利用して Java アプリケーションを構築するには、開発マシン上に以下のものをインストールし、構成する必要があります。

1. Java 開発者キット (JDK) バージョン 1.1.8 および Java Runtime Environment (JRE) バージョン 1.1.8 (OS/2 版) (IBM 社提供)。これらは、DB2 に付属しています。 (http://www.ibm.com/software/data/db2/java を参照。)
2. DB2 Java Enablement。これは OS/2 クライアントおよびサーバーに対応した DB2 ユニバーサル・データベース バージョン 7.1 で提供しています。

JDK は、拡張子が 3 文字以上になる (.java など) 長いファイル名が使えるHPFS ドライブにインストールする必要があります。さらに、Java 用の作業ディレクトリーも HPFS ドライブになければなりません。 OS/2 サーバー上で Java ストアード・プロシージャーまたは UDF を実行することになっている場合は、ストアード・プロシージャーまたは UDF の .class ファイルを %DB2PATH%¥function ディレクトリーに入れることができるように、 DB2 をサーバー上の HPFS ドライブにインストールする必要があります。

DB2 Java ストアード・プロシージャーまたは UDF を実行するには、そのマシンにインストールされている JDK のパスを含むように、サーバー上の DB2 データベース・マネージャー構成を更新する必要があります。サーバーのコマンド行で、次のように入力することによってこれを行えます。

```
db2 update dbm cfg using JDK11_PATH c:¥jdk11
```

c:¥jdk11 は、JDK がインストールされているパスです。

次のコマンドをサーバーで入力して、 DB2 データベース・マネージャー構成をチェックし、 JDK11_PATH フィールドの値が正しいことを確認できます。

```
db2 get dbm cfg
```

出力をファイルにリダイレクトすれば、一層容易に表示できます。 JDK11_PATH フィールドは、出力の先頭近くに表示されます。これらのコマンドの詳細は、*コマンド解説書* を参照してください。

JDBC プログラムや SQLJ プログラムを OS/2 上で DB2 JDBC サポートを利用して実行するために、 CLASSPATH 環境変数は以下のものを含むように DB2 のインストール時に自動的に更新されます。

- "." (現行ディレクトリー)
- ファイル %DB2PATH%¥java¥db2java.zip

SQLJ プログラムを作成するには、次のファイルを組み込むように CLASSPATH を更新します。

```
%DB2PATH%¥java¥sqlj.zip
```

SQLJ プログラムを実行するには、次のファイルを組み込むように CLASSPATH を更新します。

```
%DB2PATH%¥java¥runtime.zip
```

DYNIX/ptx

DYNIX/ptx 上で DB2 JDBC サポートを利用して Java アプリケーションを構築するには、開発マシンで次のものをインストールし、構成する必要があります。

1. ptx/JSE バージョン 3.0。これは Sun Microsystem's JDK 1.2 と等価です (http://www.ibm.com/software/data/db2/java を参照)。
2. DB2 Java Enablement。これは DB2 ユニバーサル・データベース バージョン 7.1 (NUMA-Q 版) で提供しています。

DB2 Java ストアード・プロシージャーまたは UDF を実行するには、そのマシンにインストールされている ptx/JSE のパスを含むように、サーバー上の DB2 データベース・マネージャー構成を更新する必要もあります。サーバーのコマンド行で、次のように入力することによってこれを行えます。

```
db2 update dbm cfg using JDK11_PATH /opt/jse3.0
```

/opt/jse3.0 は、ptx/JSE がインストールされているパスです。

次のコマンドをサーバーで入力して、 DB2 データベース・マネージャー構成をチェックし、 JDK11_PATH フィールドの値が正しいことを確認できます。

```
db2 get dbm cfg
```

出力をファイルにリダイレクトすれば、一層容易に表示できます。 JDK11_PATH フィールドは、出力の先頭近くに表示されます。これらのコマンドの詳細は、*コマンド解説書* を参照してください。

DB2 JDBC サポートを利用して DYNIX/ptx 上で JDBC プログラムや SQLJ プログラムを実行するために、 DYNIX/ptx の Java 環境を更新するコマンドが、データベース・マネージャー・ファイル db2profile と db2cshrc に含まれています。 DB2 インスタンスの作成時には、 .profile や .cshrc の内容を変更して以下のものを CLASSPATH に含めます。

- "." (現行ディレクトリー)
- ファイル sqllib/java/db2java.zip

SQLJ プログラムを作成するには、次のファイルを組み込むように CLASSPATH を更新します。

```
sqllib/java/sqlj.zip
```

SQLJ プログラムを実行するには、次のファイルを組み込むように CLASSPATH を更新します。

```
sqllib/java/runtime.zip
```

Silicon Graphics IRIX

DB2 JDBC をサポートする Silicon Graphics IRIX 上で Java アプリケーションを構築するには、開発マシンで次のものをインストールし、構成する必要があります。

1. Java2 ソフトウェア開発者キット バージョン 1.2.1 (JDK 1.2.1) (Silicon Graphics 社提供) (http://www.ibm.com/software/data/db2/java を参照)。
2. DB2 Java Enablement。これは Silicon Graphics IRIX クライアントに対応した DB2 ユニバーサル・データベース バージョン 7.1 で提供しています。

注: Silicon Graphics IRIX 上の SQLJ アプリケーションは、 o32 オブジェクト・タイプでのみ構築できます。 Java のデフォルト・オブジェクト・タイプを o32 に変更するには、このコマンドを使って環境変数 SGI_ABI を Korn シェルに設定してください。

```
export SGI_ABI=-o32
```

このコマンドを使って C シェルに設定してください。

```
setenv SGI_ABI -o32
```

DB2 for Silicon Graphics IRIX はクライアント専用です。 DB2 アプリケーションとアプレットを実行し、 DB2 組み込み SQL アプリケーションとアプレットを構築するには、クライアント・マシンからサーバー・マシン上の DB2 データベースにアクセスする必要があります。サーバー・マシンは、別のオペ

レーティング・システムを実行する予定です。クライアント / サーバー通信の構成については、 *DB2 ユニバーサル・データベース (UNIX 版) 概説およびインストール* を参照してください。

また、別のオペレーティング・システム上で実行しているリモート・クライアントから、サーバー上のデータベースにアクセスする予定である場合には、DB2 CLI を含むデータベース・ユーティリティーを、データベースにバインドする必要があります。詳細については、50ページの『バインド』を参照してください。

DB2 Java ストアード・プロシージャーまたは UDF を実行するには、そのマシンにインストールされている JDK のパスを含むように、サーバー上の DB2 データベース・マネージャー構成を更新する必要もあります。 サーバーのコマンド行で、次のように入力することによってこれを行えます。

```
db2 update dbm cfg using JDK11_PATH /home/db2inst/jdk11
```

`/home/db2inst/jdk11` は、JDK がインストールされているパスです。

次のコマンドをサーバーで入力して、 DB2 データベース・マネージャー構成をチェックし、 `JDK11_PATH` フィールドの値が正しいことを確認できます。

```
db2 get dbm cfg
```

出力をファイルにリダイレクトすれば、一層容易に表示できます。 `JDK11_PATH` フィールドは、出力の先頭近くに表示されます。これらのコマンドの詳細は、*コマンド解説書* を参照してください。

Silicon Graphics IRIX 上で DB2 JDBC サポートを利用して JDBC プログラムや SQLJ プログラムを実行するために、 Silicon Graphics IRIX の Java 環境を更新するコマンドが、データベース・マネージャー・ファイル `db2profile` と `db2cshrc` に含まれています。 DB2 インスタンスの作成時には、 `.profile` や `.cshrc` の内容を変更して以下のものを CLASSPATH に含めます。

- ″.″ (現行ディレクトリー)
- ファイル `sqllib/java/db2java.zip`

SQLJ プログラムを作成するには、次のファイルを組み込むように CLASSPATH を更新します。

```
sqllib/java/sqlj.zip
```

SQLJ プログラムを実行するには、次のファイルを組み込むように CLASSPATH を更新します。

```
sqllib/java/runtime.zip
```

注: Silicon Graphics IRIX では、接続コンテキストである close() メソッドがトラップの原因になることがあります。これを回避する方法としては、接続コンテキストがガーベッジ・コレクション中に自動的にクローズするようにしておきます。

Solaris

Solaris オペレーティング環境で DB2 JDBC サポートを利用して Java アプリケーションを構築するには、開発マシンで次のものをインストールし、構成する必要があります。

1. Java 開発者キット (JDK) バージョン 1.1.8 または 1.2 (Solaris 版) (Sun Microsystems 社提供) (http://www.ibm.com/software/data/db2/java を参照)。
2. DB2 Java Enablement。これは、Solaris クライアントおよびサーバーに対応した DB2 ユニバーサル・データベース バージョン 7.1 で提供しています。

JDBC 1.22 ドライバーは、 Solaris を含むすべてのオペレーティング・システム上のデフォルトのドライバーです。 JDBC 2.0 の新しい機能を使用するには、ご使用のプラットフォーム用の JDBC 2.0 ドライバーと JDK 1.2 サポートの両方をインストールしなければなりません。 Solaris 用の JDBC 2.0 ドライバーをインストールするには、 sqllib/java12 ディレクトリーからusejdbc2 コマンドを入力してください。このコマンドは、以下のタスクを実行します。

- 1.22 ドライバー・ファイル用の sqllib/java11 ディレクトリーを作成します。
- JDBC 1.22 ドライバー・ファイルを sqllib/java11 ディレクトリーにバックアップします。
- sqllib/java12 ディレクトリーから該当するディレクトリーに JDBC 2.0 ドライバー・ファイルをコピーします。

JDBC 1.22 ドライバーに切り替えるには、 sqllib/java12 ディレクトリーから usejdbc1 コマンドを実行してください。

DB2 Java ストアード・プロシージャーまたは UDF を実行するには、そのマシンにインストールされている JDK のパスを含むように、サーバー上の DB2 データベース・マネージャー構成を更新する必要があります。 サーバーのコマンド行で、次のように入力することによってこれを行えます。

```
db2 update dbm cfg using JDK11_PATH /usr/java
```

/usr/java は、JDK がインストールされているパスです。

次のコマンドをサーバーで入力して、 DB2 データベース・マネージャー構成をチェックし、 JDK11_PATH フィールドの値が正しいことを確認できます。

```
db2 get dbm cfg
```

出力をファイルにリダイレクトすれば、一層容易に表示できます。 JDK11_PATH フィールドは、出力の先頭近くに表示されます。これらのコマンドの詳細は、*コマンド解説書* を参照してください。

注: Solaris オペレーティング環境では、Java 仮想マシン・インプリメンテーションが、 ″setuid″ 環境で実行するプログラムをうまく動作しないことがあります。 Java インタープリター libjava.so が入っている共用ライブラリーが、ロードに失敗する可能性があります。これを回避する方法として、以下に示すコマンドと同様のコマンド (これは、ご使用のマシンで Java がインストールされている場所によって異なります) を使って、 /usr/lib にあるすべての必要な JVM 共用ライブラリーへの記号リンクを作成できます。

```
ln -s /usr/java/lib/*.so /usr/lib
```

これに関する詳細、および問題を回避するその他の方法については、以下に示すサイトを参照してください。

```
http://www.ibm.com/software/data/db2/java/faq.html
```

Solaris オペレーティング環境で DB2 JDBC サポートを利用して JDBC プログラムや SQLJ プログラムを実行するために、 Solaris の Java 環境を更新するコマンドが、データベース・マネージャー・ファイル db2profile と db2cshrc に含まれています。 DB2 インスタンスの作成時には、 .profile や .cshrc の内容を以下のように変更します。

1. THREADS_FLAG を ″native″ に設定します。
2. CLASSPATH に次のものを含めます。
   - ″.″ (現行ディレクトリー)
   - ファイル sqllib/java/db2java.zip

SQLJ プログラムを作成するには、次のファイルを組み込むように CLASSPATH を更新します。

```
sqllib/java/sqlj.zip
```

SQLJ プログラムを実行するには、次のファイルを組み込むように CLASSPATH を更新します。

```
sqllib/java/runtime.zip
```

Windows 32 ビット・オペレーティング・システム

Windows 32 ビット・プラットフォーム上で DB2 JDBC サポートを利用して Java アプリケーションを構築するには、開発マシンで次のものをインストールし、構成する必要があります。

1. 以下のいずれか
   - Java 開発者キット (JDK) 1.1.8 および Java Runtime Environment (JRE) 1.1.8 (Win32 版) (IBM 社提供)。これらは、DB2 とともにインストールされます。 (http://www.ibm.com/software/data/db2/java を参照。)
   - Java 開発者キット (JDK) バージョン 1.2 (Win32 版) (Sun Microsystems 社提供) (http://www.ibm.com/software/data/db2/java を参照)。
   - Microsoft Software Developer's Kit for Java、バージョン 3.1 (http://www.ibm.com/software/data/db2/java を参照)。
2. DB2 Java Enablement。これは Windows 32 ビット・オペレーティング・システムのクライアントおよびサーバーに対応した DB2 ユニバーサル・データベース バージョン 7.1 で提供しています。

JDBC 1.22 ドライバーは、 Windows 32 ビットを含むすべてのオペレーティング・システム上のデフォルトのドライバーです。 JDBC 2.0 の新しい機能を使用するには、ご使用のプラットフォーム用の JDBC 2.0 ドライバーと JDK 1.2 サポートの両方をインストールしなければなりません。 Windows 32 オペレーティング・システム用の JDBC 2.0 ドライバーをインストールするには、sqllib¥java12 ディレクトリーから usejdbc2 コマンドを入力してください。このコマンドは、以下のタスクを実行します。

- 1.22 ドライバー・ファイル用の sqllib¥java11 ディレクトリーを作成します。
- JDBC 1.22 ドライバー・ファイルを sqllib¥java11 ディレクトリーにバックアップします。
- sqllib¥java12 ディレクトリーから該当するディレクトリーに JDBC 2.0 ドライバー・ファイルをコピーします。

JDBC 1.22 ドライバーに切り替えるには、 sqllib¥java12 ディレクトリーから usejdbc1 バッチ・ファイルを実行してください。

DB2 Java ストアード・プロシージャーまたは UDF を実行するには、そのマシンにインストールされている JDK のパスを含むように、サーバー上の DB2 データベース・マネージャー構成を更新する必要があります。 サーバーのコマンド行で、次のように入力することによってこれを行えます。

```
db2 update dbm cfg using JDK11_PATH c:¥jdk11
```

c:¥jdk11 は、JDK がインストールされているパスです。

注: JDK がインストールされているパスに、 1 つまたは複数のスペースが入ったディレクトリー名が含まれている場合、パスを単一引用符で囲んでください。たとえば、次のようにします。

```
db2 update dbm cfg using JDK11_PATH 'c:¥Program Files¥jdk11'
```

次のコマンドをサーバーで入力して、 DB2 データベース・マネージャー構成をチェックし、 JDK11_PATH フィールドの値が正しいことを確認できます。

```
db2 get dbm cfg
```

出力をファイルにリダイレクトすれば、一層容易に表示できます。 JDK11_PATH フィールドは、出力の先頭近くに表示されます。これらのコマンドの詳細は、*コマンド解説書* を参照してください。

サポートされる Windows プラットフォーム上で JDBC プログラムや SQLJ プログラムを DB2 JDBC サポートを利用して実行するために、 CLASSPATH は以下のものを含むように DB2 のインストール時に自動的に更新されます。

- ″.″ (現行ディレクトリー)
- ファイル %DB2PATH%¥java¥db2java.zip

SQLJ プログラムを作成するには、次のファイルを組み込むように CLASSPATH を更新します。

```
%DB2PATH%¥java¥sqlj.zip
```

SQLJ プログラムを実行するには、次のファイルを組み込むように CLASSPATH を更新します。

```
%DB2PATH%¥java¥runtime.zip
```

DB2 SQLJ にどの Java 開発者キットを使うかを指定するため、 DB2 は Windows 32 ビット・オペレーティング・システムに環境変数 DB2JVIEW をインストールします。これはすべての DB2 SQLJ コマンド (db2profc、db2profp、 profdb、profp および sqlj) に適用されます。

デフォルト設定値である ″DB2JVIEW=0″ を使用するか DB2JVIEW を設定しない場合には、 Sun JDK が使用されます。つまり、″profp″ を呼び出すと、″java sqlj.runtime.profile.util.ProfilePrinter″ のように実行されます。また、

″DB2JVIEW=1″ であれば Microsoft SDK for Java が使用されます。つまり、″profp″ を呼び出すと、 ″jview sqlj.runtime.profile.util.ProfilePrinter″ のように実行されます。

Java サンプル・プログラム

DB2 は動的 SQL、および静的 SQL を使用する SQLJ プログラムを専用に使用する JDBC プログラムの構築と実行を例示するために、以下の節で使用されるサンプル・プログラムを提供します。 UNIX プラットフォームでは、Java のサンプルは `sqllib/samples/java` にあります。 OS/2 および Windows 32 ビット・オペレーティング・システムでは、サンプルは `%DB2PATH%¥samples¥java` にあります。サンプル・ディレクトリーには `README`、`makefile`、ビルド・ファイルなどのファイルも入っています。

Java の `makefile` は提供されているすべてのサンプル・プログラムを構築します。これは、互換性のある make 実行可能プログラムを必要としますが、このプログラムは Java 開発者キットには通常用意されていません。詳しくは、`makefile` のテキストの冒頭にあるコメントを参照してください。 Java の `makefile` コマンドの中には、他の言語と異なっているものがあります。たとえば、`make clean` コマンドは `.class` ファイルなど、 java コンパイラーが生成したすべてのファイルを除去します。 `make cleanall` コマンドはこれらのファイルだけでなく、 SQLJ 変換プログラムが生成したファイルもすべて除去します。

JDBC プログラムは比較的簡単にコマンド行で構築できるため、そのためのビルド・ファイルは特に含まれていません。 SQLJ のビルド・ファイルは 2 種類用意されており、 `bldsqlj` は SQLJ アプレットおよびアプリケーションを、また `bldsqljs` は SQLJ ストアード・プロシージャーを構築します。これらのビルド・ファイルの具体的な使用例は 93ページの『SQLJ プログラム』に示されています。

OS/2

OS/2 では、作業ディレクトリーが HPFS ドライブになければなりません。 OS/2 上に用意されている DB2 サンプル・プログラムは FAT ドライブのことを考慮してあるため、ファイル名の拡張子が最大 3 文字までとなっています。この制限に従って、 Java サンプル・ファイルの拡張子は短縮されています。 Java ファイルを作業ディレクトリーにコピーした後、以下に示すコマンドでファイルの名前を変更することができます。

```
move *.jav *.java
move *.htm *.html
move *.sql *.sqlj
```

JDBC プログラム

アプレット

DB2Applt は、 DB2 データベースにアクセスするために、 JDBC アプレット (または ″net″) ドライバーを使用する動的 SQL Java アプレットを例示します。

このアプレットを構築し実行するには、コマンド行で次のようにコマンドを入力します。

1. DB2 マシン (サーバーまたはクライアント) に Web サーバーがインストールされて実行していることを確認してください。
2. DB2Applt.html ファイルの内容を、ファイル内の指示に従って修正します。
3. DB2Applt.html で指定された TCP/IP ポートで JDBC アプレット・サーバーを開始します。たとえば DB2Applt.html では、次のように指定しています。

   ```
   db2jstrt 6789
   ```

4. 以下に示すコマンドで DB2Applt.java をコンパイルし、ファイル DB2Applt.class を生成します。

   ```
   javac DB2Applt.java
   ```

5. Web ブラウザーが、作業ディレクトリーにアクセス可能であることを確認してください。アクセス可能ではない場合は、 DB2Applt.class と DB2Applt.html をアクセス可能なディレクトリーにコピーします。
6. ファイル %DB2PATH%¥java¥db2java.zip (OS/2 または Windows 32 ビット・オペレーティング・システムの場合)、あるいは sqllib/java/db2java.zip (UNIX の場合) を DB2Applt.class や DB2Applt.html と同じディレクトリーにコピーします。
7. クライアント・マシンで Web ブラウザー (JDK 1.1 をサポートしていることが必要) を開始し、 DB2Applt.html を読み込みます。

ステップ (1)、(5)、および (7) の代わりに、次のコマンドをクライアント・マシンの作業ディレクトリーで入力して、 Java 開発者キットに付属のアプレット・ビューアーを使用することができます。

```
appletviewer DB2Applt.html
```

このプログラムは、Java の makefile を使って構築することも可能です。

アプリケーション

DB2Appl は、 DB2 データベースにアクセスするために、 JDBC アプリケーション (または ″app″) ドライバーを使用する動的 SQL Java アプリケーションを例示します。

このアプリケーションを構築し実行するには、コマンド行で次のようにコマンドを入力します。

1. 以下に示すコマンドで DB2Appl.java をコンパイルし、ファイル DB2Appl.class を生成します。

   ```
   javac DB2Appl.java
   ```

2. 次のコマンドで、アプリケーションに対して Java インタープリターを実行します。

   ```
   java DB2Appl
   ```

このプログラムは、Java の makefile を使って構築することも可能です。

ストアード・プロシージャー用のクライアント・アプリケーション

Spclient は、 Java ストアード・プロシージャー・クラス Spserver を JDBC アプリケーション・ドライバーを使って呼び出すクライアント・アプリケーションです。このクライアント・アプリケーションを構築して実行する前に、ストアード・プロシージャー・クラスをサーバー上で構築します。 92ページの『ストアード・プロシージャー』を参照してください。

このクライアント・プログラムを構築して実行するには、コマンド行で次のようにコマンドを入力します。

1. 以下に示すコマンドで Spclient.java をコンパイルし、ファイル Spclient.class を生成します。

   ```
   javac Spclient.java
   ```

2. 次のコマンドで、クライアント・プログラムに対して Java インタープリターを実行します。

   ```
   java Spclient
   ```

このプログラムは、Java の makefile を使って構築することも可能です。

ユーザー定義関数用のクライアント・アプリケーション

UDFcli は、ユーザー定義関数サーバー・プログラム UDFsrv に実装されたユーザー定義関数を、 JDBC アプリケーション・ドライバーを使って呼び出すクライアント・プログラムです。このクライアント・アプリケーションを構築して

実行する前に、ユーザー定義関数プログラム UDFsrv をサーバー上で構築します。 102ページの『ユーザー定義関数 (UDF)』を参照してください。

このクライアント・プログラムを構築して実行するには、コマンド行で次のようにコマンドを入力します。

1. 以下に示すコマンドで UDFcli.java をコンパイルし、ファイル UDFcli.class を生成します。

   ```
   javac UDFcli.java
   ```

2. 次のコマンドで、クライアント・プログラムに対して Java インタープリターを実行します。

   ```
   java UDFcli
   ```

このプログラムは、Java の makefile を使って構築することも可能です。

ストアード・プロシージャー

Spserver は、 JDBC アプリケーション・ドライバーを使用した動的 SQL PARAMETER STYLE JAVA ストアード・プロシージャーの具体的な使用例を示しています。ストアード・プロシージャーはサーバー上でコンパイルされ、格納されます。そして、クライアント・アプリケーションによって呼び出されるとサーバー・データベースにアクセスし、そのクライアント・アプリケーションに情報を戻します。

サーバー上でこのプログラムを構築して実行するには、コマンド行で次のようにコマンドを入力します。

1. 以下に示すコマンドで Spserver.java をコンパイルし、ファイル Spserver.class を生成します。

   ```
   javac Spserver.java
   ```

2. Spserver.class を、 OS/2 および Windows 32 ビット・オペレーティング・システムの場合には %DB2PATH%¥function ディレクトリーへ、また UNIX の場合には sqllib/function ディレクトリーへコピーします。
3. 次に、サーバー上で Spcreate.db2 スクリプトを実行することによってストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

   ```
   db2 connect to sample
   ```

   ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

   ```
   db2 -td@ -vf Spdrop.db2
   ```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf Spcreate.db2
```

4. カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。
5. Spclient クライアント・アプリケーションをコンパイルして実行し、ストアード・プロシージャー・クラスにアクセスします。 91ページの『ストアード・プロシージャー用のクライアント・アプリケーション』を参照してください。

このプログラムは、Java の makefile を使って構築することも可能です。

SQLJ プログラム

注: IBM Java 開発者キット (UNIX、OS/2、および Windows 32 ビット・オペレーティング・システム版) で SQLJ プログラムを構築し、実行するには、ご使用のオペレーティング・システムに該当する次のコマンドを使って、 JDK のジャストインタイム・コンパイラーをオフにしなければなりません。

OS/2 および Windows:

```
SET JAVA_COMPILER=NONE
```

UNIX: `export JAVA_COMPILER=NONE`

ビルド・ファイル bldsqlj には、 SQLJ アプレットまたはアプリケーションを構築するためのコマンドが組み込まれています。 UNIX ではスクリプト・ファイルがこれに相当します。 OS/2 ではコマンド・ファイル bldsqlj.cmd が、また Windows ではバッチ・ファイル bldsqlj.bat がこれに相当します。これらのコマンド・ファイルとバッチ・ファイルの内容は同じです。それでこのバージョンを最初に示してあり、その後に UNIX スクリプト・ファイルを示します。 アプレットとアプリケーションの構築に関するそれ以降の節では、これらのビルド・ファイルを後から参照しています。

注: DB2 に付属している SQLJ 変換プログラムは、変換後の .java ファイルを .class ファイルにコンパイルします。そのため、この節のビルド・ファイルは java コンパイラーを使用しません。

以下に示す、OS/2 および Windows 32 ビット・オペレーティング・システム用のビルド・ファイルでは、第 1 パラメーター %1 にソース・ファイルの名前

を指定します。第 2 パラメーター %2 には、接続先のデータベースの名前を指定します。 第 3 パラメーター %3 には、そのデータベースのユーザー ID を指定し、%4 にはそのパスワードを指定します。 最初のパラメーター (ソース・ファイル名) だけが、必須です。データベース名、ユーザー ID、およびパスワードは任意指定です。 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。

```
@echo off
rem bldsqlj -- OS/2 and Windows 32-bit operating systems
rem Builds a Java embedded SQL (SQLJ) program.
rem Usage: bldsqlj prog_name [ db_name [ userid password ]]
if "%1" == "" goto error
rem Translate and compile the SQLJ source file
rem and bind the package to the database.
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   sqlj %1.sqlj
   db2profc -url=jdbc:db2:sample -prepoptions="package using %1" %1_SJProfile0
   goto continue
:case2
   sqlj     -url=jdbc:db2:%2 %1.sqlj
   db2profc -url=jdbc:db2:%2 -prepoptions="package using %1" %1_SJProfile0
   goto continue
:case3
   sqlj     -url=jdbc:db2:%2 -user=%3 -password=%4  %1.sqlj
   db2profc -url=jdbc:db2:%2 -user=%3 -password=%4 -prepoptions="package using %1"
     %1_SJProfile0
   goto continue
:continue
goto exit
:error
echo Usage: bldsqlj prog_name [ db_name [ userid password ]]
:exit
@echo on
```

bldsqlj の変換プログラムとプリコンパイル・オプション

sqlj SQLJ 変換プログラム (プログラムのコンパイルも行います)。

%1.sqlj
SQLJ ソース・ファイル。

%1.java
SQLJ ソース・ファイルから変換した後の Java ファイル。

db2profc
Java プロファイル・カスタマイザーが使用する DB2。

-url データベース接続を確立するための JDBC URL を jdbc:db2:sample のように指定します。

-user ユーザー ID を指定します (任意指定パラメーター)。

-password
パスワードを指定します (任意指定パラメーター)。

-prepoptions
データベースのパッケージ名をストリング "package using %1" で指定します。 %1 は SQLJ ソース・ファイル名です。

%1_SJProfile0
プログラムのシリアル化プロファイルを指定します。

以下に示す UNIX スクリプト・ファイルでは、第 1 パラメーター $1 にソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。 第 3 パラメーター $3 にはそのデータベースのユーザー ID を、また $4 にはパスワードを指定します。最初のパラメーター (ソース・ファイル名) だけが、必須です。 データベース名、ユーザー ID、およびパスワードは任意指定です。 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。

```
#! /bin/ksh
# bldsqlj script file -- UNIX platforms
# Builds a Java embedded SQL (SQLJ) sample
# Usage: bldsqlj <prog_name> [ <db_name> [ <userid> <password> ]]
# Translate and compile the SQLJ source file
# and bind the package to the database.
if (($# < 2))
then
   sqlj $1.sqlj
   db2profc -url=jdbc:db2:sample -prepoptions="package using $1" $1_SJProfile0
elif (($# < 3))
then
   sqlj     -url=jdbc:db2:$2 $1.sqlj
   db2profc -url=jdbc:db2:$2 -prepoptions="package using $1" $1_SJProfile0
else
```

```
   sqlj     -url=jdbc:db2:$2 -user=$3 -password=$4  $1.sqlj
   db2profc -url=jdbc:db2:$2 -user=$3 -password=$4 -prepoptions="package using $1"
     $1_SJProfile0
fi
```

bldsqlj の変換プログラムとプリコンパイル・オプション

sqlj SQLJ 変換プログラム (プログラムのコンパイルも行います)。

$1.sqlj
SQLJ ソース・ファイル。

$1.java
SQLJ ソース・ファイルから変換した後の Java ファイル。

db2profc
Java プロファイル・カスタマイザーが使用する DB2。

-url データベース接続を確立するための JDBC URL を jdbc:db2:sample のように指定します。

-user ユーザー ID を指定します (任意指定パラメーター)。

-password
パスワードを指定します (任意指定パラメーター)。

-prepoptions
データベースのパッケージ名をストリング "package using $1" で指定します。 $1 は SQLJ ソース・ファイル名です。

$1_SJProfile0
プログラムのシリアル化プロファイルを指定します。

アプレット

Applt は、 DB2 データベースにアクセスする SQLJ アプレットを例示します。

このアプレットをビルド・ファイル bldsqlj で構築して実行するには、次のようにします。

1. DB2 マシン (サーバーまたはクライアント) に Web サーバーがインストールされて実行していることを確認してください。
2. Applt.html ファイルの内容を、ファイル内の指示に従って修正します。
3. Applt.html で指定された TCP/IP ポートで JDBC アプレット・サーバーを開始します。たとえば Applt.html では、 param name=port value='6789' と指定してから以下のように入力できます。

```
db2jstrt 6789
```

4. アプレットを次のコマンドで作成します。

   ```
   bldsqlj Applt [ <db_name> [ <userid> <password> ]]
   ```

   上記のコマンドで任意指定パラメーター <db_name> を指定すると、デフォルトの sample データベースではない別のデータベースにアクセスできます。任意指定パラメーター <userid> と <password> は、リモートのクライアント・マシンからサーバーにアクセスするときなど、アクセスする予定のデータベースが別のインスタンス上にある場合に指定しなければなりません。

5. Web ブラウザーが、作業ディレクトリーにアクセス可能であることを確認してください。 もしアクセス可能ではない場合は、次のファイルをディレクトリーにコピーして、アクセス可能にします。

   ```
   Applt.html,                          Applt.class,
   Applt_Cursor1.class,                 Applt_Cursor2.class,
   Applt_SJProfileKeys.class,           Applt_SJProfile0.ser
   ```

6. OS/2 および Windows 32 ビット・オペレーティング・システムの場合にはファイル %DB2PATH%¥java¥db2java.zip と %DB2PATH%¥java¥runtime.zip を、また UNIX の場合には sqllib/java/db2java.zip と sqllib/java/runtime.zip を、他の Applt ファイルと同じディレクトリーにコピーします。

7. クライアント・マシンで、 Web ブラウザー (JDK 1.1 をサポートしていなければなりません) を開始し、 Applt.html をロードします。

ステップ (1)、(5)、および (7) の代わりに、次のコマンドをクライアント・マシンの作業ディレクトリーで入力して、 Java 開発者キットに付属のアプレット・ビューアーを使用することができます。

```
appletviewer Applt.html
```

このプログラムは、Java の makefile を使って構築することも可能です。

アプリケーション

App は、 DB2 データベースにアクセスする SQLJ アプリケーションを例示します。

このアプリケーションをビルド・ファイル bldsqlj で構築するには、以下に示すコマンドを入力します。

```
bldsqlj App [ <db_name> [ <userid> <password> ]]
```

上記のコマンドで任意指定パラメーター <db_name> を指定すると、デフォルトの sample データベースではない別のデータベースにアクセスできます。任意

指定パラメーター <userid> と <password> は、リモートのクライアント・マシンからサーバーにアクセスするときなど、アクセスする予定のデータベースが別のインスタンス上にある場合に指定しなければなりません。

次のコマンドで、アプリケーションに対して Java インタープリターを実行します。

```
java App
```

このプログラムは、Java の makefile を使って構築することも可能です。

ストアード・プロシージャー用のクライアント・プログラム

Stclient は、 SQLJ ストアード・プロシージャー Stserver を JDBC アプリケーション・ドライバーを使って呼び出すクライアント・プログラムです。このクライアント・アプリケーションを構築して実行する前に、ストアード・プロシージャー・クラスをサーバー上で構築します。 99ページの『ストアード・プロシージャー』を参照してください。

このクライアント・プログラムをビルド・ファイル bldsqlj で構築するには、以下に示すコマンドを入力します。

```
bldsqlj Stclient [ <db_name> [ <userid> <password> ]]
```

上記のコマンドで任意指定パラメーター <db_name> を指定すると、デフォルトの sample データベースではない別のデータベースにアクセスできます。任意指定パラメーター <userid> と <password> は、リモートのクライアント・マシンからサーバーにアクセスするときなど、アクセスする予定のデータベースが別のインスタンス上にある場合に指定しなければなりません。

次のコマンドで、クライアント・アプリケーションに対して Java インタープリターを実行します。

```
java Stclient
```

このプログラムは、Java の makefile を使って構築することも可能です。

ユーザー定義関数用のクライアント・プログラム

UDFclie は、ユーザー定義関数サーバー・プログラム UDFsrv を、 JDBC アプリケーション・ドライバーを使って呼び出すクライアント・プログラムです。このクライアント・アプリケーションを構築して実行する前に、 UDFsrv プログラムをサーバー上で構築します。 102ページの『ユーザー定義関数 (UDF)』を参照してください。

この SQLJ クライアント・プログラムをビルド・ファイル bldsqlj で構築するには、以下に示すコマンドを入力します。

```
bldsqlj UDFclie [ <db_name> [ <userid> <password> ]]
```

上記のコマンドで任意指定パラメーター <db_name> を指定すると、デフォルトの sample データベースではない別のデータベースにアクセスできます。任意指定パラメーター <userid> と <password> は、リモートのクライアント・マシンからサーバーにアクセスするときなど、アクセスする予定のデータベースが別のインスタンス上にある場合に指定しなければなりません。

次のコマンドで、クライアント・アプリケーションに対して Java インタープリターを実行します。

```
java UDFclie
```

このプログラムは、Java の makefile を使って構築することも可能です。

ストアード・プロシージャー

ビルド・ファイル bldsqljs には、 SQLJ ストアード・プロシージャーを構築するためのコマンドが組み込まれています。 UNIX ではスクリプト・ファイルがこれに相当します。 OS/2 ではコマンド・ファイル bldsqljs.cmd が、また Windows ではバッチ・ファイル bldsqljs.bat がこれに相当します。これらのコマンド・ファイルとバッチ・ファイルの内容は同じです。それでこのバージョンを最初に示してあり、その後に UNIX スクリプト・ファイルを示します。

以下に示す、OS/2 および Windows 32 ビット・オペレーティング・システム用のビルド・ファイルでは、第 1 パラメーター %1 にソース・ファイルの名前を指定します。第 2 パラメーター %2 には、接続先のデータベースの名前を指定します。 ストアード・プロシージャーは、データベースが存在するのと同じインスタンス上に構築されなければならないので、ユーザー ID およびパスワード用のパラメーターはありません。

最初のパラメーター (ソース・ファイル名) だけが、必須です。 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。

```
@echo off
rem bldsqljs -- OS/2 and Windows 32-bit operating systems
rem Builds a Java embedded SQL (SQLJ) stored procedure
rem Usage: bldsqljs prog_name [ db_name ]
if "%1" == "" goto error
rem Translate and compile the SQLJ source file
rem and bind the package to the database.
if "%2" == "" goto case1
goto case2
:case1
   sqlj %1.sqlj
   db2profc -url=jdbc:db2:sample -prepoptions="package using %1" %1_SJProfile0
```

```
   goto continue
:case2
   sqlj     -url=jdbc:db2:%2 %1.sqlj
   db2profc -url=jdbc:db2:%2 -prepoptions="package using %1" %1_SJProfile0
:continue
rem Copy the *.class and *.ser files to the 'function' directory.
copy %1*.class "%DB2PATH%¥function"
copy %1*.ser "%DB2PATH%¥function"
goto exit
:error
echo Usage: bldsqljs prog_name [ db_name ]
:exit
@echo on
```

bldsqljs の変換プログラムとプリコンパイル・オプション

sqlj SQLJ 変換プログラム (プログラムのコンパイルも行います)。

%1.sqlj
SQLJ ソース・ファイル。

%1.java
SQLJ ソース・ファイルから変換した後の Java ファイル。

db2profc
Java プロファイル・カスタマイザーが使用する DB2。

-url データベース接続を確立するための JDBC URL を jdbc:db2:sample のように指定します。

-prepoptions
データベースのパッケージ名をストリング "package using %1" で指定します。 %1 は SQLJ ソース・ファイル名です。

%1_SJProfile0
プログラムのシリアル化プロファイルを指定します。

以下に示す UNIX スクリプト・ファイルでは、第 1 パラメーター $1 にソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

最初のパラメーター (ソース・ファイル名) だけが、必須です。 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。

```
#! /bin/ksh
# bldsqljs script file -- UNIX platforms
# Builds a Java embedded SQL (SQLJ) stored procedure
# Usage: bldsqljs <prog_name> [ <db_name> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Translate and compile the SQLJ source file
# and bind the package to the database.
if (($# < 2))
then
   sqlj $1.sqlj
   db2profc -url=jdbc:db2:sample -prepoptions="package using $1" $1_SJProfile0
else
   sqlj     -url=jdbc:db2:$2 $1.sqlj
   db2profc -url=jdbc:db2:$2 -prepoptions="package using $1" $1_SJProfile0
fi
# Copy the *.class and *.ser files to the 'function' directory.
rm -f $DB2PATH/function/$1*.class
rm -f $DB2PATH/function/$1*.ser
cp $1*.class $DB2PATH/function
cp $1*.ser $DB2PATH/function
```

bldsqljs の変換プログラムとプリコンパイル・オプション

sqlj SQLJ 変換プログラム (プログラムのコンパイルも行います)。

$1.sqlj
SQLJ ソース・ファイル。

$1.java
SQLJ ソース・ファイルから変換した後の Java ファイル。

db2profc
Java プロファイル・カスタマイザーが使用する DB2。

-url データベース接続を確立するための JDBC URL を `jdbc:db2:sample` のように指定します。

-prepoptions
データベースのパッケージ名をストリング `"package using $1"` で指定します。 `$1` は SQLJ ソース・ファイル名です。

$1_SJProfile0
プログラムのシリアル化プロファイルを指定します。

Stserver は、 DB2 データベースにアクセスするために、 JDBC アプリケーション・ドライバーを使用した PARAMETER STYLE JAVA ストアード・プロシージャーを例示します。ストアード・プロシージャーはサーバー上でコン

パイルされ、格納されます。そして、クライアント・アプリケーションによって呼び出されるとサーバー・データベースにアクセスし、そのクライアント・アプリケーションに情報を戻します。

このストアード・プロシージャー・クラスをビルド・ファイル bldsqljs で構築するには、以下に示すコマンドを入力します。

1. 以下のコマンドを入力します。

   ```
   bldsqljs Stserver [ <db_name> ]
   ```

   上記のコマンドで任意指定パラメーター <db_name> を指定すると、デフォルトの sample データベースではない別のデータベースにアクセスできます。

2. 次に、サーバー上で Stcreate.db2 スクリプトを実行することによってストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

   ```
   db2 connect to sample
   ```

   ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

   ```
   db2 -td@ -vf Stdrop.db2
   ```

   その後、次のコマンドでストアード・プロシージャーをカタログ化します。

   ```
   db2 -td@ -vf Stcreate.db2
   ```

3. カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

4. Stclient クライアント・アプリケーションをコンパイルして実行し、ストアード・プロシージャーを呼び出します。 98ページの『ストアード・プロシージャー用のクライアント・プログラム』を参照してください。

このプログラムは、Java の makefile を使って構築することも可能です。

ユーザー定義関数 (UDF)

UDFsrv は、 Java ユーザー定義関数の具体的な使用例を示しています。 UDF プログラムには SQL ステートメントが含まれていないため、 Java UDF プログラムには SQLJ ステートメントを含めることができません。 UDFsrv ライブラリーをサーバー上に作成すると、クライアント・アプリケーションはそれにアクセスすることができます。 DB2 には JDBC クライアント・アプリケーシ

ョンである UDFcli と、 SQLJ クライアント・アプリケーションである UDFclie の両方が用意されています。どちらを使っても UDFsrv ライブラリーにアクセスできます。

サーバー上で UDF プログラムを構築して実行するには、コマンド行で次のようにコマンドを入力します。

1. 以下に示すコマンドで UDFsrv.java をコンパイルし、ファイル UDFsrv.class を生成します。

   ```
   javac UDFsrv.java
   ```

2. UDFsrv.class を、 OS/2 および Windows 32 ビット・オペレーティング・システムの場合には %DB2PATH%¥function ディレクトリーへ、また UNIX の場合には sqllib/function ディレクトリーへコピーします。
3. UDFsrv ライブラリーへは、 JDBC または SQLJ のいずれのクライアント・アプリケーションでもアクセスできます。 JDBC クライアント・アプリケーション UDFcli をコンパイルして実行する方法については、 91ページの『ユーザー定義関数用のクライアント・アプリケーション』を参照してください。 SQLJ クライアント・アプリケーション UDFclie をコンパイルして実行する方法については、 98ページの『ユーザー定義関数用のクライアント・プログラム』を参照してください。

UDFcli と UDFclie には、 UDFsrv の中にある UDF をデータベースに登録するのに使用する CREATE FUNCTION SQL ステートメントが組み込まれています。 UDFcli と UDFclie にはさらに、登録が済んだ UDF を利用するための SQL ステートメントも組み込まれています。

DB2 Java アプレットの一般事項

1. いくつかの Java クラスから成る大型の JDBC または SQLJ アプレットの場合は、そのクラスすべてを単一の JAR ファイルにパッケージ化するよう選択できます。また、SQLJ アプレットでは、そのクラスに加えて、順番に並べられたプロファイルをパッケージしなければなりません。これを選択する場合、JAR ファイルを "applet" タグの archive パラメーターに追加します。詳細については、JDK バージョン 1.1 の資料を参照してください。

   SQLJ アプレットの場合: 一部のブラウザーは、アプレットに関連付けられたリソース・ファイルからの逐次化オブジェクトのロードをサポートしていません。たとえば、そのようなブラウザーでアプレット Applt をロードしようとすると、次のようなエラー・メッセージが出されます。

   ```
   java.lang.ClassNotFoundException: Applt_SJProfile0
   ```

これを回避する方法としては、逐次化プロファイルを変換して、 Java クラス形式で格納したプロファイルにするユーティリティーを使用することができます。このユーティリティーは、
sqlj.runtime.profile.util.SerProfileToClass という名前の Java クラスです。これは逐次化プロファイルのリソース・ファイルを入力として取り込み、そのプロファイルを含んだ Java クラスを出力として生成します。プロファイルは以下のいずれかのコマンドを使って変換できます。

```
profconv Applt_SJProfile0.ser
```

または

```
java sqlj.runtime.profile.util.SerProfileToClass Applt_SJProfile0.ser
```

このコマンドの結果、クラス Applt_SJProfile0.class が作成されます。アプレットが使用している .ser 形式のすべてのプロファイルを .class 形式のプロファイルに置き換えれば、問題はなくなるはずです。

2. ファイル db2java.zip (SQLJ アプレットの場合は、ファイル runtime.zip) を使用している Web サイトからロードすることのできるいくつかのアプレットによって共用されるディレクトリーに置くことができます。これらのファイルは、 OS/2 および Windows 32 ビット・オペレーティング・システムの場合には %DB2PATH%¥java ディレクトリーに、また UNIX の場合には sqllib/java ディレクトリーにあります。この場合、codebase パラメーターを、そのディレクトリーを識別する HTML ファイルの "applet" タグに追加する必要があることがあります。詳細については、JDK バージョン 1.1 の資料を参照してください。
3. DB2 バージョン 5.2 以降、 JDBC アプレット・サーバー (listener) である db2jd の機能を強化するために信号処理機能が追加されています。その結果、db2jd は Ctrl-C で強制終了することができません。そのためこの listener を終了させるには、そのプロセスを強制終了させるしか方法がありません。
4. Web サーバー、特に Domino GO Web サーバー上での DB2 Java アプレットの実行についての詳細は、次のサイトを参照してください。

   http://www.ibm.com/software/data/db2/db2lotus/gojava.htm

第5章 SQL プロシージャーの構築

この章では、DB2 SQL プロシージャーの構築について詳細に説明します。

DB2 SQL プロシージャーのサンプル・プログラムは、
sqllib/samples/sqlprocディレクトリー (UNIX プラットフォームの場合)、および %DB2PATH%¥samples¥sqlproc ディレクトリー (OS/2 および Windows 32 ビット・オペレーティング・システムの場合) にあります。サンプル・プログラムについての詳細は、 31ページの表10 を参照してください。

SQL プロシージャー環境のセットアップ

ここで扱われる内容は、39ページの『第2章 セットアップ』で扱われた、 DB2 環境のセットアップの説明を補足するものです。

ご使用の環境で SQL プロシージャーをサポートするためには、アプリケーション開発クライアントと、 DB2 がサポートしている C または C++ コンパイラーをサーバーにインストールする必要があります。アプリケーション開発クライアントのインストールについては、各プラットフォームの*概説およびインストール* を参照してください。各プラットフォームで DB2 がサポートしている C および C++ コンパイラーについては、 8ページの『各プラットフォームでサポートされるソフトウェア』を参照してください。

注: OS/2 FAT ファイル・システムでは、 SQL プロシージャーのスキーマ名が 8 文字以内に制限されています。スキーマ名が 8 文字以上になる場合は、 HPFS ファイル・システムを使用する必要があります。

コンパイラー環境の構成

SQL プロシージャーを作成するためには、次のステップに従って、サポートされている C または C++ コンパイラーをサーバー上で使用できるように DB2 を構成します。

- コンパイラーの環境をセットアップする実行可能ファイルを作成します。OS/2 ではコマンド・ファイル、UNIX では スクリプト・ファイル、Windows ではバッチ・ファイルがこれにあたります。コンパイラーには、パス、組み込み、およびライブラリーの環境変数が必要な場合があります。
- 次のコマンドを使用して、 DB2_SQLROUTINE_COMPILER_PATH DB2 レジストリー変数を実行可能ファイルとして設定します。

  ```
  db2set DB2_SQLROUTINE_COMPILER_PATH=executable_file
  ```

  ここで、*executable_file* は、 C コンパイラーの環境ファイルへのフルパス名を表します。

DB2_SQLROUTINE_COMPILER_PATH DB2 レジストリー変数が設定されない場合、 DB2 は代わりにデフォルトのファイルを実行可能ファイルとして設定します。このデフォルト・ファイルのパスとファイル名は、各オペレーティング・システムごとに、次のようになっています。

OS/2: `%DB2PATH%¥function¥routine¥sr_cpath.cmd`

UNIX: `$HOME/sqllib/function/routine/sr_cpath`

Windows:

```
%DB2PATH%¥function¥routine¥sr_cpath.bat
```

このデフォルト・ファイルを使用する場合は、これに変更を加えて、ご使用になるサーバー・オペレーティング・システムと、 C および C++ コンパイラーに必要な設定を反映させる必要があります。

注: Windows NT と Windows 2000 では、コンパイラーの環境変数を SYSTEM 変数として保管しておけば、DB2_SQLROUTINE_COMPILER_PATH DB2 レジストリー変数を設定する必要はありません。

コンパイラー・オプションのカスタマイズ

DB2 では、各プラットフォームごとに、サポートされているコンパイラーの中から 1 つのコンパイラーをデフォルト値として使用します。デフォルト値として設定されていない他のコンパイラーを使用する場合は、DB2_SQLROUTINE_COMPILE_COMMAND DB2 レジストリー変数を使用して、 SQL プロシージャーのコンパイラー・オプションを設定します。 SQL プロシージャーに、カスタマイズした C または C++ コンパイラー・オプショ

ンを指定するには、次のコマンドを使用して、DB2 レジストリーにすべてのオプションを含むコマンド行全体を保管します。

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=compiler_command
```

ここで、*compiler_command* は C または C++ コンパイル・コマンドを表しており、ストアード・プロシージャーの作成に必要なオプションやパラメーターが含まれます。

コンパイラー・コマンドにキーワード SQLROUTINE_FILENAME を使用して、生成された SQC、C、PDB、DEF、EXP、メッセージ・ログ、および共有ライブラリー・ファイルのファイル名を置き換えることができます。また、AIX の場合にのみ、キーワード SQLROUTINE_ENTRY を使用して入り口点を置き換えることができます。

サポートされている C または C++ コンパイラーの DB2_SQLROUTINE_COMPILE_COMMAND に使用されるデフォルト値の例として、以下に AIX、Solaris、および Windows 32 ビット・オペレーティング・システムのデフォルト・コンパイラー値を示します。加えて、この例には、デバッグ情報を戻すために提案されている変更も示されています。他のプラットフォームでデバッグ情報を戻す場合にも、これと同様の変更を加えることができます。

AIX 以下は、IBM C Set++ for AIX バージョン 3.6.6 で使用されるデフォルトのコンパイラー・コマンドの値です。

```
xlC_r -+ -H512 -T512 -I$HOME/sqllib/include SQLROUTINE_FILENAME.c \
  -bE:SQLROUTINE_FILENAME.exp -e SQLROUTINE_ENTRY \
  -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -lc -ldb2
```

デバッグ情報を戻すためには、次のようにデフォルトを変更して、DB2_SQLROUTINE_COMPILE_COMMAND に -g オプションを追加します。

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND="xlC_r -+ -H512 -T512 -g \
  -I$HOME/sqllib/include SQLROUTINE_FILENAME.c \
  -bE:SQLROUTINE_FILENAME.exp -e SQLROUTINE_ENTRY \
  -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -lc -ldb2"
```

ここで、"\" は改行を示すためのもので、実際の値ではありません。

注: AIX 上で 64 ビットの SQL プロシージャーをコンパイルする場合は、上のコマンドに -q64 オプションを追加してください。

Solaris

以下は、SPARCompiler C++ バージョン 4.2 および 5.0 で使用されるデフォルトのコンパイラー・コマンドの値です。

```
cc -# -Kpic -I$HOME/sqllib/include SQLROUTINE_FILENAME.c -G \
  -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -R$HOME/sqllib/lib -ldb2
```

デバッグ情報を戻すためには、次のようにデフォルトを変更して、DB2_SQLROUTINE_COMPILE_COMMAND に -g オプションを追加します。

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND="cc -# -Kpic -g \
  -I$HOME/sqllib/include SQLROUTINE_FILENAME.c -G \
  -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib \
  -R$HOME/sqllib/lib -ldb2"
```

ここで、"\" は改行を示すためのもので、実際の値ではありません。

注: Solaris 上で 64 ビットの SQL プロシージャーをコンパイルする場合は、上のコマンドに -xarch=v9 オプションを追加してください。

Windows 32 ビット・オペレーティング・システム

以下は、Microsoft Visual C++ バージョン 5.0 および 6.0 に使用されるデフォルトのコンパイラー・コマンドの値です。

```
cl -Od -W2 /TC -D_X86_=1 -I%DB2PATH%¥include SQLROUTINE_FILENAME.c
  /link -dll -def:SQLROUTINE_FILENAME.def /out:SQLROUTINE_FILENAME.dll
  %DB2PATH%¥lib¥db2api.lib
```

デバッグ情報を戻すためには、次のようにデフォルトを変更します。

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND="cl -Od -W2 /TC -D_X86_=1
  -Z7 -I%DB2PATH%¥include SQLROUTINE_FILENAME.c /link -dll
  -def:SQLROUTINE_FILENAME.def /out:SQLROUTINE_FILENAME.dll
  -debug:full -pdb:none -debugtype:cv %DB2PATH%¥lib¥db2api.lib"
```

注: Windows 32 ビット・オペレーティング・システムの場合は、コンパイラー・コマンドの値を改行せずに 1 つの行に入力しなければなりません。

デフォルトのコンパイラー・オプションを戻すには、次のコマンドを使用して、 DB2_SQLROUTINE_COMPILE_COMMAND の DB2 レジストリー値をヌルに設定します。

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=
```

中間ファイルの保存

CREATE PROCEDURE ステートメントが出されると、 DB2 はいくつもの中間ファイルを作成し、通常は、ステートメントが正常に完了されるとこれらを削除します。 SQL プロシージャーが期待した通りに実行されなかった場合には、 DB2 が作成した SQC、C、PDB、およびメッセージ・ログ・ファイルを調べることができます。 CREATE PROCEDURE ステートメントが正常に実行された場合にも DB2 が作成したファイルを保管したい場合は、次のコマンドで、DB2_SQLROUTINE_KEEP_FILES DB2 レジストリー変数の値を ″1″、″y″、または ″yes″ に設定してください。

```
db2set DB2_SQLROUTINE_KEEP_FILES=1
```

中間ファイルが保存されるディレクトリーは、各オペレーティング・システムごとに、次のようになっています。

UNIX

```
$HOME/sqllib/function/routine/sqlproc/database_name/schema_name
```

ここで、*database_name* と *schema_name* は、 SQL プロシージャーの作成に使用されたデータベースとスキーマを表しています。

OS/2 および Windows

```
%DB2PATH%¥function¥routine¥sqlproc¥database_name¥schema_name
```

ここで、*database_name* と *schema_name* は、 SQL プロシージャーの作成に使用されたデータベースとスキーマを表しています。

プリコンパイルおよびバインド・オプションのカスタマイズ

プリコンパイルおよびバインド・オプションは、DB2_SQLROUTINE_PREPOPTS DB2 レジストリー変数を設定することによってカスタマイズできます。これらのオプションを、プロシージャー・レベルでカスタマイズすることはできません。 SQL プロシージャーに、カスタマイズしたプリコンパイル・オプションを指定するには、次のコマンドを使用して、DB2 プリコンパイラーで使用するプリコンパイル・オプションのリストを DB2 レジストリーに追加します。

```
db2set DB2_SQLROUTINE_PREPOPTS=options
```

ここで、*options* は、 DB2 プリコンパイラーで使用するプリコンパイル・オプションのリストを示します。使用できるオプションは、次のものだけです。

```
BLOCKING {UNAMBIG | ALL | NO}
DATETIME {DEF | USA | EUR | ISO | JIS | LOC}
DEGREE {1 | degree-of-parallelism | ANY}
DYNAMICRULES {BIND | RUN}
EXPLAIN {NO | YES | ALL}
```

```
EXPLAINSNAP {NO | YES | ALL}
INSERT {DEF | BUF}
ISOLATION {CS |RR |UR |RS |NC}
QUERYOPT optimization-level
SYNCPOINT {ONEPHASE | TWOPHASE | NONE}
```

SQL プロシージャーの作成

sqllib/samples/sqlproc ディレクトリー (UNIX の場合)、および %DB2PATH%¥samples¥sqlproc ディレクトリー (OS/2 および Windows) にある DB2 コマンド行プロセッサーのスクリプト (最後に .db2 の拡張子が付く) は、 CREATE PROCEDURE ステートメントを実行し、サーバー上にストアード・プロシージャーを作成します。このステートメントを実行する際には、データベースへの接続が確立されていなければなりません。各 CLP スクリプトには、.sqc または .c の拡張子を持つ、同じ名前の対応するクライアント・アプリケーション・ファイルがあります。

CREATE PROCEDURE CLP スクリプトを実行する際には、次のコマンドでサンプル・データベースに接続してください。

```
db2 connect to sample user userid using password
```

ここで、*userid* と *password* は、 sample データベースが置かれているインスタンスのユーザー ID とパスワードを表します。

rsultset.db2 スクリプト・ファイルの CREATE PROCEDURE を実行するには、次のコマンドを入力します。

```
db2 -td@ -vf rsultset.db2
```

次いで、DB2 を停止し、再始動します。これで、下の『SQL プロシージャーの呼び出し』に説明されている方法で SQL プロシージャーを呼び出せるようになります。

SQL プロシージャーの呼び出し

コマンド行プロセッサー (CLP) の call コマンドを使用するか、クライアント・アプリケーションを構築することによって SQL プロシージャーを呼び出すことができます。

CALL コマンドの使用

call コマンドを使用するには、ストアード・プロシージャーの名前に加えて、そのストアード・プロシージャーが要求している IN または INOUT 引き数を入力する必要があります。 OUT パラメーターを入力することはできません。

まず、110ページの『SQL プロシージャーの作成』のステップに従って、 SQL プロシージャーを作成します。

作成された SQL プロシージャーを呼び出すには、最初にデータベースへ接続する必要があります。

```
db2 connect to sample user userid using password
```

ここで、*userid* と *password* は、 sample データベースが置かれているインスタンスのユーザー ID とパスワードを表します。

ストアード・プロシージャーのパラメーターは、プログラム・ソース・ファイルにある、そのストアード・プロシージャーの CREATE PROCEDURE ステートメントに示されています。たとえば、ソース・ファイル whiles.db2 にある、DEPT_MEDIAN プロシージャーの CREATE PROCEDURE ステートメントは、次のように始まっています。

```
CREATE PROCEDURE DEPT_MEDIAN
(IN deptNumber SMALLINT, OUT medianSalary DOUBLE)
```

この場合、このプロシージャーを呼び出すには、 IN パラメーター deptNumber に、有効な SMALLINT 値を入力する必要があります。有効な値は、サンプル・データベース内の対応するテーブルで調べることもできますし、クライアント呼び出しプログラムのソース・ファイルで、使用されている値を調べることもできます。たとえば、下の whiles.sqc では、値 ″51″ が使用されているのがわかります。

```
printf("Use CALL with Host Variables to invoke the Server Procedure "
   "named %s¥n", procname);
dept = 51;                    /* get median for dept. 51 */
```

call コマンドと共に、プロシージャー名と IN パラメーターの値を入力します。プロシージャーのパラメーターは、下の例のように必ず括弧で囲み、引用符を使用してください。

```
db2 "call DEPT_MEDIAN (51)"
```

すると、次のような結果が得られます。

```
MEDIANSALARY: 1.76545000000000e+04
```

call コマンドを使用する際は、以下の点に注意してください。

- 結果の列は、最大で 1023 文字までです。
- 呼び出すことができるのは、カタログで定義されているストアード・プロシージャーだけです。

- LOB と バイナリー・データ (FOR BIT DATA、 VARBINARY、 LONGVARBINARY、 GRAPHIC、 VARGRAPHIC、 LONGVARGRAPHIC) はサポートされていません。

OS/2 DB2 CLI クライアント・アプリケーション

%DB2PATH%¥samples¥sqlproc のコマンド・ファイル bldcli.cmd には、 SQL プロシージャー用の DB2 CLI クライアント・アプリケーションを作成するコマンドが含まれています。 bldcli.cmd についての詳細は、 252ページの『DB2 CLI アプリケーション』を参照してください。

DB2 CLI クライアント・アプリケーション rsultset を作成するには、ソース・ファイル rsultset.c から次のように入力します。

```
bldcli rsultset
```

このコマンドによって実行可能ファイル rsultset が作成されます。

ストアード・プロシージャーを呼び出すには、実行可能ファイルの名前、接続しているデータベースの名前、そしてデータベース・インスタンスのユーザー ID とパスワードを入力して、サンプル・クライアント・アプリケーションを実行します。

```
rsultset database userid password
```

OS/2 組み込み SQL クライアント・アプリケーション

%DB2PATH%¥samples¥sqlproc のコマンド・ファイル bldapp.cmd には、 SQL プロシージャー用の組み込み SQL クライアント・アプリケーションを作成するコマンドが含まれています。 bldapp.cmd についての詳細は、 258ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル basecase.sqc から組み込み SQL クライアント・アプリケーション basecase を作成するには、コマンド・ファイル名、実行可能ファイル名、接続しているデータベース、およびデータベース・インスタンスのユーザー ID とパスワードを入力します。

```
bldapp basecase database userid password
```

結果として、実行可能ファイル basecase が作成されます。

ストアード・プロシージャーを呼び出すためには、次のように入力してクライアント・アプリケーションを実行します。

```
basecase database userid password
```

UNIX DB2 CLI クライアント・アプリケーション

sqllib/samples/sqlproc のスクリプト・ファイル bldcli には、 SQL プロシージャー用の DB2 CLI クライアント・アプリケーションを作成するコマンドが含まれています。 bldcli スクリプト・ファイルについての詳細は、 UNIX プラットフォーム用の『アプリケーションの構築』という章の『DB2 CLI アプリケーション』という節を参照してください。

DB2 CLI クライアント・アプリケーション rsultset を作成するには、ソース・ファイル rsultset.c から次のように入力します。

```
bldcli rsultset
```

このコマンドによって実行可能ファイル rsultset が作成されます。

ストアード・プロシージャーを呼び出すには、実行可能ファイルの名前、接続しているデータベースの名前、そしてデータベース・インスタンスのユーザー ID とパスワードを入力して、サンプル・クライアント・アプリケーションを実行します。

rsultset *database userid password*

UNIX 組み込み SQL クライアント・アプリケーション

sqllib/samples/sqlproc のスクリプト・ファイル bldapp には、 SQL プロシージャー用の組み込み SQL クライアント・アプリケーションを作成するコマンドが含まれています。 bldapp スクリプト・ファイルについての詳細は、 UNIX プラットフォーム用の『アプリケーションの構築』という章の『DB2 API アプリケーション』という節を参照してください。

ソース・ファイル basecase.sqc から組み込み SQL クライアント・アプリケーション basecase を作成するには、スクリプト・ファイル名、実行可能ファイル名、接続しているデータベース、およびデータベース・インスタンスのユーザー ID とパスワードを入力します。

```
bldapp basecase database userid password
```

結果として、実行可能ファイル basecase が作成されます。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

basecase *database userid password*

Windows DB2 CLI クライアント・アプリケーション

%DB2PATH%¥samples¥sqlproc ディレクトリーには、 DB2 CLI クライアント・アプリケーションを作成するための 2 つのビルド・ファイル、 bldmcli (Microsoft Visual C++ コンパイラー用) と、 bldvcli (IBM VisualAge C++ コンパイラー用) が含まれています。 bldmcli についての詳細は、 372ページの『DB2 CLI アプリケーション』を参照してください。 bldvcli についての詳細は、 388ページの『DB2 CLI アプリケーション』を参照してください。

DB2 CLI クライアント・アプリケーション rsultset を作成するには、ソース・ファイル rsultset.c から、使用するコンパイラーごとに次のいずれかを入力します。

```
bldmcli rsultset
```

または

```
bldvcli rsultset
```

これらのコマンドによって実行可能ファイル rsultset が作成されます。

ストアード・プロシージャーを呼び出すには、実行可能ファイルの名前、接続しているデータベースの名前、そしてデータベース・インスタンスのユーザー ID とパスワードを入力して、サンプル・クライアント・アプリケーションを実行します。

rsultset *database userid password*

Windows 組み込み SQL クライアント・アプリケーション

%DB2PATH%¥samples¥sqlproc ディレクトリーには、組み込み SQL クライアント・アプリケーションを作成するための 2 つのビルド・ファイル、 bldmapp (Microsoft Visual C++ コンパイラー用) と、 bldvapp (IBM VisualAge C++ コンパイラー用) が含まれています。 bldmapp についての詳細は、 378ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。 bldvapp についての詳細は、 394ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル basecase.sqc から組み込み SQL クライアント・アプリケーション basecase を作成するには、スクリプト・ファイル名、実行可能ファイル名、接続しているデータベース、およびデータベース・インスタンスのユーザー ID とパスワードを入力します。使用するコンパイラーごとに、次のいずれかのコマンドを使用します。

```
bldmapp basecase database userid password
```

または

```
bldvapp basecase database userid password
```

結果として、実行可能ファイル basecase が作成されます。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

```
basecase database userid password
```

第6章 AIX アプリケーションの構築

この章は、AIX でアプリケーションを構築するための詳細な情報を提供します。スクリプト・ファイルにおいて、db2 から始まるコマンドは、コマンド行プロセッサー (CLP) のコマンドです。 CLP コマンドについての詳細な情報が必要であれば、 *コマンド解説書* を参照してください。

AIX 用の DB2 アプリケーション開発の最新の更新事項については、次の Web ページを参照してください。

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

注: この章のビルド・ファイルを使用して 64 ビットのアプリケーションを構築する場合、各ビルド・ファイルで指示されているコマンドをコメント解除することもできますし、以下のコマンドを使用して 64 ビットのオブジェクト・モード環境を設定することもできます。

```
export OBJECT_MODE=64
```

重要な考慮事項

この節では、サポートされている各種コンパイラーで DB2 アプリケーションを構築するための情報として、 AIX に固有の情報を提供します。次のような内容を扱います。

- IBM および Micro Focus COBOL の実行
- ストアード・プロシージャーおよび UDF 用の入り口点
- ストアード・プロシージャーおよび CALL ステートメント
- UDF および CREATE FUNCTION ステートメント

IBM および Micro Focus COBOL の実行

AIX がストアード・プロシージャーをロードしたり、その中にあるライブラリー参照を解決したりする方法のために、 COBOL をインストールする方法についての要件があります。これらの要件は、 COBOL プログラムが実行時に共用ライブラリー (ストアード・プロシージャー) をロードするときの要素となります。

ストアード・プロシージャーをロードするときには、それが参照する一連のライブラリーの連鎖もロードする必要があります。プログラムで間接的にのみ参照するライブラリーを AIX が探索するときには、言語プロバイダー (IBM COBOL または Micro Focus COBOL) の作成した参照ライブラリーにコンパイルされたパスを使用する必要があります。このパスは必ずしも、コンパイラーがインストールされたパスと同じとは限りません。連鎖に含まれるライブラリーをバインドできない場合、ストアード・プロシージャーのロードは失敗し、SQLCODE -10013 を受け取ります。

これが生じないようにするには、必要なときには常にコンパイラーをインストールし、その後、すべての言語ライブラリーの記号リンクを、インストール・ディレクトリーから `/usr/lib` (ライブラリーのロードが必要なときには、ほぼ必ず探索されるディレクトリー) に作成します。ライブラリーを `sqllib/function` (ストアード・プロシージャーのディレクトリー) にリンクできますが、これは、1 つのデータベース・インスタンスに対してしか作動しません。 `/usr/lib` は、マシン上のすべてのデータベース・インスタンスに対し

て作動します。ライブラリーをコピーしないことを強くお勧めします。 (特に、ライブラリーの複数のコピーが存在するときの Micro Focus COBOL の場合。)

Micro Focus COBOL のサンプルの記号リンクを下記に示します (/usr/lpp/cobdirにインストールされていると仮定する)。

```
[1]> su root
[2]> cd /usr/lib
[1]> ln -sf /usr/lpp/cobdir/coblib/*.a .
```

ストアード・プロシージャーおよび UDF 用の入り口点

ストアード・プロシージャーは、データベースにアクセスし、クライアント・アプリケーションに情報を戻すプログラムです。ユーザー定義関数 (UDF) は、ユーザー独自のスカラー関数または表関数です。ストアード・プロシージャーおよび UDF は、サーバー上でコンパイルされ、サーバー上の共用ライブラリーに保管されて実行されます。これらの共用ライブラリーは、ストアード・プロシージャーおよび UDF をコンパイルするときに作成されます。

共用ライブラリーには入り口点が 1 つずつあります。入り口点は、サーバーから呼び出され、共用ライブラリー内のプロシージャーにアクセスします。 AIX 上の IBM C コンパイラーでは、ライブラリー内のエクスポートされた関数名を、デフォルトの入り口点として指定することができます。これは、ストアード・プロシージャーの呼び出しまたは CREATE FUNCTION ステートメントで、ライブラリー名だけを指定した場合に呼び出される関数です。これを行うには、リンク・ステップで -e オプションを指定します。たとえば、次のようにします。

```
-e funcname
```

これは、funcname をデフォルトの入り口点とします。 CREATE FUNCTION ステートメントとの関連については、 121ページの『UDF および CREATE FUNCTION ステートメント』を参照してください。

他の UNIX プラットフォームでは、そのような機構は存在しないので、 DB2 はデフォルトの入り口点がライブラリーと同じ名前であるとみなします。

AIX では、外部から呼び出し可能な、ライブラリー内のグローバル関数を指定するエクスポート・ファイルを提供することが必要です。このファイルは、ライブラリー内のすべてのストアード・プロシージャーまたはユーザー定義関数 (あるいはその両方) の名前を含んでいる必要があります。他の UNIX プラッ

トフォームは単に、ライブラリー内のすべてのグローバル関数をエクスポートするだけです。次は、AIX エクスポート・ファイルの例です。

```
#! outsrv export file
outsrv
```

エクスポート・ファイルの outsrv.exp は、ストアード・プロシージャー outsrv をリストします。リンカーは outsrv.exp を使用して、同じ名前のストアード・プロシージャーを含む共用ライブラリー outsrv を作成します。

注: 共用ライブラリーを作成した後、一般に、 DB2 がそれにアクセスするディレクトリーにコピーします。ストアード・プロシージャーまたはユーザー定義関数の共用ライブラリーを置換するには、 /usr/sbin/slibclean を実行して AIX 共用ライブラリーのキャッシュをフラッシュするか、またはライブラリーを宛先ディレクトリーから除去した後、ソース・ディレクトリーから宛先ディレクトリーにライブラリーをコピーする必要があります。そうしなければ、参照されるライブラリーのキャッシュを AIX が保持し、ライブラリーの重ね書きを認めないため、コピー操作が失敗します。

AIX コンパイラーの資料には、エクスポート・ファイルに関する追加情報が記載されています。

ストアード・プロシージャーおよび CALL ステートメント

アプリケーション開発の手引き は、ストアード・プロシージャーのコーディング方法を説明しています。 *SQL 解説書* は、 CALL ステートメントを使用するデータベースのロケーションでストアード・プロシージャーを呼び出す方法を説明しています。この節では、ストアード・プロシージャーをコンパイルおよびリンクする方法と、 CALL ステートメントで提供する情報について説明します。

プログラムをコンパイルしリンクするときには、次の 2 つの方法で関数を識別することができます。

- -e オプションを使用する。

  たとえば、リンク・ステップで次のように指定できます。

  ```
  -e modify
  ```

  これは、リンク・ライブラリーのデフォルト入り口点が関数 modify であることを示します。

/u/mydir/procs ディレクトリー中のライブラリー mystored をリンクしており、上記のようにデフォルト入り口点 modify を使用したい場合、 CALL ステートメントに次のものを含めてください。

```
CALL '/u/mydir/procs/mystored'
```

ライブラリー mystored は、メモリーにロードされ、関数 modify はデフォルト入り口点として DB2 によってピックアップされて実行されます。

- -bE: オプションを使って指定されたエクスポート・ファイルを使用します。

  一般には、ライブラリー中に複数のストアード・プロシージャーがあり、追加の関数にストアード・プロシージャーとしてアクセスしたい場合に、このリンク・オプションを使います。

  上記の例から続けるために、ライブラリー mystored が 3 つのストアード・プロシージャー、すなわち modify、remove、そして add を持っているものとします。 modify を上記のようにデフォルト入り口点として指定し、remove および add をエクスポート・ファイル中に含めることによって、それらが追加入り口点であることをリンク・ステップに通知します。

  リンク・ステップにおいて、次のように指定します。

  ```
  -bE:mystored.exp
  ```

  これは、エクスポート・ファイル mystored.exp を指定します。

  エクスポート・ファイルは、ストアード・プロシージャーの関数がリストされたもので、最初にデフォルトの入り口点がリストされます。

  ```
  modify
  remove
  add
  ```

  最後に、ストアード・プロシージャーの 2 つの CALL ステートメント、すなわち remove 関数および add 関数を呼び出すステートメントを、次のようにコーディングします。

  ```
  CALL '/u/mydir/procs/mystored!remove'
  ```

  および

  ```
  CALL '/u/mydir/procs/mystored!add'
  ```

UDF および CREATE FUNCTION ステートメント

アプリケーション開発の手引き は、UDF のコーディング方法を説明します。*SQL 解説書* は、 CREATE FUNCTION ステートメントを使って UDF を DB2 に登録する方法を説明しています。この節は、UDF をコンパイルし、リ

ンクする方法と、 CREATE FUNCTION ステートメントの EXTERNAL NAME 文節中で提供されている情報との間に見られる関係を説明します。

プログラムをコンパイルしリンクするときには、次の 2 つの方法で関数を識別することができます。

- -e オプションを使用する。

  たとえば、リンク・ステップで次のように指定できます。

  ```
  -e modify
  ```

  これは、リンク・ライブラリーのデフォルト入り口点が関数 modify であることを示します。

  /u/mydir/procs ディレクトリー中のライブラリー myudfs をリンクしており、上記のようにデフォルト入り口点 modify を使用したい場合、 CREATE FUNCTION ステートメントに次のものを含めてください。

  ```
  EXTERNAL NAME '/u/mydir/procs/myudfs'
  ```

  DB2 はライブラリー myudfs のデフォルトの入り口点をピックアップします。これは、関数 modify です。

- -bE: オプションを使って指定されたエクスポート・ファイルを使用します。

  一般的に、ライブラリー中に複数の UDF があり、追加の関数に UDF としてアクセスしたい場合に、このリンク・オプションを使います。

  上の例から続けるために、ライブラリー myudfs が 3 つの UDF、すなわち上記の modify、 remove、そして add を持っているものとします。 modify を上記のようにデフォルト入り口点として指定し、 remove および add をエクスポート・ファイル中に含めることによって、それらが追加入り口点であることをリンク・ステップに通知します。

  リンク・ステップにおいて、次のように指定します。

  ```
  -bE:myudfs.exp
  ```

  これは、エクスポート・ファイル myudfs.exp を指定します。

  エクスポート・ファイルは、次のようになります。

```
* additional entry points for myudfs
#!
remove
add
```

最後に、 UDF 用の 2 つの CREATE FUNCTION ステートメント (remove および add 関数によって実行する) に、以下の EXTERNAL NAME 文節を含めます。

```
EXTERNAL NAME '/u/mydir/procs/myudfs!remove'
```

および

```
EXTERNAL NAME '/u/mydir/procs/myudfs!add'
```

IBM C

この節では、次に示す DB2 インターフェースとともに IBM C を使用する方法について説明します。

- DB2 CLI
- DB2 API
- 組み込み SQL

DB2 CLI アプリケーション

sqllib/samples/cli にあるスクリプト・ファイル bldcli には、 DB2 CLI プログラムを作成するためのコマンドが含まれています。パラメーター $1 には、ソース・ファイルの名前を指定します。

必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない CLI プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

プログラムに組み込み SQL が含まれている場合 (拡張子が .sqc の場合) は、embprep スクリプトが呼び出されてそのプログラムをプリコンパイルし、 .c という拡張子のプログラム・ファイルを生成します。

```
#! /bin/ksh
# bldcli script file -- AIX
# Builds a CLI program with IBM C.
# Usage: bldcli <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
```

```
fi
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi
# Compile the error-checking utility.
xlc $CFLAGS_64 -I$DB2PATH/include -c utilcli.c
# Compile the program.
xlc $CFLAGS_64 -I$DB2PATH/include -c $1.c
# Link the program.
xlc $CFLAGS_64 -o $1 $1.o utilcli.o -L$DB2PATH/lib -ldb2
```

bldcli のコンパイルおよびリンク・オプション

コンパイル・オプション

xlc IBM C コンパイラー。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldcli のコンパイルおよびリンク・オプション
リンク・オプション **xlc** コンパイラーをリンカーのフロントエンドとして使用します。 **$CFLAGS_64** 'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。 **-o $1** 実行可能プログラムを指定します。 **$1.o** オブジェクト・ファイルを指定します。 **utilcli.o** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル tbinfo.c からサンプル・プログラム tbinfo を構築するには、次のようにします。

```
bldcli tbinfo
```

結果として、実行可能ファイル tbinfo が作成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
tbinfo
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル dbusemx.sqc から組み込み SQL アプリケーション dbusemx を作成する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldcli dbusemx
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldcli dbusemx database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

```
bldcli dbusemx database userid password
```

結果として、実行可能ファイル dbusemx が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   dbusemx
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   dbusemx database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   dbusemx database userid password
   ```

DB2 API を使用する DB2 CLI アプリケーション

DB2 には、CLI サンプル・プログラムが含まれています。このサンプル・プログラムは、DB2 API を使用してデータベースを作成およびドロップし、 CLI 機能を複数のデータベースで使用する方法を示します。 DB2 API を使用するサンプルは、 27ページの表7 にある CLI サンプル・プログラムの説明の中に示されています。

sqllib/samples/cli にあるスクリプト・ファイル bldapi には、 DB2 API を使用して DB2 CLI プログラムを作成するためのコマンドが入っています。このファイルは、データベースを作成およびドロップするための DB2 API が入った utilapi ユーティリティー・ファイルでコンパイルおよびリンクします。この点が、このスクリプト・ファイルと bldcli スクリプトの唯一の違いです。 bldapi と bldcli の両方に共通するコンパイルとリンクのオプションについては、 123ページの『DB2 CLI アプリケーション』 を参照してください。

ソース・ファイル dbmconn.c からサンプル・プログラム dbmconn を作成するには、次のようにします。

```
bldapi dbmconn
```

結果として、実行可能ファイル dbmconn が作成されます。この実行可能ファイルを実行するには、次の実行可能名を入力します。

```
dbmconn
```

DB2 CLI ストアード・プロシージャー

sqllib/samples/cli のスクリプト・ファイル bldclisp には、 DB2 CLI ストアード・プロシージャーを構築するコマンドが入っています。パラメーター $1 には、ソース・ファイルの名前を指定します。 $2 には、共用ライブラリーへの入り口点になっているストアード・プロシージャー関数を指定します。

```
#! /bin/ksh
# bldclisp script file -- AIX
# Builds a CLI stored procedure in IBM C.
# Usage: bldclisp <prog_name> [ <entry_point> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi
# Compile the error-checking utility.
xlc $CFLAGS_64 -I$DB2PATH/include -c utilcli.c
# Compile the program.
xlc $CFLAGS_64 -I$DB2PATH/include -c $1.c
# Link the program.
xlc $CFLAGS_64 -o $1 $1.o utilcli.o -L$DB2PATH/lib ¥
  -ldb2 -lm -H512 -T512 -bE:$1.exp -e $2
# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldclisp のコンパイルおよびリンク・オプション

コンパイル・オプション

xlc IBM C コンパイラー。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、 $HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

bldclisp のコンパイルおよびリンク・オプション
リンク・オプション **xlc** コンパイラーをリンカーのフロントエンドとして使用します。 **$CFLAGS_64** 'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。 **-o $1** 実行可能プログラムを指定します。 **$1.o** オブジェクト・ファイルを指定します。 **utilcli.o** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。 **-ldb2** DB2 ライブラリーとリンクします。 **-lm** 数学ライブラリーとリンクします。 **-H512** 出力ファイル位置合わせを指定します。 **-T512** 出力ファイル・テキスト・セグメントの開始アドレスを指定します。 **-bE:$.exp** エクスポート・ファイルを指定します。エクスポート・ファイルには、ストアード・プロシージャーのリストが含まれています。 **-e $2** 共用ライブラリーに対するデフォルト入り口点を指定します。 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。

ソース・ファイル spserver.c からサンプル・プログラム spserver を構築するには、そのビルド・ファイル名、プログラム名、および共用ライブラリーへの入り口点になっているストアード・プロシージャー関数を入力します。

```
bldclisp spserver outlanguage
```

スクリプト・ファイルは、ストアード・プロシージャーを sqllib/function ディレクトリーにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、 CLI クライアント・アプリケーション spclient を構築することができます。これは、共用ライブラリー内のストアード・プロシージャーを呼び出すアプリケーションです。

spclient は、スクリプト・ファイル bldcli を使用して構築することができます。詳細については、123ページの『DB2 CLI アプリケーション』を参照してください。

共用ライブラリーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

DB2 API と組み込み SQL アプリケーション

sqllib/samples/c にあるビルド・ファイル bldapp には、 DB2 アプリケーション・プログラムを構築するコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない DB2 API プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- AIX
# Builds a C application program.
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi
# If embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.c error-checking utility.
  xlc $CFLAGS_64 -I$DB2PATH/include -c utilemb.c
else
  # Compile the utilapi.c error-checking utility.
  xlc $CFLAGS_64 -I$DB2PATH/include -c utilapi.c
fi
# Compile the program.
xlc $CFLAGS_64 -I$DB2PATH/include -c $1.c
if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o
  xlc $CFLAGS_64 -o $1 $1.o utilemb.o -ldb2 -L$DB2PATH/lib
else
  # Link the program with utilapi.o
  xlc $CFLAGS_64 -o $1 $1.o utilapi.o -ldb2 -L$DB2PATH/lib
fi
```

bldapp のコンパイルおよびリンク・オプション
コンパイル・オプション **xlc** IBM C コンパイラー。 **$CFLAGS_64** 'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。 **-I$DB2PATH/include** DB2 インクルード・ファイルのロケーションを指定します。たとえば、 $HOME/sqllib/include。 **-c** コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。
リンク・オプション **xlc** コンパイラーをリンカーのフロントエンドとして使用します。 **$CFLAGS_64** 'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。 **-o $** 実行可能プログラムを指定します。 **$1.o** プログラム・オブジェクト・ファイルを指定します。 **utilemb.o** 組み込み SQL プログラムの場合に、エラー・チェックを行う組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。 **utilapi.o** 組み込み SQL プログラムでない場合に、エラー・チェックを行う DB2 API ユーティリティー・オブジェクト・ファイルを含みます。 **-ldb2** データベース・マネージャー・ライブラリーとリンクします。 **-L$DB2PATH/lib** DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。 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。

ソース・ファイル client.c から DB2 API 非組み込み SQL サンプル・プログラム client を構築するには、次のようにします。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。

この実行可能ファイルを実行するには、ファイル名を入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqc から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

sqllib/samples/c にあるスクリプト・ファイル bldsrv には、ストアード・プロシージャーを構築するためのコマンドが含まれています。スクリプト・ファ

イルは、ストアード・プロシージャーを共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、共用ライブラリーへの入り口点になっているストアード・プロシージャー関数を指定します。第 3 パラメーター $3 には、接続先のデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

最初の 2 つのパラメーター (ソース・ファイル名と入り口点) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldsrv script file -- AIX
# Builds a C stored procedure
# Usage: bldsrv <prog_name> <entry_point> [ <db_name> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $3
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi
# Compile the program.
xlc $CFLAGS_64 -I$DB2PATH/include -c $1.c
# Link the program using the export file $1.exp,
# creating shared library $1 with entry point $2.
xlc $CFLAGS_64 -o $1 $1.o -ldb2 -L$DB2PATH/lib -H512 -T512 -bE:$1.exp -e $2
# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldsrv のコンパイルおよびリンク・オプション
コンパイル・オプション **xlc** IBM C コンパイラー。 **$CFLAGS_64** 'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。 **-I$DB2PATH/include** DB2 インクルード・ファイルのロケーションを指定します。たとえば、 $HOME/sqllib/include。 **-c** コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。
リンク・オプション **xlc** コンパイラーをリンカーのフロントエンドとして使用します。 **$CFLAGS_64** 'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。 **-o $1** 出力を、共用ライブラリー・ファイルとして指定します。 **$1.o** ストアード・プロシージャー・オブジェクト・ファイルを指定します。 **-ldb2** DB2 ライブラリーとリンクします。 **-L$DB2PATH/lib** DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。 **-H512** 出力ファイル位置合わせを指定します。 **-T512** 出力ファイル・テキスト・セグメントの開始アドレスを指定します。 **-bE:$1.exp** エクスポート・ファイルを指定します。エクスポート・ファイルには、ストアード・プロシージャーのリストが含まれています。 **-e $1** 共用ライブラリーに対するデフォルト入り口点を指定します。 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。

ソース・ファイル spserver.sqc からサンプル・プログラム spserver を構築する場合、 sample データベースに接続しているときは、ビルド・ファイル

名、プログラム名、および共用ライブラリーの入り口点になっているストアード・プロシージャー関数の名前を入力します。

```
bldsrv spserver outlanguage
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv spserver outlanguage database
```

このスクリプト・ファイルは、ストアード・プロシージャーをサーバー上の sqllib/function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、共用ライブラリーにアクセスするクライアント・アプリケーション spclient を構築することができます。

spclient は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 129ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

ユーザー定義関数 (UDF)

スクリプト・ファイル bldudf は、 sqllib/samples/c にあり、 UDF を作成するためのコマンドが含まれています。 UDF は、ストアード・プロシージャーと同じようにコンパイルされます。 UDF の中に SQL ステートメントを含めることはできません。したがって、UDF プログラムを作成する場合、データベースへの接続、プログラムのプリコンパイル、およびバインドは行いません。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、共用ライブラリーへの入り口点になっているストアード・プロシージャー関数を指定します。スクリプト・ファイルは、ソース・ファイル名 $1 を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldudf script file -- AIX
# Builds a C UDF library
# Usage: bldudf <prog_name> <entry_point>
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi
# Compile the program.
xlc $CFLAGS_64 -I$DB2PATH/include -c $1.c
# Link the program.
xlc $CFLAGS_64 -o $1 $1.o -ldb2 -ldb2apie -L$DB2PATH/lib -H512 -T512 -bE:$1.exp -e $2
# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldudf のコンパイルおよびリンク・オプション

コンパイル・オプション

xlc IBM C コンパイラー。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、 `$HOME/sqllib/include`。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

bldudf のコンパイルおよびリンク・オプション

リンク・オプション

xlc コンパイラーをリンカーのフロントエンドとして使用します。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。

-o $1 出力を、共用ライブラリー・ファイルとして指定します。

$1.o 共用ライブラリー・オブジェクト・ファイルを指定します。

-ldb2 データベース・マネージャー・ライブラリーとリンクします。

-ldb2apie
DB2 API エンジン・ライブラリーとリンクして、LOB ロケーターを使用できるようにします。

-L$DB2PATH/lib
DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。

-H512 出力ファイル位置合わせを指定します。

-T512 出力ファイル・テキスト・セグメントの開始アドレスを指定します。

-bE:$1.exp
エクスポート・ファイルを指定します。エクスポート・ファイルには、UDF のリストが含まれています。

-e $2 共用ライブラリーに対するデフォルト入り口点を指定します。

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。
UDF の作成に関する詳細については、 121ページの『UDF および CREATE FUNCTION ステートメント』を参照してください。

ソース・ファイル udfsrv.c からユーザー定義関数プログラム udfsrv を構築するには、そのビルド・ファイル名、プログラム名、および共用ライブラリーへの入り口点になっている UDF 関数を入力します。

```
bldudf udfsrv ScalarUDF
```

スクリプト・ファイルは、 UDF を sqllib/function ディレクトリーにコピーします。

必要であれば、UDF にファイル・モードを設定してクライアント・アプリケーションから実行できるようにします。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。このプログラムには DB2 CLI バージョンと組み込み SQL バージョンがあります。

DB2 CLI udfcli プログラム は、スクリプト・ファイル bldcli を使用して、sqllib/samples/cli にあるソース・ファイル udfcli.c から作成できます。詳細については、123ページの『DB2 CLI アプリケーション』を参照してください。

組み込み SQL udfcli プログラムは、スクリプト・ファイル bldapp を使用して、 sqllib/samples/c にあるソース・ファイル udfcli.sqc から構築することができます。詳細については、129ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF を呼び出すには、次の実行可能ファイル名を入力して、サンプルの呼び出しアプリケーションを実行します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv ライブラリーから ScalarUDF 関数を呼び出します。

マルチスレッド・アプリケーション

AIX バージョン 4 上で実行する C マルチスレッド・アプリケーションは、xlc コンパイラーの代わりに xlc_r コンパイラーを、 C++ の場合は、 xlC コンパイラーの代わりに xlC_r コンパイラーを使用して、コンパイルおよびリンクする必要があります。 32 ビットのアプリケーションに AIX 4.3 以降を使用する場合は、 xlc_r7 コンパイラーまたは xlC_r7 コンパイラーを使用してください。 _r バージョン (および他のマルチスレッド・アプリケーションのフロントエンド) では、マルチスレッド用のコンパイルを定義している適当なプリプロセッサーが設定され、適当なスレッド・ライブラリー名がリンカーに付けられます。

マルチスレッド・コンパイラーのフロントエンドを使用したコンパイラーおよびリンク・フラグの設定についてのさらに詳しい情報は、 /etc/xlC.cfg (3.1 コンパイラーを使用する場合)、または /etc/ibmcxx.cfg (3.6 以降のコンパイラーを使用する場合) を参照してください。

スクリプト・ファイル bldmt は sqllib/samples/c にあり、組み込み SQL マルチスレッド・プログラムを作成するためのコマンドが含まれています。第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。パラメーター $3

はそのデータベースのユーザー ID を、 $4 はパスワードを指定します。第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名、ユーザー ID、および、パスワードは任意指定です。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldmt script file -- AIX
# Builds a C multi-threaded embedded SQL program.
# Usage:  bldmt <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2 $3 $4
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi
# Compile the program.
xlc_r $CFLAGS_64 -I$DB2PATH/include -c $1.c
# Link the program.
xlc_r $CFLAGS_64 -o $1 $1.o -L$DB2PATH/lib -ldb2
```

上記の xlc_r コンパイラーや、リンクされているユーティリティー・ファイルがないという点だけでなく、コンパイルおよびリンク・オプションも、組み込み SQL スクリプト・ファイル bldapp で使用されているものと同じです。これらのオプションについては、 129ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル thdsrver.sqc からマルチスレッド・サンプル・プログラム thdsrver を作成するには、次のように入力します。

```
bldmt thdsrver
```

結果として、実行可能ファイル thdsrver が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
thdsrver
```

IBM C Set++

この節では以下のトピックを取り上げています。

- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー
- ユーザー定義関数 (UDF)
- マルチスレッド・アプリケーション

DB2 API と組み込み SQL アプリケーション

sqllib/samples/cpp スレッド・ライブラリーにあるビルド・ファイル bldapp には、 DB2 API と組み込み SQL アプリケーションを構築するコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない DB2 API プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- AIX
# Builds a C++ application program.
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqC" ]]
```

```
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.C error-checking utility.
  xlC $CFLAGS_64 -I$DB2PATH/include -c utilemb.C
else
  # Compile the utilapi.C error-checking utility.
  xlC $CFLAGS_64 -I$DB2PATH/include -c utilapi.C
fi
# Compile the program.
xlC $CFLAGS_64 -I$DB2PATH/include -c $1.C
if [[ -f $1".sqC" ]]
then
  # Link the program with utilemb.o
  xlC $CFLAGS_64 -o $1 $1.o utilemb.o -ldb2 -L$DB2PATH/lib
else
  # Link the program with utilapi.o
  xlC $CFLAGS_64 -o $1 $1.o utilapi.o -ldb2 -L$DB2PATH/lib
fi
```

bldapp のコンパイルおよびリンク・オプション

コンパイル・オプション

xlC　IBM C Set++ コンパイラー。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c　コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。

bldapp のコンパイルおよびリンク・オプション
リンク・オプション
xlC コンパイラーをリンカーのフロントエンドとして使用します。
$CFLAGS_64 'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。
-o $1 実行可能プログラムを指定します。
-o $1 プログラム・オブジェクト・ファイルを指定します。
utilapi.o 非組み込み SQL プログラムの場合に、 API ユーティリティー・オブジェクト・ファイルを含みます。
utilemb.o 組み込み SQL プログラムの場合に、組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。
-ldb2 データベース・マネージャー・ライブラリーとリンクします。
-L$DB2PATH/lib DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。
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。

ソース・ファイル client.C から非組み込み SQL サンプル・プログラム client を作成するには、次のように入力します。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqC から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

注: 71ページの『UDF およびストアード・プロシージャーに関する C++ 考慮事項』にある、 C++ のストアード・プロシージャーの構築に関する情報を参照してください。

sqllib/samples/cpp にあるスクリプト・ファイル bldsrv には、ストアード・プロシージャーを構築するためのコマンドが含まれています。スクリプト・ファイルは、ストアード・プロシージャーを共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、共用ライブラリーへの入り口点になっているストアード・プロシージャー関数を指定します。第 3 パラメーター $3 には、接続先のデータベースの名前を指定します。ストアード・プロシージャーは、必ずデー

タベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターは必要ありません。

最初の 2 つのパラメーター (ソース・ファイル名と入り口点) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldsrv script file -- AIX
# Builds a C++ stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
  LFLAGS_64=-X64
else
  CFLAGS_64=
  LFLAGS_64=
fi
# Compile the program.
xlC $CFLAGS_64 -I$DB2PATH/include -c $1.C
# Link using export file $1.exp, creating shared library $1
makeC++SharedLib $LFLAGS_64 -p 1024 -o $1 $1.o -L$DB2PATH/lib -ldb2 -E $1.exp
# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

xlC IBM C Set++ コンパイラー。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。

bldsrv のコンパイルおよびリンク・オプション
リンク・オプション **makeC++SharedLib** 静的コンストラクターを持つストアード・プロシージャーのリンカー・スクリプト。 **$LFLAGS_64** 'BUILD_64BIT=true' がコメント解除されている場合に、 "-X64" の値を含みます。それ以外の場合は、値を含みません。 **-p 1024** 優先順位を仮に 1024 の値に設定します。 **-o $1** 出力を、共用ライブラリー・ファイルとして指定します。 **$1.o** プログラム・オブジェクト・ファイルを指定します。 **-L$DB2PATH/lib** DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。 **-ldb2** データベース・マネージャー・ライブラリーとリンクします。 **-E $1.exp** エクスポート・ファイルを指定します。エクスポート・ファイルには、ストアード・プロシージャーのリストが含まれています。 **-e $2** 共用ライブラリーに対する入り口点を指定します。 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。

ソース・ファイル spserver.sqC からサンプル・プログラム spserver を構築する場合、 sample データベースに接続しているときは、ビルド・ファイル名、プログラム名、および共用ライブラリーの入り口点になっているストアード・プロシージャー関数の名前を入力します。

```
bldsrv spserver outlanguage
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv spserver outlanguage database
```

スクリプト・ファイルは、共用ライブラリーをサーバー上の sqllib/function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、クライアント・アプリケーション spclient を構築することができます。これは、共用ライブラリー内のストアード・プロシージャーを呼び出すアプリケーションです。

spclient は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 141ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。この出力は、クライアント・アプリケーションに戻されます。

ユーザー定義関数 (UDF)

注: 71ページの『UDF およびストアード・プロシージャーに関する C++ 考慮事項』にある、 C++ UDF の構築に関する情報を参照してください。

スクリプト・ファイル bldudf は sqllib/samples/cpp にあり、 UDF を作成するためのコマンドが含まれています。 UDF に組み込み SQL ステートメントを含めることはできません。したがって、UDF プログラムを作成する際に、データベースへの接続、プログラムのプリコンパイル、およびバインドは必要ありません。

パラメーター $1 には、ソース・ファイルの名前を指定します。パラメーター $2 には、共用ライブラリーへの入り口点になっているストアード・プロシージャー関数を指定します。スクリプト・ファイルは、ソース・ファイル名 $1 を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldudf script file -- AIX
# Builds a C++ UDF library
# Usage: bldudf <prog_name>
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
  LFLAGS_64=-X64
else
  CFLAGS_64=
  LFLAGS_64=
fi
# Compile the program.
if [[ -f $1".c" ]]
then
  xlC $CFLAGS_64 -I$DB2PATH/include -c $1.c
elif [[ -f $1".C" ]]
then
  xlC $CFLAGS_64 -I$DB2PATH/include -c $1.C
fi
# Link using export file $1.exp, creating shared library $1
makeC++SharedLib $LFLAGS_64 -p 1024  -o $1 $1.o -L$DB2PATH/lib -ldb2 -ldb2apie -E $1.exp
# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldudf のコンパイルおよびリンク・オプション
コンパイル・オプション **xlC** IBM C Set++ コンパイラー。 **$CFLAGS_64** 'BUILD_64BIT=true' がコメント解除されている場合に、 "-q64" の値を含みます。それ以外の場合は、値を含みません。 **-I$DB2PATH/include** DB2 インクルード・ファイルのロケーションを指定します。たとえば、 $HOME/sqllib/include。 **-c** コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

bldudf のコンパイルおよびリンク・オプション

リンク・オプション

makeC++SharedLib
静的コンストラクターを持つストアード・プロシージャーのリンカー・スクリプト。

$LFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 "-X64" の値を含みます。それ以外の場合は、値を含みません。

-p 1024
優先順位を仮に 1024 の値に設定します。

-o $1 出力を、共用ライブラリー・ファイルとして指定します。

$1.o プログラム・オブジェクト・ファイルを指定します。

-L$DB2PATH/lib
DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。

-ldb2 データベース・マネージャー・ライブラリーとリンクします。

-ldb2apie
DB2 API エンジン・ライブラリーとリンクして、LOB ロケーターを使用できるようにします。

-E $1.exp
エクスポート・ファイルを指定します。エクスポート・ファイルには、ストアード・プロシージャーのリストが含まれています。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。
UDF の作成に関する詳細については、 121ページの『UDF および CREATE FUNCTION ステートメント』を参照してください。

ソース・ファイル udfsrv.c からユーザー定義関数プログラム udfsrv を構築するには、そのビルド・ファイル名、プログラム名、および共用ライブラリーへの入り口点になっている UDF 関数を入力します。

```
bldudf udfsrv ScalarUDF
```

スクリプト・ファイルは、サーバー上の sqllib/function というパスに UDF をコピーします。

必要であれば、UDF にファイル・モードを設定して DB2 インスタンスがそれを実行できるようにしてください。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。 udfcli プログラムは、スクリプト・ファイル bldapp を使用して、 sqllib/samples/cpp にあるソース・ファイル udfcli.sqC から作成します。詳細については、141ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF を呼び出すには、次の実行可能ファイル名を入力して、サンプルの呼び出しアプリケーションを実行します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv ライブラリーの ScalarUDF 関数を呼び出します。

マルチスレッド・アプリケーション

AIX バージョン 4 上で実行する C++ マルチスレッド・アプリケーションは、xlc コンパイラーの代わりに xlc_r コンパイラーを、 C の場合は、 xlC コンパイラーの代わりに xlC_r コンパイラーを使用して、コンパイルおよびリンクする必要があります。 32 ビットのアプリケーションに AIX 4.3 以降を使用する場合は、 xlC_r7 コンパイラーまたは xlc_r7 コンパイラーを使用してください。 _r バージョン (および他のマルチスレッド・アプリケーションのフロントエンド) では、マルチスレッド用のコンパイルを定義している適当なプリプロセッサーが設定され、適当なスレッド・ライブラリー名がリンカーに付けられます。

マルチスレッド・コンパイラーのフロントエンドを使用したコンパイラーおよびリンク・フラグの設定についてのさらに詳しい情報は、 /etc/xlC.cfg (3.1 コンパイラーを使用する場合)、または /etc/ibmcxx.cfg (3.6 以降のコンパイラーを使用する場合) を参照してください。

スクリプト・ファイル bldmt は sqllib/samples/cpp にあり、組み込み SQL マルチスレッド・プログラムを作成するためのコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。パラメーター $3 はそのデータベースのユーザー ID を、 $4 はパスワードを指定します。第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名、ユーザー ID、および、パスワードは任意指定です。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldmt script file -- AIX
# Builds a C++ multi-threaded embedded SQL program
```

```
# Usage:  bldmt <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2 $3 $4
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi
# Compile the program.
xlC_r $CFLAGS_64 -I$DB2PATH/include -c $1.C
# Link the program.
xlC_r $CFLAGS_64 -o $1 $1.o -L$DB2PATH/lib -ldb2
```

上記の xlC_r コンパイラーや、リンクされているユーティリティー・ファイルがないという点だけでなく、コンパイルおよびリンク・オプションも、組み込み SQL スクリプト・ファイル bldapp で使用されているものと同じです。これらのオプションについては、 141ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル thdsrver.sqC からマルチスレッド・サンプル・プログラム thdsrver を作成するには、次のように入力します。

```
bldmt thdsrver
```

結果として、実行可能ファイル thdsrver が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
thdsrver
```

VisualAge C++ バージョン 4.0

この VisualAge C++ コンパイラーは、AIX、OS/2、および Windows 32 ビット・オペレーティング・システム用です。この節の内容は、これらすべてのプラットフォームに適用されます。

VisualAge C++ コンパイラーは、本書で紹介される他のコンパイラーとは異なります。 VisualAge C++ バージョン 4.0 を使用してプログラムをコンパイルするには、まず構成ファイルを作成する必要があります。このことについてさらに知りたい場合は、コンパイラーに付属する資料を参照してください。

DB2 は、 VisualAge C++ コンパイラーで作成できるさまざまなタイプの DB2 プログラム用の構成ファイルを提供します。 DB2 構成ファイルを使用するためには、まず、コンパイルするプログラム名に合わせて環境変数を設定します。次に、VisualAge C++ が提供しているコマンドを使用して、プログラムをコンパイルします。 DB2 が提供する構成ファイル、およびプログラムをコンパイルする際の使用方法について説明している節は、以下のとおりです。

cli.icc
DB2 CLI 構成ファイル。詳細については、『DB2 CLI アプリケーション』を参照してください。

cliapi.icc
DB2 API を使用する DB2 CLI の構成ファイル。詳細については、156 ページの『DB2 API を使用する DB2 CLI アプリケーション』を参照してください。

clis.icc
DB2 CLI ストアード・プロシージャー構成ファイル。詳細については、157ページの『DB2 CLI ストアード・プロシージャー』を参照してください。

api.icc
DB2 API 構成ファイル。詳細については、160ページの『DB2 API アプリケーション』を参照してください。

emb.icc
組み込み SQL 構成ファイル。詳細については、162ページの『組み込み SQL アプリケーション』を参照してください。

stp.icc
組み込み SQL ストアード・プロシージャー構成ファイル。詳細については、164ページの『組み込み SQL ストアード・プロシージャー』を参照してください。

udf.icc
ユーザー定義関数構成ファイル。詳細については、 167ページの『ユーザー定義関数 (UDF)』を参照してください。

DB2 CLI アプリケーション

DB2 CLI プログラムは、`sqllib/samples/cli` (AIX の場合)、および `%DB2PATH%¥samples¥cli` (OS/2 および Windows 32 ビット・オペレーティング・システムの場合) にある構成ファイル `cli.icc` を使用して構築することができます。

```
// cli.icc configuration file for DB2 CLI applications
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export CLI=prog_name'
// To use on OS/2 and Windows, enter: 'set CLI=prog_name'
// Then compile the program by entering: 'vacbld cli.icc'
if defined( $CLI )
{
  prog_name = $CLI
}
else
{
  error "Environment Variable CLI is not defined."
}
infile   = prog_name".c"
utilcli = "utilcli.c"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".exe"
  group lib   = "db2cli.lib"
  option opts = link( libsearchpath, db2path"¥¥lib" ),
                incl( searchPath, db2path"¥¥include" )
}
option opts

{
  target type(exe) outfile
  {
    source infile
    source utilcli
    source lib
  }
}
```

VisualAge C++ バージョン 4.0 は、インストール先のオペレーティング・システムに応じて、 __TOS_AIX__、__TOS_OS2__、 __TOS_WIN__ のいずれかの環境変数を定義します。

構成ファイルを使用して、ソース・ファイル tbinfo.c から DB2 CLI サンプル・プログラム tbinfo を構築するには、以下のようにします。

1. 次のように入力して、CLI 環境変数をプログラム名に設定します。

```
export CLI=tbinfo
```

2. cli.icc ファイルを使用して異なるプログラムを作成することによって生成された cli.ics ファイルが作業ディレクトリーにある場合は、次のコマンドで cli.ics ファイルを削除してください。

   ```
   rm cli.ics
   ```

   既存の cli.ics ファイルが、再構築するその同じプログラム用に生成されているのであれば、削除する必要はありません。

3. サンプル・プログラムを以下のように入力してコンパイルします。

   ```
   vacbld cli.icc
   ```

   注: vacbld コマンドは、 VisualAge C++ バージョン 4.0 で提供されます。

結果として、実行可能ファイル tbinfo が作成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
tbinfo
```

組み込み SQL アプリケーションの構築および実行

cli.icc 構成ファイルを使用して、ファイル embprep (AIX)、embprep.cmd (OS/2)、または embprep.bat (Windows 32 ビット・オペレーティング・システム) でプリコンパイルされた組み込み SQL プログラムをコンパイルすることができます。このファイルは、ソース・ファイルをプリコンパイルし、プログラムをデータベースにバインドします。

ソース・ファイル dbusemx.sqc から組み込み SQL アプリケーション dbusemx をプリコンパイルする方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   embprep dbusemx
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   embprep dbusemx database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   embprep dbusemx database userid password
   ```

結果として、プリコンパイルされた C ファイル dbusemx.c が作成されます。

プリコンパイルした後、この C ファイルは、次のようにして cli.icc ファイルでコンパイルすることができます。

1. 次のように入力して、CLI 環境変数をプログラム名に設定します。

   ```
   export CLI=dbusemx
   ```

2. cli.icc ファイルを使用して異なるプログラムを作成することによって生成された cli.ics ファイルが作業ディレクトリーにある場合は、次のコマンドで cli.ics ファイルを削除してください。

   ```
   rm cli.ics
   ```

   既存の cli.ics ファイルが、再構築するその同じプログラム用に生成されているのであれば、削除する必要はありません。

3. サンプル・プログラムを以下のように入力してコンパイルします。

   ```
   vacbld cli.icc
   ```

   注: vacbld コマンドは、 VisualAge C++ バージョン 4.0 で提供されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   dbusemx
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   dbusemx database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   dbusemx database userid password
   ```

DB2 API を使用する DB2 CLI アプリケーション

DB2 には、CLI サンプル・プログラムが含まれています。このサンプル・プログラムは、DB2 API を使用してデータベースを作成およびドロップし、 CLI 機能を複数のデータベースで使用する方法を示します。 DB2 API を使用するサンプルは、 27ページの表7 にある CLI サンプル・プログラムの説明の中に示されています。 DB2 API を使用する DB2 CLI プログラムは、sqllib/samples/cli (AIX の場合)、および %DB2PATH%¥samples¥cli (OS/2 お

よび Windows 32 ビット・オペレーティング・システムの場合) にある構成ファイル cliapi.icc を使用して構築することができます。

このファイルは、データベースを作成およびドロップするための DB2 API が入った utilapi ユーティリティー・ファイルでコンパイルおよびリンクします。この点が、このファイルと cli.icc 構成ファイルの唯一の違いです。

ソース・ファイル dbmconn.c から DB2 CLI サンプル・プログラム dbmconn を構築するには、次のようにします。

1. 次のように入力して、CLIAPI 環境変数をプログラム名に設定します。

   ```
   export CLIAPI=dbmconn
   ```

2. cliapi.icc ファイルを使用して異なるプログラムを構築することによって生成された cliapi.ics ファイルが作業ディレクトリー内にある場合は、次のコマンドを使用して cliapi.ics ファイルを削除します。

   ```
   rm cliapi.ics
   ```

   既存の cliapi.ics ファイルが、再構築するその同じプログラム用に生成されているのであれば、削除する必要はありません。

3. サンプル・プログラムを以下のように入力してコンパイルします。

   ```
   vacbld cliapi.icc
   ```

   注: vacbld コマンドは、 VisualAge C++ バージョン 4.0 で提供されます。

結果として、実行可能ファイル dbmconn が生成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
dbmconn
```

DB2 CLI ストアード・プロシージャー

DB2 CLI ストアード・プロシージャーは、sqllib/samples/cli (AIX の場合)、および %DB2PATH%¥samples¥cli (OS/2 および Windows 32 ビット・オペレーティング・システムの場合) にある構成ファイル clis.icc を使用して構築することができます。

```
// clis.icc configuration file for DB2 CLI stored procedures
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export CLIS=prog_name'
// To use on OS/2 and Windows, enter: 'set CLIS=prog_name'
// Then compile the program by entering: 'vacbld clis.icc'
if defined( $CLIS )
{
  prog_name = $CLIS
}
```

```
else
{
  error "Environment Variable CLIS is not defined."
}
infile   = prog_name".c"
utilcli = "utilcli.c"
expfile = prog_name".exp"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
  cpcmd       = "cp"
  funcdir     = db2path"/function"
}
else /* if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ ) */
{
  db2path     = $DB2PATH
  outfile     = prog_name".dll"
  if defined( $__TOS_WIN__ )
  {
    expfile = prog_name"v4.exp"
  }
  group lib   = "db2cli.lib"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"¥¥lib" ),
                incl( searchPath, db2path"¥¥include" )
  cpcmd       = "copy"
  funcdir     = db2path"¥¥function"
}
option opts

{
  target type(dll) outfile
  {
    source infile
    source utilcli
    source lib
  }
}

if defined( $__TOS_AIX__ )
{
  rmcmd       = "rm -f"
  run after rmcmd " " funcdir "/" outfile
}
run after cpcmd " " outfile " " funcdir
```

VisualAge C++ バージョン 4.0 は、インストール先のオペレーティング・システムに応じて、 __TOS_AIX__、__TOS_OS2__、 __TOS_WIN__ のいずれかの環境変数を定義します。

構成ファイルを使用して、ソース・ファイル spserver.c から DB2 CLI ストアード・プロシージャー spserver を作成するには、以下のようにします。

1. 次のように入力して、CLIS 環境変数をプログラム名に設定します。

   ```
   export CLIS=spserver
   ```

2. clis.icc ファイルを使用して異なるプログラムを作成することによって生成された clis.ics ファイルが作業ディレクトリーにある場合は、次のコマンドで clis.ics ファイルを削除してください。

   ```
   rm clis.ics
   ```

   既存の clis.ics ファイルが、再構築するその同じプログラム用に生成されているのであれば、削除する必要はありません。

3. サンプル・プログラムを以下のように入力してコンパイルします。

   ```
   vacbld clis.icc
   ```

   注: vacbld コマンドは、 VisualAge C++ バージョン 4.0 で提供されます。

ストアード・プロシージャーは、サーバー上の sqllib/function (AIX) および %DB2PATH%¥function (OS/2 および Windows 32 ビット・オペレーティング・システム) というパスにコピーされます。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースがあるインスタンスのユーザー ID とパスワードを使用して、データベースに接続します。

```
db2 connect to sample userid password
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

ストアード・プロシージャー spserver を作成したなら、そのストアード・プロシージャーを呼び出す CLI クライアント・アプリケーション spclient を構築できます。 spclient は、構成ファイル cli.icc を使用して構築することができます。詳細については、153ページの『DB2 CLI アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample またはそのリモート別名、あるいはその他の名前にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

DB2 API アプリケーション

DB2 API プログラムは、sqllib/samples/c および sqllib/samples/cpp (AIX)、そして %DB2PATH%¥samples¥c および %DB2PATH%¥samples¥c (OS/2 および Windows 32 ビット・オペレーティング・システム) にある構成ファイル api.icc を使用して、 C または C++ で構築することができます。

```
// api.icc configuration file for DB2 API programs
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export API=prog_name'
// To use on OS/2 and Windows, enter: 'set API=prog_name'
// Then compile the program by entering: 'vacbld api.icc'
if defined( $API )
{
  prog_name = $API
}
```

```
else
{
  error "Environment Variable API is not defined."
}
infile   = prog_name".c"
util   = "utilapi.c"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".exe"
  group lib   = "db2api.lib"
  option opts = link( libsearchpath, db2path"¥¥lib" ),
                incl( searchPath, db2path"¥¥include" )
}
option opts

{
  target type(exe) outfile
  {
    source infile
    source util
    source lib
  }
}
```

VisualAge C++ バージョン 4.0 は、インストール先のオペレーティング・システムに応じて、 __TOS_AIX__、__TOS_OS2__、 __TOS_WIN__ のいずれかの環境変数を定義します。

構成ファイルを使用して、ソース・ファイル client.c から DB2 API サンプル・プログラム client を作成するには、以下のようにします。

1. 次のように入力して、API 環境変数をプログラム名に設定します。

   ```
   export API=client
   ```

2. api.icc ファイルを使用して異なるプログラムを作成することによって生成された api.ics ファイルが作業ディレクトリーにある場合は、次のコマンドで api.ics ファイルを削除してください。

   ```
   rm api.ics
   ```

既存の api.ics ファイルが、再構築するその同じプログラム用に生成されているのであれば、削除する必要はありません。

3. サンプル・プログラムを以下のように入力してコンパイルします。

```
vacbld api.icc
```

注: vacbld コマンドは、 VisualAge C++ バージョン 4.0 で提供されます。

結果として、実行可能ファイル client が作成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
client
```

組み込み SQL アプリケーション

DB2 組み込み SQL アプリケーションは、 sqllib/samples/c および sqllib/samples/cpp (AIX)、そして %DB2PATH%¥samples¥c および %DB2PATH%¥samples¥cpp (OS/2 および Windows 32 ビット・オペレーティング・システム) にある構成ファイル emb.icc を使用して、 C および C++ で構築することができます。

```
// emb.icc configuration file for embedded SQL applications
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export EMB=prog_name'
// To use on OS/2 and Windows, enter: 'set EMB=prog_name'
// Then compile the program by entering: 'vacbld emb.icc'
if defined( $EMB )
{
  prog_name = $EMB
}
else
{
  error "Environment Variable EMB is not defined."
}
// To connect to another database, replace "sample"
// For user ID and password, update 'user' and 'passwd'
// and take out the comment in the line: 'run before "embprep "'
dbname = "sample"
user   = ""
passwd = ""
// Precompiling the source program file
run before "embprep " prog_name " " dbname // " " user " " passwd
infile   = prog_name".c"
util = "utilemb.sqc"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
```

```
  group lib   = "libdb2.a"
  option opts = link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".exe"
  group lib   = "db2api.lib"
  option opts = link( libsearchpath, db2path"¥¥lib" ),
                incl( searchPath, db2path"¥¥include" )
}
option opts

{
  target type(exe) outfile
  {
    source infile
    source util
    source lib
  }
}
```

VisualAge C++ バージョン 4.0 は、インストール先のオペレーティング・システムに応じて、 __TOS_AIX__、__TOS_OS2__、 __TOS_WIN__ のいずれかの環境変数を定義します。

構成ファイルを使用してソース・ファイル updat.sqc から組み込み SQL アプリケーション updat を構築するには、次のようにします。

1. 次のように入力して、EMB 環境変数をプログラム名に設定します。

   ```
   export EMB=updat
   ```

2. emb.icc ファイルを使用して異なるプログラムを作成することによって生成された emb.ics ファイルが作業ディレクトリーにある場合は、次のコマンドで emb.ics ファイルを削除してください。

   ```
   rm emb.ics
   ```

   既存の emb.ics ファイルが、再構築するその同じプログラム用に生成されているのであれば、削除する必要はありません。

3. サンプル・プログラムを以下のように入力してコンパイルします。

   ```
   vacbld emb.icc
   ```

   注: vacbld コマンドは、 VisualAge C++ バージョン 4.0 で提供されます。

結果として、実行可能ファイル updat が作成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
updat
```

組み込み SQL ストアード・プロシージャー

DB2 組み込み SQL ストアード・プロシージャーは、 sqllib/samples/c および sqllib/samples/cpp (AIX)、そして %DB2PATH%¥samples¥c および %DB2PATH%¥samples¥cpp (OS/2 および Windows 32 ビット・オペレーティング・システム) にある構成ファイル stp.icc を使用して、 C および C++ で構築することができます。

```
// stp.icc configuration file for embedded SQL stored procedures
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export STP=prog_name'
// To use on OS/2 and Windows, enter: 'set STP=prog_name'
// Then compile the program by entering: 'vacbld emb.icc'
if defined( $STP )
{
  prog_name = $STP
}
else
{
  error "Environment Variable STP is not defined."
}
// To connect to another database, replace "sample"
// For user ID and password, update 'user' and 'passwd'
// and take out the comment in the line: 'run before "embprep "'
dbname = "sample"
user   = ""
passwd = ""
// Precompiling the source program file
run before "embprep " prog_name " " dbname // " " user " " passwd
infile   = prog_name".c"
expfile = prog_name".exp"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
  cpcmd       = "cp"
  funcdir     = db2path"/function"
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".dll"
  if defined( $__TOS_WIN__ )
  {
    expfile = prog_name"v4.exp"
```

```
  }
  group lib   = "db2api.lib"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"¥¥lib" ),
                incl( searchPath, db2path"¥¥include" )
  cpcmd       = "copy"
  funcdir     = db2path"¥¥function"
}
option opts

{
  target type(dll) outfile
  {
    source infile
    source lib
  }
}

if defined( $__TOS_AIX__ )
{
  rmcmd       = "rm -f"
  run after rmcmd " " funcdir "/" outfile
}
run after cpcmd " " outfile " " funcdir
```

VisualAge C++ バージョン 4.0 は、インストール先のオペレーティング・システムに応じて、 __TOS_AIX__、__TOS_OS2__、 __TOS_WIN__ のいずれかの環境変数を定義します。

構成ファイルを使用して、ソース・ファイル spserver.sqc から組み込み SQL ストアード・プロシージャー spserver を作成するには、以下のようにします。

1. 次のように入力して、STP 環境変数をプログラム名に設定します。

   ```
   export STP=spserver
   ```

2. stp.icc を使用して異なるプログラムを作成することによって生成された stp.ics ファイルが作業ディレクトリーにある場合は、次のコマンドで stp.ics ファイルを削除してください。

   ```
   rm stp.ics
   ```

   既存の stp.ics ファイルが、再構築するその同じプログラム用に生成されているのであれば、削除する必要はありません。

3. サンプル・プログラムを以下のように入力してコンパイルします。

   ```
   vacbld stp.icc
   ```

   注: vacbld コマンドは、 VisualAge C++ バージョン 4.0 で提供されます。

ストアード・プロシージャーは、サーバー上の sqllib/function (AIX) および %DB2PATH%¥function (OS/2 および Windows 32 ビット・オペレーティング・システム) というパスにコピーされます。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

ストアード・プロシージャー spserver を作成したなら、そのストアード・プロシージャーを呼び出すクライアント・アプリケーション spclient を構築できます。 spclient は、構成ファイル emb.icc を使用して構築することができます。詳細については、162ページの『組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample またはそのリモート別名、あるいはその他の名前にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

ユーザー定義関数 (UDF)

ユーザー定義関数は、 sqllib/samples/c および sqllib/samples/cpp (AIX)、そして %DB2PATH%¥samples¥c および %DB2PATH%¥samples¥cpp (OS/2 および Windows 32 ビット・オペレーティング・システム) にある構成ファイル udf.icc を使用して、 C および C++ で構築することができます。

```
// udf.icc configuration file for user-defined functions
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export UDF=prog_name'
// To use on OS/2 and Windows, enter: 'set UDF=prog_name'
// Then compile the program by entering: 'vacbld udf.icc'
if defined( $UDF )
{
  prog_name = $UDF
}
else
{
  error "Environment Variable UDF is not defined."
}
infile   = prog_name".c"
expfile = prog_name".exp"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a", "libdb2apie.a"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
  cpcmd       = "cp"
  funcdir     = db2path"/function"
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".dll"
  if defined( $__TOS_WIN__ )
  {
    expfile = prog_name"v4.exp"
  }
  group lib   = "db2api.lib", "db2apie.lib"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"¥¥lib" ),
                incl( searchPath, db2path"¥¥include" )
  cpcmd       = "copy"
```

```
  funcdir     = db2path"¥¥function"
}
option opts

{
  target type(dll) outfile
  {
    source infile
    source lib
  }
}

if defined( $__TOS_AIX__ )
{
  rmcmd       = "rm -f"
  run after rmcmd " " funcdir "/" outfile
}
run after cpcmd " " outfile " " funcdir
```

VisualAge C++ バージョン 4.0 は、インストール先のオペレーティング・システムに応じて、 __TOS_AIX__、__TOS_OS2__、 __TOS_WIN__ のいずれかの環境変数を定義します。

構成ファイルを使用して、ソース・ファイル udf.c からユーザー定義関数プログラム udfsrv を作成するには、以下のようにします。

1. 次のように入力して、UDF 環境変数をプログラム名に設定します。

   ```
   export UDF=udfsrv
   ```

2. udf.icc ファイルを使用して異なるプログラムを作成することによって生成された udf.ics ファイルが作業ディレクトリーにある場合は、次のコマンドで udf.ics ファイルを削除してください。

   ```
   rm udf.ics
   ```

   既存の udf.ics ファイルが、再構築するその同じプログラム用に生成されているのであれば、削除する必要はありません。

3. サンプル・プログラムを以下のように入力してコンパイルします。

   ```
   vacbld udf.icc
   ```

   注: vacbld コマンドは、 VisualAge C++ バージョン 4.0 で提供されます。

UDF ライブラリーは、サーバー上の sqllib/function というパスにコピーされます。

必要であれば、ユーザー定義関数にファイル・モードを設定して DB2 インスタンスがそれを実行できるようにしてください。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。このプログラムには DB2 CLI バージョンと組み込み SQL バージョンがあります。

DB2 CLI udfcli プログラムは、構成ファイル cli.icc を使用して、sqllib/samples/cli (AIX)、および %DB2PATH%¥samples¥cli (OS/2 および Windows 32 ビット・オペレーティング・システム) にあるソース・ファイル udfcli.c から構築することができます。詳細については、153ページの『DB2 CLI アプリケーション』を参照してください。

組み込み SQL udfcli プログラムは、構成ファイル emb.icc を使用して、sqllib/samples/c (AIX)、および %DB2PATH%¥samples¥cli (OS/2 および Windows 32 ビット・オペレーティング・システム) にあるソース・ファイル udfcli.sqc から構築することができます。詳細については、162ページの『組み込み SQL アプリケーション』を参照してください。

UDF を呼び出すには、次の実行可能ファイル名を入力して、サンプルの呼び出しアプリケーションを実行します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv ライブラリーから ScalarUDF 関数を呼び出します。

IBM COBOL Set for AIX

この節では以下のトピックを取り上げています。

- コンパイラーの使用法
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー

コンパイラーの使用法

組み込み SQL および DB2 API 呼び出しを含むアプリケーションを開発しており、 IBM COBOL Set for AIX コンパイラーを使用している場合には、以下の点に留意してください。

- コマンド行プロセッサーのコマンド db2 prep を使ってアプリケーションをプリコンパイルする場合は、 target ibmcob オプションを使ってください。
- ソース・ファイルの中でタブ文字を使用しないでください。
- コンパイル・オプションを設定するためには、ソース・ファイルの 1 行目で PROCESS および CBL キーワードを使うことができます。

- アプリケーションに組み込み SQL のみが含まれていて、 DB2 API 呼び出しは含まれない場合には、 pgmname(mixed) コンパイル・オプションを使う必要はありません。 DB2 API 呼び出しを使用する場合には、 pgmname(mixed) コンパイル・オプションを使う必要があります。
- IBM COBOL Set for AIX コンパイラーの「システム/390 ホスト・データ型サポート」機能を使用している場合、アプリケーション用の DB2 組み込みファイルは、次のディレクトリー中にあります。

  ```
  $HOME/sqllib/include/cobol_i
  ```

  提供されたスクリプト・ファイルを使って DB2 サンプル・プログラムを作成している場合、スクリプト・ファイルで指定された組み込みファイルのパスは、 cobol_a ディレクトリーではなく、 cobol_i ディレクトリーを指すように変更しなければなりません。

  IBM COBOL Set for AIX コンパイラーの「システム/390 ホスト・データ型サポート」機能を使用していない場合、またはこのコンパイラーのそれよりも前のバージョンを使用している場合、アプリケーション用の DB2 組み込みファイルは、次のディレクトリー中にあります。

  ```
  $HOME/sqllib/include/cobol_a
  ```

  .cbl 拡張子を含めるには、次のようにして COPY ファイル名を指定します。

  ```
  COPY "sql.cbl".
  ```

DB2 API と組み込み SQL アプリケーション

sqllib/samples/cobol にあるビルド・ファイル bldapp には、 DB2 アプリケーション・プログラムを構築するコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を含まないプログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザ

ー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- AIX
# Builds an IBM COBOL application program
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqb" ]]
then
  embprep $1 $2 $3 $4
fi
# Compile the checkerr.cbl error checking utility.
cob2 -qpgmname¥(mixed¥) -qlib -I$DB2PATH/include/cobol_a \
     -c checkerr.cbl
# Compile the program.
cob2 -qpgmname¥(mixed¥) -qlib -I$DB2PATH/include/cobol_a \
     -c $1.cbl
# Link the program.
cob2 -o $1 $1.o checkerr.o -ldb2 -L$DB2PATH/lib
```

bldapp のコンパイルおよびリンク・オプション

コンパイル・オプション

cob2 IBM COBOL Set コンパイラー。

-qpgmname¥(mixed¥)
コンパイラーに、大文字小文字混合の名前を持つライブラリー入り口点の CALL を許可するように指示します。

-qlib コンパイラーに COPY ステートメントを処理するように指示します。

-I$DB2PATH/include/cobol_a
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include/cobol_a。

-c コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。

bldapp のコンパイルおよびリンク・オプション
リンク・オプション
cob2 コンパイラーをリンカーのフロントエンドとして使用します。
-o $1 実行可能プログラムを指定します。
$1.o プログラム・オブジェクト・ファイルを指定します。
checkerr.o エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。
-ldb2 データベース・マネージャー・ライブラリーとリンクします。
-L$DB2PATH/lib DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。
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。

ソース・ファイル client.cbl から非組み込み SQL サンプル・プログラム client を作成するには、次のように入力します。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqb から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

sqllib/samples/cobol にあるスクリプト・ファイル bldsrv には、ストアード・プロシージャーを構築するためのコマンドが含まれています。スクリプト・ファイルは、ストアード・プロシージャーを共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

スクリプト・ファイルは、ソース・ファイル名 $1 を共用ライブラリー名として、また共用ライブラリーへの入り口点として使います。構築しようとしているストアード・プロシージャーの入り口点関数名の名前がソース・ファイルの名前とは異なる場合、入り口点の別のパラメーターを受け入れるようにスクリプト・ファイルを変更することができます。データベース・パラメーターの名前を、 $3 に変更するようお勧めします。それから、入り口点のリンク・オプションを -e $2 に変更し、スクリプト・ファイルの実行時にコマンド行で追加パラメーターを指定します。

```
#! /bin/ksh
# bldsrv script file -- AIX
# Builds an IBM COBOL stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2
# Compile the checkerr.cbl error checking utility.
cob2 -qpgmname¥(mixed¥) -qlib -I$DB2PATH/include/cobol_a \
     -c checkerr.cbl
# Compile the program.
cob2 -qpgmname¥(mixed¥) -qlib -c -I$DB2PATH/include/cobol_a $1.cbl
# Link the program using the export file $1.exp
# creating shared library $1 with entry point $1.
cob2 -o $1 $1.o checkerr.o -H512 -T512 -e $1 -bE:$1.exp \
     -L$DB2PATH/lib -ldb2
# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

cob2 IBM COBOL Set コンパイラー。

-qpgmname¥(mixed¥)
コンパイラーに、大文字小文字混合の名前を持つライブラリー入り口点の CALL を許可するように指示します。

-qlib コンパイラーに COPY ステートメントを処理するように指示します。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

-I$DB2PATH/include/cobol_a
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include/cobol_a。

bldsrv のコンパイルおよびリンク・オプション
リンク・オプション
cob2 リンク編集にコンパイラーを使用します。
-o $1 出力を、共用ライブラリー・ファイルとして指定します。
$1.o ストアード・プロシージャー・オブジェクト・ファイルを指定します。
checkerr.o エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。
-H512 出力ファイル位置合わせを指定します。
-T512 出力ファイル・テキスト・セグメントの開始アドレスを指定します。
-e $1 共用ライブラリーに対するデフォルト入り口点を指定します。
-bE:$1.exp エクスポート・ファイルを指定します。エクスポート・ファイルには、ストアード・プロシージャーのリストが含まれています。
-L$DB2PATH/lib DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。
-ldb2 データベース・マネージャー・ライブラリーとリンクします。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

サンプル・データベースに接続している場合に、ソース・ファイル outsrv.sqb からサンプル・プログラム outsrv を構築するには、次のように入力します。

```
bldsrv outsrv
```

他のデータベースに接続しているときは、さらにデータベース名も含めます。

```
bldsrv outsrv database
```

このスクリプト・ファイルは、ストアード・プロシージャーをサーバー上の sqllib/function というパスにコピーします。

必要であれば、ストアード・プロシージャーにファイル・モードを設定して、クライアント・アプリケーションからアクセスできるようにします。

ストアード・プロシージャー outsrv を構築してしまえば、そのストアード・プロシージャーを呼び出すクライアント・アプリケーション outcli を構築で

きます。 outcli は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 170ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

```
outcli database userid password
```

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample またはそのリモート別名、あるいはその他の名前にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー outsrv にアクセスし、ストアード・プロシージャー関数をサーバー・データベース上で実行します。この出力は、クライアント・アプリケーションに戻されます。

Micro Focus COBOL

この節では以下のトピックを取り上げています。

- コンパイラーの使用法
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー

コンパイラーの使用法

組み込み SQL および DB2 API 呼び出しを含むアプリケーションを開発しており、 Micro Focus COBOL コンパイラーを使用している場合には、以下の点に留意してください。

- コマンド行プロセッサーのコマンド db2 prep を使ってアプリケーションをプリコンパイルする場合は、 target mfcob オプション (デフォルト) を使ってください。
- 組み込みプリコンパイラー・フロントエンド、実行時解釈プログラム、または Animator デバッガーを使うためには、次のように Micro Focus が提供する mkrts コマンドを実行して、 DB2 Generic API 入り口点を Micro Focus 実行時モジュール rts32 に追加します。

1. root としてログインします。
2. 次のディレクトリーにある引き数を指定して、mkrts を実行します。

   ```
   /usr/lpp/db2_06_01/lib/db2mkrts.args
   ```

- DB2 COBOL COPY ファイル・ディレクトリーを、 Micro Focus COBOL 環境変数 COBCPY に含める必要があります。 COBCPY 環境変数は、COPY ファイルのロケーションを指定します。 Micro Focus COBOL 用の DB2 COPY ファイルは、データベース・インスタンス・ディレクトリーの下にある sqllib/include/cobol_mf にあります。

  このディレクトリーを組み込むには、次のように入力します。

  ```
  export COBCPY=$COBCPY:$HOME/sqllib/include/cobol_mf
  ```

  注: COBCPY を .profile ファイル中に設定することもできます。

DB2 API と組み込み SQL アプリケーション

sqllib/samples/cobol_mf にあるビルド・ファイル bldapp には、 DB2 アプリケーション・プログラムを構築するコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を含まないプログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- AIX
# Builds a Micro Focus COBOL application program
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqb" ]]
then
  embprep $1 $2 $3 $4
fi
# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$DB2PATH/include/cobol_mf:$COBCPY
# Compile the checkerr.cbl error checking utility.
cob -c -x checkerr.cbl
# Compile the program.
cob -c -x $1.cbl
# Link the program.
cob -x -o $1 $1.o checkerr.o -ldb2 -ldb2gmf -L$DB2PATH/lib
```

bldmfapi のコンパイルおよびリンク・オプション

コンパイル・オプション

cob COBOL コンパイラー。

-c コンパイルのみを実行し、リンクは実行しません。

-x 実行可能プログラムを作成します。

リンク・オプション

cob コンパイラーをリンカーのフロントエンドとして使用します。

-x 実行可能プログラムを作成します。

-o $1 実行可能プログラムを指定します。

$1.o プログラム・オブジェクト・ファイルを指定します。

-ldb2 DB2 ライブラリーとリンクします。

-ldb2gmf
Micro Focus COBOL 用 DB2 例外ハンドラー・ライブラリーとリンクします。

-L$DB2PATH/lib
DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。

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。

ソース・ファイル client.cbl から非組み込み SQL サンプル・プログラム client を作成するには、次のように入力します。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqb から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

注:

1. Micro Focus 4.1 コンパイラーを使用して AIX 4.3 でストアード・プロシージャーを構築する前に、次のコマンドを実行してください。

   ```
   db2stop
   db2set DB2LIBPATH=$LIBPATH
   db2set DB2ENVLIST="COBDIR LIBPATH"
   db2set
   db2start
   ```

   db2stop がデータベースを停止し、 LIBPATH がシェル環境に正しく設定されていることを確認してください。最後の db2set コマンドは、設定を表示させるコマンドです。 DB2LIBPATH と DB2ENVLIST が正しく設定されていることを確認してください。

2. AIX バージョン 4 プラットフォーム上で使用される Micro Focus COBOL コンパイラーの最近のバージョンの中には、静的にリンクされたストアード・プロシージャーを作成するのに使えないものもあります。 makefile とスクリプト・ファイルの bldsrv 自体は、動的にリンクされたストアード・プロシージャーを作成できるように変更されています。

   この動的にリンクされたストアード・プロシージャーをリモート・クライアント・アプリケーションが正常に呼び出すためには、ストアード・プロシージャーが実行される直前にそのストアード・プロシージャーが常駐するサーバーで呼び出すために Micro Focus COBOL ルーチンの cobinit() が必要です。 makefile またはスクリプト・ファイル bldsrv の実行中に、これを成し遂げるラッパー・プログラムが作成されます。次に、ラッパー・プログラムはストアード・プロシージャー・コードとリンクしてストアード・プロシージャーの共用ライブラリーを形成します。このラッパー・プログラムを使用するために、クライアント・アプリケーションが x という名前のストアード・プロシージャーを呼び出すには、 x の代わりに x_wrap を呼び出さなければなりません。

   ラッパー・プログラムの詳細については、この節で後述します。

sqllib/samples/cobol_mf にあるスクリプト・ファイル bldsrv には、ストアード・プロシージャーを構築するためのコマンドが含まれています。スクリプト・ファイルは、ストアード・プロシージャーを共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。ストアー

ド・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

スクリプト・ファイルは、ソース・ファイル名 $1 を共用ライブラリー名として、また共用ライブラリーへの入り口点として使います。構築しようとしているストアード・プロシージャーの入り口点関数名の名前がソース・ファイルの名前とは異なる場合、入り口点の別のパラメーターを受け入れるようにスクリプト・ファイルを変更することができます。データベース・パラメーターの名前を、 $3 に変更するようお勧めします。それから、入り口点のリンク・オプションを -e $2 に変更し、スクリプト・ファイルの実行時にコマンド行で追加パラメーターを指定します。

```
#! /bin/ksh
# bldsrv script file -- AIX
# Builds a Micro Focus COBOL stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2
# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$DB2PATH/include/cobol_mf:$COBCPY
# Compile the program.
cob -c -x $1.cbl
# Create the wrapper program for the stored procedure.
wrapsrv $1
# Link the program using export file ${1}_wrap.exp
# creating shared library $1 with entry point ${1}_wrap.
cob -x -o $1 ${1}_wrap.c $1.o -Q -bE:${1}_wrap.exp -Q "-e $1" \
-Q -bI:$DB2PATH/lib/db2g.imp -ldb2gmf -L$DB2PATH/lib
# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

cob COBOL コンパイラー。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

-x 実行可能プログラムを作成します。

リンク・オプション

cob リンク編集にコンパイラーを使用します。

-x 実行可能プログラムを作成します。

-o $1 実行可能プログラムを指定します。

-o ${1}_wrap.c
ラッパー・プログラムを指定します。

$1.o プログラム・オブジェクト・ファイルを指定します。

-Q -bE:${1}_wrap.exp
エクスポート・ファイルを指定します。エクスポート・ファイルには、ストアード・プロシージャーの入り口点のリストが含まれています。ストアード・プロシージャーが x という場合、入り口点は x_wrap になります。

-Q "-e $1"
共用ライブラリーに対するデフォルト入り口点を指定します。

-Q -bI:$DB2PATH/lib/db2g.imp
DB2 アプリケーション・ライブラリーに対する入り口点のリストを提供します。

-ldb2gmf
Micro Focus COBOL 用 DB2 例外ハンドラー・ライブラリーとリンクします。

-L$DB2PATH/lib
DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。

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。

ラッパー・プログラム wrapsrv は、ストアード・プロシージャーが実行される直前に、 Micro Focus COBOL のルーチン cobinit() が呼び出される原因になります。その内容は以下のとおりです。

```
#! /bin/ksh
# wrapsrv script file
# Creates the wrapper program for Micro Focus COBOL stored procedures
# Usage: wrapsrv <stored_proc>
# Note: The client program calls "<stored_proc>_wrap" not "<stored_proc>"
# Create the wrapper program for the stored procedure.
cat << WRAPPER_CODE > ${1}_wrap.c
#include <stdio.h>
void cobinit(void);
int $1(void *p0, void *p1, void *p2, void *p3);
int main(void)
{
  return 0;
}
int ${1}_wrap(void *p0, void *p1, void *p2, void *p3)
{
  cobinit();
  return $1(p0, p1, p2, p3);
}
WRAPPER_CODE
# Create the export file for the wrapper program
echo $1_wrap > ${1}_wrap.exp
```

サンプル・データベースに接続している場合に、ソース・ファイル outsrv.sqb からサンプル・プログラム outsrv を構築するには、次のように入力します。

```
bldsrv outsrv
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv outsrv database
```

スクリプト・ファイルは、共用ライブラリーをサーバー上の sqllib/function というパスにコピーします。

必要であれば、共用ライブラリーにファイル・モードを設定して、クライアント・アプリケーションからアクセスできるようにします。

ストアード・プロシージャー outsrv を構築してしまえば、そのストアード・プロシージャーを呼び出すクライアント・アプリケーション outcli を構築できます。 outcli は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 177ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

```
outcli database userid password
```

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー outsrv にアクセスし、ストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

ストアード・プロシージャーの終了

ストアード・プロシージャーを開発したならば、次のステートメントを使って、それを終了します。

```
move SQLZ-HOLD-PROC to return-code.
```

このステートメントで、ストアード・プロシージャーはクライアント・アプリケーションに正しく戻ります。ストアード・プロシージャーが、ローカル COBOL クライアント・アプリケーションによって呼び出された場合、これは特に重要です。

REXX

REXX プログラムはプリコンパイルまたはバインドしません。

AIX 上で DB2 REXX/SQL プログラムを実行するには、 DB2 インストール・ディレクトリーの下に sqllib/lib を組み込むように、 LIBPATH 環境変数を設定する必要があります。

次のように入力します。

```
export LIBPATH=$LIBPATH:/lib:/usr/lib:/usr/lpp/db2_07_01/sqllib/lib
```

AIX 上で、アプリケーション・ファイルには、任意のファイル拡張子を付けることができます。アプリケーションは、次の 2 つの方法で実行することができます。

1. シェル・コマンド・プロンプトで、rexx *name* と入力します。 (*name* は REXX プログラムの名前。)
2. REXX プログラムの最初の行に「マジック・ナンバー」(#!) が含まれており、それが REXX/6000 解釈プログラムの常駐するディレクトリーを識別する場合は、シェル・コマンド・プロンプトでその名前を入力すれば、REXX プログラムを実行することができます。たとえば、 REXX/6000 解釈プログラム・ファイルが /usr/bin ディレクトリーにある場合は、次の行を、REXX プログラムの最初の行として組み込みます。

   ```
   #! /usr/bin/rexx
   ```

   そうすれば、シェル・コマンド・プロンプトで次のコマンドを入力することによって、プログラムを実行可能にできます。

   ```
   chmod +x name
   ```

   シェル・コマンド・プロンプトでファイル名を入力することによって、REXX プログラムを実行します。

REXX サンプル・プログラムは、 sqllib/samples/rexx ディレクトリーにあります。サンプル REXX プログラム updat.cmd を実行するには、次のいずれかを行います。

- プログラム・ソース・ファイルの上部に行 "#! /usr/bin/rexx" がなければ、それを追加してから、次のように入力してプログラムを直接実行してください。

  ```
  updat.cmd
  ```

- 次のように入力して、REXX 解釈プログラムおよびプログラムを指定してください。

  ```
  rexx updat.cmd
  ```

REXX および DB2 の詳細については、 *アプリケーション開発の手引き* の『REXX でのプログラミング』を参照してください。

第7章 HP-UX アプリケーションの構築

この章では、HP-UX で DB2 アプリケーションを構築するための詳細な情報を提供します。スクリプト・ファイルにおいて、db2 から始まるコマンドは、コマンド行プロセッサー (CLP) のコマンドです。 CLP コマンドについての詳細な情報が必要であれば、 *コマンド解説書* を参照してください。

HP-UX 環境での DB2 アプリケーション開発の最新の更新事項については、次の DB2 アプリケーション開発 Web ページを参照してください。

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

注:

1. DB2 ビルド・ファイルおよび makefile のコンパイルおよびリンクのステップでは、 +DAportable オプションが使用されます。このオプションは、PA_RISC 1.1 および 2.0 ワークステーションでの互換性を持つコードを生成します。このオプションは、パフォーマンスに若干の影響を与えます。パフォーマンスを上げたい場合は、 sqllib/samples ディレクトリーのビルド・ファイルと makefile から、 +DAportable オプションを取り外すことができます。このオプションを使用しないで HP-UX プログラムを構築すると、次のような警告が出される場合があります。

   ```
   (Warning) At least one PA 2.0 object file (<filename>.o) was detected.
   The linked object may not run on a PA 1.x system.
   ```

ここで、<filename> はコンパイルするプログラム・ファイルです。

PA_RISC 1.1 または 2.0 以外のシステムをご使用の場合、この警告は表示されません。

2. HP-UX バージョン 10 または HP-UX バージョン 11 以前から DB2 を移行している場合、 DB2 プログラムは HP-UX バージョン 11 (組み込み SQL がある場合) 上の DB2 で再びプリコンパイルして、再コンパイルしなければなりません。これには、すべての DB2 アプリケーション、ストアード・プロシージャー、ユーザー定義関数、およびユーザー出口プログラムが含まれます。さらに、HP-UX バージョン 11 上でコンパイルされた DB2 プログラムは、 HP-UX バージョン 10 以前の上では実行できない可能性があります。 HP-UX バージョン 10 上でコンパイルされ、実行される DB2 プログラムは、 HP-UX バージョン 11 サーバーにリモートで接続することができます。

HP-UX C

この節では以下のトピックを取り上げています。

- DB2 CLI アプリケーション
- DB2 CLI ストアード・プロシージャー
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー
- ユーザー定義関数 (UDF)
- マルチスレッド・アプリケーション

DB2 CLI アプリケーション

sqllib/samples/cli にあるスクリプト・ファイル bldcli には、 DB2 CLI プログラムを作成するためのコマンドが含まれています。パラメーター $1 には、ソース・ファイルの名前を指定します。

必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない CLI プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

プログラムに組み込み SQL が含まれている場合 (拡張子が .sqc の場合) は、embprep スクリプトが呼び出されてそのプログラムをプリコンパイルし、 .c という拡張子のプログラム・ファイルを生成します。

```
#! /bin/ksh
# bldcli script file -- HP-UX
# Builds a CLI program with HP-UX C.
# Usage: bldcli <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
embprep $1 $2 $3 $4
fi

# Compile the error-checking utility.
cc +DAportable -Aa +e -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc +DAportable -Aa +e -I$DB2PATH/include -c $1.c

# Link the program.
cc +DAportable -o $1 $1.o utilcli.o -L$DB2PATH/lib -ldb2
```

bldcli のコンパイルおよびリンク・オプション

コンパイル・オプション

cc
C コンパイラーを使用します。

+DAportable
PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを生成します。

-Aa
ANSI 規格モードを使用します。

+e
ANSI C モードでコンパイル中に、HP 値追加機能を使用可能にします。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c
コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。

bldcli のコンパイルおよびリンク・オプション
リンク・オプション **cc**　コンパイラーをリンカーのフロントエンドとして使用します。 **+DAportable** PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを使用します。 **-o $1**　実行可能プログラムを指定します。 **-o $1.o** オブジェクト・ファイルを指定します。 **utilcli.o** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 **-ldb2**　データベース・マネージャー・ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル tbinfo.c からサンプル・プログラム tbinfo を構築するには、次のようにします。

```
bldcli tbinfo
```

結果として、実行可能ファイル tbinfo が生成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
tbinfo
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル dbusemx.sqc から組み込み SQL アプリケーション dbusemx を作成する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldcli dbusemx
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldcli dbusemx database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldcli dbusemx database userid password
   ```

結果として、実行可能ファイル dbusemx が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   dbusemx
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   dbusemx database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   dbusemx database userid password
   ```

DB2 API を使用する DB2 CLI アプリケーション

DB2 には、CLI サンプル・プログラムが含まれています。このサンプル・プログラムは、DB2 API を使用してデータベースを作成およびドロップし、 CLI 機能を複数のデータベースで使用する方法を示します。 DB2 API を使用するサンプルは、 27ページの表7 にある CLI サンプル・プログラムの説明の中に示されています。

sqllib/samples/cli にあるスクリプト・ファイル bldapi には、 DB2 API を使用して DB2 CLI プログラムを作成するためのコマンドが入っています。このファイルは、データベースを作成およびドロップするための DB2 API が入った utilapi ユーティリティー・ファイルでコンパイルおよびリンクします。この点が、このスクリプト・ファイルと bldcli スクリプトの唯一の違いです。 bldapi と bldcli の両方に共通するコンパイルとリンクのオプションについては、 188ページの『DB2 CLI アプリケーション』 を参照してください。

ソース・ファイル dbmconn.c からサンプル・プログラム dbmconn を作成するには、次のようにします。

```
bldapi dbmconn
```

結果として、実行可能ファイル dbmconn が作成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
dbmconn
```

DB2 CLI ストアード・プロシージャー

sqllib/samples/cli のスクリプト・ファイル bldclisp には、 DB2 CLI ストアード・プロシージャーを構築するコマンドが入っています。パラメーター $1 には、ソース・ファイルの名前を指定します。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldclisp script file -- HP-UX
# Builds a CLI stored procedure in HP-UX C.
# Usage: bldclisp <prog_name>
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the error-checking utility.
cc +DAportable +u1 +z -Aa +e -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc +DAportable +u1 +z -Aa +e -I$DB2PATH/include -c $1.c

# Link the program.
ld -b -o $1 $1.o utilcli.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldclisp のコンパイルおよびリンク・オプション
コンパイル・オプション **cc** C コンパイラー。 **+DAportable** PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを生成します。 **+u1** 位置合わせしないデータ・アクセスを認めます。アプリケーションが位置合わせしないデータを使用する場合にのみ使用します。 **+z** 位置独立コードを生成します。 **-Aa** ANSI 規格モードを使用します (C コンパイラー専用)。 **+e** ANSI C モードでコンパイル中に、HP 値追加機能を使用可能にします。 **-I$DB2PATH/include** DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。 **-c** コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。
リンク・オプション **ld** リンク編集にリンカーを使用します。 **-b** 通常の実行可能ファイルではなく、共用ライブラリーを作成します。 **-o $1** 実行可能ファイルを指定します。 **$1.o** オブジェクト・ファイルを指定します。 **-L$DB2PATH/lib** DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、-L$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

sample データベースに接続している場合に、ソース・ファイル spserver.c からサンプル・プログラム spserver を構築するには、次のように入力します。

```
bldclisp spserver
```

スクリプト・ファイルは、共用ライブラリーをサーバー上の sqllib/function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、共用ライブラリーにアクセスする CLI クライアント・アプリケーション spclient を構築することができます。

spclient は、スクリプト・ファイル bldcli を使用して構築することができます。詳細については、188ページの『DB2 CLI アプリケーション』を参照してください。

共用ライブラリーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。この出力は、クライアント・アプリケーションに戻されます。

DB2 API と組み込み SQL アプリケーション

sqllib/samples/c にあるスクリプト・ファイル bldapp には、 DB2 アプリケーション・プログラムを構築するコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない DB2 API プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- HP-UX
# Builds a C application program
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
embprep $1 $2 $3 $4
  # Compile the utilemb.c error-checking utility.
  cc +DAportable -Aa +e -I$DB2PATH/include -c utilemb.c
else
  # Compile the utilapi.c error-checking utility.
  cc +DAportable -Aa +e -I$DB2PATH/include -c utilapi.c
fi
# Compile the program.
cc +DAportable -Aa +e -I$DB2PATH/include -c $1.c

if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o
  cc +DAportable -o $1 $1.o utilemb.o -L$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o
  cc +DAportable -o $1 $1.o utilapi.o -L$DB2PATH/lib -ldb2
fi
```

bldapp のコンパイルおよびリンク・オプション

コンパイル・オプション

cc C コンパイラー。

+DAportable
PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを生成します。

-Aa ANSI 規格モードを使用します (C コンパイラー専用)。

+e ANSI C モードでコンパイル中に、HP 値追加機能を使用可能にします。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、-I$DB2PATH/include。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

リンク・オプション

cc リンク編集にコンパイラーを使用します。

+DAportable
PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを使用します。

-o $1 実行可能ファイルを指定します。

$1.o プログラム・オブジェクト・ファイルを指定します。

utilemb.o
組み込み SQL プログラムの場合に、エラー・チェックを行う組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。

utilapi.o
非組み込み SQL プログラムの場合に、エラー・チェックを行う DB2 API ユーティリティー・オブジェクト・ファイルを含みます。

-L$DB2PATH/lib
DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、 -L$DB2PATH/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。

-ldb2 DB2 ライブラリーとリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.c から DB2 API 非組み込み SQL サンプル・プログラム client を構築するには、次のようにします。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。

この実行可能ファイルを実行するには、ファイル名を入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqc から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

sqllib/samples/c にあるスクリプト・ファイル bldsrv には、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが含まれています。スクリプト・ファイルは、ストアード・プロシージャーを共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターは必要ありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldsrv script file -- HP-UX
# Builds a C stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2
# Compile the program.
cc +DAportable +u1 +z -Aa +e -I$DB2PATH/include -c $1.c

# Link the program to create a shared library
ld -b -o $1 $1.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

cc C コンパイラー。

+DAportable
PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを生成します。

+u1 位置合わせしないデータ・アクセスを認めます。アプリケーションが位置合わせしないデータを使用する場合にのみ使用します。

-Aa ANSI 規格モードを使用します (C コンパイラー専用)。

+z 位置独立コードを生成します。

+e ANSI C モードでコンパイル中に、HP 値追加機能を使用可能にします。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、-I$DB2PATH/include。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

bldsrv のコンパイルおよびリンク・オプション
リンク・オプション **ld** リンク編集にリンカーを使用します。 **-b** 通常の実行可能ファイルではなく、共用ライブラリーを作成します。 **-o $1** 出力を、共用ライブラリー・ファイルとして指定します。 **$1.o** プログラム・オブジェクト・ファイルを指定します。 **-L$DB2PATH/lib** DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

sample データベースに接続している場合に、ソース・ファイル spserver.sqc からサンプル・プログラム spserver を構築するには、次のように入力します。

```
bldsrv spserver
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv spserver database
```

スクリプト・ファイルは、共用ライブラリーをサーバー上の sqllib/function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、この共用ライブラリーにアクセスするクライアント・アプリケーション spclient を構築することができます。

spclient は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 194ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

共用ライブラリーにアクセスするには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

ユーザー定義関数 (UDF)

スクリプト・ファイル bldudf は、 sqllib/samples/c にあり、 UDF を作成するためのコマンドが含まれています。 UDF は、ストアード・プロシージャーと同じようにコンパイルされます。 UDF に組み込み SQL ステートメントを含めることはできません。したがって、UDF プログラムを作成する際に、データベースへの接続、プログラムのプリコンパイル、およびバインドは必要ありません。

パラメーター $1 には、ソース・ファイルの名前を指定します。スクリプト・ファイルは、そのソース・ファイル名を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldudf script file -- HP-UX
# Builds a C UDF library
# Usage: bldudf <prog_name>
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Compile the program.
cc +DAportable +u1 +z -Aa +e -I$DB2PATH/include -c $1.c

# Link the program and create a shared library.
ld -b -o $1 $1.o -L$DB2PATH/lib -ldb2 -ldb2apie

# Copy the shared library to the sqllib/function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldudf のコンパイルおよびリンク・オプション

コンパイル・オプション

cc C コンパイラー。

+DAportable
PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを生成します。

+u1 位置合わせしないデータ・アクセスを認めます。アプリケーションが位置合わせしないデータを使用する場合にのみ使用します。

-Aa ANSI 規格モードを使用します (C コンパイラー専用)。

+z 位置独立コードを生成します。

+e ANSI C モードでコンパイル中に、HP 値追加機能を使用可能にします。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

bldudf のコンパイルおよびリンク・オプション
リンク・オプション **ld**　リンク編集にリンカーを使用します。 **-b**　通常の実行可能ファイルではなく、共用ライブラリーを作成します。 **-o $1**　出力を、共用ライブラリー・ファイルとして指定します。 **$1.o**　プログラム・オブジェクト・ファイルを指定します。 **-L$DB2PATH/lib** DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-ldb2**　DB2 ライブラリーとリンクします。 **-ldb2apie** DB2 API エンジン・ライブラリーとリンクして、LOB ロケーターを使用できるようにします。 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。

ユーザー定義関数プログラム udfsrv をソース・ファイル udfsrv.c から作成するには、次のように入力します。

```
bldudf udfsrv
```

スクリプト・ファイルは、 UDF を sqllib/function ディレクトリーにコピーします。

必要であれば、UDF にファイル・モードを設定してクライアント・アプリケーションから実行できるようにします。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。このプログラムには DB2 CLI バージョンと組み込み SQL バージョンがあります。

DB2 CLI udfcli プログラム は、スクリプト・ファイル bldcli を使用して、sqllib/samples/cli にあるソース・ファイル udfcli.c から作成できます。詳細については、188ページの『DB2 CLI アプリケーション』を参照してください。

組み込み SQL udfcli プログラムは、スクリプト・ファイル bldapp を使用して、 sqllib/samples/c にあるソース・ファイル udfcli.sqc から構築することができます。詳細については、 194ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF を呼び出すには、次のように実行可能ファイル名を入力して、サンプルの呼び出しアプリケーションを実行します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv ライブラリーから ScalarUDF 関数を呼び出します。

マルチスレッド・アプリケーション

注: HP-UX には、POSIX スレッド・ライブラリーと DCE スレッド・ライブラリーがあります。 POSIX スレッド・ライブラリーを使用するマルチスレッド・アプリケーションは、 DB2 がサポートします。

HP-UX のマルチスレッド・アプリケーションは、コンパイルするために _REENTRANT 定義が必要です。 HP-UX の資料では、 -D_POSIX_C_SOURCE=199506L でコンパイルすることをお勧めします。 _REENTRANT が定義されていることも必ず確認してください。また、アプリケーションは、-lpthread とリンクされていることも必要です。

スクリプト・ファイル bldmt は sqllib/samples/c にあり、組み込み SQL マルチスレッド・プログラムを作成するためのコマンドが含まれています。第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。第 3 パラメーター $3 にはそのデータベースのユーザー ID を、また $4 にはパスワードを指定します。第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名、ユーザー ID、およびパスワードは任意指定です。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldmt script file -- HP-UX
# Builds a C multi-threaded embedded SQL program
# Usage:  bldmt <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2 $3 $4
# Compile the program.
cc +DAportable -Aa +e -I$DB2PATH/include -D_POSIX_C_SOURCE=199506L -c $1.c

# Link the program
cc +DAportable -o $1 $1.o -L$DB2PATH/lib -ldb2 -lpthread
```

上記の -D_POSIX_C_SOURCE=199506L コンパイラー・オプションと -lpthread リンク・オプション、そしてリンクされているユーティリティー・ファイルが

ないという点だけでなく、残りのコンパイルとリンクのオプションも、bldapp ファイルで使われているものと同じです。これらのオプションについては、194ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル thdsrver.sqc からサンプル・プログラム thdsrver を作成するには、次のように入力します。

```
bldmt thdsrver
```

結果として、実行可能ファイル thdsrver が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
thdsrver
```

HP-UX C++

この節では、次のトピックを取り上げます。

- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー
- ユーザー定義関数 (UDF)
- マルチスレッド・アプリケーション

DB2 API と組み込み SQL アプリケーション

sqllib/samples/cpp にあるスクリプト・ファイル bldapp には、 DB2 アプリケーション・プログラムを構築するコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を含まないプログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- HP-UX
# Builds a C++ application program
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqC" ]]
then
embprep $1 $2 $3 $4
  # Compile the utilemb.C error-checking utility.
  CC +DAportable +a1 -ext -I$DB2PATH/include -c utilemb.C
else
  # Compile the utilapi.C error-checking utility.
  CC +DAportable +a1 -ext -I$DB2PATH/include -c utilapi.C
fi

# Compile the program.
CC +DAportable +a1 -ext -I$DB2PATH/include -c $1.C
if [[ -f $1".sqC" ]]
then
  # Link the program with utilemb.o
  CC +DAportable -o $1 $1.o utilemb.o -L$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o
  CC +DAportable -o $1 $1.o utilapi.o -L$DB2PATH/lib -ldb2
fi
```

bldapp のコンパイルおよびリンク・オプション

コンパイル・オプション

CC C コンパイラー。

+DAportable
PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを生成します。

+a1 コンパイラーに ANSI C/C++ を使用するように指示します。

-ext "long long" サポートを含むさまざまな C++ 拡張子を許可します。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

bldapp のコンパイルおよびリンク・オプション
リンク・オプション **CC**　コンパイラーをリンカーのフロントエンドとして使用します。 **+DAportable** PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを使用します。 **-o $1**　実行可能ファイルを指定します。 **$1.o**　プログラム・オブジェクト・ファイルを指定します。 **utilemb.o** 組み込み SQL プログラムの場合に、エラー・チェックを行う組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。 **utilapi.o** 非組み込み SQL プログラムの場合に、エラー・チェックを行う DB2 API ユーティリティー・オブジェクト・ファイルを含みます。 **-L$DB2PATH/lib** DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-ldb2**　DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.C から非組み込み SQL DB2 API サンプル・プログラム client を作成するには、次のように入力します。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqC から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

```
bldapp updat database
```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

```
bldapp updat database userid password
```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

```
updat
```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

```
updat database
```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

```
updat database userid password
```

組み込み SQL ストアード・プロシージャー

注: 71ページの『UDF およびストアード・プロシージャーに関する C++ 考慮事項』にある、 C++ のストアード・プロシージャーの構築に関する情報を参照してください。

スクリプト・ファイル bldsrv は sqllib/samples/cpp にあり、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが含まれています。スクリプト・ファイルは、ストアード・プロシージャーを共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 では、ソース・ファイルの名前を指定します。第 2 パラメーター $2 では、接続したいデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターは必要ありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

スクリプト・ファイルは、ソース・ファイル名 $1 を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldsrv script file -- HP-UX
# Builds a C++ stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2
# Compile the program. First ensure it is coded with extern "C".
CC +DAportable +a1 +z -ext -I$DB2PATH/include -c $1.C
# Link the program to create a shared library.
ld -b -o $1 $1.o -L$DB2PATH/lib -ldb2
# Copy the shared library to the sqllib/function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

CC C++ コンパイラー。

+DAportable
PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを生成します。

+a1 コンパイラーに ANSI C/C++ を使用するように指示します。

+z 位置独立コードを生成します。

-ext ″long long″ サポートを含むさまざまな C++ 拡張子を許可します。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$DB2PATH/include。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

bldsrv のコンパイルおよびリンク・オプション
リンク・オプション **ld**　リンク編集にリンカーを使用します。 **-b**　通常の実行可能ファイルではなく、共用ライブラリーを作成します。 **-o $1**　実行可能ファイルを指定します。 **$1.o**　プログラム・オブジェクト・ファイルを指定します。 **-L$DB2PATH/lib** DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、 -L$DB2PATH/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-ldb2**　DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

sample データベースに接続している場合に、ソース・ファイル spserver.sqC からサンプル・プログラム spserver を構築するには、次のように入力します。

```
bldsrv spserver
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv spserver database
```

スクリプト・ファイルは、共用ライブラリーをサーバー上の sqllib/function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、クライアント・アプリケーション spclient を構築することができます。これは、共用ライブラリー内のストアード・プロシージャーを呼び出すアプリケーションです。

spclient は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 204ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

共用ライブラリーにアクセスするには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。ストアード・プロシージャーは、出力をクライアント・アプリケーションに戻します。

ユーザー定義関数 (UDF)

注: 71ページの『UDF およびストアード・プロシージャーに関する C++ 考慮事項』にある、 C++ UDF の構築に関する情報を参照してください。

スクリプト・ファイル bldudf は、 sqllib/samples/c にあり、 UDF を作成するためのコマンドが含まれています。ユーザー定義プログラムに組み込み SQL ステートメントを含めることはできません。したがって、UDF プログラムを作成する際に、データベースへの接続、プログラムのプリコンパイル、およびバインドは必要ありません。

パラメーター $1 には、ソース・ファイルの名前を指定します。スクリプト・ファイルは、そのソース・ファイル名を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldudf script file -- HP-UX
# Builds a C or C++ UDF library
# Usage: bldudf <prog_name>
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Compile the program.
if [[ -f $1".c" ]]
then
  CC +DAportable -ext +a1 +z -I$DB2PATH/include -c $1.c
elif [[ -f $1".C" ]]
then
  CC +DAportable -ext +a1 +z -I$DB2PATH/include -c $1.C
fi
# Link the program.
CC +DAportable -b -o $1 $1.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldudf のコンパイルおよびリンク・オプション

コンパイル・オプション

CC C++ コンパイラー。

+DAportable
PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを生成します。

-ext ″long long″ サポートを含むさまざまな C++ 拡張子を許可します。

+a1 コンパイラーに ANSI C/C++ を使用するように指示します。

+z 位置独立コードを生成します。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$DB2PATH/include。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

bldudf のコンパイルおよびリンク・オプション
リンク・オプション **CC** コンパイラーをリンカーのフロントエンドとして使用します。 **-b** 通常の実行可能ファイルではなく、共用ライブラリーを作成します。 **-o $1** 実行可能ファイルを指定します。 **$1.o** プログラム・オブジェクト・ファイルを指定します。 **-L$DB2PATH/lib** DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、 -L$DB2PATH/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ユーザー定義関数プログラム udfsrv をソース・ファイル udfsrv.c から作成するには、次のように入力します。

```
bldudf udfsrv
```

スクリプト・ファイルは、 UDF を sqllib/function ディレクトリーにコピーします。

必要であれば、UDF にファイル・モードを設定してクライアント・アプリケーションから実行できるようにします。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。 udfcli プログラムは、スクリプト・ファイル bldapp を使用して、 sqllib/samples/cpp にあるソース・ファイル udfcli.sqC から作成します。詳細については、 204ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF を呼び出すには、次のように実行可能ファイル名を入力して、サンプルの呼び出しアプリケーションを実行します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv ライブラリーの ScalarUDF 関数を呼び出します。

マルチスレッド・アプリケーション

注: HP-UX には、POSIX スレッド・ライブラリーと DCE スレッド・ライブラリーがあります。 POSIX スレッド・ライブラリーを使用するマルチスレッド・アプリケーションは、 HP-UX 上の DB2 がサポートします。

HP-UX のマルチスレッド・アプリケーションは、コンパイルするために _REENTRANT 定義が必要です。 HP-UX の資料では、 -D_POSIX_C_SOURCE=199506L でコンパイルすることをお勧めします。 _REENTRANT が定義されていることも必ず確認してください。 HP-UX C++ コンパイラーの場合、 -D_HPUX_SOURCE を使用して rand_r を定義することが必要です。また、アプリケーションは、-lpthread とリンクされていることも必要です。

sqllib/samples/cpp にあるスクリプト・ファイル bldmt には、組み込み SQL マルチスレッド・プログラムを作成するためのコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。第 3 パラメーター $3 にはそのデータベースのユーザー ID を、また $4 にはパスワードを指定します。第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名、ユーザー ID、およびパスワードは任意指定です。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldmt script file -- HP-UX
# Builds a C++ multi-threaded embedded SQL program
# Usage:  bldmt <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2 $3 $4
# Compile the program.
CC +DAportable +a1 -ext -I$DB2PATH/include \
   -D_HPUX_SOURCE -D_POSIX_C_SOURCE=199506L -c $1.C
# Link the program
CC -o $1 $1.o -L$DB2PATH/lib -ldb2 -lpthread
```

上記の -D_HPUX_SOURCE および -D_POSIX_C_SOURCE=199506L コンパイル・オプションと -lpthread リンク・オプション、そしてリンクされているユーティリティー・ファイルがないという点だけでなく、残りのコンパイルとリンクのオプションも、組み込み SQL スクリプト・ファイル bldapp で使われているも

のと同じです。これらのオプションについては、 204ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル thdsrver.sqC からサンプル・プログラム thdsrver を作成するには、次のように入力します。

```
bldmt thdsrver
```

結果として、実行可能ファイル thdsrver が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
thdsrver
```

Micro Focus COBOL

この節では以下のトピックを取り上げています。

- コンパイラーの使用法
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー

コンパイラーの使用

組み込み SQL および DB2 API 呼び出しを含むアプリケーションを開発しており、 Micro Focus COBOL コンパイラーを使用している場合には、以下の点に留意してください。

- コマンド行プロセッサーのコマンド db2 prep を使ってアプリケーションをプリコンパイルする場合は、 target mfcob オプション (デフォルト) を使ってください。
- 組み込みプリコンパイラー・フロントエンド、実行時解釈プログラム、または Animator デバッガーを使うためには、次のように Micro Focus が提供する mkrts コマンドを実行して、 DB2 Generic API 入り口点を Micro Focus 実行時モジュール rts32 に追加する必要があります。また、mkcheck を更新して検査ファイルを更新する必要があります。 mkcheck が実行されない場合、SQLGSTRT に 173 エラーが戻されます。

  mkrts および mkcheck を実行するには、次のステップに従って COBOPT を設定する必要があります。

  1. root としてログインします。
  2. ディレクトリー $COBDIR/src/rts から次のように入力します。

```
COBOPT=/opt/IBMdb2/V7.1/lib/db2mkrts.args; export COBOPT
ksh mkrts
mv $COBDIR/rts32 $COBDIR/rts32.orig
cp rts32 $COBDIR/rts32
```

3. さらに、Hewlett-Packard 社が製品にあらかじめ配備している、 check 実行可能ファイルを再作成する必要があります。 $COBDIR ディレクトリーの check 実行可能ファイルを再作成しないで cob -C SQL を使用したコンパイルを試行すると、 DB2 プリプロセッサーは DB2 ライブラリーを呼び出すため、コンパイルは失敗し、実行時システム 173 エラーが戻されます。 check を再作成するには、 $COBDIR ルート・ディレクトリーの下にある src/sql ディレクトリーに移動し、 mkcheck スクリプトを実行します。スクリプトが完了したら、結果として作成された check 実行可能ファイルを、 $COBDIR ディレクトリーに移動させる必要があります。ディレクトリー $COBDIR/src/sql から、次のように入力します。

```
COBOPT=/opt/IBMdb2/V7.1/lib/db2mkrts.args; export COBOPT
ksh mkcheck
mv $COBDIR/check $COBDIR/check.orig
cp check $COBDIR/check
```

これで、次のディレクトリーにある引き数を指定して、mkrts を実行することができます。

```
/opt/IBMdb2/V7.1/lib/db2mkrts.args
```

- DB2 COBOL COPY ファイル・ディレクトリーを、 Micro Focus COBOL 環境変数 COBCPY に含める必要があります。 COBCPY 環境変数には、COPY ファイルのロケーションを指定します。 Micro Focus COBOL 用の DB2 COPY ファイルは、データベース・インスタンス・ディレクトリーの下にある sqllib/include/cobol_mf にあります。

  このディレクトリーを組み込むには、次のように入力します。

  ```
  export COBCPY=$COBCPY:/opt/IBMdb2/V7.1/include/cobol_mf
  ```

  注: COBCPY を .profile ファイル中に設定することもできます。

DB2 API と組み込み SQL アプリケーション

sqllib/samples/cobol_mf にあるスクリプト・ファイル bldapp には、 DB2 API および組み込み SQL アプリケーション・プログラムを構築するコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない DB2 API プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーター

がオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- HP-UX
# Builds a Micro Focus COBOL application program
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqb" ]]
then
  embprep $1 $2 $3 $4
fi
# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$COBCPY:$DB2PATH/include/cobol_mf

# Compile the checkerr.cbl error checking utility.
cob +DAportable -cx checkerr.cbl

# Compile the program.
cob +DAportable -cx $1.cbl

# Link the program.
cob +DAportable -x $1.o checkerr.o -L$DB2PATH/lib -ldb2 -ldb2gmf
```

bldapp のコンパイルおよびリンク・オプション

コンパイル・オプション

cob Micro Focus COBOL コンパイラー。

+DAportable
PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを生成します。

-cx オブジェクト・モジュールにコンパイルします。

bldapp のコンパイルおよびリンク・オプション
リンク・オプション **cob**　コンパイラーをリンカーのフロントエンドとして使用します。 **+DAportable** PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを使用します。 **-x**　実行可能プログラムを指定します。 **$1.o**　プログラムのオブジェクト・ファイルを組み込みます。 **checkerr.o** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 **-ldb2**　DB2 ライブラリーとリンクします。 **-ldb2gmf** Micro Focus COBOL 用 DB2 例外ハンドラー・ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.cbl から非組み込み SQL サンプル・プログラム client を作成するには、次のように入力します。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqb から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

```
bldapp updat database userid password
```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

sqllib/samples/cobol_mf にあるスクリプト・ファイル bldsrv には、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが含まれています。スクリプト・ファイルは、ストアード・プロシージャーをサーバー上の共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。スクリプト・ファイルは、ソース・ファイル名 $1 を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldsrv script file -- HP-UX
# Builds a Micro Focus COBOL stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
```

```
# Precompile and bind the program.
embprep $1 $2
# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$COBCPY:$DB2PATH/include/cobol_mf

# Compile the program.
cob +DAportable +z -cx $1.cbl

# Link the program.
ld -b -o $1 $1.o -L$DB2PATH/lib -ldb2 -ldb2gmf \
   -L$COBDIR/coblib -lcobol -lcrtn

# Copy the shared library to the sqllib/function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

cob COBOL コンパイラー。

+DAportable
PA_RISC 1 および 2.0 ワークステーションでの互換性を持つコードを生成します。

+z 位置独立コードを生成します。

-cx オブジェクト・モジュールにコンパイルします。

リンク・オプション

ld リンク編集にリンカーを使用します。

-b 通常の実行可能ファイルではなく、共用ライブラリーを作成します。

-o $1 実行可能ファイルを指定します。

$1.o プログラムのオブジェクト・ファイルを組み込みます。

-L$DB2PATH/lib
DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。

-ldb2 DB2 共用ライブラリーにリンクします。

-ldb2gmf
Micro Focus COBOL 用 DB2 例外ハンドラー・ライブラリーとリンクします。

-L$COBDIR/coblib
COBOL 実行時ライブラリーのロケーションを指定します。

-lcobol
COBOL ライブラリーにリンクします。

-lcrtn crtn ライブラリーにリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

サンプル・データベースに接続している場合に、ソース・ファイル outsrv.sqb からサンプル・プログラム outsrv を構築するには、次のように入力します。

```
bldsrv outsrv
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv outsrv database
```

スクリプト・ファイルは、ストアード・プロシージャーを sqllib/function ディレクトリーにコピーします。

必要であれば、ストアード・プロシージャーにファイル・モードを設定して、クライアント・アプリケーションからアクセスできるようにします。

ストアード・プロシージャー outsrv を作成したなら、そのストアード・プロシージャーを呼び出すクライアント・アプリケーション outcli を構築できます。 outcli は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 215ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

```
outcli database userid password
```

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションはストアード・プロシージャー・ライブラリー outsrv にアクセスし、ストアード・プロシージャー関数をサーバー・データベース上で実行します。この出力は、クライアント・アプリケーションに戻されます。

ストアード・プロシージャーの終了

ストアード・プロシージャーを開発したならば、次のステートメントを使って、それを終了します。

```
move SQLZ-HOLD-PROC to return-code.
```

このステートメントで、ストアード・プロシージャーはクライアント・アプリケーションに正しく戻ります。

第8章 Linux アプリケーションの構築

この章は、Linux でアプリケーションを構築するための詳細な情報を提供します。スクリプト・ファイルにおいて、db2 から始まるコマンドは、コマンド行プロセッサー (CLP) のコマンドです。 CLP コマンドについての詳しい情報が必要であれば、 *コマンド解説書* を参照してください。

Linux 環境での DB2 アプリケーション開発の最新の更新事項については、次の Web ページを参照してください。

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

Linux C

この節では以下のトピックを取り上げています。

- DB2 CLI アプリケーション
- DB2 CLI ストアード・プロシージャー
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー
- ユーザー定義関数 (UDF)
- マルチスレッド・アプリケーション

DB2 CLI アプリケーション

sqllib/samples/cli のスクリプト・ファイル bldcli には、 DB2 CLI プログラムを作成するためのコマンドが入っています。パラメーター $1 には、ソース・ファイルの名前を指定します。

必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない CLI プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

プログラムに組み込み SQL が含まれている場合 (拡張子が .sqc の場合) は、embprep スクリプトが呼び出されてそのプログラムをプリコンパイルし、 .c という拡張子のプログラム・ファイルを生成します。

```
#! /bin/ksh
# bldcli script file -- Linux
# Builds a CLI program with Linux C
# Usage: bldcli <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
embprep $1 $2 $3 $4
fi
# Compile the error-checking utility.
cc -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o utilcli.o -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
```

bldcli のコンパイルおよびリンク・オプション

コンパイル・オプション

cc C コンパイラー。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldcli のコンパイルおよびリンク・オプション
リンク・オプション **cc** コンパイラーをリンカーのフロントエンドとして使用します。 **-o $1** 実行可能ファイルを指定します。 **$1.o** プログラムのオブジェクト・ファイルを組み込みます。 **utilcli.o** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-Wl,-rpath,$DB2PATH/lib** 実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル tbinfo.c からサンプル・プログラム tbinfo を構築するには、次のようにします。

```
bldcli tbinfo
```

結果として、実行可能ファイル tbinfo が生成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
tbinfo
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル dbusemx.sqc から組み込み SQL アプリケーション dbusemx を作成する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldcli dbusemx
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldcli dbusemx database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldcli dbusemx database userid password
   ```

結果として、実行可能ファイル dbusemx が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   dbusemx
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   dbusemx database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   dbusemx database userid password
   ```

DB2 API を使用する DB2 CLI アプリケーション

DB2 には、CLI サンプル・プログラムが含まれています。このサンプル・プログラムは、DB2 API を使用してデータベースを作成およびドロップし、 CLI 機能を複数のデータベースで使用する方法を示します。 DB2 API を使用するサンプルは、 27ページの表7 にある CLI サンプル・プログラムの説明の中に示されています。

sqllib/samples/cli にあるスクリプト・ファイル bldapi には、 DB2 API を使用して DB2 CLI プログラムを作成するためのコマンドが入っています。このファイルは、データベースを作成およびドロップするための DB2 API が入った utilapi ユーティリティー・ファイルでコンパイルおよびリンクします。この点が、このスクリプト・ファイルと bldcli スクリプトの唯一の違いです。 bldapi と bldcli の両方に共通するコンパイルとリンクのオプションについては、 223ページの『DB2 CLI アプリケーション』 を参照してください。

ソース・ファイル dbmconn.c からサンプル・プログラム dbmconn を作成するには、次のようにします。

```
bldapi dbmconn
```

結果として、実行可能ファイル dbmconn が作成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
dbmconn
```

DB2 CLI ストアード・プロシージャー

sqllib/samples/cli のスクリプト・ファイル bldclisp には、 DB2 CLI ストアード・プロシージャーを構築するコマンドが入っています。パラメーター $1 には、ソース・ファイルの名前を指定します。

```
#! /bin/ksh
# bldclisp script file -- Linux
# Builds a CLI stored procedure in Linux C.
# Usage: bldclisp <prog_name>
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the error-checking utility.
cc -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o utilcli.o -shared -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2

# Copy the shared library to the function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldclisp のコンパイルおよびリンク・オプション

コンパイル・オプション

cc C コンパイラー。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldclisp のコンパイルおよびリンク・オプション
リンク・オプション **cc** コンパイラーをリンカーのフロントエンドとして使用します。 **-o $1** 実行可能ファイルを指定します。 **$1.o** プログラムのオブジェクト・ファイルを組み込みます。 **utilcli.o** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-shared** 共用ライブラリーを生成します。 **-L$DB2PATH/lib** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-Wl,-rpath,$DB2PATH/lib** 実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル spserver.c からサンプル・プログラム spserver を構築するには、次のように入力します。

```
bldclisp spserver
```

スクリプト・ファイルは、共用ライブラリーを sqllib/function ディレクトリーにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、共用ライブラリーを呼び出す CLI クライアント・アプリケーション spclient を構築することができます。

spclient は、スクリプト・ファイル bldcli を使用して構築することができます。詳細については、223ページの『DB2 CLI アプリケーション』を参照してください。

共用ライブラリーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

DB2 API と組み込み SQL アプリケーション

sqllib/samples/c にあるスクリプト・ファイル bldapp には、 DB2 アプリケーション・プログラムを構築するコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない DB2 API プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続

するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのスクリプト embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- Linux
# Builds a C application program.
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.c error-checking utility.
  cc -I$DB2PATH/include -c utilemb.c
else
  # Compile the utilapi.c error-checking utility.
  cc -I$DB2PATH/include -c utilapi.c
fi
# Compile the program.
cc -I$DB2PATH/include -c $1.c
if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o.
  cc -o $1 $1.o utilemb.o -L$DB2PATH/lib \
    -Wl,-rpath,$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o.
  cc -o $1 $1.o utilapi.o -L$DB2PATH/lib \
    -Wl,-rpath,$DB2PATH/lib -ldb2
fi
```

bldapp のコンパイルおよびリンク・オプション
コンパイル・オプション **cc** C コンパイラー。 **-I$DB2PATH/include** DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。 **-c** コンパイルのみを実行し、リンクは実行しません。このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。
リンク・オプション **cc** コンパイラーをリンカーのフロントエンドとして使用します。 **-o $1** 実行可能ファイルを指定します。 **$1.o** オブジェクト・ファイルを指定します。 **utilemb.o** 組み込み SQL プログラムの場合に、エラー・チェックを行う組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。 **utilapi.o** 非組み込み SQL プログラムの場合に、エラー・チェックを行う DB2 API ユーティリティー・オブジェクト・ファイルを含みます。 **-L$DB2PATH/lib** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-Wl,-rpath,$DB2PATH/lib** 実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.c から DB2 API 非組み込み SQL サンプル・プログラム client を構築するには、次のようにします。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。

この実行可能ファイルを実行するには、ファイル名を入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqc から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

sqllib/samples/c にあるスクリプト・ファイル bldsrv には、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが含まれています。スクリプト・ファイルは、ストアード・プロシージャーを共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターは必要ありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldsrv script file -- Linux
# Builds a C stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2
# Compile the program.
cc -I$DB2PATH/include -c $1.c
# Link the program and create a shared library
cc -shared -o $1 $1.o -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
# Copy the shared library to the function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

cc C コンパイラー。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldsrv のコンパイルおよびリンク・オプション
リンク・オプション **cc** コンパイラーをリンカーのフロントエンドとして使用します。 **-shared** 共用ライブラリーを生成します。 **-o $1** 実行可能ファイルを指定します。 **$1.o** プログラムのオブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-Wl,-rpath,$DB2PATH/lib** 実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

sample データベースに接続している場合に、ソース・ファイル spserver.sqc からサンプル・プログラム spserver を構築するには、次のように入力します。

```
bldsrv spserver
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv spserver database
```

スクリプト・ファイルは、共用ライブラリーをサーバー上の sqllib/function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、この共用ライブラリーにアクセスするクライアント・アプリケーション spclient を構築することができます。

spclient は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 229ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

共用ライブラリーにアクセスするには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションはストアード・プロシージャー・ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

ユーザー定義関数 (UDF)

sqllib/samples/c のスクリプト・ファイル bldudf には、 UDF を作成するためのコマンドが入っています。 UDF には、組み込み SQL ステートメントは含まれません。したがって、UDF プログラムを作成する場合は、データベースへの接続、あるいはプログラムのプリコンパイルおよびバインドは必要ありません。

パラメーター $1 には、ソース・ファイルの名前を指定します。スクリプト・ファイルは、このソース・ファイル名を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldudf script file -- Linux
# Builds a C UDF library
# Usage: bldudf <prog_name>
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Compile the program.
cc -I$DB2PATH/include -c $1.c
# Link the program and create a shared library.
cc -o $1 $1.o -shared -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2 -ldb2apie
# Copy the shared library to the function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldudf のコンパイルおよびリンク・オプション

コンパイル・オプション

cc C コンパイラー。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldudf のコンパイルおよびリンク・オプション
リンク・オプション
cc コンパイラーをリンカーのフロントエンドとして使用します。
-o $1 実行可能ファイルを指定します。
$1.o プログラムのオブジェクト・ファイルを組み込みます。
-shared 共用ライブラリーを生成します。
-L$DB2PATH/lib リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。
-Wl,-rpath,$DB2PATH/lib 実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。
-ldb2 DB2 ライブラリーとリンクします。
-ldb2apie DB2 API エンジン・ライブラリーとリンクして、LOB ロケーターを使用できるようにします。
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。

ユーザー定義関数プログラム udfsrv をソース・ファイル udfsrv.c から作成するには、次のように入力します。

```
bldudf udfsrv
```

スクリプト・ファイルは、 UDF を sqllib/function ディレクトリーにコピーします。

必要であれば、UDF にファイル・モードを設定して DB2 インスタンスがそれを実行できるようにしてください。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。このプログラムには DB2 CLI バージョンと組み込み SQL バージョンがあります。

DB2 CLI udfcli プログラム は、スクリプト・ファイル bldcli を使用して、sqllib/samples/cli にあるソース・ファイル udfcli.c から作成できます。詳細については、223ページの『DB2 CLI アプリケーション』を参照してください。

組み込み SQL udfcli プログラムは、スクリプト・ファイル bldapp を使用して、 sqllib/samples/c にあるソース・ファイル udfcli.sqc から構築することができます。詳細については、 229ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF を呼び出すには、次のように実行可能ファイル名を入力して、サンプルの呼び出しアプリケーションを実行します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv ライブラリーから ScalarUDF 関数を呼び出します。

マルチスレッド・アプリケーション

Linux C を使用するマルチスレッド・アプリケーションは、 -D_REENTRANT でコンパイルし、 -lpthread とリンクする必要があります。

スクリプト・ファイル bldmt は sqllib/samples/c にあり、組み込み SQL マルチスレッド・プログラムを作成するためのコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。第 3 パラメーター $3 にはそのデータベースのユーザー ID を、また $4 にはパスワードを指定します。第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名、ユーザー ID、および、パスワードは任意指定です。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldmt script file -- Linux
# Builds a C multi-threaded embedded SQL program.
# Usage:  bldmt <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2 $3 $4
# Compile the program.
cc -I$DB2PATH/include -c $1.c -D_REENTRANT
# Link the program.
cc -o $1 $1.o -lpthread -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
```

上記の -D_REENTRANT オプションと -lpthread オプション、そしてリンクされているユーティリティー・ファイルがないという点だけでなく、残りのコンパイルとリンクのオプションも、組み込み SQL スクリプト・ファイル bldapp で使用されているものと同じです。これらのオプションについては、229ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル thdsrver.sqc からサンプル・プログラム thdsrver を作成するには、次のように入力します。

```
bldmt thdsrver
```

結果として、実行可能ファイル thdsrver が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
thdsrver
```

Linux C++

この節では、以下のトピックを取り上げています。

- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー
- ユーザー定義関数
- マルチスレッド・アプリケーション

DB2 API と組み込み SQL アプリケーション

スクリプト・ファイル bldapp は、sqllib/samples/cpp にあり、サンプル C++ プログラムを作成するためのコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない DB2 API プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザ

ー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- Linux
# Builds a C++ application program.
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqC" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.C error-checking utility.
  g++ -I$DB2PATH/include -c utilemb.C
else
  # Compile the utilapi.C error-checking utility.
  g++ -I$DB2PATH/include -c utilapi.C
fi
# Compile the program.
  g++ -I$DB2PATH/include -c $1.C
if [[ -f $1".sqC" ]]
then
  # Link the program with utilemb.o
  g++ -o $1 $1.o utilemb.o -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o
  g++ -o $1 $1.o utilapi.o -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
fi
```

bldapp のコンパイルおよびリンク・オプション

コンパイル・オプション

g++ C++ コンパイラー。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldapp のコンパイルおよびリンク・オプション
リンク・オプション
g++ コンパイラーをリンカーのフロントエンドとして使用します。
-o $1 実行可能ファイルを指定します。
$1.o プログラムのオブジェクト・ファイルを組み込みます。
utilemb.o 組み込み SQL プログラムの場合に、エラー・チェックを行う組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。
utilapi.o 非組み込み SQL プログラムの場合に、エラー・チェックを行う DB2 API ユーティリティー・オブジェクト・ファイルを含みます。
-L$DB2PATH/lib リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。
-Wl,-rpath,$DB2PATH/lib 実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。
-ldb2 DB2 ライブラリーとリンクします。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.C から非組み込み SQL サンプル・プログラム client を作成するには、次のように入力します。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqC から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

注: 71ページの『UDF およびストアード・プロシージャーに関する C++ 考慮事項』にある、 C++ のストアード・プロシージャーの構築に関する情報を参照してください。

スクリプト・ファイル bldsrv は sqllib/samples/cpp にあり、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが含まれています。スクリプト・ファイルは、ストアード・プロシージャーを共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 では、ソース・ファイルの名前を指定します。第 2 パラメーター $2 では、接続したいデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターは必要ありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。スクリプト・ファイルは、ソース・ファイル名 $1 を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldsrv script file -- Linux
# Builds a C++ stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2
# Compile the program.
  g++ -I$DB2PATH/include -c $1.C
# Link the program and create a shared library.
g++ -shared -o $1 $1.o -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2

# Copy the shared library to the function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

g++ C++ コンパイラー。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldsrv のコンパイルおよびリンク・オプション
リンク・オプション **g++** コンパイラーをリンカーのフロントエンドとして使用します。 **-shared** 共用ライブラリーを生成します。 **-o $1** 実行可能ファイルを指定します。 **$1.o** プログラムのオブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-Wl,-rpath,$DB2PATH/lib** 実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

sample データベースに接続している場合に、ソース・ファイル spserver.sqC からサンプル・プログラム spserver を構築するには、次のように入力します。

```
bldsrv spserver
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv spserver database
```

スクリプト・ファイルは、共用ライブラリーをサーバー上の sqllib/function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、クライアント・アプリケーション spclient を構築することができます。これは、共用ライブラリー内のストアード・プロシージャーを呼び出すアプリケーションです。

spclient は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 239ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

共用ライブラリーにアクセスするには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。ストアード・プロシージャーは、出力をクライアント・アプリケーションに戻します。

ユーザー定義関数 (UDF)

注: 71ページの『UDF およびストアード・プロシージャーに関する C++ 考慮事項』 にある、 C++ UDF の構築に関する情報を参照してください。

スクリプト・ファイル bldudf は sqllib/samples/cpp にあり、 UDF を作成するためのコマンドが含まれています。 UDF には、組み込み SQL ステート

メントは含まれません。したがって、UDF プログラムを作成する際に、データベースへの接続、プログラムのプリコンパイル、およびバインドは必要ありません。

パラメーター $1 には、ソース・ファイルの名前を指定します。スクリプト・ファイルは、このソース・ファイル名を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldudf script file -- Linux
# Builds a C++ UDF library
# Usage: bldudf <prog_name>
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Compile the program.
if [[ -f $1".c" ]]
then
  g++ -I$DB2PATH/include -c $1.c

elif [[ -f $1".C" ]]
then
  g++ -I$DB2PATH/include -c $1.C
fi
# Link the program.
g++ -o $1 $1.o -shared -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2 -ldb2apie

# Copy the shared library to the function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldudf のコンパイルおよびリンク・オプション

コンパイル・オプション

g++ C++ コンパイラー。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldudf のコンパイルおよびリンク・オプション
リンク・オプション
g++ コンパイラーをリンカーのフロントエンドとして使用します。
-o $1 実行可能ファイルを指定します。
$1.o プログラムのオブジェクト・ファイルを組み込みます。
-shared 共用ライブラリーを生成します。
-L$DB2PATH/lib リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。
-Wl,-rpath,$DB2PATH/lib 実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。
-ldb2 DB2 ライブラリーとリンクします。
-ldb2apie DB2 API エンジン・ライブラリーとリンクして、LOB ロケーターを使用できるようにします。
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。

ユーザー定義関数プログラム udfsrv をソース・ファイル udfsrv.c から作成するには、次のように入力します。

```
bldudf udfsrv
```

スクリプト・ファイルは、 UDF を sqllib/function ディレクトリーにコピーします。

必要であれば、UDF にファイル・モードを設定してクライアント・アプリケーションから実行できるようにします。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。 udfcli プログラムは、スクリプト・ファイル bldapp を使用して、 sqllib/samples/cpp にあるソース・ファイル udfcli.sqC から作成します。詳細については、 239ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF を呼び出すには、次のように実行可能ファイル名を入力して、サンプルの呼び出しアプリケーションを実行します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv ライブラリーの ScalarUDF 関数を呼び出します。

マルチスレッド・アプリケーション

Linux C++ を使用するマルチスレッド・アプリケーションは、 -D_REENTRANT でコンパイルし、 -lpthread とリンクする必要があります。

sqllib/samples/cpp にあるスクリプト・ファイル bldmt には、組み込み SQL マルチスレッド・プログラムを作成するためのコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。第 3 パラメーター $3 にはそのデータベースのユーザー ID を、また $4 にはパスワードを指定します。第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名、ユーザー ID、および、パスワードは任意指定です。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldmt script file -- Linux
# Builds a C++ multi-threaded embedded SQL program.
# Usage:  bldmt <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
embprep $1 $2 $3 $4
# Compile the program.
g++ -D_REENTRANT -I$DB2PATH/include -c $1.C
# Link the program.
g++ -lpthread -o $1 $1.o -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
```

上記の -D_REENTRANT オプションと -lpthread オプション、そしてリンクされているユーティリティー・ファイルがないという点だけでなく、残りのコンパイルとリンクのオプションも、組み込み SQL スクリプト・ファイル bldapp で使用されているものと同じです。これらのオプションについては、 239ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル thdsrver.sqC からサンプル・プログラム thdsrver を作成するには、次のように入力します。

```
bldmt thdsrver
```

結果として、実行可能ファイル thdsrver が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
thdsrver
```

第9章 OS/2 アプリケーションの構築

この章には、OS/2 上でアプリケーションを構築するための詳細情報が記載されています。コマンド・ファイルでは、db2 で始まるコマンドは、コマンド行プロセッサー (CLP) コマンドです。 CLP コマンドについての詳しい情報が必要であれば、 *コマンド解説書* を参照してください。

OS/2 に関する DB2 アプリケーション開発の最新の更新事項については、次の Web ページを参照してください。

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

注: ユーザー定義の SQLDA を含む複合 SQL ステートメントは、 OS/2 上の 16 ビット・アプリケーションでは許可されていません。

IBM VisualAge C++ for OS/2 バージョン 3

この節では以下のトピックを取り上げています。

- DB2 CLI アプリケーション
- DB2 CLI ストアード・プロシージャー
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー
- ユーザー定義関数 (UDF)

注: VisualAge C++ コンパイラーは、 %DB2PATH%¥samples¥c および %DB2PATH%¥samples¥cpp ディレクトリーにある C と C++ の両方のサンプル・プログラムに使用されます。これらのいずれのディレクトリーでも、同じコマンド・ファイルが使用されます。これらのファイルには、ファイルの拡張子に応じて、 C または C++ のいずれかのソース・ファイルを受け入れるコマンドが含まれています。

DB2 CLI アプリケーション

%DB2PATH%¥samples¥cli にある bldcli には、 DB2 CLI プログラムを作成するためのコマンドが入っています。このパラメーター %1 には、ソース・ファイルの名前を指定します。

必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない CLI プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

プログラムに組み込み SQL が含まれている場合 (拡張子が .sqc の場合) は、embprep コマンド・ファイルが呼び出されてそのプログラムをプリコンパイルし、 .c という拡張子のプログラム・ファイルを生成します。

```
@echo off
rem  bldcli command file - OS/2
rem  Builds a CLI program with IBM VisualAge C++.
rem  Usage: bldcli prog_name [ db_name [ userid password ]]
if exist "%1.sqc" call embprep %1 %2 %3 %4
if exist "%1.sqx" call embprep %1 %2 %3 %4
if "%1" == "" goto error
rem Compile the error-checking utility.

icc -C+ -O- -Ti+ utilcli.c
rem Compile the program.
if exist "%1.sqx" goto cpp
icc -C+ -O- -Ti+ %1.c

goto link_step
:cpp
icc -C+ -O- -Ti+ %1.cxx
rem Link the program.
:link_step
ilink /NOFREE /NOI /DEBUG /ST:64000 /PM:VIO %1.obj utilcli.obj,%1.exe,NUL,db2cli.lib;
goto exit
:error

echo Usage: bldcli prog_name [ db_name [ userid password ]]
:exit
@echo on
```

bldcli のコンパイルおよびリンク・オプション
コンパイル・オプション **icc** IBM VisualAge C++ コンパイラー。 **-C+** コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。 **-O-** 最適化なし。最適化をオフにしてデバッガーを使用する方が簡単です。 **-Ti+** デバッガー情報を生成します。
リンク・オプション **ilink** リンク編集に ilink リンカーを使用します。 **/NOFREE** 自由書式ではありません。 **/NOI** 大 / 小文字を無視しません。大文字小文字を区別する識別子を強制します。 **/DEBUG** デバッグ情報を組み込みます。 **/ST:64000** スタック・サイズとして少なくとも 64 000 を指定します。 **/PM:VIO** プログラムが OS/2 ウィンドウで稼働することを可能にします。 **%1.obj** オブジェクト・ファイルを組み込みます。 **utilcli.obj** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **%1.exe** 実行可能ファイルを指定します。 **NUL** デフォルト値を受け入れます。 **db2cli.lib** DB2 CLI ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル tbinfo.c からサンプル・プログラム tbinfo を構築するには、次のようにします。

```
bldcli tbinfo
```

結果として、実行可能ファイル tbinfo.exe が作成されます。この実行可能ファイルを実行するには、次の実行可能名 (拡張子なし) を入力します。

```
tbinfo
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル dbusemx.sqc から組み込み SQL アプリケーション dbusemx を作成する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

```
bldcli dbusemx
```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldcli dbusemx database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldcli dbusemx database userid password
   ```

結果として、実行可能ファイル dbusemx.exe が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前 (拡張子なし) を入力します。

   ```
   dbusemx
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   dbusemx database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   dbusemx database userid password
   ```

DB2 API を使用する DB2 CLI アプリケーション

DB2 には、CLI サンプル・プログラムが含まれています。このサンプル・プログラムは、DB2 API を使用してデータベースを作成およびドロップし、 CLI 機能を複数のデータベースで使用する方法を示します。 DB2 API を使用するサンプルは、 27ページの表7 にある CLI サンプル・プログラムの説明の中に示されています。

%DB2PATH%¥samples¥cli のコマンド・ファイル bldapi には、 DB2 API を使用する DB2 CLI プログラムを構築するコマンドが入っています。このファイルは、データベースを作成およびドロップするための DB2 API が入った utilapi ユーティリティー・ファイルでコンパイルおよびリンクします。この点が、このファイルと bldcli コマンド・ファイルの唯一の違いです。 bldapi と bldcli の両方に共通するコンパイルとリンクのオプションについては、252ページの『DB2 CLI アプリケーション』 を参照してください。

ソース・ファイル dbmconn.c からサンプル・プログラム dbmconn を作成するには、次のようにします。

```
bldapi dbmconn
```

結果として、実行可能ファイル dbmconn.exe が作成されます。この実行可能ファイルを実行するには、次の実行可能名 (拡張子なし) を入力します。

```
dbmconn
```

DB2 CLI ストアード・プロシージャー

%DB2PATH%¥samples¥cli にあるコマンド・ファイル bldclisp には、 CLI ストアード・プロシージャーを作成するためのコマンドが入っています。このコマンド・ファイルは、ストアード・プロシージャーをサーバー上の DLL 内に作成します。

このパラメーター %1 には、ソース・ファイルの名前を指定します。コマンド・ファイルでは、ソース・ファイル名 %1 を DLL 名に使用します。

```
@echo off
rem  bldclisp command file - OS/2
rem Builds a CLI stored procedure using the IBM VisualAge C++ compiler.
rem  Usage: bldclisp <prog_name>
if "%1" == "" goto error
rem Compile the error-checking utility.

icc -C+ -Ti+ -Ge- -Gm+ -W2 utilcli.c
rem Compile the program.
if exist "%1.cxx" goto cpp
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.c
goto link_step
:cpp
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.cxx
:link_step
rem Link the program and produce a DLL.
ilink /NOFREE /MAP /NOI /DEBUG /ST:64000 %1.obj utilcli.obj,%1.dll,,db2cli.lib,%1.def;
rem Copy the stored procedure DLL to the 'function' directory
copy %1.dll %DB2PATH%¥function
goto exit
:error

echo Usage: bldclisp prog_name
:exit
@echo on
```

bldclisp のコンパイルおよびリンク・オプション

コンパイル・オプション

icc IBM VisualAge C++ コンパイラー。

-C+ コンパイルのみを実行し、リンクは実行しません。このコマンド・ファイルでは、コンパイルとリンクは別個のステップです。

-Ti+ デバッガー情報を生成します。

-Ge- .DLL ファイルを作成します。静的にリンクされる実行時ライブラリーのバージョンを使用します。

-Gm+ マルチタスキング・ライブラリーとリンクします。

-W2 警告、エラー、重大、および回復不能エラー・メッセージを出力します。

リンク・オプション

ilink リンク編集に ilink リンカーを使用します。

/NOFREE
自由書式ではありません。

/MAP マップ・ファイルを生成します。

/NOI 大 / 小文字を無視しません。大文字小文字を区別する識別子を強制します。

/DEBUG デバッグ情報を組み込みます。

/ST:64000
スタック・サイズとして少なくとも 64000 を指定します。

%1.obj オブジェクト・ファイルを組み込みます。

%1.dll ダイナミック・リンク・ライブラリーを作成します。

db2cli.lib
DB2 CLI ライブラリーとリンクします。

%1.def モジュール定義ファイル。

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。

ソース・ファイル spserver.c からサンプル・プログラム spserver を構築するには、次のように入力します。

```
bldclisp spserver
```

スクリプト・ファイルは、共用ライブラリーをサーバー上の %DB2PATH%¥function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、 CLI クライアント・アプリケーション spclient を構築することができます。これは、共用ライブラリー内のストアード・プロシージャーを呼び出すアプリケーションです。

spclient は、コマンド・ファイル bldcli を使用して構築することができます。詳細については、 252ページの『DB2 CLI アプリケーション』を参照してください。

共用ライブラリーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

DB2 API と組み込み SQL アプリケーション

%DB2PATH%¥samples¥c と %DB2PATH%¥samples¥cpp にあるコマンド・ファイル bldapp.cmd には、 DB2 アプリケーション・プログラムを作成するためのコマンドが入っています。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を含まないプログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは %2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは %3 で、データベースのユーザー ID を指定します。そしてもう 1 つが %4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのコマンド・ファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
@echo off
rem bldapp command file -- OS/2
rem Builds a VisualAge C++ application program
rem Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
if exist "%1.sqx" goto embedded
if exist "%1.sqc" goto embedded
goto non_embedded
:embedded
rem Precompile and bind the program.
call embprep %1 %2 %3 %4
rem Compile the program.
if exist "%1.cxx" goto cpp_embedded
icc -c utilemb.c
icc -C+ -O- -Ti+ %1.c

goto link_embedded
:cpp_embedded
icc -c utilemb.cxx
icc -C+ -O- -Ti+ %1.cxx
goto link_embedded
:non_embedded
rem Compile the program.
if exist "%1.cxx" goto cpp
icc -c utilapi.c
icc -C+ -O- -Ti+ %1.c

goto link_non_embedded
:cpp
icc -c utilapi.cxx
```

```
icc -C+ -O- -Ti+ %1.cxx
goto link_non_embedded
rem Link the program.
:link_embedded
ilink /NOFREE /NOI /DEBUG /ST:64000 /PM:VIO %1.obj utilemb.obj,,,db2api;
goto exit
:link_non_embedded
ilink /NOFREE /NOI /DEBUG /ST:64000 /PM:VIO %1.obj utilapi.obj,,,db2api;
:exit
@echo on
```

bldapp のコンパイルおよびリンク・オプション

コンパイル・オプション

icc IBM VisualAge C++ コンパイラー。

-C+ コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

-O- 最適化なし。最適化をオフにしてデバッガーを使用する方が簡単です。

-Ti+ デバッガー情報を生成します。

リンク・オプション

ilink リンク編集に ilink リンカーを使用します。

/NOFREE
自由書式ではありません。

/NOI 大 / 小文字を無視しません。大文字小文字を区別する識別子を強制します。

/DEBUG デバッグ情報を組み込みます。

/ST:64000
スタック・サイズとして少なくとも 64000 を指定します。

/PM:VIO
プログラムが OS/2 ウィンドウで稼働することを可能にします。

utilemb.obj
組み込み SQL プログラムの場合に、エラー・チェックを行う組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。

utilapi.obj
組み込み SQL プログラムでない場合に、エラー・チェックを行う DB2 API ユーティリティー・オブジェクト・ファイルを含みます。

db2api DB2 ライブラリーとリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.c から DB2 API 非組み込み SQL サンプル・プログラム client を構築するには、次のようにします。

```
bldapp client
```

結果として、実行可能ファイル client.exe が作成されます。

この実行可能ファイルを実行するには、実行可能ファイル名 (拡張子なし) を入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqc から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat.exe が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前 (拡張子なし) を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

%DB2PATH%¥samples¥c と %DB2PATH%¥samples¥cpp にあるコマンド・ファイル bldsrv には、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが入っています。コマンド・ファイルは、ストアード・プロシージャーをサーバー上の DLL にコンパイルします。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。第 2 パラメーター %2 には、接続先のデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

コマンド・ファイルでは、ソース・ファイル名 %1 を DLL 名に使用します。

```
@echo off
rem bldsrv command file -- OS/2
rem Builds a VisualAge C++ stored procedure
rem Usage: bldsrv <prog_name> [ <db_name> ]
rem Precompile and bind the program.
call embprep %1 %2
rem Compile the program.
if exist "%1.cxx" goto cpp
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.c
goto link_step
:cpp
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.cxx
:link_step
rem Link the program.
ilink /NOFREE /NOI /DEBUG /ST:64000 %1.obj,%1.dll,,db2api,%1.def;
rem Copy the stored procedure to the %DB2PATH%¥function directory.
copy %1.dll %DB2PATH%¥function
@echo on
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

icc IBM VisualAge C++ コンパイラー。

-C+ コンパイルのみを実行し、リンクは実行しません。このコマンド・ファイルでは、コンパイルとリンクは別個のステップです。

-Ti+ デバッガー情報を生成します。

-Ge- .DLL ファイルを作成します。静的にリンクされる実行時ライブラリーのバージョンを使用します。

-Gm+ マルチタスキング・ライブラリーとリンクします。

-W2 警告、エラー、重大、および回復不能エラー・メッセージを出力します。

bldsrv のコンパイルおよびリンク・オプション
リンク・オプション **ilink**　リンク編集に ilink リンカーを使用します。 **/NOFREE** 自由書式ではありません。 **/NOI**　大 / 小文字を無視しません。大文字小文字を区別する識別子を強制します。 **/DEBUG**　デバッグ情報を組み込みます。 **/ST:64000** スタック・サイズとして少なくとも 64000 を指定します。 **%1.dll**　ダイナミック・リンク・ライブラリーを作成します。 **db2api**　DB2 ライブラリーとリンクします。 **%1.def**　モジュール定義ファイル。 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。

sample データベースに接続している場合に、ソース・ファイル spserver.sqc からサンプル・プログラム spserver を構築するには、次のように入力します。

```
bldsrv spserver
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv spserver database
```

コマンド・ファイルは、共用ライブラリーをサーバー上の %DB2PATH%¥function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、共用ライブラリーにアクセスするクライアント・アプリケーション spclient を構築することができます。

spclient は、コマンド・ファイル bldapp を使用して構築することができます。詳細については、 258ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

ユーザー定義関数 (UDF)

%DB2PATH%¥samples¥c と %DB2PATH%¥samples¥cpp にあるコマンド・ファイル bldudf には、 UDF を作成するためのコマンドが入っています。

UDF に組み込み SQL ステートメントを含めることはできません。このため、UDF プログラムを作成する際に、データベースに接続してプログラムをプリコンパイルおよびバインドすることはしません。

コマンド・ファイルは、ソース・ファイルの名前を指定する、 %1 というパラメーターを取ります。ソース・ファイル名 %1 を DLL 名に使用します。

```
@echo off
rem  bldudf command file -- OS/2
rem Builds a VisualAge C++ user-defined function (UDF)
rem  Usage: bldudf <prog_name>
if "%1" == "" goto error
rem Compile the program.
if exist "%1.cxx" goto cpp
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.c
goto link_step
:cpp
rem icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.cxx
:link_step
rem Link the program.
ilink /NOFREE /MAP /NOI /DEBUG /ST:64000 %1.obj,%1.dll,,db2api db2apie,%1.def;
rem Copy the UDF to the %DB2PATH%¥function directory
copy %1.dll %DB2PATH%¥function
goto exit
:error

echo Usage: bldudf prog_name
:exit
@echo on
```

bldudf のコンパイルおよびリンク・オプション

コンパイル・オプション

icc IBM VisualAge C++ コンパイラー。

-C+ コンパイルのみを実行し、リンクは実行しません。このコマンド・ファイルでは、コンパイルとリンクは別個のステップです。

-Ti+ デバッガー情報を生成します。

-Ge- .DLL ファイルを作成します。静的にリンクされる実行時ライブラリーのバージョンを使用します。

-Gm+ マルチタスキング・ライブラリーとリンクします。

-W2 警告、エラー、重大、および回復不能エラー・メッセージを出力します。

bldudf のコンパイルおよびリンク・オプション
リンク・オプション **ilink** リンク編集に ilink リンカーを使用します。 **/NOFREE** 自由書式ではありません。 **/MAP** マップ・ファイルを生成します。 **/NOI** 大 / 小文字を無視しません。大文字小文字を区別する識別子を強制します。 **/DEBUG** デバッグ情報を組み込みます。 **/ST:64000** スタック・サイズとして少なくとも 64000 を指定します。 **%1.dll** ダイナミック・リンク・ライブラリーを作成します。 **db2api** DB2 ライブラリーとリンクします。 **db2apie** DB2 API エンジン・ライブラリーとリンクします。 **%1.def** モジュール定義ファイル。 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。

ユーザー定義関数プログラム udfsrv をソース・ファイル udfsrv.c から作成するには、次のように入力します。

```
bldudf udfsrv
```

スクリプト・ファイルは、 UDF をサーバー上の %DB2PATH%¥function というパスにコピーします。

必要であれば、UDF にファイル・モードを設定してクライアント・アプリケーションから実行できるようにします。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。このプログラムには DB2 CLI バージョンと組み込み SQL バージョンがあります。

DB2 CLI udfcli プログラムは、コマンド・ファイル bldcli.cmd を使用して、 %DB2PATH%¥samples¥cli の udfcli.c ソース・ファイルから作成できます。詳細については、252ページの『DB2 CLI アプリケーション』 を参照してください。

組み込み SQL udfcli プログラムは、コマンド・ファイル bldapp を使用して、 %DB2PATH%¥samples¥c にあるソース・ファイル udfcli.sqc から構築することができます。詳細については、258ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF を呼び出すには、次のように実行可能ファイル名 (拡張子なし) を入力して、サンプルの呼び出しアプリケーションを実行します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv ライブラリーから ScalarUDF 関数を呼び出します。

IBM VisualAge C++ for OS/2 バージョン 4.0

VisualAge C++ バージョン 4 コンパイラーのアプリケーション構築情報は、AIX、OS/2 および Windows 32 ビット・オペレーティング・システムで共通です。この情報については、152ページの『VisualAge C++ バージョン 4.0』を参照してください。

IBM VisualAge COBOL for OS/2

この節では以下のトピックを取り上げています。

- コンパイラーの使用
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー

コンパイラーの使用

以下に記載する点は、DB2 で IBM VisualAge COBOL を使用する場合に役立ちます。

バインド・ファイル作成時の解決策

DB2 (OS/2 版) および IBM COBOL を使用するアプリケーションを構築すると、 DB2 プリコンパイラーがバインド・ファイルの作成に失敗することがよくあります。原因は、OS/2 内のファイル処理限界にあります。

修正を実行すると、OS/2 は、コンパイルを実行するマシン上でさらに多くのファイルを処理できるようになります。 DB2 (OS/2 版) がインストールされているマシンの CONFIG.SYS ファイルに、次の行を挿入する必要があります。

```
SET SHELLHANDLESINC=20
```

コンパイル時に NODATA オプション (IBM Cobol オプション) を使用することもできます。

組み込み SQL および DB2 API 呼び出し

組み込み SQL および DB2 API 呼び出しを含むアプリケーションを開発していて、 IBM VisualAge COBOL コンパイラーを使用している場合には、以下の点に留意してください。

- コマンド行プロセッサーのコマンド db2 prep を使ってアプリケーションをプリコンパイルする場合は、 target ibmcob オプションを使ってください。
- ソース・ファイルの中でタブ文字を使用しないでください。
- ソース・ファイルで PROCESS および CBL キーワードを使用して、コンパイル・オプションを設定できます。キーワードは 8〜72 桁目だけに置いてください。
- アプリケーションに組み込み SQL のみが含まれていて、 DB2 API 呼び出しは含まれない場合には、 pgmname(mixed) コンパイル・オプションを使う必要はありません。 DB2 API 呼び出しを使用する場合には、 pgmname(mixed) コンパイル・オプションを使う必要があります。
- IBM VisualAge COBOL コンパイラーの「システム/390 ホスト・データ型サポート」機能を使用している場合、アプリケーション用の DB2 組み込みファイルは、次のディレクトリー中にあります。

  ```
  %DB2PATH%¥include¥cobol_i
  ```

  提供されたスクリプト・ファイルを使って DB2 サンプル・プログラムを作成している場合、コマンド・ファイルで指定された組み込みファイルのパスは、 cobol_a ディレクトリーではなく、 cobol_i ディレクトリーを指すように変更しなければなりません。

  IBM VisualAge COBOL コンパイラーの「システム/390 ホスト・データ型サポート」機能を使用していない場合、またはこのコンパイラーのそれよりも前のバージョンを使用している場合、アプリケーション用の DB2 組み込みファイルは、次のディレクトリー中にあります。

  ```
  %DB2PATH%¥include¥cobol_a
  ```

  .cbl 拡張子を含めるには、次のようにして COPY ファイル名を指定します。

  ```
  COPY "sql.cbl".
  ```

組み込み SQL アプリケーション

%DB2PATH%¥samples¥cobol にあるコマンド・ファイル bldapp.cmd には、 DB2 アプリケーション・プログラムを作成するためのコマンドが入っています。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を含まないプログラムに必要な唯一のパラメーターです。組み込

み SQL プログラムを作成するためにはデータベースへの接続が必要なため、3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは %2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは %3 で、データベースのユーザー ID を指定します。そしてもう 1 つが %4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのコマンド・ファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
@echo off
rem bldapp command file -- OS/2
rem Builds a VisualAge COBOL application program
rem Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
rem If an embedded SQL program, precompile and bind it.
if exist "%1.sqb" goto prepbind
goto compile_step
:prepbind
call embprep %1 %2 %3 %4
:compile_step
rem Compile the checkerr error checking utility.
cob2 -c -g -qpgmname(mixed) -qlib -I%DB2PATH%¥include¥cobol_a checkerr.cbl
rem Compile the program.
cob2 -c -g -qpgmname(mixed) -qlib -I%DB2PATH%¥include¥cobol_a %1.cbl
rem Link the program.
ilink %1.obj checkerr.obj db2api.lib /ST:64000 /PM:VIO /NOI /DEBUG
@echo on
```

bldapp のコンパイルおよびリンク・オプション

コンパイル・オプション

cob2 IBM VisualAge COBOL コンパイラー。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

-g デバッグ情報を組み込みます。

-qpgmname(mixed)
コンパイラーに、大文字小文字混合の名前を持つライブラリー入り口点の CALL を許可するように指示します。

-qlib コンパイラーに COPY ステートメントを処理するように指示します。

-I*path* DB2 インクルード・ファイルのロケーションを指定します。たとえば、-I%DB2PATH%¥include¥cobol_a。

bldapp のコンパイルおよびリンク・オプション
リンク・オプション **ilink**　リンク編集に ilink リンカーを使用します。 **checkerr.obj** エラー検査ユーティリティー・オブジェクト・ファイルを組み込みます。 **db2api.lib** DB2 ライブラリーとリンクします。 **/ST:64000** スタック・サイズとして少なくとも 64000 を指定します。 **/PM:VIO** プログラムが OS/2 ウィンドウで稼働することを可能にします。 **/NOI**　リンク時に大 / 小文字を無視しません。 **/DEBUG**　デバッグ情報を組み込みます。 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。

ソース・ファイル client.cbl から非組み込み SQL サンプル・プログラム client を作成するには、次のように入力します。

```
bldapp client
```

結果として、実行可能ファイル client.exe が作成されます。この実行可能ファイルを sample データベースに対して実行するには、次の実行可能ファイル名 (ファイル拡張子なし) を入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqb から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat.exe が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイル名 (ファイル拡張子なし) を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

%DB2PATH%¥samples¥cobol にあるコマンド・ファイル bldsrv には、ストアード・プロシージャーを作成するためのコマンドが入っています。コマンド・ファイルは、ストアード・プロシージャーをサーバー上の DLL にコンパイルします。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。第 2 パラメーター %2 には、接続先のデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

コマンド・ファイルでは、ソース・ファイル名 %1 を DLL 名に使用します。

```
@echo off
rem bldsrv command file -- OS/2
rem Builds a VisualAge COBOL stored procedure
rem Usage: bldsrv <prog_name> [ <db_name> ]
rem Precompile and bind the program.
call embprep %1 %2
rem Compile the program.
cob2 -c -g -qpgmname(mixed) -qlib -I%DB2PATH%¥include¥cobol_a %1.cbl
rem Link the program.
```

```
ilink %1.obj checkerr.obj %1.def db2api.lib /ST:64000 /PM:VIO /NOI /DEBUG
rem Copy stored procedure to the %DB2PATH%¥function directory.
copy %1.dll %DB2PATH%¥function
@echo on
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

cob2 IBM VisualAge COBOL コンパイラー。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

-g デバッグ情報を組み込みます。

-qpgmname(mixed)
コンパイラーに、大文字小文字混合の名前を持つライブラリー入り口点の CALL を許可するように指示します。

-qlib コンパイラーに COPY ステートメントを処理するように指示します。

-I*path* DB2 インクルード・ファイルのロケーションを指定します。たとえば、-I%DB2PATH%¥include¥cobol_a。

リンク・オプション

ilink リンク編集に ilink リンカーを使用します。

checkerr.obj
エラー検査ユーティリティー・オブジェクト・ファイルを組み込みます。

%1.def モジュール定義ファイル。

db2api.lib
DB2 ライブラリーとリンクします。

/ST:64000
スタック・サイズとして少なくとも 64000 を指定します。

/PM:VIO
プログラムが OS/2 ウィンドウで稼働することを可能にします。

/NOI リンク時に大 / 小文字を無視しません。

/DEBUG デバッグ情報を組み込みます。

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。

サンプル・データベースに接続している場合に、ソース・ファイル outsrv.sqb からサンプル・プログラム outsrv を構築するには、次のように入力します。

```
bldsrv outsrv
```

他のデータベースに接続しているときは、さらにデータベース名も含めます。

```
bldsrv outsrv database
```

このコマンド・ファイルは、サンプル・プログラムと同じディレクトリーに入っている、モジュール定義ファイル outsrv.def を使用して DLL を作成します。このコマンド・ファイルは、ストアード・プロシージャー DLL の outsrv.dll をサーバー上の %DB2PATH%¥function というパスにコピーします。

必要であれば、DLL にファイル・モードを設定して、クライアント・アプリケーションからアクセスできるようにします。

DLL outsrv を構築してしまえば、その DLL にアクセスするクライアント・アプリケーション outcli を構築できます。 outcli は、コマンド・ファイル bldapp を使用して構築することができます。詳細については、 267ページの『組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してクライアント・アプリケーションを実行します。

```
outcli database userid password
```

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample またはそのリモート別名、あるいはその他の名前にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは DLL outsrv にアクセスし、ストアード・プロシージャー関数をサーバー・データベース上で実行します。この出力は、クライアント・アプリケーションに戻されます。

Micro Focus COBOL

この節では以下のトピックを取り上げています。

- コンパイラーの使用
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー

コンパイラーの使用

DB2 は、 Micro Focus COBOL コンパイラーに付属している link386 リンカーをサポートしていません。 DB2 Micro Focus COBOL プログラムにリンク

するには、 IBM コンパイラー製品から入手できる ilink リンカーを使用する必要があります。この節のスクリプト・ファイルで使用されている cbllink コマンドで、 ilink リンカーを呼び出します。

組み込み SQL および DB2 API 呼び出しを含むアプリケーションを Micro Focus COBOL コンパイラーを使用して作成している場合には、以下の点に留意してください。

- コマンド行プロセッサーのコマンド db2 prep を使ってアプリケーションをプリコンパイルする場合は、 target mfcob オプション (デフォルト) を使用してください。
- 以下のように入力して、必ず LIB 環境変数が %DB2PATH%¥lib を指すようにしてください。

  ```
  set LIB=%DB2PATH%¥lib;%LIB%
  ```

- Micro Focus COBOL 用の DB2 COPY ファイルは、%DB2PATH%¥include¥cobol_mf にあります。 COBCPY 環境変数を、以下のようにディレクトリーを含めて設定してください。

  ```
  set COBCPY=%DB2PATH%¥include¥cobol_mf;%COBCPY%
  ```

すべての DB2 アプリケーション・プログラミング・インターフェースへの呼び出しは、呼び出し規則 8 を使用して実行しなければなりません。 DB2 COBOL プリコンパイラーは、自動的に CALL-CONVENTION 節を SPECIAL-NAMES 段落に挿入します。 SPECIAL-NAMES 段落が存在しない場合、 DB2 COBOL プリコンパイラーはそれを以下のように作成します。

```
Identification Division
Program-ID. "static".
special-names.
       call-convention 8 is DB2API.
```

さらにプリコンパイラーは、呼び出し規則を識別するために使用する記号 DB2API を、 DB2 API が呼び出されるたびに必ず call キーワードの後に自動的に置きます。これはたとえば、プリコンパイラーが DB2 API 実行時呼び出しを組み込み SQL ステートメントから生成する場合にも必ず実行されます。

DB2 API への呼び出しをプリコンパイルされていないアプリケーションで実行する場合、前述したものと同様の SPECIAL-NAMES 段落を、手操作で入力してアプリケーションに作成する必要があります。 DB2 API を直接呼び出す場合は、 call キーワードの後に DB2API 記号を手動で追加する必要があります。

DB2 API と組み込み SQL アプリケーション

%DB2PATH%¥samples¥cobol_mf にあるコマンド・ファイル bldapp には、 DB2 アプリケーション・プログラムを作成するためのコマンドが入っています。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を含まないプログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは %2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは %3 で、データベースのユーザー ID を指定します。そしてもう 1 つが %4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのコマンド・ファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
@echo off
rem bldapp command file -- OS/2
rem Builds a Micro Focus COBOL application program
rem Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
rem If an embedded SQL program, precompile and bind it.
if exist "%1.sqb" goto prepbind
goto compile_step
:prepbind
call embprep %1 %2 %3 %4
:compile_step
rem Compile the error-checking utility.

cobol checkerr.cbl;
rem Compile the program.
cobol %1.cbl;
rem Link the program.
cbllink %1.obj checkerr.obj db2api.lib db2gmf32.lib

@echo on
```

bldapp のコンパイルおよびリンク・オプション
コンパイル・オプション
cobol　Micro Focus COBOL コンパイラー。

bldapp のコンパイルおよびリンク・オプション
リンク・オプション **cbllink** リンク編集にリンカーを使用します。 **checkerr.obj** エラー検査ユーティリティー・オブジェクト・ファイルを組み込みます。 **db2api.lib** DB2 API ライブラリーとリンクします。 **db2gmf32.lib** M. F. COBOL 用 DB2 例外ハンドラー・ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.cbl から非組み込み SQL サンプル・プログラム client を作成するには、次のように入力します。

```
bldapp client
```

結果として、実行可能ファイル client.exe が作成されます。この実行可能ファイルを sample データベースに対して実行するには、次の実行可能ファイル名 (ファイル拡張子なし) を入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqb から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat.exe が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイル名 (ファイル拡張子なし) を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

%DB2PATH%¥samples¥cobol_mf にあるコマンド・ファイル bldsrv には、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが入っています。コマンド・ファイルは、ストアード・プロシージャーをサーバー上の DLL にコンパイルします。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。第 2 パラメーター %2 には、接続先のデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。コマンド・ファイルでは、ソース・ファイル名 %1 を DLL 名に使用します。

```
@echo off
rem bldsrv command file -- OS/2
rem Builds a Micro Focus COBOL stored procedure
rem Usage: bldsrv <prog_name> [ <db_name> ]
rem Precompile and bind the program.
call embprep %1 %2
rem  Compile the stored procedure.
cobol %1.cbl;
rem  Link the stored procedure and create a shared library.
cbllink /d %1.obj db2api.lib db2gmf32.lib
rem Copy the stored procedure to the %DB2PATH%¥function directory.
copy %1.dll %DB2PATH%¥function
@echo on
```

bldsrv のコンパイルおよびリンク・オプション
コンパイル・オプション **cobol**　Micro Focus COBOL コンパイラー。
リンク・オプション **cbllink** リンク編集のために Micro Focus COBOL リンカーを使用します。 **/d**　.DLL ファイルを作成します。 **db2api.lib** DB2 API ライブラリーを組み込みます。 **db2gmf32.lib** M. F. COBOL 用 DB2 例外ハンドラー・ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

サンプル・データベースに接続している場合に、ソース・ファイル outsrv.sqb からサンプル・プログラム outsrv を構築するには、次のように入力します。

```
bldsrv outsrv
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv outsrv database
```

リンカーは、ユーザーによって指定されていないデフォルトの入り口点を使用します。ストアード・プロシージャーを作成するために、 /d オプションを使用して .dll ファイルが作成されます。このコマンド・ファイルは、ストアード・プロシージャー DLL の outsrv.dll をサーバー上の %DB2PATH%¥function というパスにコピーします。

必要であれば、DLL にファイル・モードを設定して、クライアント・アプリケーションからアクセスできるようにします。

DLL outsrv を構築してしまえば、その DLL にアクセスするクライアント・アプリケーション outcli を構築できます。 outcli は、コマンド・ファイル bldapp を使用して構築することができます。詳細については、 274ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

```
outcli database userid password
```

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは DLL outsrv にアクセスし、ストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

REXX

REXX プログラムはコンパイルまたはバインドしません。

OS/2 上でアプリケーション・ファイルには、拡張子 .cmd がなければなりません。作成後、オペレーティング・システムのコマンド・プロンプトから直接アプリケーションを実行することが可能です。

OS/2 REXX プログラムには、コメントをバッチ・コマンドと区別するための、第 1 行の第 1 列から始まる以下のコメントが入っていなければなりません。

```
/* Any comment will do. */
```

REXX サンプル・プログラムは、ディレクトリー %DB2PATH%¥samples¥rexx にあります。サンプル REXX プログラム updat を実行するには、次のように入力します。

```
updat
```

REXX および DB2 の詳細については、 *アプリケーション開発の手引き* の『REXX でのプログラミング』を参照してください。

第10章 DYNIX/ptx アプリケーションの構築

この章では、 NUMA-Q 用の DB2 を使用した DYNIX/ptx アプリケーションの構築について詳しく説明します。スクリプト・ファイルにおいて、db2 から始まるコマンドは、コマンド行プロセッサー (CLP) のコマンドです。 CLP コマンドについての詳細な情報が必要であれば、 *コマンド解説書* を参照してください。

DYNIX/ptx に関する DB2 アプリケーション開発の最新の更新事項については、次の Web ページを参照してください。

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

ptx/C

この節では、以下のトピックを取り上げています。

- DB2 CLI アプリケーション
- DB2 API を使用する DB2 CLI アプリケーション
- DB2 CLI ストアード・プロシージャー
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー
- ユーザー定義関数 (UDF)
- マルチスレッド・アプリケーション

DB2 CLI アプリケーション

sqllib/samples/cli のスクリプト・ファイル bldcli には、 DB2 CLI プログラムを作成するためのコマンドが入っています。パラメーター $1 には、ソース・ファイルの名前を指定します。

必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない CLI プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

プログラムに組み込み SQL が含まれている場合 (拡張子が .sqc の場合) は、embprep スクリプトが呼び出されてそのプログラムをプリコンパイルし、 .c という拡張子のプログラム・ファイルを生成します。

```
#! /bin/ksh
# bldcli script file -- DYNIX/ptx
# Builds a DB2 CLI program
# Usage: bldcli <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
fi

# Compile the error-checking utility.
cc -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o utilcli.o -L$DB2PATH/lib -ldb2
```

bldcli のコンパイルおよびリンク・オプション
コンパイル・オプション **cc** C コンパイラーを使用します。 **-I$DB2PATH/include** DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include のように指定します。 **-c** コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。
リンク・オプション **cc** コンパイラーをリンカーのフロントエンドとして使用します。 **-o $1** 実行可能プログラムを指定します。 **$1.o** プログラムのオブジェクト・ファイルを組み込みます。 **utilcli.o** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル tbinfo.c からサンプル・プログラム tbinfo を構築するには、次のようにします。

```
bldcli tbinfo
```

結果として、実行可能ファイル tbinfo が生成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
tbinfo
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル dbusemx.sqc から組み込み SQL アプリケーション dbusemx を作成する場合、次の 3 つの方法があります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

```
bldcli dbusemx
```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldcli dbusemx database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldcli dbusemx database userid password
   ```

結果として、実行可能ファイル dbusemx が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   dbusemx
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   dbusemx database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   dbusemx database userid password
   ```

DB2 API を使用する DB2 CLI アプリケーション

DB2 には、CLI サンプル・プログラムが含まれています。このサンプル・プログラムは、DB2 API を使用してデータベースを作成およびドロップし、 CLI 機能を複数のデータベースで使用する方法を示します。 DB2 API を使用するサンプルは、 27ページの表7 にある CLI サンプル・プログラムの説明の中に示されています。

sqllib/samples/cli にあるスクリプト・ファイル bldapi には、 DB2 API を使用して DB2 CLI プログラムを作成するためのコマンドが入っています。このファイルは、データベースを作成およびドロップするための DB2 API が入った utilapi ユーティリティー・ファイルでコンパイルおよびリンクします。この点が、このスクリプト・ファイルと bldcli スクリプトの唯一の違いです。 bldapi と bldcli の両方に共通するコンパイルとリンクのオプションについては、 280ページの『DB2 CLI アプリケーション』 を参照してください。

ソース・ファイル dbmconn.c からサンプル・プログラム dbmconn を作成するには、次のようにします。

```
bldapi dbmconn
```

結果として、実行可能ファイル dbmconn が作成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
dbmconn
```

DB2 CLI ストアード・プロシージャー

sqllib/samples/cli のスクリプト・ファイル bldclisp には、 DB2 CLI ストアード・プロシージャーを構築するコマンドが入っています。パラメーター $1 には、ソース・ファイルの名前を指定します。

```
#! /bin/ksh
# bldclisp script file -- DYNIX/ptx
# Builds a DB2 CLI stored procedure
# Usage: bldclisp <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the error-checking utility.
cc -KPIC -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc -KPIC -I$DB2PATH/include -c $1.c

# Link the program.
cc -G -o $1 $1.o utilcli.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldclisp のコンパイルおよびリンク・オプション
コンパイル・オプション **cc** C コンパイラー。 **-KPIC** 共用ライブラリー用の位置独立コードを生成します。 **-I$DB2PATH/include** DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。 **-c** コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。
リンク・オプション **cc** コンパイラーをリンカーのフロントエンドとして使用します。 **-G** 共用ライブラリーを生成します。 **-o $1** 実行可能ファイルを指定します。 **$1.o** プログラムのオブジェクト・ファイルを組み込みます。 **utilcli.o** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル spserver.c からサンプル・プログラム spserver を構築するには、次のように入力します。

```
bldclisp spserver
```

スクリプト・ファイルは、共用ライブラリーをサーバー上の sqllib/function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、共用ライブラリーにアクセスする CLI クライアント・アプリケーション spclient を構築することができます。

spclient は、スクリプト・ファイル bldcli を使用して構築することができます。詳細については、 280ページの『DB2 CLI アプリケーション』を参照してください。

共用ライブラリーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。この出力は、クライアント・アプリケーションに戻されます。

DB2 API と組み込み SQL アプリケーション

sqllib/samples/c のビルド・ファイル bldapp には、 DB2 API と組み込み SQL プログラムを構築するコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない DB2 API プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- DYNIX/ptx
# Builds a C application program.
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to the location where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# if an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
embprep $1 $2 $3 $4
  # Compile the utilemb.c error-checking utility.
  cc -I$DB2PATH/include -c utilemb.c
else
  # Compile the utilapi.c error-checking utility.
  cc -I$DB2PATH/include -c utilapi.c
fi
# Compile the program.
cc -I$DB2PATH/include -c $1.c

if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o
  cc -o $1 $1.o utilemb.o -L$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o
  cc -o $1 $1.o utilapi.o -L$DB2PATH/lib -ldb2
fi
```

bldapp のコンパイルおよびリンク・オプション
コンパイル・オプション **cc** C コンパイラー。 **-I$DB2PATH/include** DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。 **-c** コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。
リンク・オプション **cc** コンパイラーをリンカーのフロントエンドとして使用します。 **-o $1** 実行可能ファイルを指定します。 **$1.o** プログラムのオブジェクト・ファイルを組み込みます。 **util.o** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.c から非組み込み SQL サンプル・プログラム client を作成するには、次のように入力します。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。

この実行可能ファイルを実行するには、ファイル名を入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqc から組み込み SQL アプリケーション updat を構築する場合、次の 3 つの方法があります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

sqllib/samples/c にあるスクリプト・ファイル bldsrv には、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが含まれています。スクリプト・ファイルは、ストアード・プロシージャーを共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターは必要ありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldsrv script file -- DYNIX/ptx
# Builds a C stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2

# Compile the program.
cc -KPIC -I$DB2PATH/include -c $1.c

# Link the program and create a shared library.
cc -G -o $1 $1.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

cc C コンパイラー。

-KPIC 共用ライブラリー用の位置独立コードを生成します。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、-I$DB2PATH/include。

-c コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。

bldsrv のコンパイルおよびリンク・オプション
リンク・オプション
cc コンパイラーをリンカーのフロントエンドとして使用します。
-G 共用ライブラリーを生成します。
-o $1 実行可能ファイルを指定します。
$1.o プログラムのオブジェクト・ファイルを組み込みます。
-L$DB2PATH/lib リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。
-ldb2 DB2 ライブラリーとリンクします。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

sample データベースに接続している場合に、ソース・ファイル spserver.sqc からサンプル・プログラム spserver を構築するには、次のように入力します。

```
bldsrv spserver
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv spserver database
```

スクリプト・ファイルは、共用ライブラリーをサーバー上の sqllib/function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、この共用ライブラリーにアクセスするクライアント・アプリケーション spclient を構築することができます。

spclient は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 285ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

共用ライブラリーにアクセスするには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

ユーザー定義関数 (UDF)

sqllib/samples/c のスクリプト・ファイル bldudf には、 UDF を作成するためのコマンドが含まれています。 UDF には組み込み SQL ステートメントは含まれていません。したがって、UDF プログラムを作成する際に、データベースに接続したり、プログラムをプリコンパイルおよびバインドする必要はありません。

パラメーター $1 には、ソース・ファイルの名前を指定します。スクリプト・ファイルは、そのソース・ファイル名を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldudf script file -- DYNIX/ptx
# Builds a C user-defined function library
# Usage: bldudf <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the program.
cc -KPIC -I$DB2PATH/include -c $1.c

# Link the program and create a shared library.
cc -G -o $1 $1.o -L$DB2PATH/lib -ldb2 -ldb2apie

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldudf のコンパイルおよびリンク・オプション

コンパイル・オプション

cc C コンパイラー。

-KPIC 共用ライブラリー用の位置独立コードを生成します。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。

bldudf のコンパイルおよびリンク・オプション
リンク・オプション
cc コンパイラーをリンカーのフロントエンドとして使用します。
-G 共用ライブラリーを生成します。
-o $1 実行可能ファイルを指定します。
$1.o プログラムのオブジェクト・ファイルを組み込みます。
-L$DB2PATH/lib リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。
-ldb2 DB2 ライブラリーとリンクします。
-ldb2apie DB2 API エンジン・ライブラリーとリンクして、LOB ロケーターを使用できるようにします。
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。

ユーザー定義関数プログラム udfsrv をソース・ファイル udfsrv.c から作成するには、次のように入力します。

```
bldudf udfsrv
```

スクリプト・ファイルは、 UDF を sqllib/function ディレクトリーにコピーします。

必要であれば、UDF にファイル・モードを設定してクライアント・アプリケーションから実行できるようにします。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。このプログラムには DB2 CLI バージョンと組み込み SQL バージョンがあります。

DB2 CLI udfcli プログラム は、スクリプト・ファイル bldcli を使用して、sqllib/samples/cli にあるソース・ファイル udfcli.c から作成できます。詳細については、 280ページの『DB2 CLI アプリケーション』を参照してください。

組み込み SQL udfcli プログラムは、スクリプト・ファイル bldapp を使用して、 sqllib/samples/c にあるソース・ファイル udfcli.sqc から構築するこ

とができます。詳細については、285ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF を呼び出すには、次の実行可能ファイル名を入力して、サンプルの呼び出しアプリケーションを実行します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv ライブラリーから ScalarUDF 関数を呼び出します。

マルチスレッド・アプリケーション

ptx/C を使用するマルチスレッド・アプリケーションは、 -Kthread でコンパイルおよびリンクする必要があります。

sqllib/samples/c のスクリプト・ファイル bldmt には、組み込み SQL マルチスレッド・プログラムを作成するためのコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。第 3 パラメーター $3 にはそのデータベースのユーザー ID を、また $4 にはパスワードを指定します。第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名、ユーザー ID、およびパスワードは任意指定です。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldmt script file -- DYNIX/ptx
# Builds a C multi-threaded embedded SQL program.
# Usage:  bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to the location where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2 $3 $4
# Compile the program.
cc -Kthread -I$DB2PATH/include -c $1.c

# Link the program.
cc -Kthread -o $1 $1.o -L$DB2PATH/lib -ldb2
```

上記の -Kthread オプションや、リンクされているユーティリティー・ファイルがないという点だけでなく、残りのコンパイルとリンクのオプションも、組み込み SQL スクリプト・ファイル bldapp で使用されているものと同じで

す。これらのオプションについては、285ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル thdsrver.sqc からサンプル・プログラム thdsrver を作成するには、次のように入力します。

```
bldmt thdsrver
```

結果として、実行可能ファイル thdsrver が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
thdsrver
```

ptx/C++

この節では、以下のトピックを取り上げています。

- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー
- ユーザー定義関数 (UDF)
- マルチスレッド・アプリケーション

DB2 API と組み込み SQL アプリケーション

sqllib/samples/cpp のビルド・ファイル bldapp には、 DB2 API と組み込み SQL プログラムを構築するコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。必要なパラメーターはこのパラメーターだけであり、組み込み SQL を含まない DB2 API プログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- DYNIX/ptx
# Builds a C++ application program
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to the location where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# if an embedded SQL program, precompile and bind it.
if [[ -f $1".sqC" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.C error-checking utility.
  c++ -I$DB2PATH/include -D_RWSTD_COMPILE_INSTANTIATE=0 -c utilemb.C
else
  # Compile the utilapi.C error-checking utility.
  c++ -I$DB2PATH/include -D_RWSTD_COMPILE_INSTANTIATE=0 -c utilapi.C
fi

# Compile the program.
c++ -I$DB2PATH/include -D_RWSTD_COMPILE_INSTANTIATE=0 -c $1.C
if [[ -f $1".sqC" ]]
then
  # Link the program with utilemb.o
  c++ -o $1 $1.o utilemb.o -L$DB2PATH/lib -ldb2 -lseq
else
  # Link the program with utilapi.o
  c++ -o $1 $1.o utilapi.o -L$DB2PATH/lib -ldb2 -lseq
fi
```

bldapp のコンパイルおよびリンク・オプション

コンパイル・オプション

c++ C++ コンパイラー。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-D_RWSTD_COMPILE_INSTANTIATE=0
Rogue Wave のクラスをインスタンス化しません。

-c コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。

bldapp のコンパイルおよびリンク・オプション
リンク・オプション
c++ コンパイラーをリンカーのフロントエンドとして使用します。
-o $1 実行可能ファイルを指定します。
$1.o プログラムのオブジェクト・ファイルを組み込みます。
utilemb.o 組み込み SQL プログラムの場合に、エラー・チェックを行う組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。
utilapi.o 非組み込み SQL プログラムの場合に、エラー・チェックを行う DB2 API ユーティリティー・オブジェクト・ファイルを含みます。
-L$DB2PATH/lib リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。
-ldb2 DB2 ライブラリーとリンクします。
-lseq Sequent ライブラリーとリンクします。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.C から DB2 API 非組み込み SQL サンプル・プログラム client を構築するには、次のようにします。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。

この実行可能ファイルを実行するには、ファイル名を入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqC から組み込み SQL アプリケーション updat を構築する場合、次の 3 つの方法があります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名を加えます。

```
bldapp updat database
```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードを加えます。

```
bldapp updat database userid password
```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

```
updat
```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

```
updat database
```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

```
updat database userid password
```

組み込み SQL ストアード・プロシージャー

注: 71ページの『UDF およびストアード・プロシージャーに関する C++ 考慮事項』にある、 C++ のストアード・プロシージャーの構築に関する情報を参照してください。

sqllib/samples/cpp にあるスクリプト・ファイル bldsrv には、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが含まれています。スクリプト・ファイルは、ストアード・プロシージャーを共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。必須パラメーターはこのパラメーターだけです。ただし、組み込み SQL プログラムの構築にはデータベースへの接続が必要なので、付加的なオプションとして、接続先のデータベース名を指定するパラメーター $2 が用意されています。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを入力する

ための付加的なパラメーターは必要ありません。スクリプト・ファイル bldsrv は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。

共用ライブラリーの名前には、ソース・ファイル名 $1 が使用されます。

```
#! /bin/ksh
# bldsrv script file -- DYNIX/ptx
# Builds a C++ stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2

# Compile the program. First ensure it is coded with extern "C".
c++ -KPIC -I$DB2PATH/include -D_RWSTD_COMPILE_INSTANTIATE=0 -c $1.C
# Link the program and create a shared library.
c++ -G -o $1 $1.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1.so $DB2PATH/function/$1
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

c++ C++ コンパイラー。

-KPIC 共用ライブラリー用の位置独立コードを生成します。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、-I$DB2PATH/include。

-D_RWSTD_COMPILE_INSTANTIATE=0
Rogue Wave のクラスをインスタンス化しません。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

bldsrv のコンパイルおよびリンク・オプション	
リンク・オプション	
c++	コンパイラーをリンカーのフロントエンドとして使用します。
-G	共用ライブラリーを生成します。
-o $1	実行可能ファイルを指定します。
$1.o	プログラムのオブジェクト・ファイルを組み込みます。
-L$DB2PATH/lib	リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。
-ldb2	DB2 ライブラリーとリンクします。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。	

sample データベースに接続している場合に、ソース・ファイル spserver.sqC からサンプル・プログラム spserver を構築するには、次のように入力します。

```
bldsrv spserver
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv spserver database
```

スクリプト・ファイルは、共用ライブラリーをサーバー上の sqllib/function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、クライアント・アプリケーション spclient を構築することができます。これは、共用ライブラリー内のストアード・プロシージャーを呼び出すアプリケーションです。

spclient は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 295ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

共用ライブラリーにアクセスするには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。ストアード・プロシージャーは、出力をクライアント・アプリケーションに戻します。

ユーザー定義関数 (UDF)

注: 71ページの『UDF およびストアード・プロシージャーに関する C++ 考慮事項』にある、 C++ UDF の構築に関する情報を参照してください。

sqllib/samples/cpp のスクリプト・ファイル bldudf には、 UDF を作成するためのコマンドが含まれています。 UDF には組み込み SQL ステートメントは含まれていません。したがって、UDF プログラムを作成する際に、データベースに接続したり、プログラムをプリコンパイルおよびバインドする必要はありません。

パラメーター $1 には、ソース・ファイルの名前を指定します。スクリプト・ファイルは、そのソース・ファイル名を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldudf script file -- DYNIX/ptx
# Builds a C++ user-defined function library
# Usage: bldudf <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the program.
c++ -KPIC -I$DB2PATH/include  -D_RWSTD_COMPILE_INSTANTIATE=0 -c $1.c

# Link the program and create a shared library.
c++ -G -o $1 $1.o -L$DB2PATH/lib -ldb2 -ldb2apie

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1.so $DB2PATH/function/$1
```

bldudf のコンパイルおよびリンク・オプション

コンパイル・オプション

c++ C++ コンパイラー。

-KPIC 共用ライブラリー用の位置独立コードを生成します。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include。

-D_RWSTD_COMPILE_INSTANTIATE=0
Rogue Wave のクラスをインスタンス化しません。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

bldudf のコンパイルおよびリンク・オプション
リンク・オプション **c++** コンパイラーをリンカーのフロントエンドとして使用します。 **-G** 共用ライブラリーを生成します。 **-o $1** 実行可能ファイルを指定します。 **$1.o** プログラムのオブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-ldb2** DB2 ライブラリーとリンクします。 **-ldb2apie** DB2 API エンジン・ライブラリーとリンクして、LOB ロケーターを使用できるようにします。 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。

ユーザー定義関数プログラム udfsrv をソース・ファイル udfsrv.c から作成するには、次のように入力します。

```
bldudf udfsrv
```

スクリプト・ファイルは、 UDF をサーバー上の sqllib/function というパスにコピーします。

必要であれば、UDF にファイル・モードを設定してクライアント・アプリケーションから実行できるようにします。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。 udfcli プログラムは、スクリプト・ファイル bldapp を使用して、 sqllib/samples/cpp にあるソース・ファイル udfcli.sqC から作成します。詳細については、 295ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF を呼び出すには、次の実行可能ファイル名を入力して、サンプルの呼び出しアプリケーションを実行します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv ライブラリーの ScalarUDF 関数を呼び出します。

マルチスレッド・アプリケーション

ptx/C++ を使用するマルチスレッド・アプリケーションは、 -Kthread でコンパイルおよびリンクする必要があります。

sqllib/samples/cpp にあるスクリプト・ファイル bldmt には、組み込み SQL マルチスレッド・プログラムを作成するためのコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。第 3 パラメーター $3 にはそのデータベースのユーザー ID を、また $4 にはパスワードを指定します。第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名、ユーザー ID、およびパスワードは任意指定です。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldmt script file -- DYNIX/ptx
# Builds a C++ multi-threaded embedded SQL program
# Usage:  bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to the location where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2 $3 $4

# Compile the program.
c++ -Kthread -I$DB2PATH/include -D_RWSTD_COMPILE_INSTANTIATE=0 -c $1.C
# Link the program.
c++ -Kthread -o $1 $1.o -L$DB2PATH/lib -ldb2 -lseq
```

上記の -Kthread オプションや、リンクされているユーティリティー・ファイルがないという点だけでなく、残りのコンパイルとリンクのオプションも、組み込み SQL スクリプト・ファイル bldapp で使用されているものと同じです。これらのオプションについては、 295ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル thdsrver.sqC からサンプル・プログラム thdsrver を作成するには、次のように入力します。

```
bldmt thdsrver
```

結果として、実行可能ファイル thdsrver が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
thdsrver
```

第11章 Silicon Graphics IRIX アプリケーションの構築

この章では、 Silicon Graphics IRIX で DB2 アプリケーションを構築するための詳細な情報を提供します。スクリプト・ファイルにおいて、db2 から始まるコマンドは、コマンド行プロセッサー (CLP) のコマンドです。 CLP コマンドについての詳細な情報が必要であれば、 *コマンド解説書* を参照してください。

Silicon Graphics IRIX 用の DB2 アプリケーション開発の最新の更新事項については、次の Web ページを参照してください。

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

DB2 for Silicon Graphics IRIX はクライアント専用です。 DB2 アプリケーションを実行し、DB2 組み込み SQL アプリケーションを構築するには、クライアント・マシンからサーバー・マシン上の DB2 データベースにアクセスする必要があります。なお、サーバー・マシンでは、別のオペレーティング・システムを実行します。クライアント / サーバー通信の構成については、*DB2 ユニバーサル・データベース (UNIX 版) 概説およびインストール* を参照してください。

加えて、別のオペレーティング・システムで実行しているリモート・クライアントからサーバー上のデータベースにアクセスするためには、 DB2 CLI を含

むデータベース・ユーティリティーをデータベースにバインドする必要があります。詳細については、50ページの『バインド』を参照してください。

DB2 ライブラリー・サポート

Silicon Graphics IRIX には、o32 (デフォルト)、 n32 (新しい 32 オブジェクト・タイプ)、そして 64 (64 オブジェクト・タイプ) という、互いに独立して互換性を持たない 3 つのオブジェクト・タイプがあります。 DB2 ではまだ 64 をサポートしていませんが、 o32 オブジェクト・タイプと n32 オブジェクト・タイプをサポートしています。

このオペレーティング・システムのスレッド API には、 sproc インターフェースと POSIX スレッド API という、互いに独立して互換性を持たない 2 つのバージョンがあります。 DB2 でサポートされているのは、POSIX スレッド API です。

sproc インターフェースを使用しているアプリケーションでは、非スレッド・バージョンの DB2 ライブラリー libdb2 を利用することができますが、これはスレッド・セーフではありません。 libdb2 は sproc セーフではないため、sproc インターフェースを使う際には注意が必要です。

このような機能に対応するため、 DB2 には以下のようなライブラリー・サポートがあります。

lib/libdb2.so
: o32、スレッドなし

lib/libdb2_th.so
: o32、POSIX スレッド

lib32/libdb2.so
: n32、スレッドなし

lib32/libdb2_th.so
: n32、POSIX スレッド

n32 オブジェクト・タイプを利用する場合は、 -n32 オプションでプログラムをコンパイルしてリンクさせ、さらに lib32/libdb2.so ライブラリーか lib32/libdb2_th.so ライブラリーにリンクさせてください。デフォルトの o32 オブジェクト・タイプを利用する場合は、-n32 オプションを指定しないで、lib/libdb2.so ライブラリーか lib/libdb2_th.so ライブラリーにプログラムをリンクさせてください。

注: この章で扱われているビルド・ファイルを使用して n32 オブジェクト・タイプのアプリケーションを構築するには、指示されているコマンドをコメント解除してください。

MIPSpro C

この節では、以下に示す DB2 インターフェースで MIPSpro C を使用する方法について説明します。

- DB2 CLI
- DB2 API
- 組み込み SQL

DB2 CLI アプリケーション

sqllib/samples/cli のスクリプト・ファイル bldcli には、 DB2 CLI プログラムを作成するためのコマンドが入っています。パラメーター $1 には、ソース・ファイルの名前を指定します。

これは、唯一の必須パラメーターであり、組み込み SQL を含まない CLI プログラムに必要なパラメーターはこのパラメーターだけです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、オプションとして 3 つのパラメーターが用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

プログラムに組み込み SQL が含まれている場合 (拡張子が .sqc の場合) は、embprep スクリプトが呼び出されてそのプログラムをプリコンパイルし、 .c という拡張子のプログラム・ファイルを生成します。

```
#! /bin/ksh
# bldcli script file -- Silicon Graphics IRIX
# Builds a CLI program with MIPSpro C.
# Usage: bldcli <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
   # The default is the instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
fi
# To compile with n32 object support, uncomment the following line.
# IRIX_OBJECT_MODE=-n32
if [ "$IRIX_OBJECT_MODE" = "-n32" ] ; then
  # Link with db2 n32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib32
else
  # Link with db2 o32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib
```

```
fi
# Compile the error-checking utility.
cc $IRIX_OBJECT_MODE -I$DB2PATH/include -c utilcli.c
# Compile the program.
cc $IRIX_OBJECT_MODE -I$DB2PATH/include -c $1.c
# Link the program.
cc $IRIX_OBJECT_MODE -o $1 $1.o utilcli.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH -lm -ldb2
```

bldcli のコンパイルおよびリンク・オプション

コンパイル・オプション

cc C コンパイラーを使用します。

$IRIX_OBJECT_MODE
'IRIX_OBJECT_MODE=-n32' がコメント解除されている場合に、 "-n32" を含みます。それ以外の場合は、値を含みません。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、 $HOME/sqllib/include。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

bldcli のコンパイルおよびリンク・オプション
リンク・オプション
cc コンパイラーをリンカーのフロントエンドとして使用します。
$IRIX_OBJECT_MODE 'IRIX_OBJECT_MODE=-n32' がコメント解除されている場合に、 "-n32" を含みます。それ以外の場合は、値を含みません。
-o $1 実行可能ファイルを指定します。
$1.o プログラムのオブジェクト・ファイルを組み込みます。
utilcli.o エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。
-L$DB2_LIBPATH リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを示します。 o32 オブジェクト・タイプの場合は $DB2PATH/lib を、 n32 オブジェクト・タイプの場合は $DB2PATH/lib32 を示します。 -L オプションが指定されない場合、コンパイラーはパスとして /usr/lib:/lib を想定します。
-rpath $DB2_LIBPATH 実行時の DB2 共用ライブラリーのロケーションを示します。 o32 オブジェクト・タイプの場合は $DB2PATH/lib を、 n32 オブジェクト・タイプの場合は $DB2PATH/lib32 を示します。
-lm 数学ライブラリーとリンクします。
-ldb2 DB2 ライブラリーとリンクします。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル tbinfo.c からサンプル・プログラム tbinfo を構築するには、次のようにします。

```
bldcli tbinfo
```

結果として、実行可能ファイル tbinfo が生成されます。この実行可能ファイルを実行するには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力します。

```
tbinfo database userid password
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル dbusemx.sqc から dbusemx を構築するには、データベースのパラメーターと、データベースが置かれているインスタンスのユーザー ID とパスワードを指定するパラメーターを組み込みます。

```
bldcli dbusemx database userid password
```

結果として、実行可能ファイル dbusemx が作成されます。

この組み込み SQL アプリケーションを実行するには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力します。

```
dbusemx database userid password
```

DB2 API を使用する DB2 CLI アプリケーション

DB2 には、CLI サンプル・プログラムが含まれています。このサンプル・プログラムは、DB2 API を使用してデータベースを作成およびドロップし、 CLI 機能を複数のデータベースで使用する方法を示します。 DB2 API を使用するサンプルは、 27ページの表7 にある CLI サンプル・プログラムの説明の中に示されています。

sqllib/samples/cli のスクリプト・ファイル bldapi には、 DB2 API を使用して DB2 CLI プログラムを作成するためのコマンドが入っています。このファイルは、データベースを作成およびドロップするための DB2 API が入った utilapi ユーティリティー・ファイルでコンパイルおよびリンクします。この点が、このスクリプト・ファイルと bldcli スクリプトの唯一の違いです。 bldapi と bldcli の両方に共通するコンパイルとリンクのオプションについては、 307ページの『DB2 CLI アプリケーション』 を参照してください。

ソース・ファイル dbmconn.c からサンプル・プログラム dbmconn を作成するには、次のようにします。

```
bldapi dbmconn
```

結果として、実行可能ファイル dbmconn が作成されます。この実行可能ファイルを実行するには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力します。

```
dbmconn database userid password
```

ストアード・プロシージャー用の DB2 CLI クライアント・アプリケーション

ストアード・プロシージャーは、データベースにアクセスしてクライアント・アプリケーションに情報を戻すプログラムです。ストアード・プロシージャーのコンパイルと保管は、サーバー上で行います。なお、サーバーは、別のプラットフォームで実行します。

DB2 がサポートするプラットフォームのサーバー上で DB2 CLI ストアード・プロシージャー spserver を作成する場合は、本書のそのプラットフォーム用

の『アプリケーションの構築』という章を参照してください。 DB2 クライアントからアクセスできる他のサーバーについては、 7ページの『サポートされるサーバー』を参照してください。

ストアード・プロシージャー spserver を構築すると、スクリプト・ファイル bldcli を使用して、ストアード・プロシージャー spclient を呼び出すクライアント・アプリケーションをソース・ファイル spclient.c から構築することができます。詳細については、307ページの『DB2 CLI アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力します。

```
spclient database userid password
```

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

UDF 用の DB2 CLI クライアント・アプリケーション

ユーザー定義関数 (UDF) とは、ユーザー独自のスカラー関数や表関数のことであり、これらの関数はサーバー上でコンパイルされ、サーバー上に保管されます。なお、サーバーは別のプラットフォームで実行します。 DB2 がサポートする各プラットフォームのサーバー上でユーザー定義関数 udfsrv を作成する場合は、本書のそのプラットフォーム用の『アプリケーションの構築』という章を参照してください。 DB2 クライアントからアクセスできる他のサーバーについては、 7ページの『サポートされるサーバー』を参照してください。

udfsrv を構築したなら、DB2 CLI スクリプト・ファイル bldcli を使用して、それを呼び出すための DB2 CLI クライアント・アプリケーション udfcli を sqllib/samples/cli のソース・ファイル udfcli.c から構築することができます。詳細については、307ページの『DB2 CLI アプリケーション』を参照してください。

UDF プログラムを呼び出すには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力して、呼び出しアプリケーションを実行します。

```
udfcli database userid password
```

この呼び出しアプリケーションは、 udfsrv ライブラリーから ScalarUDF 関数を呼び出します。

DB2 API と組み込み SQL アプリケーション

sqllib/samples/c にあるスクリプト・ファイル bldapp には、 DB2 アプリケーション・プログラムを構築するコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。このパラメーターは、唯一の必須パラメーターであり、組み込み SQL を含まない DB2 API プログラムに必要なパラメーターはこのパラメーターだけです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、オプションとして 3 つのパラメーターが用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。データベース名が指定されない場合は、デフォルトの sample データベースが使用されます。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- Silicon Graphics IRIX
# Builds a C application program.
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# To compile with n32 object support, uncomment the following line.
# IRIX_OBJECT_MODE=-n32
if [ "$IRIX_OBJECT_MODE" = "-n32" ] ; then
  # Link with db2 n32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib32
else
  # Link with db2 o32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib
fi
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.c error-checking utility.
  cc $IRIX_OBJECT_MODE -I$DB2PATH/include -c utilemb.c
else
  # Compile the utilapi.c error-checking utility.
  cc $IRIX_OBJECT_MODE -I$DB2PATH/include -c utilapi.c
fi
# Compile the program.
cc $IRIX_OBJECT_MODE -I$DB2PATH/include -c $1.c
if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o
  cc $IRIX_OBJECT_MODE -o $1 $1.o utilemb.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH -lm -ldb2
else
  # Link the program with utilapi.o
  cc $IRIX_OBJECT_MODE -o $1 $1.o utilapi.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH -lm -ldb2
fi
```

bldapp のコンパイルおよびリンク・オプション
コンパイル・オプション **cc**　C コンパイラーを使用します。 **$IRIX_OBJECT_MODE** 'IRIX_OBJECT_MODE=-n32' がコメント解除されている場合に、 "-n32" を含みます。それ以外の場合は、値を含みません。 **-I$DB2PATH/include** DB2 インクルード・ファイルのロケーションを指定します。たとえば、 `$HOME/sqllib/include`。 **-c**　コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

bldapp のコンパイルおよびリンク・オプション
リンク・オプション **cc** コンパイラーをリンカーのフロントエンドとして使用します。 **$IRIX_OBJECT_MODE** 'IRIX_OBJECT_MODE=-n32' がコメント解除されている場合に、 "-n32" を含みます。それ以外の場合は、値を含みません。 **-o $1** 実行可能ファイルを指定します。 **$1.o** プログラムのオブジェクト・ファイルを組み込みます。 **utilemb.o** 組み込み SQL プログラムの場合に、エラー・チェックを行う組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。 **utilapi.o** 非組み込み SQL プログラムの場合に、エラー・チェックを行う DB2 API ユーティリティー・オブジェクト・ファイルを含みます。 **-L$DB2_LIBPATH** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを示します。 o32 オブジェクト・タイプの場合は $DB2PATH/lib を、 n32 オブジェクト・タイプの場合は $DB2PATH/lib32 を示します。 -L オプションが指定されない場合、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-rpath $DB2_LIBPATH** 実行時の DB2 共用ライブラリーのロケーションを示します。 o32 オブジェクト・タイプの場合は $DB2PATH/lib を、 n32 オブジェクト・タイプの場合は $DB2PATH/lib32 を示します。 **-lm** 数学ライブラリーとリンクします。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.c から DB2 API 非組み込み SQL サンプル・プログラム client を構築するには、次のようにします。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。

この実行可能ファイルを実行するには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力します。

```
client database userid password
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqc からサンプル・プログラム updat を構築するには、データベースのパラメーターと、データベースが置かれているインスタンスのユーザー ID とパスワードを指定するパラメーターを組み込みます。

```
bldapp updat database userid password
```

結果として、実行可能ファイル updat が作成されます。この実行可能ファイルを sample データベースに対して実行するには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力します。

```
updat database userid password
```

ストアード・プロシージャー用の組み込み SQL クライアント・アプリケーション

ストアード・プロシージャーは、データベースにアクセスしてクライアント・アプリケーションに情報を戻すプログラムです。ストアード・プロシージャーのコンパイルと保管は、サーバー上で行われます。なお、サーバーは、別のプラットフォームで実行します。

DB2 がサポートするプラットフォームのサーバー上で組み込み SQL ストアード・プロシージャー spserver を作成する場合は、本書のそのプラットフォーム用の『アプリケーションの構築』という章を参照してください。 DB2 クライアントからアクセスできる他のサーバーについては、 7ページの『サポートされるサーバー』を参照してください。

ストアード・プロシージャー spserver を作成したなら、そのストアード・プロシージャーを呼び出すクライアント・アプリケーションを構築できます。spclient は、スクリプト・ファイル bldapp を使用して、ソース・ファイル spclient.sqc から構築することができます。詳細については、312ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力して、クライアント・アプリケーションを実行します。

```
spclient database userid password
```

クライアント・アプリケーションはストアード・プロシージャー・ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

ユーザー定義関数 (UDF) 用のクライアント・アプリケーション

ユーザー定義関数 (UDF) とは、ユーザー独自のスカラー関数や表関数のことであり、これらの関数はサーバー上でコンパイルされ、サーバー上に保管されます。なお、サーバーは別のプラットフォームで実行します。 DB2 がサポートする各プラットフォームのサーバー上でユーザー定義関数 udfsrv を作成する場合は、本書のそのプラットフォーム用の『アプリケーションの構築』という章を参照してください。 DB2 クライアントからアクセスできる他のサーバーについては、 7ページの『サポートされるサーバー』を参照してください。

udfsrv を構築したなら、スクリプト・ファイル bldapp を使用して、それを呼び出すための組み込み SQL クライアント・アプリケーション udfcli を sqllib/samples/c のソース・ファイル udfcli.sqc から構築することができます。詳細については、312ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF プログラムを呼び出すには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力して、呼び出しアプリケーションを実行します。

```
udfcli database userid password
```

この呼び出しアプリケーションは、 udfsrv ライブラリーから ScalarUDF 関数を呼び出します。

マルチスレッド・アプリケーション

Silicon Graphics IRIX 上のマルチスレッド・アプリケーションは、 -ldb2_th および -lpthread リンク・オプションを使用して、 o32 か n32 のオブジェクト・タイプに対応した POSIX スレッド・バージョンの DB2 ライブラリーにリンクさせる必要があります。

sqllib/samples/c のスクリプト・ファイル bldmt には、組み込み SQL マルチスレッド・プログラムを作成するためのコマンドが含まれています。第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。第 3 パラメーター $3 にはそのデータベースのユーザー ID を、また $4 にはパスワードを指定します。

```
#! /bin/ksh
# bldmt script file -- Silicon Graphics IRIX
# Builds a C multi-threaded embedded SQL program
# Usage:  bldmt <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# To compile with n32 object support, uncomment the following line.
# IRIX_OBJECT_MODE=-n32
if [ "$IRIX_OBJECT_MODE" = "-n32" ] ; then
  # Link with db2 n32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib32
else
  # Link with db2 o32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib
fi
# Precompile and bind the program.
embprep $1 $2 $3 $4
# Compile the program.
cc $IRIX_OBJECT_MODE -I$DB2PATH/include -c $1.c
# Link the program.
cc $IRIX_OBJECT_MODE -o $1 $1.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH -lm -ldb2_th -lpthread
```

上記の -ldb2_th および -lpthread リンク・オプションや、リンクされているユーティリティー・ファイルがないという点だけでなく、残りのコンパイルとリンクのオプションも組み込み SQL スクリプト・ファイル bldapp で使用されているものと同じです。これらのオプションについては、312ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル thdsrver.sqc からサンプル・プログラム thdsrver を構築するには、データベースのパラメーターと、データベースが置かれているインスタンスのユーザー ID とパスワードを指定するパラメーターを組み込みます。

```
bldmt thdsrver database userid password
```

結果として、実行可能ファイル thdsrver が作成されます。

この実行可能ファイルを実行するには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力します。

```
thdsrver database userid password
```

MIPSpro C++

この節では以下のトピックを取り上げています。

- DB2 API と組み込み SQL アプリケーション
- マルチスレッド・アプリケーション

DB2 API と組み込み SQL アプリケーション

sqllib/samples/cpp にあるスクリプト・ファイル bldapp には、 DB2 アプリケーション・プログラムを構築するコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。非組み込み SQL アプリケーションの必須パラメーターはこのパラメーターだけです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、オプションとして 3 つのパラメーターが用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。

```
#! /bin/ksh
# bldapp script file -- Silicon Graphics IRIX
# Builds a C++ application program
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# To compile with n32 object support, uncomment the following line.
# IRIX_OBJECT_MODE=-n32
if [ "$IRIX_OBJECT_MODE" = "-n32" ] ; then
  # Link with db2 n32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib32
else
  # Link with db2 o32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib
fi
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqC" ]]
then
embprep $1 $2 $3 $4
  # Compile the utilemb.C error-checking utility.
  CC $IRIX_OBJECT_MODE -I$DB2PATH/include -c utilemb.C
else
  # Compile the utilapi.c error-checking utility.
  CC $IRIX_OBJECT_MODE -I$DB2PATH/include -c utilapi.C
fi
# Compile the program.
CC $IRIX_OBJECT_MODE -I$DB2PATH/include -c $1.C
if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o
  CC $IRIX_OBJECT_MODE -o $1 $1.o utilemb.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH -lm -ldb2
else
  # Link the program with utilapi.o
  CC $IRIX_OBJECT_MODE -o $1 $1.o utilapi.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH -lm -ldb2
fi
```

bldapp のコンパイルおよびリンク・オプション
コンパイル・オプション **CC**　C++ コンパイラーを使用します。 **$IRIX_OBJECT_MODE** 'IRIX_OBJECT_MODE=-n32' がコメント解除されている場合に、 "-n32" を含みます。それ以外の場合は、値を含みません。 **-I$DB2PATH/include** DB2 インクルード・ファイルのロケーションを指定します。たとえば、 $HOME/sqllib/include。 **-c**　コンパイルのみを実行し、リンクは実行しません。このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldapp のコンパイルおよびリンク・オプション

リンク・オプション

CC　コンパイラーをリンカーのフロントエンドとして使用します。

$IRIX_OBJECT_MODE
'IRIX_OBJECT_MODE=-n32' がコメント解除されている場合に、 "-n32" を含みます。それ以外の場合は、値を含みません。

-o $1　実行可能ファイルを指定します。

$1.o　プログラムのオブジェクト・ファイルを組み込みます。

utilemb.o
組み込み SQL プログラムの場合に、エラー・チェックを行う組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。

utilapi.o
非組み込み SQL プログラムの場合に、エラー・チェックを行う DB2 API ユーティリティー・オブジェクト・ファイルを含みます。

-L$DB2_LIBPATH
リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを示します。 o32 オブジェクト・タイプの場合は $DB2PATH/lib を、 n32 オブジェクト・タイプの場合は $DB2PATH/lib32 を示します。 -L オプションが指定されない場合、コンパイラーはパスとして /usr/lib:/lib を想定します。

-rpath $DB2_LIBPATH
実行時の DB2 共用ライブラリーのロケーションを示します。 o32 オブジェクト・タイプの場合は $DB2PATH/lib を、 n32 オブジェクト・タイプの場合は $DB2PATH/lib32 を示します。

-lm　数学ライブラリーとリンクします。

-ldb2　DB2 ライブラリーとリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル updat.sqC からサンプル・プログラム updat を構築するには、データベースのパラメーターと、データベースが置かれているインスタンスのユーザー ID とパスワードを組み込みます。

```
bldapp updat database userid password
```

結果として実行可能ファイル updat が作成されます。この実行可能ファイルを実行するには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力します。

```
updat database userid password
```

ストアード・プロシージャー用の組み込み SQL クライアント・アプリケーション

ストアード・プロシージャーは、データベースにアクセスしてクライアント・アプリケーションに情報を戻すプログラムです。ストアード・プロシージャーのコンパイルと保管は、サーバー上で行われます。なお、サーバーは別のプラットフォームで実行します。

DB2 がサポートするプラットフォームのサーバー上で組み込み SQL ストアード・プロシージャー spserver を作成する場合は、本書のそのプラットフォーム用の『アプリケーションの構築』という章を参照してください。 DB2 クライアントからアクセスできる他のサーバーについては、 7ページの『サポートされるサーバー』を参照してください。

ストアード・プロシージャー spserver を作成したなら、そのストアード・プロシージャーを呼び出すクライアント・アプリケーションを構築できます。spclient は、スクリプト・ファイル bldapp を使用して、ソース・ファイル spclient.sqC から作成することができます。詳細については、318ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力して、クライアント・アプリケーションを実行します。

```
spclient database userid password
```

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。ストアード・プロシージャーは、出力をクライアント・アプリケーションに戻します。

UDF 用の組み込み SQL クライアント・アプリケーション

ユーザー定義関数 (UDF) とは、ユーザー独自のスカラー関数のことであり、この関数はサーバー上でコンパイルされ、サーバー上に保管されます。なお、サーバーは別のプラットフォームで実行します。 DB2 がサポートする各プラットフォームのサーバー上でユーザー定義関数 udfsrv を作成する場合は、本書のそのプラットフォーム用の『アプリケーションの構築』という章を参照してください。 DB2 クライアントからアクセスできる他のサーバーについては、 7ページの『サポートされるサーバー』を参照してください。

udfsrv を構築したなら、スクリプト・ファイル bldapp を使用して、それを呼び出すための組み込み SQL クライアント・アプリケーション udfcli を

sqllib/samples/cpp のソース・ファイル udfcli.sqC から構築することができます。詳細については、318ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF プログラムを呼び出すには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力して、呼び出しアプリケーションを実行します。

```
udfcli database userid password
```

この呼び出しアプリケーションは、 udfsrv ライブラリーから ScalarUDF 関数を呼び出します。

マルチスレッド・アプリケーション

Silicon Graphics IRIX 上のマルチスレッド・アプリケーションは、 -ldb2_th および -lpthread リンク・オプションを使用して、 o32 か n32 のオブジェクト・タイプに対応した POSIX スレッド・バージョンの DB2 ライブラリーにリンクさせる必要があります。

sqllib/samples/cpp のスクリプト・ファイル bldmt には、組み込み SQL マルチスレッド・プログラムを作成するためのコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。パラメーター $3 にはそのデータベースのユーザー ID を、 $4 にはパスワードを指定します。

```
#! /bin/ksh
# bldmt script file -- Silicon Graphics IRIX
# Builds a C++ multi-threaded embedded SQL program
# Usage:  bldmt <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# To compile with n32 object support, uncomment the following line.
# IRIX_OBJECT_MODE=-n32
if [ "$IRIX_OBJECT_MODE" = "-n32" ] ; then
  # Link with db2 n32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib32
else
  # Link with db2 o32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib
fi
# Precompile and bind the program.
embprep $1 $2 $3 $4
# Compile the program.
CC $IRIX_OBJECT_MODE -I$DB2PATH/include -c $1.C
# Link the program.
CC $IRIX_OBJECT_MODE -o $1 $1.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH -lm -ldb2_th -lpthread
```

上記の -ldb2_th および -lpthread リンク・オプションや、リンクされているユーティリティー・ファイルがないという点だけでなく、残りのコンパイルと

リンクのオプションも、組み込み SQL スクリプト・ファイル bldapp で使用されているものと同じです。これらのオプションについては、318ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル thdsrver.sqC からサンプル・プログラム thdsrver を構築するには、データベースのパラメーターと、データベースが置かれているインスタンスのユーザー ID とパスワードを指定するパラメーターを組み込みます。

```
bldmt thdsrver database userid password
```

結果として、実行可能ファイル thdsrver が作成されます。

この実行可能ファイルを sample データベースに対して実行するには、実行可能ファイル名、データベース名、およびデータベースが置かれているインスタンスのユーザー ID とパスワードを入力します。

```
thdsrver database userid password
```

第12章 Solaris アプリケーションの構築

この章は、Solaris オペレーティング環境でアプリケーションを構築するための詳細な情報を提供します。スクリプト・ファイルにおいて、db2 から始まるコマンドは、コマンド行プロセッサー (CLP) コマンドです。 CLP コマンドについての詳しい情報が必要であれば、 *コマンド解説書* を参照してください。

Solaris オペレーティング環境用の DB2 アプリケーション開発の最新の更新事項については、次の Web ページを参照してください。

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

注:

1. -mt マルチスレッド・オプションは、スレッドが Solaris オペレーティング環境で実装される方法のため、 DB2 ビルド・ファイルおよび makefile のリンク・ステップで使用されます。この場合は、パフォーマンス・コストがわずかながらかかります。最適パフォーマンスが考慮事項である場合、このオプションを指定せず、非スレッド libdb2.so ライブラリーを指定して、アプリケーションへのリンクを試みることができます。ただし、-mt スイッチを使用しないと、アプリケーションの実行時に以下のようなエラーが発生することがあります。

   ```
   libc internal error: _rmutex_unlock: rmutex not held
   ```

または、アプリケーションが停止し、エラー・メッセージが発行されないことがあります。

2. この章で示されているビルド・ファイルを使って 64 ビット・アプリケーションを構築するには、提供されているコマンドをコメント解除してください。

SPARCompiler C

この節では以下のトピックを取り上げています。

- DB2 CLI アプリケーション
- DB2 API を使用する DB2 CLI アプリケーション
- DB2 CLI ストアード・プロシージャー
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー
- ユーザー定義関数 (UDF)
- マルチスレッド・アプリケーション

DB2 CLI アプリケーション

sqllib/samples/cli にあるスクリプト・ファイル bldcli には、 DB2 CLI プログラムを作成するためのコマンドが入っています。パラメーター $1 には、ソース・ファイルの名前を指定します。

これは、組み込み SQL を含んでいない CLI プログラム用の唯一の必須パラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、オプションとして 3 つのパラメーターが用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

プログラムに組み込み SQL が含まれている場合 (拡張子が .sqc の場合) は、embprep スクリプトが呼び出されてそのプログラムをプリコンパイルし、 .c という拡張子のプログラム・ファイルを生成します。

```
#! /bin/ksh
# bldcli script file -- Solaris
# Builds a DB2 CLI program.
# Usage: bldcli <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
```

```
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
fi

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi
# Compile the error-checking utility.
cc $CFLAGS_64 -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc $CFLAGS_64 -I$DB2PATH/include  -c $1.c
# Link the program.
cc $CFLAGS_64 -o $1 $1.o utilcli.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
```

bldcli のコンパイルおよびリンク・オプション

コンパイル・オプション

cc
C コンパイラーを使用します。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 "-xarch=v9" の値を含みます。それ以外の場合は、値を含みません。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、 `$HOME/sqllib/include` のように指定します。

-c
コンパイルのみを実行し、リンクは実行しません。このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldcli のコンパイルおよびリンク・オプション
リンク・オプション **cc** コンパイラーをリンカーのフロントエンドとして使用します。 **$CFLAGS_64** 'BUILD_64BIT=true' がコメント解除されている場合に、 "-xarch=v9" の値を含みます。それ以外の場合は、値を含みません。 **-o $1** 実行可能プログラムを指定します。 **$1.o** プログラム・オブジェクト・ファイルを組み込みます。 **utilcli.o** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 **-R$DB2PATH/lib** 実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル tbinfo.c からサンプル・プログラム tbinfo を作成するには、次のように入力します。

```
bldcli tbinfo
```

結果として、実行可能ファイル tbinfo が作成されます。この実行可能ファイルを実行するには、次の実行可能名を入力します。

```
tbinfo
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル dbusemx.sqc から組み込み SQL アプリケーション dbusemx を作成する場合、次の 3 つの方法があります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldcli dbusemx
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

```
bldcli dbusemx database
```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

```
bldcli dbusemx database userid password
```

結果として、実行可能ファイル dbusemx が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

```
dbusemx
```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

```
dbusemx database
```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

```
dbusemx database userid password
```

DB2 API を使用する DB2 CLI アプリケーション

DB2 には、CLI サンプル・プログラムが含まれています。このサンプル・プログラムは、DB2 API を使用してデータベースを作成およびドロップし、 CLI 機能を複数のデータベースで使用する方法を示します。 DB2 API を使用するサンプルは、 27ページの表7 にある CLI サンプル・プログラムの説明の中に示されています。

sqllib/samples/cli にあるスクリプト・ファイル bldapi には、 DB2 API を使用して DB2 CLI プログラムを作成するためのコマンドが入っています。このファイルは、データベースを作成およびドロップするための DB2 API が入った utilapi ユーティリティー・ファイルでコンパイルおよびリンクします。この点が、このスクリプト・ファイルと bldcli スクリプトの唯一の違いです。 bldapi と bldcli の両方に共通のコンパイルおよびリンク・オプションについては、 326ページの『DB2 CLI アプリケーション』 を参照してください。

ソース・ファイル dbmconn.c からサンプル・プログラム dbmconn を作成するには、次のようにします。

```
bldapi dbmconn
```

結果として、実行可能ファイル dbmconn が作成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を入力します。

```
dbmconn
```

DB2 CLI ストアード・プロシージャー

sqllib/samples/cli にあるスクリプト・ファイル bldclisp には、 DB2 CLI ストアード・プロシージャーを作成するためのコマンドが入っています。パラメーター $1 には、ソース・ファイルの名前を指定します。

```
#! /bin/ksh
# bldclisp script file -- Solaris
# Builds a DB2 CLI stored procedure.
# Usage: bldclisp <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi
# Compile the error-checking utility.
cc $CFLAGS_64 -Kpic -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc $CFLAGS_64 -Kpic -I$DB2PATH/include -c $1.c

# Link the program.
cc $CFLAGS_64 -G -o $1 $1.o utilcli.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldclisp のコンパイルおよびリンク・オプション

コンパイル・オプション

cc　C コンパイラー。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 "-xarch=v9" の値を含みます。それ以外の場合は、値を含みません。

-Kpic　共用ライブラリー用の位置独立コードを生成します。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、 `$HOME/sqllib/include` のように指定します。

-c　コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

リンク・オプション

cc　コンパイラーをリンカーのフロントエンドとして使用します。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 "-xarch=v9" の値を含みます。それ以外の場合は、値を含みません。

-o $1　実行可能ファイルを指定します。

$1.o　プログラム・オブジェクト・ファイルを組み込みます。

utilcli.o
エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。

-L$DB2PATH/lib
リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、 `$HOME/sqllib/lib`。 `-L` オプションを指定しないと、コンパイラーはパスとして `/usr/lib:/lib` を想定します。

-R$DB2PATH/lib
実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、 `$HOME/sqllib/lib`。

-ldb2　DB2 ライブラリーとリンクします。

-G　共用ライブラリーを生成します。

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。

ソース・ファイル `spserver.c` からサンプル・プログラム `spserver` を作成するには、次のように入力します。

```
bldclisp spserver
```

このスクリプト・ファイルは、パス sqllib/function 内のサーバーにストアード・プロシージャーをコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

ストアード・プロシージャー spserver を作成したなら、そのストアード・プロシージャーを呼び出す CLI クライアント・アプリケーション spclient を構築できます。

spclient は、スクリプト・ファイル bldcli を使用して構築することができます。詳細については、326ページの『DB2 CLI アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

データベース
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは、ストアード・プロシージャー・ライブラリー spserver にアクセスし、サーバー・データベース上のいくつかのストアード・プロシージャー関数を実行します。出力は、クライアント・アプリケーションに戻されます。

DB2 API と組み込み SQL アプリケーション

スクリプト・ファイル bldapp は sqllib/samples/c にあり、 DB2 アプリケーション・プログラムを作成するためのコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を含まないプログラムに必要な唯一のパラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、オプションとして 3 つのパラメーターが用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。 データベース名を指定しない場合は、デフォルトの sample データベースを使用します。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- Solaris
# Builds a C application program.
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.c error-checking utility.
  cc $CFLAGS_64 -I$DB2PATH/include -c utilemb.c
else
  # Compile the utilapi.c error-checking utility.
  cc $CFLAGS_64 -I$DB2PATH/include -c utilapi.c
fi
```

```
# Compile the program.
cc $CFLAGS_64 -I$DB2PATH/include -c $1.c
if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o
  cc $CFLAGS_64 -o $1 $1.o utilemb.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o
  cc $CFLAGS_64 -o $1 $1.o utilapi.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
fi
```

bldapp のコンパイルおよびリンク・オプション

コンパイル・オプション

cc C コンパイラー。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 "-xarch=v9" の値を含みます。それ以外の場合は、値を含みません。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。 たとえば、 $HOME/sqllib/include のように指定します。

-c コンパイルのみを実行し、リンクは実行しません。 このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldapp のコンパイルおよびリンク・オプション
リンク・オプション
cc コンパイラーをリンカーのフロントエンドとして使用します。
$CFLAGS_64 'BUILD_64BIT=true' がコメント解除されている場合に、 "-xarch=v9" の値を含みます。それ以外の場合は、値を含みません。
-o $1 実行可能ファイルを指定します。
$1.o プログラム・オブジェクト・ファイルを組み込みます。
utilemb.o 組み込み SQL プログラムの場合に、エラー・チェックを行う組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。
utilapi.o 組み込み SQL プログラムでない場合に、エラー・チェックを行う DB2 API ユーティリティー・オブジェクト・ファイルを含みます。
-L$DB2PATH/lib リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。
-R$DB2PATH/lib 実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。
-ldb2 DB2 ライブラリーとリンクします。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.c から DB2 API 非組み込み SQL サンプル・プログラム client を構築するには、次のようにします。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。

この実行可能ファイルを実行するには、ファイル名を入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqc から組み込み SQL アプリケーション updat を構築する場合、次の 3 つの方法があります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

スクリプト・ファイル bldsrv は sqllib/samples/c にあり、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが含まれています。スクリプト・ファイルは、ストアード・プロシージャーを共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

最初のパラメーター (ソース・ファイル名) だけが、必須です。 データベース名は任意で指定します。データベース名を指定しない場合、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。

```
#! /bin/ksh
# bldsrv script file -- Solaris
# Builds a C stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi
# Compile the program.
cc $CFLAGS_64 -Kpic -I$DB2PATH/include -c $1.c

# Link the program and create a shared library
cc $CFLAGS_64 -G -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

cc C コンパイラー。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 ″-xarch=v9″ の値を含みます。それ以外の場合は、値を含みません。

-Kpic 共用ライブラリー用の位置独立コードを生成します。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、 -I$DB2PATH/include。

-c コンパイルのみを実行し、リンクは実行しません。 このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldsrv のコンパイルおよびリンク・オプション
リンク・オプション **cc** コンパイラーをリンカーのフロントエンドとして使用します。 **$CFLAGS_64** 'BUILD_64BIT=true' がコメント解除されている場合に、 ″-xarch=v9″ の値を含みます。それ以外の場合は、値を含みません。 **-G** 共用ライブラリーを生成します。 **-o $1** 実行可能ファイルを指定します。 **$1.o** プログラム・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-R$DB2PATH/lib** 実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。 **-ldb2** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

sample データベースに接続している場合、ソース・ファイル spserver.sqc からサンプル・プログラム spserver を作成するには、次のように入力します。

```
bldsrv spserver
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv spserver database
```

このスクリプト・ファイルは、sqllib/function ディレクトリーにストアード・プロシージャーをコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

ストアード・プロシージャー spserver を作成したなら、そのストアード・プロシージャーを呼び出すクライアント・アプリケーション spclient を構築できます。

spclient は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 333ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションはストアード・プロシージャー・ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

ユーザー定義関数 (UDF)

スクリプト・ファイル bldudf は sqllib/samples/c にあり、 UDF を作成するためのコマンドが含まれています。 UDF には組み込み SQL ステートメントは含まれていません。 したがって、データベースへの接続、またはプログラムのプリコンパイルおよびバインドは行いません。

パラメーター $1 には、ソース・ファイルの名前を指定します。 スクリプト・ファイルは、そのソース・ファイル名を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldudf script file -- Solaris
# Builds a C UDF library
# Usage: bldudf <prog_name>
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi
# Compile the program.
cc $CFLAGS_64 -Kpic -I$DB2PATH/include -c $1.c

# Link the program and create a shared library.
cc $CFLAGS_64 -G -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -ldb2apie

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldudf のコンパイルおよびリンク・オプション

コンパイル・オプション

cc C コンパイラー。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 ″-xarch=v9″ の値を含みます。それ以外の場合は、値を含みません。

-Kpic 共用ライブラリー用の位置独立コードを生成します。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、$HOME/sqllib/include のように指定します。

-c コンパイルのみを実行し、リンクは実行しません。このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldudf のコンパイルおよびリンク・オプション
リンク・オプション
cc コンパイラーをリンカーのフロントエンドとして使用します。
$CFLAGS_64 ’BUILD_64BIT=true’ がコメント解除されている場合に、 ″-xarch=v9″ の値を含みます。それ以外の場合は、値を含みません。
-o $1 実行可能ファイルを指定します。
$1.o プログラム・オブジェクト・ファイルを組み込みます。
-L$DB2PATH/lib リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。
-R$DB2PATH/lib 実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、$HOME/sqllib/lib。
-ldb2 DB2 ライブラリーとリンクします。
-ldb2apie DB2 API エンジン・ライブラリーとリンクして、LOB ロケーターを使用できるようにします。
-G 共用ライブラリーを生成します。
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。

ユーザー定義関数プログラム udfsrv をソース・ファイル udfsrv.c から作成するには、次のように入力します。

```
bldudf udfsrv
```

スクリプト・ファイルは、 UDF を sqllib/function ディレクトリーにコピーします。

必要であれば、UDF にファイル・モードを設定してクライアント・アプリケーションから実行できるようにします。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。このプログラムの DB2 CLI および組み込み SQL バージョンが提供されています。

DB2 CLI udfcli プログラム は、スクリプト・ファイル bldcli を使用して、sqllib/samples/cli にあるソース・ファイル udfcli.c から作成できます。詳細については、326ページの『DB2 CLI アプリケーション』を参照してください。

組み込み SQL udfcli プログラムは、 スクリプト・ファイル bldapp を使用して、 sqllib/samples/c にあるソース・ファイル udfcli.sqc から作成できます。 詳細については、 333ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF を呼び出すには、次の実行可能名を入力して、サンプルの呼び出しアプリケーションを実行します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv ライブラリーから ScalarUDF 関数を呼び出します。

マルチスレッド・アプリケーション

Solaris 上で SPARCompiler C を使用するマルチスレッド・アプリケーションは、 -mt でコンパイルおよびリンクする必要があります。これは、-D_REENTRANT をプリプロセッサーに渡し、 -lthread をリンカーに渡します。 POSIX スレッドには、リンカーに渡すための -lpthread も必要です。さらに、コンパイラー・オプション -D_POSIX_PTHREAD_SEMANTICS を使うことによって、 POSIX 変形関数 (getpwnam_r() など) を使用できます。

スクリプト・ファイル bldmt は sqllib/samples/c にあり、組み込み SQL マルチスレッド・プログラムを作成するためのコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。 第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。第 3 パラメーター $3 にはそのデータベースのユーザー ID を、また $4 にはパスワードを指定します。 第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名、ユーザー ID、およびパスワードは任意指定です。 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。

```
#! /bin/ksh
# bldmt script file -- Solaris
# Builds a C multi-threaded embedded SQL program.
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
```

```
# Precompile and bind the program.
embprep $1 $2 $3 $4
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi
# Compile the program.
cc $CFLAGS_64 -mt -D_POSIX_PTHREAD_SEMANTICS -I$DB2PATH/include -c $1.c
# Link the program.
cc $CFLAGS_64  -mt -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -lpthread
```

上記の -mt、-D_POSIX_PTHREAD_SEMANTICS、-lpthread の各オプションだけでなく、残りのコンパイルとリンクのオプションも組み込み SQL スクリプト・ファイル bldapp で使われているものと同じです。 これらのオプションについては、 333ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル thdsrver.sqc からサンプル・プログラム thdsrver を作成するには、次のように入力します。

```
bldmt thdsrver
```

結果として、実行可能ファイル thdsrver が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
thdsrver
```

SPARCompiler C++

この節では以下のトピックを取り上げています。

- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー
- ユーザー定義関数 (UDF)
- マルチスレッド・アプリケーション

DB2 API と組み込み SQL アプリケーション

スクリプト・ファイル bldapp は sqllib/samples/cpp にあり、 DB2 アプリケーション・プログラムを作成するためのコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を含まないプログラムに必要な唯一のパラメーターです。組み込

み SQL プログラムを作成するためにはデータベースへの接続が必要なため、3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。 データベース名を指定しない場合は、デフォルトの sample データベースを使用します。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- Solaris
# Builds a C++ application program.
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqC" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.C error-checking utility.
  CC $CFLAGS_64 -I$DB2PATH/include -c utilemb.C
else
  # Compile the utilapi.C error-checking utility.
  CC $CFLAGS_64 -I$DB2PATH/include -c utilapi.C
fi
# Compile the program.
CC $CFLAGS_64 -I$DB2PATH/include -c $1.C
if [[ -f $1".sqC" ]]
then
  # Link the program with utilemb.o
  CC $CFLAGS_64 -o $1 $1.o utilemb.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -mt
else
  # Link the program with utilapi.o
  CC $CFLAGS_64 -o $1 $1.o utilapi.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -mt
fi
```

bldapp のコンパイルおよびリンク・オプション

コンパイル・オプション

CC C++ コンパイラー。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 ″-xarch=v9″ の値を含みます。それ以外の場合は、値を含みません。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。 たとえば、 `$HOME/sqllib/include` のように指定します。

-c コンパイルのみを実行し、リンクは実行しません。 このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

リンク・オプション

CC コンパイラーをリンカーのフロントエンドとして使用します。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 ″-xarch=v9″ の値を含みます。それ以外の場合は、値を含みません。

-o $1 実行可能ファイルを指定します。

$1.o プログラム・オブジェクト・ファイルを組み込みます。

utilemb.o
組み込み SQL プログラムの場合に、エラー・チェックを行う組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。

utilapi.o
非組み込み SQL プログラムの場合に、エラー・チェックを行う DB2 API ユーティリティー・オブジェクト・ファイルを含みます。

-L$DB2PATH/lib
リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、 `$HOME/sqllib/lib`。 `-L` オプションを指定しないと、コンパイラーはパスとして `/usr/lib:/lib` を想定します。

-R$DB2PATH/lib
実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、 `$HOME/sqllib/lib`。

-ldb2 DB2 ライブラリーとリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.C から組み込み SQL を含まない DB2 API サンプル・プログラム client を作成するには、次のように入力します。

```
bldapp client
```

結果として、実行可能ファイル client が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqC から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

注: 71ページの『UDF およびストアード・プロシージャーに関する C++ 考慮事項』にある、 C++ ストアード・プロシージャーの構築に関する情報を参照してください。

スクリプト・ファイル bldsrv は sqllib/samples/cpp にあり、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが含まれています。スクリプト・ファイルは、ストアード・プロシージャーを共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 では、ソース・ファイルの名前を指定します。第 2 パラメーター $2 では、接続したいデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターは必要ありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。 データベース名は任意で指定します。データベース名を指定しない場合、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。

スクリプト・ファイルは、ソース・ファイル名 $1 を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldsrv script file -- Solaris
# Builds a C++ stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi
# Compile the program.
CC $CFLAGS_64 -Kpic -I$DB2PATH/include -c $1.C
# Link the program and create a shared library
CC $CFLAGS_64 -G -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -mt
```

```
# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

CC C++ コンパイラー。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 "-xarch=v9" の値を含みます。それ以外の場合は、値を含みません。

-Kpic 共用ライブラリー用の位置独立コードを生成します。

-I$DB2PATH/include
DB2 インクルード・ファイルのロケーションを指定します。たとえば、 -I$DB2PATH/include。

-c コンパイルのみを実行し、リンクは実行しません。 このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

リンク・オプション

CC コンパイラーをリンカーのフロントエンドとして使用します。

$CFLAGS_64
'BUILD_64BIT=true' がコメント解除されている場合に、 "-xarch=v9" の値を含みます。それ以外の場合は、値を含みません。

-G 共用ライブラリーを生成します。

-o $1 実行可能ファイルを指定します。

$1.o プログラム・オブジェクト・ファイルを組み込みます。

-L$DB2PATH/lib
リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。

-R$DB2PATH/lib
実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。

-ldb2 DB2 ライブラリーとリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

sample データベースに接続している場合、ソース・ファイル spserver.sqC からサンプル・プログラム spserver を作成するには、次のように入力します。

```
bldsrv spserver
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv spserver database
```

スクリプト・ファイルは、共用ライブラリーをサーバー上の sqllib/function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

共用ライブラリー spserver を作成したなら、クライアント・アプリケーション spclient を構築することができます。これは、共用ライブラリー内のストアード・プロシージャーを呼び出すアプリケーションです。

spclient は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 343ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

共用ライブラリーにアクセスするには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは共用ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。ストアード・プロシージャーは、出力をクライアント・アプリケーションに戻します。

ユーザー定義関数 (UDF)

注: 71ページの『UDF およびストアード・プロシージャーに関する C++ 考慮事項』 にある、 C++ UDF の構築に関する情報を参照してください。

スクリプト・ファイル bldudf は sqllib/samples/cpp にあり、 UDF を作成するためのコマンドが含まれています。 UDF には組み込み SQL ステートメントは含まれていません。 したがって、データベースへの接続、またはプログラムのプリコンパイルおよびバインドは行いません。

パラメーター $1 には、ソース・ファイルの名前を指定します。 スクリプト・ファイルは、そのソース・ファイル名を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldudf script file -- Solaris
# Builds a C++ UDF library
# Usage: bldudf <prog_name>
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi
# Compile the program.
if [[ -f $1".c" ]]
then
  CC $CFLAGS_64 -Kpic -I$DB2PATH/include -c $1.c
elif [[ -f $1".C" ]]
then
  CC $CFLAGS_64 -Kpic -I$DB2PATH/include -c $1.C
```

```
fi
# Link the program and create a shared library.
CC $CFLAGS_64 -G -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -ldb2apie
# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldudf のコンパイルおよびリンク・オプション
コンパイル・オプション **CC** C++ コンパイラー。 **$CFLAGS_64** 'BUILD_64BIT=true' がコメント解除されている場合に、 ″-xarch=v9″ の値を含みます。それ以外の場合は、値を含みません。 **-Kpic** 共用ライブラリー用の位置独立コードを生成します。 **-I$DB2PATH/include** DB2 インクルード・ファイルのロケーションを指定します。たとえば、 $HOME/sqllib/include のように指定します。 **-c** コンパイルのみを実行し、リンクは実行しません。このスクリプト・ファイルでは、コンパイルとリンクは別個のステップです。

bldudf のコンパイルおよびリンク・オプション
リンク・オプション **CC** コンパイラーをリンカーのフロントエンドとして使用します。 **$CFLAGS_64** 'BUILD_64BIT=true' がコメント解除されている場合に、 "-xarch=v9" の値を含みます。それ以外の場合は、値を含みません。 **-G** 共用ライブラリーを生成します。 **-o $1** 実行可能ファイルを指定します。 **$1.o** プログラム・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/lib** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーはパスとして /usr/lib:/lib を想定します。 **-R$DB2PATH/lib** 実行時の DB2 共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 **-ldb2** DB2 ライブラリーとリンクします。 **-ldb2apie** DB2 API エンジン・ライブラリーとリンクして、LOB ロケーターを使用できるようにします。 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。

ユーザー定義関数プログラム udfsrv をソース・ファイル udfsrv.c から作成するには、次のように入力します。

```
bldudf udfsrv
```

スクリプト・ファイルは、 UDF を sqllib/function ディレクトリーにコピーします。

必要であれば、UDF にファイル・モードを設定してクライアント・アプリケーションから実行できるようにします。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。udfcli プログラムは、スクリプト・ファイル bldapp を使用して、 sqllib/samples/cpp にあるソース・ファイル udfcli.sqC から作成します。 詳細については、 343ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF を呼び出すには、次の実行可能名を入力して、サンプルの呼び出しアプリケーションを実行します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv ライブラリーの ScalarUDF 関数を呼び出します。

マルチスレッド・アプリケーション

Solaris 上で SPARCompiler C++ を使用するマルチスレッド・アプリケーションは、 -mt でコンパイルおよびリンクする必要があります。これは、-D_REENTRANT をプリプロセッサーに渡し、 -lthread をリンカーに渡します。 POSIX スレッドには、リンカーに渡すための -lpthread も必要です。さらに、コンパイラー・オプション -D_POSIX_PTHREAD_SEMANTICS を使うことによって、 POSIX 変形関数 (getpwnam_r() など) を使用できます。

スクリプト・ファイル bldmt は sqllib/samples/cpp にあり、組み込み SQL マルチスレッド・プログラムを作成するためのコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。 第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。第 3 パラメーター $3 にはそのデータベースのユーザー ID を、また $4 にはパスワードを指定します。 第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名、ユーザー ID、およびパスワードは任意指定です。 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。

```
#! /bin/ksh
# bldmt script file -- Solaris
# Builds a C++ multi-threaded embedded SQL program
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2 $3 $4
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi
# Compile the program.
CC $CFLAGS_64 -mt -D_POSIX_PTHREAD_SEMANTICS -I$DB2PATH/include -c $1.C
# Link the program.
CC $CFLAGS_64 -mt -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -lpthread
```

上記の -mt、-D_POSIX_PTHREAD_SEMANTICS、-lpthread の各オプションだけでなく、残りのコンパイルとリンクのオプションも組み込み SQL スクリプト・ファイル bldapp で使われているものと同じです。 これらのオプションについては、 343ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ソース・ファイル thdsrver.sqC からサンプル・プログラム thdsrver を作成するには、次のように入力します。

```
bldmt thdsrver
```

結果として、実行可能ファイル thdsrver が作成されます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
thdsrver
```

Micro Focus COBOL

この節では、以下のトピックについて記載します。

- コンパイラーの使用
- DB2 API と DB2 組み込みアプリケーション
- 組み込み SQL ストアード・プロシージャー

コンパイラーの使用

組み込み SQL および DB2 API 呼び出しを含むアプリケーションを開発しており、 Micro Focus COBOL コンパイラーを使用している場合には、以下の点に留意してください。

- コマンド行プロセッサーのコマンド db2 prep を使ってアプリケーションをプリコンパイルする場合は、 target mfcob オプション (デフォルト) を使ってください。
- 組み込みプリコンパイラー・フロントエンド、実行時解釈プログラム、または Animator デバッガーを使うためには、次のように Micro Focus が提供する mkrts コマンドを実行して、 DB2 Generic API 入り口点を Micro Focus 実行時モジュール rts32 に追加します。
  1. root としてログインします。
  2. 次のディレクトリーにある引き数を指定して、mkrts を実行します。

     ```
     /opt/IBMdb2/V7.1/lib/db2mkrts.args
     ```
- DB2 COBOL COPY ファイル・ディレクトリーを、 Micro Focus COBOL 環境変数 COBCPY に含める必要があります。 COBCPY 環境変数には、COPY ファイルのロケーションを指定します。 Micro Focus COBOL 用の

DB2 COPY ファイルは、データベース・インスタンス・ディレクトリーの下にある sqllib/include/cobol_mf にあります。

このディレクトリーを含めるには、次のように入力します。

```
export COBCPY=$COBCPY:$HOME/sqllib/include/cobol_mf
```

注: COBCPY を .profile ファイル中に設定することもできます。

DB2 API と組み込み SQL アプリケーション

sqllib/samples/cobol_mf にあるスクリプト・ファイル bldapp には、 DB2 アプリケーション・プログラムを作成するためのコマンドが含まれています。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を含まないプログラムに必要な唯一のパラメーターです。 組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは $2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは $3 で、データベースのユーザー ID を指定します。そしてもう 1 つが $4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。 データベース名を指定しない場合は、デフォルトの sample データベースを使用します。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
#! /bin/ksh
# bldapp script file -- Solaris
# Builds a Micro Focus COBOL application program
# Usage: bldapp [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqb" ]]
then
  embprep $1 $2 $3 $4
fi
# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$DB2PATH/include/cobol_mf:$COBCPY

# Compile the checkerr.cbl error-checking utility.
cob -cx checkerr.cbl

# Compile the program.
cob -cx $1.cbl
```

```
# Link the program.
cob -x $1.o checkerr.o -L$DB2PATH/lib -ldb2 -ldb2gmf
```

bldapp のコンパイルおよびリンク・オプション
コンパイル・オプション
cob Micro Focus COBOL コンパイラー。
-cx オブジェクト・モジュールにコンパイルします。
リンク・オプション
cob コンパイラーをリンカーのフロントエンドとして使用します。
-x 実行可能プログラムを指定します。
$1.o プログラム・オブジェクト・ファイルを組み込みます。
checkerr.o エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。
-L$DB2PATH/lib リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。
-ldb2 DB2 ライブラリーとリンクします。
-ldb2gmf Micro Focus COBOL 用 DB2 例外ハンドラー・ライブラリーとリンクします。
他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.cbl から組み込み SQL を含まないサンプル・プログラム client を作成するには、次のように入力します。

```
bldapp client
```

結果として、実行可能ファイル client ができます。 sample データベースに対してこの実行可能ファイルを実行するには、次のように入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqb から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

```
bldapp updat
```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

注:

1. Solaris 上で Micro Focus ストアード・プロシージャーを作成する前に、以下のコマンドを実行してください。

   ```
   db2stop
   db2set DB2LIBPATH=$LD_LIBRARY_PATH
   db2set DB2ENVLIST="COBDIR LD_LIBRARY_PATH"
   db2set
   db2start
   ```

   db2stop がデータベースを停止するようにしてください。最後の db2set コマンドが設定値を検査するために出されます。 DB2LIBPATH および DB2ENVLIST が正しく設定されるようにしてください。

2. Solaris 上で使用される Micro Focus COBOL コンパイラーの最近のバージョンの中には、静的にリンクされたストアード・プロシージャーを作成する

のに使えないものもあります。 makefile とスクリプト・ファイルの bldsrv 自体は、動的にリンクされたストアード・プロシージャーを作成できるように変更されています。

この動的にリンクされたストアード・プロシージャーをリモート・クライアント・アプリケーションが正常に呼び出すためには、ストアード・プロシージャーが実行される直前にそのストアード・プロシージャーが常駐するサーバーで呼び出すために Micro Focus COBOL ルーチンの cobinit() が必要です。 makefile またはスクリプト・ファイル bldsrv の実行中に、これを成し遂げるラッパー・プログラムが作成されます。 次に、ラッパー・プログラムはストアード・プロシージャー・コードとリンクしてストアード・プロシージャーの共用ライブラリーを形成します。このラッパー・プログラムを使用するために、クライアント・アプリケーションが x という名前のストアード・プロシージャーを呼び出すには、 x の代わりに x_wrap を呼び出さなければなりません。

ラッパー・プログラムの詳細については、この節で後述します。

sqllib/samples/cobol_mf にあるスクリプト・ファイル bldsrv には、ストアード・プロシージャーを作成するためのコマンドが含まれています。 スクリプト・ファイルは、ストアード・プロシージャーを共用ライブラリーの中にコンパイルしますが、それはクライアント・アプリケーションから呼び出すことができます。

第 1 パラメーター $1 には、ソース・ファイルの名前を指定します。第 2 パラメーター $2 には、接続先のデータベースの名前を指定します。 ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

最初のパラメーター (ソース・ファイル名) だけが、必須です。 データベース名は任意で指定します。データベース名を指定しない場合、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。

スクリプト・ファイルは、ソース・ファイル名 $1 を共用ライブラリー名として使います。

```
#! /bin/ksh
# bldsrv script file -- Solaris
# Builds a Micro Focus COBOL stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Precompile and bind the program.
embprep $1 $2
```

```
# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$DB2PATH/include/cobol_mf:$COBCPY

# Compile the program.
cob -cx $1.cbl
# Create the wrapper program for the stored procedure.
wrapsrv $1

# Link the program creating shared library $1 with main entry point ${1}_wrap
cob -x -o $1 ${1}_wrap.c $1.o -Q -G -L$DB2PATH/lib -ldb2 -ldb2gmf
# Copy the shared library to the sqllib/function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

cob COBOL コンパイラー。

-cx オブジェクト・モジュールにコンパイルします。

リンク・オプション

cob リンク編集をするコンパイラーを使用します。

-x 実行可能プログラムを作成します。

-o $1 実行可能プログラムを指定します。

${1}_wrap.c
ラッパー・プログラムを指定します。

$1.o プログラム・オブジェクト・ファイルを指定します。

-Q

-G

-L$DB2PATH/lib
DB2 実行時共用ライブラリーのロケーションを指定します。たとえば、 $HOME/sqllib/lib。 -L オプションを指定しないと、コンパイラーは次のパスを想定します。 /usr/lib:/lib。

-ldb2 DB2 ライブラリーとリンクします。

-ldb2gmf
Micro Focus COBOL 用 DB2 例外ハンドラー・ライブラリーとリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ラッパー・プログラム wrapsrv は、ストアード・プロシージャーが実行される直前に、 Micro Focus COBOL のルーチン cobinit() が呼び出される原因になります。 その内容は以下のとおりです。

```
#! /bin/ksh
# wrapsrv script file
# Creates the wrapper program for Micro Focus COBOL stored procedures
# Usage: wrapsrv <stored_proc>
# Note: The client program calls "<stored_proc>_wrap" not "<stored_proc>"

# Create the wrapper program for the stored procedure.
cat << WRAPPER_CODE > ${1}_wrap.c
#include <stdio.h>
void cobinit(void);
int $1(void *p0, void *p1, void *p2, void *p3);
int main(void)
{
  return 0;
}
int ${1}_wrap(void *p0, void *p1, void *p2, void *p3)
{
  cobinit();
  return $1(p0, p1, p2, p3);
}
WRAPPER_CODE
```

サンプル・データベースに接続している場合、ソース・ファイル outsrv.sqb からサンプル・プログラム outsrv を作成するには、次のように入力します。

```
bldsrv outsrv
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv outsrv database
```

このスクリプト・ファイルは、パス sqllib/function 内のサーバーにストアード・プロシージャーをコピーします。

必要であれば、ストアード・プロシージャーにファイル・モードを設定して、クライアント・アプリケーションからアクセスできるようにします。

ストアード・プロシージャー outsrv を構築してしまえば、そのストアード・プロシージャーを呼び出すクライアント・アプリケーション outcli を構築できます。 outcli は、スクリプト・ファイル bldapp を使用して構築することができます。詳細については、 355ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

```
outcli database userid password
```

ここで、それぞれは次のものを表します。

データベース
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションはストアード・プロシージャー・ライブラリー outsrv にアクセスし、ストアード・プロシージャー関数をサーバー・データベース上で実行します。この出力は、クライアント・アプリケーションに戻されます。

ストアード・プロシージャーの終了

ストアード・プロシージャーを開発したならば、次のステートメントを使って、それを終了します。

```
move SQLZ-HOLD-PROC to return-code.
```

このステートメントで、ストアード・プロシージャーはクライアント・アプリケーションに正しく戻ります。 ストアード・プロシージャーが、ローカル COBOL クライアント・アプリケーションによって呼び出された場合、これは特に重要です。

第13章 Windows 32 ビット・オペレーティング・システムで使用するアプリケーションの構築

この章は、 Windows 32 ビット・オペレーティング・システムでアプリケーションを構築するための詳細な情報を提供します。バッチ・ファイルでは、 db2 で始まるコマンドは、コマンド行プロセッサー (CLP) コマンドです。 DB2 コマンドについてのさらに詳細な情報が必要であれば、 *コマンド解説書* を参照してください。

Windows 32 ビット・オペレーティング・システム用の DB2 アプリケーション開発の最新の更新事項については、次の Web ページを参照してください。

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

注:

1. Windows 32 ビット・アプリケーション上のすべてのアプリケーション (組み込み SQL と非組み込み SQL の両方) は、オペレーティング・システムのコマンド・プロンプトではなく、 DB2 コマンド・ウィンドウで構築されなければなりません。
2. 変数 %DB2PATH% を含むご使用のプログラムで使用されるパス名は、"%DB2PATH%¥function" のように引用符で囲まなければなりません。Windows 32 ビット・オペレーティング・システム バージョン 7.1 上での DB2 のデフォルトのインストールでは、 ¥Program Files¥sqllib で、これにはスペースが含まれます。引用符で囲まないと、″the syntax of the command is incorrect″ などのエラーを受け取ることがあります。この章では、コマンドまたはコード例の一部として指定される場合にのみパス名を引用符で囲みます。

WCHARTYPE CONVERT プリコンパイル・オプション

WCHARTYPE プリコンパイル・オプションは、 wchar_t データ・タイプを使用する、マルチバイト形式またはワイド文字形式のいずれかで図形データを処理します。 このオプションの詳細については、*アプリケーション開発の手引き* を参照することができます。

Windows 32 ビット・オペレーティング・システムの場合、 WCHARTYPE CONVERT オプションは、 Microsoft Visual C++ コンパイラーでコンパイルされたアプリケーション用にサポートされています。アプリケーションがデータを DB2 データベースにデータベース・コード・ページとは異なるコード・ページで挿入する場合は、このコンパイラーで CONVERT オプションを使用しないでください。 DB2 は通常はこのような状況でコード・ページ変換を実行します。しかし、Microsoft C 実行時環境は、特定の 2 バイト文字の代入文字は処理しません。 これは、実行時変換エラーとなる場合があります。

WCHARTYPE CONVERT オプションは、 IBM VisualAge C++ コンパイラーでコンパイルされたアプリケーション用にはサポートされていません。 このコンパイラーをご使用の場合は、 WCHARTYPE にはデフォルトの NOCONVERT オプションを使用してください。 NOCONVERT オプションを使用すると、アプリケーションとデータベース・マネージャーとの間の暗黙の文字変換は起きなくなります。図形ホスト変数のデータは、変換されない 2 バイト文字セット (DBCS) 文字として、データベース・マネージャーとの間で送受信されます。

図形データをワイド文字形式からマルチバイト形式に変換したい場合は、wcstombs() 関数を使用してください。たとえば、次のようにします。

```
wchar_t widechar[200];
wchar_t mb[200];
wcstombs((char *)mb,widechar,200);
EXEC SQL INSERT INTO TABLENAME VALUES(:mb);
```

同様に、mbstowcs() 関数を使用して、マルチバイト形式をワイド文字形式に変換することができます。

アプリケーションが静的に C 実行時ライブラリーにバインドされている場合は、アプリケーションから setlocale() 呼び出しを出さないでください。これは、C 実行時変換エラーとなる可能性があります。アプリケーションが動的に C 実行時ライブラリーにバインドされている場合は、 setlocale() の使用は問題になりません。これは、ストアード・プロシージャーにも当てはまります。

オブジェクトのリンクと埋め込みデータベース (OLE DB) 表関数

DB2 は、OLE DB 表関数をサポートします。これらの関数については、CREATE FUNCTION DDL を作成する他にアプリケーションを構築する必要はありません。 DB2 の %DB2PATH%¥samples¥oledb ディレクトリーに、 OLE DB 表関数のサンプル・ファイルが提供されています。これらは、コマンド行プロセッサー (CLP) のファイルです。これらのファイルは、以下のステップで構築できます。

1. database_name への db2 の接続
2. db2 -t -v -f file_name.db2
3. db2 の終了

database_name は接続先のデータベース、 file_name は CLP ファイルの名前 (拡張子は .db2) です。

OLE DB 表関数の完全な説明については、*アプリケーション開発の手引き* を参照してください。

Microsoft Visual Basic

注: Windows 32 ビット・オペレーティング・システム用の DB2 AD クライアントでは、 Microsoft Visual Basic のプリコンパイラーを提供していません。

この節では、以下のトピックを取り上げています。

- ActiveX Data Objects (ADO)
- リモート・データ・オブジェクト (RDO)
- オブジェクトのリンクと埋め込み (OLE) オートメーション

ActiveX Data Objects (ADO)

ActiveX Data Object (ADO) を使用すれば、 OLE DB Provider を使用して、データベース・サーバー内のデータにアクセスしたり、操作したりするアプリケーションを作成できます。 ADO の主要な利点は、速度が速く、使用が容易で、メモリーのオーバーヘッドが少なく、ディスク・フットプリントが小さいことです。

ADO を Microsoft Visual Basic で使用する場合は、 ADO タイプのライブラリーへの参照を設定する必要があります。以下のようにします。

1. プロジェクト・メニューから「参照 (References)」を選択する。
2. 「Microsoft ActiveX Data Object <version_number> ライブラリー (Microsoft ActiveX Data Objects <version_number> Library)」のボックスにチェックマークを付ける。
3. 「OK」をクリックします。

<version_number> は、ADO ライブラリーの現行バージョンです。

このようにすると、 VBA オブジェクト・ブラウザーと IDE エディターを介して、 ADO オブジェクト、メソッド、および特性にアクセスできるようになります。

完全な Visual Basic プログラムには、フォーム、その他のグラフィカルな要素が含まれており、このプログラムは Visual Basic 環境の内部で表示する必要があります。以下に、 DB2 `sample` にアクセスするプログラムの一部をなすコマンドを示します。これらのコマンドは、ODBC のカタログに記録されます。

接続を確立します。

```
Dim db As Connection
Set db = New Connection
```

ローカル・カーソル・ライブラリーによって提供されるクライアント側のカーソルを設定します。

```
db.CursorLocation = adUseClient
```

ADO が Microsoft ODBC Driver を使用するようにプロバイダーを設定し、ユーザー ID/パスワードなしで (つまり、現在のユーザーを使用) ″sample″ データベースをオープンします。

```
db.Open "SAMPLE"
```

レコード・セットを作成します。

```
Set adoPrimaryRS = New Recordset
```

SELECT ステートメントを使用して、レコード・セットにデータを入れます。

```
adoPrimaryRS.Open "select EMPNO,LASTNAME,FIRSTNME,MIDINIT,EDLEVEL,JOB
from EMPLOYEE Order by EMPNO", db
```

この時点で、プログラマーは ADO メソッドを使用して、次のレコード・セットに移動する場合など、データにアクセスできるようになります。

```
adoPrimaryRS.MoveNext
```

レコード・セットの現行レコードを削除します。

```
adoPrimaryRS.Delete
```

またプログラマーは、以下のようにして、個々のフィールドにアクセスすることもできます。

```
Dim Text1 as String
Text1 = adoPrimaryRS!LASTNAME
```

DB2 の %DB2PATH%¥samples¥ADO¥VB に、 Visual Basic ADO のサンプル・プログラムがあります。

Remote Data Object (RDO)

Remote Data Objects (RDO) は、 ODBC を介してリモート・データ・ソースにアクセスするための情報モデルを提供します。 RDO が提供するオブジェクトのセットを使用すれば、データベースへの接続、照会の実行、ストアード・プロシージャーの実行、結果の操作、変更のサーバーへのコミットが容易になります。 RDO は、リモート ODBC リレーショナル・データ・ソースにアクセスするために特別に設計されたものであり、複雑なアプリケーション・コードを使用せずに ODBC を使用することが容易になっています。そのため、ODBC ドライバーによって公開されるリレーショナル・データベースにアクセスするための主な手段となっています。 RDO はオープン・データベース・コネクティビティー (ODBC) API を介したシン・コード層を実装しています。また、接続の確立、結果セットとカーソルの作成を行い、ワークステーション資源を最小限に抑えて複雑なプロシージャーを実行するドライバー・マネージャーも実装しています。

RDO を Microsoft Visual Basic で使用する場合は、 Visual Basic のプロジェクトへの参照を設定する必要があります。以下のようにします。

1. プロジェクト・メニューから「参照 (References)」を選択する。
2. 「Microsoft Remote Data Object <Version Number>」のボックスにチェックマークを付ける。
3. 「OK」をクリックします。

<version_number> は、RDO の現行バージョンです。

完全な Visual Basic プログラムには、フォーム、その他のグラフィカルな要素が含まれており、このプログラムは Visual Basic 環境の内部で表示する必要があります。以下に、 DB2 プログラムの一部を成す Visual Basic コマンドを示します。 DB2 プログラムは、sample データベースに接続して、 EMPLOYEE 表のすべての列を選択するレコード・セットをオープンし、メッセージ・ウィンドウに従業員名を一人一人表示します。

```
Dim rdoEn As rdoEngine
Dim rdoEv As rdoEnvironment
Dim rdoCn As rdoConnection
Dim Cnct$
Dim rdoRS As rdoResultset
Dim SQLQueryDB As String
```

接続ストリングを割り当てます。

```
Cnct$ = "DSN=SAMPLE;UID=;PWD=;"
```

RDO 環境を設定します。

```
Set rdoEn = rdoEngine
Set rdoEv = rdoEn.rdoEnvironments(0)
```

データベースに接続します。

```
Set rdoCn = rdoEv.OpenConnection("", , , Cnct$)
```

レコード・セットの SELECT ステートメントを割り当てます。

```
SQLQueryDB = "SELECT * FROM EMPLOYEE"
```

レコード・セットをオープンして、照会を実行します。

```
Set rdoRS = rdoCn.OpenResultset(SQLQueryDB)
```

レコード・セットの終わりに While not を置き、メッセージ・ボックスに、表にあるラストネームとファーストネーム (1 回につき 1 人の従業員) を表示します。

```
While Not rdoRS.EOF
MsgBox rdoRS!LASTNAME & ", " & rdoRS!FIRSTNME
```

レコード・セット内の次の行に移動します。

```
rdoRS.MoveNext
Wend
```

プログラムをクローズします。

```
rdoRS.Close
rdoCn.Close
rdoEv.Close
```

DB2 の %DB2PATH%¥samples¥RDO に、 Visual Basic RDO のサンプル・プログラムがあります。

オブジェクトのリンクと埋め込み (OLE) オートメーション

この節では、 Microsoft Visual Basic でのオブジェクトのリンクと埋め込み (OLE) オートメーション UDF、およびストアード・プロシージャーの OLE オートメーション制御プログラム・サンプルへのアクセスについて説明します。

OLE オートメーション UDF およびストアード・プロシージャーは、 OLE が言語独立であるので、 OLE オートメーション・サーバーのメソッドを公開し、そのメソッドを DB2 を使用する UDF として登録することによって、任意の言語で実現できます。 OLE オートメーション・サーバーの開発をサポートするアプリケーション開発環境には、以下の特定のバージョンが含まれます。 Microsoft Visual Basic、Microsoft Visual C++、Microsoft Visual J++、Microsoft FoxPro、Borland Delphi、Powersoft PowerBuilder、および Micro Focus COBOL。 さらに、たとえば Microsoft Visual J++ に付属するような、OLE 用に同梱されている Java の簡単なオブジェクトも、 OLE オートメーションを介してアクセスできます。

OLE オートメーション・サーバーの開発の詳細については、該当するアプリケーション開発環境の資料を参照する必要があります。 OLE オートメーションを使用した DB2 プログラミングの詳細については、 *アプリケーション開発の手引き* を参照してください。

OLE オートメーション UDF およびストアード・プロシージャー

Microsoft Visual Basic は、OLE オートメーション・サーバーの作成をサポートします。 新しい種類のオブジェクトは、 Visual Basic ではクラス・モジュールを Visual Basic プロジェクトに追加することによって作成します。 メソッドは、公用サブプロシージャーをクラス・モジュールに追加することによって作成します。 これらの公用プロシージャーは、 DB2 に OLE オートメーション UDF およびストアード・プロシージャーとして登録できます。 OLE サーバーの作成および構築の詳細については、 Microsoft Visual Basic のマニュアル、 *Creating OLE Servers, Microsoft Corporation, 1995*、および Microsoft Visual Basic によって提供される OLE サンプルを参照してください。

DB2 は、 Microsoft Visual Basic に含まれている OLE オートメーション UDF およびストアード・プロシージャーのサンプルを提供しており、それはディレクトリー %DB2PATH%¥samples¥ole¥msvb にあります。 OLE オートメーション UDF およびストアード・プロシージャーのサンプルを作成して実行する方法については、 %DB2PATH%¥samples¥ole の README ファイルを参照してください。

Microsoft Visual C++

この節では以下のトピックを取り上げています。

- ActiveX Data Objects (ADO)
- オブジェクトのリンクと埋め込み (OLE) オートメーション
- DB2 CLI アプリケーション
- DB2 CLI ストアード・プロシージャー
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー
- ユーザー定義関数 (UDF)

注: Visual C++ コンパイラーは、 %DB2PATH%¥samples¥c ディレクトリーと %DB2PATH%¥samples¥cpp ディレクトリーにある、 C と C++ の両方のサンプル・プログラムに使用されます。 同じバッチ・ファイルがこれらの 2 つのディレクトリー内に置かれています。これらのディレクトリーにあるバッチ・ファイルには、ファイルの拡張子に応じて、 C または C++ のどちらかのソース・ファイルを受け入れるコマンドが含まれています。 バッチ・ファイルを使用しない最初の 2 つのトピック「ActiveX Data Objects (ADO)」と「オブジェクトのリンクと埋め込み (OLE) オートメーション」を除き、この節ではバッチ・ファイルを使用してプログラムを作成する方法を例を挙げて説明します。

ActiveX Data Objects (ADO)

以下に示す変更を行うと、Visual C++ を使用する DB2 ADO プログラムは、正規の C++ プログラムと同じようにコンパイルできるようになります。

C++ ソース・プログラムを ADO プログラムとして実行するには、以下の IMPORT ステートメントをソース・プログラム・ファイルの先頭に置くことができます。

```
#import "C:¥program files¥common files¥system¥ado¥msado<VERSION NUMBER>.dll" \
no_namespace \
rename( "EOF", "adoEOF")
```

<VERSION NUMBER> は、ADO ライブラリーのバージョン番号です。

プログラムがコンパイルされたら、ユーザーは msado<VERSION NUMBER>.dll が指定されたパスにあるかどうか検証する必要があります。

C:¥program files¥common files¥system¥ado を環境変数 LIBPATH に追加し、短くした IMPORT ステートメントを以下のようにソース・ファイルで使用することもできます。

```
#import <msado<VERSION NUMBER>.dll> \
no_namespace \
rename( "EOF", "adoEOF")
```

DB2 サンプル・プログラム BLOBAccess.dsp では、この方法が使用されています。

この IMPORT ステートメントにより、 DB2 プログラムには ADO ライブラリーへのアクセス権が与えられます。これで、 Visual C++ プログラムも他のプログラムと同様にコンパイルできるようになります。また、DB2 API または DB2 CLI など、別のプログラミング・インターフェースを使用する場合、プログラム作成の詳細については、この章の該当する節を参照してください。

DB2 の %DB2PATH%¥samples¥ADO¥VC に、 Visual C++ ADO のサンプル・プログラムがあります。

オブジェクトのリンクと埋め込み (OLE) オートメーション

この節では、 Microsoft Visual C++ でのオブジェクトのリンクと埋め込み (OLE) オートメーション UDF、およびストアード・プロシージャーの OLE オートメーション制御プログラム・サンプルについて説明します。

OLE オートメーション UDF およびストアード・プロシージャーは、 OLE が言語独立であるので、 OLE オートメーション・サーバーのメソッドを公開し、そのメソッドを DB2 を使用する UDF として登録することによって、任意の言語で実現できます。 OLE オートメーション・サーバーの開発をサポートするアプリケーション開発環境には、以下の特定のバージョンが含まれます。 Microsoft Visual Basic、Microsoft Visual C++、Microsoft Visual J++、Microsoft FoxPro、Borland Delphi、Powersoft PowerBuilder、および Micro Focus COBOL。 さらに、たとえば Microsoft Visual J++ に付属するような、OLE 用に同梱されている Java の簡単なオブジェクトも、 OLE オートメーションを介してアクセスできます。

OLE オートメーション・サーバーの開発の詳細については、該当するアプリケーション開発環境の資料を参照する必要があります。 OLE オートメーションを使用した DB2 プログラミングの詳細については、 *アプリケーション開発の手引き* を参照してください。

OLE オートメーション UDF およびストアード・プロシージャー

Microsoft Visual C++ は、 OLE オートメーション・サーバーの作成をサポートします。サーバーは、 Microsoft Foundation Classes および Microsoft Foundation Class アプリケーション・ウィザードを使用して、または Win32 アプリケーションとして実現することができます。 サーバーは、DLL または EXE にすることができます。 詳細については、 Microsoft Visual C++ の資料および Microsoft Visual C++ によって提供される OLE サンプルを参照してください。 DB2 用の Visual C++ UDF を作成するための情報については、 385 ページの『ユーザー定義関数 (UDF)』を参照してください。 DB2 CLI を使って Visual C++ ストアード・プロシージャーを作成するための情報については、 376ページの『DB2 CLI ストアード・プロシージャー』を参照してください。 DB2 用の Visual C++ 組み込み SQL ストアード・プロシージャーを作成するための情報については、 382ページの『組み込み SQL ストアード・プロシージャー』を参照してください。

DB2 は、 Microsoft Visual C++ に含まれている OLE オートメーション UDF およびストアード・プロシージャーのサンプルを提供しており、それはディレクトリー `%DB2PATH%¥samples¥ole¥msvc` にあります。 OLE オートメーション UDF およびストアード・プロシージャーのサンプルを作成して実行する方法については、 `%DB2PATH%¥samples¥ole` の `README` ファイルを参照してください。

DB2 CLI アプリケーション

`%DB2PATH%¥samples¥cli` にあるバッチ・ファイル `bldmcli.bat` には、 DB2 CLI プログラムを作成するためのコマンドが入っています。

このパラメーター `%1` には、ソース・ファイルの名前を指定します。

これは、唯一の必須パラメーターであり、組み込み SQL を含まない CLI プログラムに必要なパラメーターはこのパラメーターだけです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは `%2` で、接続するデータベースの名前を指定します。 3 番目のパラメーターは `%3` で、データベースのユーザー ID を指定します。そしてもう 1 つが `%4` で、データベースのパスワードを指定します。

プログラムに組み込み SQL (`.sqc` または `.sqx` 拡張子が付いている) が含まれている場合、 `embprep` バッチ・ファイルは、`.c` または `.cxx` 拡張子を持つプログラム・ファイルを生成して、プログラムをプリコンパイルするために呼び出されます。

```
@echo off
rem  bldmcli batch file - Windows 32-bit Operating Systems
rem  Builds a CLI program with Microsoft Visual C++.
rem  Usage: bldmcli prog_name [ db_name [ userid password ]]
if exist "%1.sqc" call embprep %1 %2 %3 %4
if exist "%1.sqx" call embprep %1 %2 %3 %4
rem Compile the error-checking utility.
cl -Z7 -Od -c -W1 -D_X86=1 -DWIN32 utilcli.c
rem  Compile the program.
if exist "%1.sqx" goto cpp
cl -Z7 -Od -c -W1 -D_X86=1 -DWIN32 %1.c
goto link_step
:cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx
rem  Link the program.
:link_step
link -debug:full -debugtype:cv -OUT:%1.exe %1.obj utilcli.obj db2cli.lib
@echo on
```

bldmcli のコンパイルおよびリンク・オプション

コンパイル・オプション

cl Microsoft Visual C++ コンパイラー。

-Z7 生成される C7 スタイル CodeView 情報。

-Od 最適化なし。最適化をオフにしてデバッガーを使用する方が簡単です。

-c コンパイルのみを実行し、リンクは実行しません。 本書では、コンパイルとリンクが別個のステップであることを前提としています。

-W1 警告レベルを設定します。

-D_X86_=1
32 ビット・オペレーティング・システムを Intel ベースのコンピューターで実行するために必要なコンパイラー・オプション。

-DWIN32
32 ビット・オペレーティング・システムに必要なコンパイラー・オプション。

bldmcli のコンパイルおよびリンク・オプション
リンク・オプション **link** リンク編集に 32 ビットのリンカーを使用します。 **-debug:full** デバッグ情報を組み込みます。 **-debugtype:cv** デバッガー・タイプを指定します。 **-OUT:%1.exe** 実行可能ファイルを指定します。 **%1.obj** オブジェクト・ファイルを組み込みます。 **utilcli.obj** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **db2cli.lib** DB2 CLI ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル tbinfo.c からサンプル・プログラム tbinfo を作成するには、次のように入力します。

```
bldmcli tbinfo
```

結果として、実行可能ファイル tbinfo が作成されます。この実行可能ファイルを実行するには、次の実行可能名を入力します。

```
tbinfo
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル dbusemx.sqc から組み込み SQL アプリケーション dbusemx を作成する場合、次の 3 つの方法があります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldmcli dbusemx
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldmcli dbusemx database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldmcli dbusemx database userid password
   ```

結果として、実行可能ファイル dbusemx が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   dbusemx
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   dbusemx database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   dbusemx database userid password
   ```

DB2 API を使用する DB2 CLI アプリケーション

DB2 には、CLI サンプル・プログラムが含まれています。このサンプル・プログラムは、DB2 API を使用してデータベースを作成およびドロップし、 CLI 機能を複数のデータベースで使用する方法を示します。 DB2 API を使用するサンプルは、 27ページの表7 にある CLI サンプル・プログラムの説明の中に示されています。

sqllib/samples/cli にあるスクリプト・ファイル bldmapi には、 DB2 API を持つ DB2 CLI プログラムを作成するためのコマンドが入っています。 このファイルは、データベースを作成およびドロップするための DB2 API が入った utilapi ユーティリティー・ファイルでコンパイルおよびリンクします。この点が、このファイルと bldmcli バッチ・ファイルの唯一の違いです。bldmapi と bldmcli の両方に共通のコンパイルおよびリンク・オプションについては、 372ページの『DB2 CLI アプリケーション』 を参照してください。

ソース・ファイル dbmconn.c からサンプル・プログラム dbmconn を作成するには、次のように入力します。

```
bldmapi dbmconn
```

結果として、実行可能ファイル dbmconn が作成されます。この実行可能ファイルを実行するには、次の実行可能名を入力します。

```
dbmconn
```

DB2 CLI ストアード・プロシージャー

%DB2PATH%¥samples¥cli にあるバッチ・ファイル bldmclis.bat には、 CLI ストアード・プロシージャーを作成するためのコマンドが入っています。バッチ・ファイルは、ストアード・プロシージャーをサーバー上の DLL 内に作成します。

このパラメーター %1 には、ソース・ファイルの名前を指定します。バッチ・ファイルでは、ソース・ファイル名 %1 を DLL 名に使用します。

```
@echo off
rem bldmclis.bat file - Windows 32-bit Operating Systems
rem Builds a CLI stored procedure using the Microsoft Visual C++ compiler.
rem Usage: bldmclis prog_name
if "%1" == "" goto error
rem Compile the program.
if exist "%1.cxx" goto cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c  utilcli.c
goto link_step
:cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx  utilcli.c
rem Link the program.
:link_step
link -debug:full -debugtype:cv -dll -out:%1.dll %1.obj utilcli.obj db2cli.lib -def:%1.def
rem Copy the stored procedure DLL to the 'function' directory
copy %1.dll "%DB2PATH%¥function"
goto exit
:error
echo Usage: bldmclis prog_name
:exit
@echo on
```

bldmclis のコンパイルおよびリンク・オプション

コンパイル・オプション

cl Microsoft Visual C++ コンパイラー。

-Z7 生成される C7 スタイル CodeView 情報。

-Od 最適化なし。最適化をオフにしてデバッガーを使用する方が簡単です。

-c コンパイルのみを実行し、リンクは実行しません。 本書では、コンパイルとリンクが別個のステップであることを前提としています。

-W2 警告レベルを設定します。

-D_X86_=1
32 ビット・オペレーティング・システムを Intel ベースのコンピューターで実行するために必要なコンパイラー・オプション。

-DWIN32
32 ビット・オペレーティング・システムに必要なコンパイラー・オプション。

bldmclis のコンパイルおよびリンク・オプション
リンク・オプション **link**　リンク編集に 32 ビットのリンカーを使用します。 **-debug:full** デバッグ情報を組み込みます。 **-debugtype:cv** デバッガー・タイプを指定します。 **-OUT:%1.dll** .DLL ファイルを作成します。 **%1.obj**　オブジェクト・ファイルを組み込みます。 **utilcli.obj** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **db2cli.lib** DB2 CLI ライブラリーとリンクします。 **-def:%1.def** モジュール定義ファイルを使用します。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル spserver.c から spserver ストアード・プロシージャーを作成するには、次のように入力します。

```
bldmclis spserver
```

このバッチ・ファイルは、CLI サンプル・プログラムと同じディレクトリーに入っている、モジュール定義ファイル spserver.def を使用してストアード・プロシージャーを作成します。 このバッチ・ファイルは、ストアード・プロシージャー DLL の spserver.dll をサーバー上の %DB2PATH%¥function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

ストアード・プロシージャー spserver を作成したなら、そのストアード・プロシージャーを呼び出す CLI クライアント・アプリケーション spclient を構築できます。

spclient は、スクリプト・ファイル bldmcli を使用して構築することができます。詳細については、 372ページの『DB2 CLI アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは、ストアード・プロシージャー・ライブラリー spserver にアクセスし、サーバー・データベース上のいくつかのストアード・プロシージャー関数を実行します。出力は、クライアント・アプリケーションに戻されます。

DB2 API と組み込み SQL アプリケーション

%DB2PATH%¥samples¥c と %DB2PATH%¥samples¥cpp にあるバッチ・ファイル bldmapp.bat には、組み込み SQL プログラムを作成するためのコマンドが入っています。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を含まないプログラムに必要な唯一のパラメーターです。 組み込み SQL プログラムを構築するにはデータベースへの接続が必要なので、 3 つ

のオプション・パラメーターが提供されます。第 2 パラメーター %2 は接続したいデータベースの名前を指定し、第 3 パラメーター %3 はデータベースのユーザー ID を指定し、 %4 はパスワードを指定します。

組み込み SQL プログラムの場合、 bldmapp は、プリコンパイルおよびバインド・ファイル embprep にパラメーターを渡します。データベース名を指定しない場合は、デフォルトの sample データベースを使用します。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
@echo off
rem bldmapp.bat -- Windows 32-bit operating systems
rem Builds a Microsoft Visual C++ application program
rem Usage: bldmapp prog_name [ db_name [ userid password ]]
if exist "%1.sqx" goto embedded
if exist "%1.sqc" goto embedded
goto non_embedded
:embedded
rem Precompile and bind the program.
call embprep %1 %2 %3 %4
rem  Compile the program.
if exist "%1.cxx" goto cpp_emb
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c utilemb.c
goto link_embedded
:cpp_emb
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx utilemb.cxx
rem Link the program.
:link_embedded
link -debug:full -debugtype:cv -out:%1.exe %1.obj utilemb.obj db2api.lib
goto exit
:non_embedded
rem  Compile the program.
if exist "%1.cxx" goto cpp_non
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c utilapi.c
goto link_non_embedded
:cpp_non
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx utilapi.cxx
rem Link the program.
:link_non_embedded
link -debug:full -debugtype:cv -out:%1.exe %1.obj utilapi.obj db2api.lib
:exit
@echo on
```

bldmapp のコンパイルおよびリンク・オプション

コンパイル・オプション

cl Microsoft Visual C++ コンパイラー。

-Z7 生成される C7 スタイル CodeView 情報。

-Od 最適化なし。最適化をオフにしてデバッガーを使用する方が簡単です。

-c コンパイルのみを実行し、リンクは実行しません。 本書では、コンパイルとリンクが別個のステップであることを前提としています。

-W2 警告レベルを設定します。

-D_X86_=1
32 ビット・オペレーティング・システムを Intel ベースのコンピューターで実行するために必要なコンパイラー・オプション。

-DWIN32
32 ビット・オペレーティング・システムに必要なコンパイラー・オプション。

リンク・オプション

link リンク編集に 32 ビットのリンカーを使用します。

-debug:full
デバッグ情報を組み込みます。

-debugtype:cv
デバッガー・タイプを指定します。

-out:%1.exe
ファイル名を指定します。

%1.obj オブジェクト・ファイルを組み込みます。

utilemb.obj
組み込み SQL プログラムの場合に、エラー・チェックを行う組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。

utilapi.obj
組み込み SQL プログラムでない場合に、エラー・チェックを行う DB2 API ユーティリティー・オブジェクト・ファイルを含みます。

db2api.lib
DB2 ライブラリーとリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

%DB2PATH%¥samples¥c のソース・ファイル client.c、または %DB2PATH%¥samples¥cpp のソース・ファイル client.cxx のどちらかから、DB2 API の組み込み SQL を含まないサンプル・プログラム client を作成するには、次のように入力します。

```
bldmapp client
```

結果として、実行可能ファイル client.exe が作成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を (拡張子なしで) コマンド行に入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

%DB2PATH%¥samples¥c の C ソース・ファイル updat.sqc、または %DB2PATH%¥samples¥cpp の C++ ソース・ファイル updat.sqx から組み込み SQL アプリケーション updat を構築する方法は 3 つあります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldmapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldmapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldmapp updat database userid password
   ```

結果として、実行可能ファイル updat.exe が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

updat *database userid password*

組み込み SQL ストアード・プロシージャー

%DB2PATH%¥samples¥c と %DB2PATH%¥samples¥cpp にあるバッチ・ファイル bldmsrv.bat には、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが入っています。 バッチ・ファイルは、ストアード・プロシージャーをサーバー上の DLL 内に作成します。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。第 2 パラメーター %2 には、接続先のデータベースの名前を指定します。 ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。 データベース名は任意で指定します。データベース名を指定しない場合、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。

バッチ・ファイルでは、ソース・ファイル名 %1 を DLL 名に使用します。

```
@echo off
rem bldmsrv.bat -- Windows 32-bit operating systems
rem Builds a Microsoft Visual C++ stored procedure
rem Usage: bldmsrv prog_name [ db_name ]
rem Precompile and bind the program.
call embprep %1 %2
rem Compile the program.
if exist "%1.cxx" goto cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c
goto link_step
:cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx
:link_step
rem Link the program.
link -debug:full -debugtype:cv -out:%1.dll -dll %1.obj db2api.lib -def:%1.def
rem Copy the stored procedure DLL to the 'function' directory
copy %1.dll "%DB2PATH%¥function"
@echo on
```

bldmsrv のコンパイルおよびリンク・オプション
コンパイル・オプション **cl** Microsoft Visual C++ コンパイラー。 **-Z7** 生成される C7 スタイル CodeView 情報。 **-Od** 最適化なし。 **-c** コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。 **-W2** 警告、エラー、重大、および回復不能エラー・メッセージを出力します。 **-D_X86_=1** 32 ビット・オペレーティング・システムを Intel ベースのコンピューターで実行するために必要なコンパイラー・オプション。 **-DWIN32** 32 ビット・オペレーティング・システムに必要なコンパイラー・オプション。
リンク・オプション **link** リンク編集にリンカーを使用します。 **-debug:full** デバッグ情報を組み込みます。 **-debugtype:cv** デバッガー・タイプを指定します。 **-out:%1.dll** .DLL ファイルを作成します。 **%1.obj** オブジェクト・ファイルを組み込みます。 **db2api.lib** DB2 ライブラリーとリンクします。 **-def:%1.def** モジュール定義ファイル。 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 ソース・ファイル spserver.sqc、または C++ ソース・ファイル spserver.sqx のどちらかから spserver ストアード・プロシージャーを作成するには、次のように入力します。

```
bldmsrv spserver
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldmsrv spserver database
```

このバッチ・ファイルは、サンプル・プログラムと同じディレクトリーに入っている、モジュール定義ファイル spserver.def を使用して DLL を作成します。 このバッチ・ファイルは、 DLL の spserver.dll をサーバー上の %DB2PATH%¥function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

ストアード・プロシージャー DLL spserver を作成したなら、そのストアード・プロシージャーを呼び出すクライアント・アプリケーション spclient を構築できます。

spclient は、スクリプト・ファイル bldmapp を使用して構築することができます。詳細については、 378ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは、ストアード・プロシージャー DLL spserver にアクセスし、サーバー・データベース上のいくつかのストアード・プロシージャー関数を実行します。出力は、クライアント・アプリケーションに戻されます。

ユーザー定義関数 (UDF)

%DB2PATH%¥samples¥c と %DB2PATH%¥samples¥cpp にあるバッチ・ファイル bldmudf には、 UDF を作成するためのコマンドが入っています。

UDF には、組み込み SQL ステートメントは含められません。このため、UDF プログラムを作成するには、プログラムをプリコンパイルおよびバインドするためにデータベースに接続する必要はありません。

バッチ・ファイルは、ソース・ファイルの名前を指定する、 %1 というパラメーターを取ります。ソース・ファイル名 %1 を DLL 名に使用します。

```
@echo off
rem bldmudf.bat -- Windows 32-bit operating systems
rem Builds a Microsoft Visual C++ user-defined function (UDF).
rem Usage: bldmudf udf_prog_name
if "%1" == "" goto error
rem Compile the program.
if exist "%1.cxx" goto cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c
goto link_step
:cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx
:link_step
rem Link the program.
link -debug:full -debugtype:cv -dll -out:%1.dll %1.obj db2api.lib db2apie.lib -def:%1.def
rem Copy the UDF DLL to the 'function' directory
copy %1.dll "%DB2PATH%¥function"
goto exit
:error
echo Usage: bldmudf prog_name
:exit
@echo on
```

bldmudf のコンパイルおよびリンク・オプション

コンパイル・オプション

cl Microsoft Visual C++ コンパイラー。

-Z7 生成される C7 スタイル CodeView 情報。

-Od 最適化なし。

-c コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。

-W2 警告、エラー、重大、および回復不能エラー・メッセージを出力します。

-D_X86_=1
32 ビット・オペレーティング・システムを Intel ベースのコンピューターで実行するために必要なコンパイラー・オプション。

-DWIN32
32 ビット・オペレーティング・システムに必要なコンパイラー・オプション。

リンク・オプション

link リンク編集にリンカーを使用します。

-debug:full
デバッグ情報を組み込みます。

-debugtype:cv
デバッガー・タイプを指定します。

-dll DLL を作成します。

-out:%1.dll
.DLL ファイルを作成します。

%1.obj オブジェクト・ファイルを組み込みます。

db2api.lib
DB2 ライブラリーとリンクします。

db2apie.lib
DB2 API エンジン・ライブラリーとリンクします。

-def:%1.def
モジュール定義ファイル。

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。

ユーザー定義関数 udfsrv をソース・ファイル udfsrv.c から作成するには、次のように入力します。

```
bldmudf udfsrv
```

このバッチ・ファイルは、サンプル・プログラムと同じディレクトリーに入っている、モジュール定義ファイル udfsrv.def を使用してユーザー定義関数を作成します。 このバッチ・ファイルは、ユーザー定義関数 DLL の udfsrv.dll をサーバー上の %DB2PATH%¥function というパスにコピーします。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。 DB2 CLI が、このプログラムの組み込み SQL C および C++ バージョンとともに提供されます。

DB2 CLI udfcli プログラムは、 %DB2PATH%¥samples¥cli のバッチ・ファイル bldmcli を使用して、 udfcli.c ソース・ファイルから作成できます。 詳細については、372ページの『DB2 CLI アプリケーション』 を参照してください。

組み込み SQL C udfcli プログラムは、 %DB2PATH%¥samples¥c のバッチ・ファイル bldmapp を使用して、 udfcli.sqc ソース・ファイルから作成できます。 詳細については、 378ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

組み込み SQL C++ udfcli プログラムは、 %DB2PATH%¥samples¥cpp のバッチ・ファイル bldmapp を使用して、 udfcli.sqx ソース・ファイルから作成できます。 詳細については、 378ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF を実行するには、次のように入力します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv DLL から ScalarUDF 関数を呼び出します。

IBM VisualAge C++ バージョン 3.5

この節では、以下のトピックについて記載します。

- DB2 CLI アプリケーション
- DB2 CLI ストアード・プロシージャー
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー

- ユーザー定義関数 (UDF)

注: VisualAge C++ コンパイラーは、 %DB2PATH%¥samples¥c ディレクトリーと %DB2PATH%¥samples¥cpp ディレクトリーにある、 C と C++ の両方のサンプル・プログラムに使用されます。 同じバッチ・ファイルがこれらの 2 つのディレクトリー内に置かれています。 これらのディレクトリーにあるバッチ・ファイルには、ファイルの拡張子に応じて、 C または C++ のどちらかのソース・ファイルを受け入れるコマンドが含まれています。

DB2 CLI アプリケーション

%DB2PATH%¥samples¥cli にあるバッチ・ファイル bldvcli.bat には、 IBM VisualAge C++ で DB2 CLI プログラムを作成するためのコマンドが入っています。

このパラメーター %1 には、ソース・ファイルの名前を指定します。

これは、組み込み SQL を含んでいない CLI プログラム用の唯一の必須パラメーターです。組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、 3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは %2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは %3 で、データベースのユーザー ID を指定します。そしてもう 1 つが %4 で、データベースのパスワードを指定します。

プログラムに組み込み SQL (.sqc または .sqx 拡張子が付いている) が含まれている場合、 embprep バッチ・ファイルは、.c または .cxx 拡張子を持つプログラム・ファイルを生成して、プログラムをプリコンパイルするために呼び出されます。

```
@echo off
rem  bldvcli batch file - Windows 32-bit Operating Systems
rem  Builds a CLI program with IBM VisualAge C++.
rem  Usage: bldvcli prog_name
if exist "%1.sqc" call embprep %1 %2 %3 %4
if exist "%1.sqx" call embprep %1 %2 %3 %4
rem Compile the error-checking utility.
icc -c -Ti -W1 /I"%DB2PATH%¥include" utilcli.c
rem  Compile the program.
if exist "%1.sqx" goto cpp
icc -c -Ti -W1 /I"%DB2PATH%¥include" %1.c
goto link_step
:cpp
icc -c -Ti -W1 /I"%DB2PATH%¥include" %1.cxx
rem  Link the program.
:link_step
ilink /MAP /DEBUG /ST:32000 /PM:VIO %1.obj utilcli.obj db2cli.lib
@echo on
```

bldvcli のコンパイルおよびリンク・オプション
コンパイル・オプション **icc** IBM VisualAge C++ コンパイラー。 **-c** コンパイルのみを実行し、リンクは実行しません。 本書では、コンパイルとリンクが別個のステップであることを前提としています。 **-Ti** デバッガー情報を生成します。 **-W1** 警告、エラー、重大、および回復不能エラー・メッセージを出力します。
リンク・オプション **ilink** リンク編集に資源リンカーを使用します。 **/MAP** マップ・ファイルを生成します。 **/DEBUG** デバッグ情報を組み込みます。 **/ST:32000** スタック・サイズとして少なくとも 32 000 を指定します。 **/PM:VIO** プログラムがウィンドウ表示または全画面表示で稼働することを可能にします。 **%1.obj** オブジェクト・ファイルを組み込みます。 **utilcli.obj** エラー検査用のユーティリティー・オブジェクト・ファイルを組み込みます。 **db2cli.lib** DB2 CLI ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル tbinfo.c からサンプル・プログラム tbinfo を作成するには、次のように入力します。

```
bldvcli tbinfo
```

結果として、実行可能ファイル tbinfo が作成されます。この実行可能ファイルを実行するには、次の実行可能名を入力します。

```
tbinfo
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル dbusemx.sqc から組み込み SQL アプリケーション dbusemx を作成する場合、次の 3 つの方法があります。

1. 同じインスタンスにあるサンプル・データベースに接続している場合は、次のように入力します。

   ```
   bldvcli dbusemx
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldvcli dbusemx database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldvcli dbusemx database userid password
   ```

結果として、実行可能ファイル dbusemx が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   dbusemx
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   dbusemx database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   dbusemx database userid password
   ```

DB2 API を使用する DB2 CLI アプリケーション

DB2 には、CLI サンプル・プログラムが含まれています。このサンプル・プログラムは、DB2 API を使用してデータベースを作成およびドロップし、 CLI 機能を複数のデータベースで使用する方法を示します。 DB2 API を使用するサンプルは、 27ページの表7 にある CLI サンプル・プログラムの説明の中に示されています。

sqllib/samples/cli にあるスクリプト・ファイル bldvapi には、 DB2 API を持つ DB2 CLI プログラムを作成するためのコマンドが入っています。 このファイルは、データベースを作成およびドロップするための DB2 API が入った utilapi ユーティリティー・ファイルでコンパイルおよびリンクします。この点が、このファイルと bldvcli バッチ・ファイルの唯一の違いです。

bldvapi と bldvcli の両方に共通のコンパイルおよびリンク・オプションについては、 388ページの『DB2 CLI アプリケーション』 を参照してください。

ソース・ファイル dbmconn.c からサンプル・プログラム dbmconn を作成するには、次のように入力します。

```
bldvapi dbmconn
```

結果として、実行可能ファイル dbmconn が作成されます。この実行可能ファイルを実行するには、次の実行可能名を入力します。

```
dbmconn
```

DB2 CLI ストアード・プロシージャー

%DB2PATH%¥samples¥cli にあるバッチ・ファイル bldvclis.bat には、 CLI ストアード・プロシージャーを作成するためのコマンドが入っています。バッチ・ファイルは、ストアード・プロシージャーをサーバー上の DLL 内に作成します。

このパラメーター %1 には、ソース・ファイルの名前を指定します。バッチ・ファイルでは、ソース・ファイル名 %1 を DLL 名に使用します。

```
@echo off
rem bldvclis.bat file - Windows 32-bit Operating Systems
rem Builds a CLI stored procedure using the IBM VisualAge C++ compiler
rem Usage: bldvclis prog_name
if "%1" == "" goto error
rem Compile the program.
if exist "%1.cxx" goto cpp
icc -c+ -Ti -Ge- -Gm+ -W1 %1.c  utilcli.c
goto link_step
:cpp
icc -c+ -Ti -Ge- -Gm+ -W1 %1.cxx  utilcli.c
:link_step
rem Import the library and create an export file.
rem The function name in the .def file must be decorated to be consistent
rem with the function name in the .map file.  Typically, this is done by
rem prepending "_" and appending "@" and the number of bytes of arguments,
rem as in: "@16". In spserverva.def, for example, the IBM VisualAge C++
rem compiler requires "EXPORTS _outlanguage@16" and not "EXPORTS outlanguage".
ilib /GI %1va.def
rem Link the program and produce a DLL.
ilink /ST:64000 /PM:VIO /MAP /DLL %1.obj utilcli.obj %1va.exp db2cli.lib
rem Copy the stored procedure DLL to the 'function' directory
copy %1.dll "%DB2PATH%¥function"
goto exit
:error
echo Usage: bldvclis prog_name
:exit
@echo on
```

bldvclis のコンパイルおよびリンク・オプション
コンパイル・オプション **icc** IBM VisualAge C++ コンパイラー。 **-c+** コンパイルのみを実行し、リンクは実行しません。 このバッチ・ファイルでは、コンパイルとリンクは別個のステップです。 **-Ti** デバッガー情報を生成します。 **-Ge-** .DLL ファイルを作成します。 静的にリンクされる実行時ライブラリーのバージョンを使用します。 **-Gm+** マルチタスキング・ライブラリーとリンクします。 **-W1** 警告、エラー、重大、および回復不能エラー・メッセージを出力します。
リンク・オプション **ilink** リンク編集に資源リンカーを使用します。 **/ST:64000** スタック・サイズとして少なくとも 64 000 を指定します。 **/PM:VIO** プログラムがウィンドウ表示または全画面表示で稼働することを可能にします。 **/MAP** マップ・ファイルを生成します。 **/DLL** .DLL ファイルを作成します。 **%1.obj** オブジェクト・ファイルを組み込みます。 **%1.exp** VisualAge エクスポート・ファイルを組み込みます。 **db2cli.lib** DB2 CLI ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル spserver.c から spserver ストアード・プロシージャーを作成するには、次のように入力します。

```
bldvclis spserver
```

このバッチ・ファイルは、CLI サンプル・プログラムと同じディレクトリーに入っている、モジュール定義ファイル spserverva.def を使用してストアード・プロシージャーを作成します。 このバッチ・ファイルは、ストアード・プロシージャー DLL の spserver.dll をサーバー上の %DB2PATH%¥function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

ストアード・プロシージャー spserver を作成したなら、そのストアード・プロシージャーを呼び出す CLI クライアント・アプリケーション spclient を構築できます。

spclient は、バッチ・ファイル bldvcli を使用して構築することができます。詳細については、 388ページの『DB2 CLI アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションはストアード・プロシージャー・ライブラリー spserver にアクセスし、様々なストアード・プロシージャー関数をサーバー・データベース上で実行します。出力は、クライアント・アプリケーションに戻されます。

DB2 API と組み込み SQL アプリケーション

%DB2PATH%¥samples¥c と %DB2PATH%¥samples¥cpp にあるバッチ・ファイル bldvapp.bat には、 DB2 アプリケーション・プログラムを作成するためのコマンドが入っています。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を含まないプログラムに必要な唯一のパラメーターです。 組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは %2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは %3 で、データベースのユーザー ID を指定します。そしてもう 1 つが %4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、 bldvapp は、プリコンパイルおよびバインド・ファイル embprep にパラメーターを渡します。データベース名を指定しない場合は、デフォルトの sample データベースを使用します。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
@echo off
rem bldvapp.bat -- Windows 32-bit operating systems
rem Builds a VisualAge C++ application program
rem Usage: bldvapp prog_name [ db_name [  userid password ]]
if exist "%1.sqx" goto embedded
if exist "%1.sqc" goto embedded
goto non_embedded
:embedded
rem Precompile and bind the program.
call embprep %1 %2 %3 %4
rem  Compile the program.
if exist "%1.cxx" goto cpp_emb
icc -c -Ti -W1 %1.c utilemb.c
goto link_embedded
:cpp_emb
icc -c -Ti -W1 %1.cxx utilemb.cxx
rem Link the program.
:link_embedded
ilink /MAP /DEBUG /ST:32000 /PM:VIO %1.obj utilemb.obj db2api.lib
goto exit
:non_embedded
rem  Compile the program.
if exist "%1.cxx" goto cpp_non
icc -c -Ti -W1 %1.c utilapi.c
goto link_non_embedded
:cpp_non
icc -c -Ti -W1 %1.cxx utilapi.cxx
rem Link the program.
```

```
:link_non_embedded
ilink /MAP /DEBUG /ST:32000 /PM:VIO %1.obj utilapi.obj db2api.lib
:exit
@echo on
```

bldvapp のコンパイルおよびリンク・オプション

コンパイル・オプション

icc IBM VisualAge C++ コンパイラー。

-c コンパイルのみを実行し、リンクは実行しません。 本書では、コンパイルとリンクが別個のステップであることを前提としています。

-Ti デバッガー情報を生成します。

-W1 警告、エラー、重大、および回復不能エラー・メッセージを出力します。

リンク・オプション

ilink リンク編集に資源リンカーを使用します。

/MAP マップ・ファイルを生成します。

/DEBUG デバッグ情報を組み込みます。

/ST:32000
スタック・サイズとして少なくとも 32 000 を指定します。

/PM:VIO
プログラムがウィンドウ表示または全画面表示で稼働することを可能にします。

%1.obj オブジェクト・ファイルを組み込みます。

utilemb.obj
組み込み SQL プログラムの場合に、エラー・チェックを行う組み込み SQL ユーティリティー・オブジェクト・ファイルを含みます。

utilapi.obj
組み込み SQL プログラムでない場合に、エラー・チェックを行う DB2 API ユーティリティー・オブジェクト・ファイルを含みます。

db2api.lib
DB2 ライブラリーとリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

%DB2PATH%¥samples¥c のソース・ファイル client.c、または %DB2PATH%¥samples¥cpp のソース・ファイル client.cxx のどちらかから、DB2 API の組み込み SQL を含まないサンプル・プログラム client を作成するには、次のように入力します。

```
bldvapp client
```

結果として、実行可能ファイル client.exe が作成されます。この実行可能ファイルを実行するには、次の実行可能ファイル名を (拡張子なしで) コマンド行に入力します。

```
client
```

組み込み SQL アプリケーションの構築と実行

%DB2PATH%¥samples¥c の C ソース・ファイル updat.sqc、または %DB2PATH%¥samples¥cpp の C++ ソース・ファイル updat.sqx から組み込み SQL アプリケーション updat を構築する方法は 3 つあります。

1. 同じインスタンス上のサンプル・データベースに接続している場合には、次のように入力します。

   ```
   bldvapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldvapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldvapp updat database userid password
   ```

結果として、実行可能ファイル updat.exe が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

updat *database userid password*

組み込み SQL ストアード・プロシージャー

%DB2PATH%¥samples¥c と %DB2PATH%¥samples¥cpp にあるバッチ・ファイル bldvsrv.bat には、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが入っています。 バッチ・ファイルは、ストアード・プロシージャーをコンパイルし、サーバー上の DLL に格納します。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。 第 2 パラメーター %2 には、接続先のデータベースの名前を指定します。ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

バッチ・ファイルでは、ソース・ファイル名 %1 を DLL 名に使用します。

```
@echo off
rem bldvsrv.bat -- Windows 32-bit operating systems
rem Builds a VisualAge C++ stored procedure
rem Usage: bldvsrv prog_name [ db_name ]
rem Precompile and bind the program.
call embprep %1 %2
rem Compile the program.
if exist "%1.cxx" goto cpp
icc -c+ -Ti -Ge- -Gm+ -W1 %1.c
goto link_step
:cpp
icc -c+ -Ti -Ge- -Gm+ -W1 %1.cxx
:link_step
rem Import the library and create a definition file.
rem The function name in the .def file must be decorated to be consistent
rem with the function name in the .map file. Typically, this is done by
rem prepending "_" and appending "@" and the number of bytes of arguments,
rem for example, "@16". In spserverva.def, the IBM VisualAge C++ compiler requires
rem "EXPORTS _outlanguage@16" and not "EXPORTS outlanguage".
ilib /GI %1va.def
rem Link the program and produce a DLL.
ilink /ST:64000 /PM:VIO /MAP /DLL %1.obj %1va.exp db2api.lib
rem Copy the Stored Procedure DLL to the 'function' directory.
copy %1.dll "%DB2PATH%¥function"
@echo on
```

bldvsrv のコンパイルおよびリンク・オプション
コンパイル・オプション **icc** IBM VisualAge C++ コンパイラー。 **-c+** コンパイルのみを実行し、リンクは実行しません。このバッチ・ファイルでは、コンパイルとリンクは別個のステップです。 **-Ti** デバッガー情報を生成します。 **-Ge-** .DLL ファイルを作成します。静的にリンクされる実行時ライブラリーのバージョンを使用します。 **-Gm+** マルチタスキング・ライブラリーとリンクします。 **-W1** 警告、エラー、重大、および回復不能エラー・メッセージを出力します。
リンク・オプション **ilink** リンク編集に資源リンカーを使用します。 **/ST:64000** スタック・サイズとして少なくとも 64 000 を指定します。 **/PM:VIO** プログラムがウィンドウ表示または全画面表示で稼働することを可能にします。 **/MAP** MAP ファイルを生成します。 **/DLL** .DLL ファイルを作成します。 **%1.obj** オブジェクト・ファイルを組み込みます。 **%1va.exp** VisualAge エクスポート・ファイル。 **db2api.lib** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

C ソース・ファイル spserver.sqc、または C++ ソース・ファイル spserver.sqx のどちらかから spserver ストアード・プロシージャーを作成するには、次のように入力します。

```
bldvsrv spserver
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldmsrv spserver database
```

このバッチ・ファイルは、サンプル・プログラムと同じディレクトリーに入っている、モジュール定義ファイル spserverva.def を使用してストアード・プロシージャーを作成します。 このバッチ・ファイルは、ストアード・プロシージャー DLL の spserver.dll をサーバー上の %DB2PATH%¥function というパスにコピーします。

次に、サーバー上で spcreate.db2 スクリプトを実行して、ストアード・プロシージャーをカタログ化します。まず、データベースに接続します。

```
db2 connect to sample
```

ストアード・プロシージャーがすでにカタログ化されている場合は、次のコマンドを使用してそれらをドロップすることができます。

```
db2 -td@ -vf spdrop.db2
```

その後、次のコマンドでストアード・プロシージャーをカタログ化します。

```
db2 -td@ -vf spcreate.db2
```

カタログ化が終了したら、データベースを 1 度停止してから再始動し、新しい共用ライブラリーが認識されるようにします。必要であれば、共用ライブラリーにファイル・モードを設定して、 DB2 インスタンスからアクセスできるようにします。

ストアード・プロシージャー DLL spserver を作成したなら、そのストアード・プロシージャーを呼び出すクライアント・アプリケーション spclient を構築できます。

spclient は、スクリプト・ファイル bldvapp を使用して構築することができます。詳細については、 394ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

spclient *database userid password*

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは、ストアード・プロシージャー DLL spserver にアクセスし、サーバー・データベース上のいくつかのストアード・プロシージャー関数を実行します。出力は、クライアント・アプリケーションに戻されます。

ユーザー定義関数 (UDF)

%DB2PATH%¥samples¥c と %DB2PATH%¥samples¥cpp にあるバッチ・ファイル bldvudf には、 UDF を作成するためのコマンドが入っています。

UDF には、組み込み SQL ステートメントは含められません。したがって、UDF プログラムを作成するには、データベースへの接続、またはプログラムのプリコンパイルおよびバインドは行いません。

このパラメーター %1 には、ソース・ファイルの名前を指定します。 バッチ・ファイルでは、ソース・ファイル名 %1 を DLL 名に使用します。

```
@echo off
rem bldvudf.bat -- Windows 32-bit operating systems
rem Builds a VisualAge C++ user-defined function (UDF)
rem Usage: bldvudf program_name
if "%1" == "" goto error
rem Compile the program.
if exist "%1.cxx" goto cpp
icc -Ti -c+ -Ge- -Gm+ -W1 %1.c
goto link_step
:cpp
icc -Ti -c+ -Ge- -Gm+ -W1 %1.cxx
:link_step
rem Generate an import library and export file using a definition file.
rem Function(s) in the .def file are prepended with an underscore, and
rem appended with the @ sign and number of bytes of arguments (in decimal).
rem Parameters of less than four bytes are rounded up to four bytes.
rem Structure size is rounded up to a multiple of four bytes.
rem For example, function fred prototyped as: "int fred(int, int, short);"
rem would appear as:  "_fred@12" in the .def file.
rem These decorated function names can also be found in %1.map
rem after running the following ilink command without %1va.exp.
ilib /gi %1va.def
rem Link the program to a dynamic link library
ilink /ST:64000 /PM:VIO /MAP /DLL %1.obj %1va.exp db2api.lib db2apie.lib
rem Copy the UDF DLL to the 'function' directory.
copy %1.dll "%DB2PATH%¥function"
goto exit
```

```
:error
echo Usage: bldvudf prog_name
:exit
@echo on
```

bldvudf のコンパイルおよびリンク・オプション
コンパイル・オプション **icc** IBM VisualAge C++ コンパイラー。 **-Ti** デバッガー情報を生成します。 **-c+** コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。 **-Ge-** .DLL ファイルを作成します。静的にリンクされる実行時ライブラリーのバージョンを使用します。 **-Gm+** マルチタスキング・ライブラリーとリンクします。 **-W1** 警告、エラー、重大、および回復不能エラー・メッセージを出力します。
リンク・オプション **ilink** リンク編集に資源リンカーを使用します。 **/ST:64000** スタック・サイズとして少なくとも 64000 を指定します。 **/PM:VIO** プログラムがウィンドウ表示または全画面表示で稼働することを可能にします。 **/MAP** MAP ファイルを生成します。 **/DLL** .DLL ファイルを作成します。 **%1.obj** オブジェクト・ファイルを組み込みます。 **%1va.exp** VisualAge エクスポート・ファイルを組み込みます。 **db2api.lib** DB2 ライブラリーとリンクします。 **db2apie.lib** DB2 API エンジン・ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ユーザー定義関数 `udfsrv` をソース・ファイル `udf.c` から作成するには、次のように入力します。

```
bldvudf udfsrv
```

このバッチ・ファイルは、サンプル・プログラムと同じディレクトリーに入っている、モジュール定義ファイル udfsrv.def を使用してユーザー定義関数を作成します。 このバッチ・ファイルは、ユーザー定義関数 DLL の udfsrv.dll をサーバー上の %DB2PATH%¥function というパスにコピーします。

udfsrv を作成したなら、それを呼び出すクライアント・アプリケーション udfcli を構築できます。 DB2 CLI が、このプログラムの組み込み SQL C および C++ バージョンとともに提供されます。

DB2 CLI udfcli プログラムは、 %DB2PATH%¥samples¥cli のバッチ・ファイル bldvcli を使用して、 udfcli.c ソース・ファイルから作成できます。 詳細については、388ページの『DB2 CLI アプリケーション』 を参照してください。

組み込み SQL C udfcli プログラムは、 %DB2PATH%¥samples¥c のバッチ・ファイル bldvapp を使用して、 udfcli.sqc ソース・ファイルから作成できます。 詳細については、 394ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

組み込み SQL C++ udfcli プログラムは、 %DB2PATH%¥samples¥cpp のバッチ・ファイル bldvapp を使用して、 udfcli.sqx ソース・ファイルから作成できます。 詳細については、 394ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

UDF を実行するには、次のように入力します。

```
udfcli
```

この呼び出しアプリケーションは、 udfsrv DLL から ScalarUDF 関数を呼び出します。

IBM VisualAge C++ バージョン 4.0

VisualAge C++ バージョン 4 コンパイラーのアプリケーション構築情報は、AIX、OS/2 および Windows 32 ビット・オペレーティング・システムで共通です。この情報については、152ページの『VisualAge C++ バージョン 4.0』を参照してください。

IBM VisualAge COBOL

この節では、以下のトピックについて記載します。

- コンパイラーの使用
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー

コンパイラーの使用

組み込み SQL および DB2 API 呼び出しを含むアプリケーションを開発しており、 IBM VisualAge COBOL コンパイラーを使用している場合には、以下の点に留意してください。

- コマンド行プロセッサーのコマンド db2 prep を使ってアプリケーションをプリコンパイルする場合は、 target ibmcob オプション (デフォルト) を使用してください。
- ソース・ファイルの中でタブ文字を使用しないでください。
- ソース・ファイルで PROCESS および CBL キーワードを使用して、コンパイル・オプションを設定できます。キーワードは 8〜72 桁目だけに置いてください。
- アプリケーションに組み込み SQL のみが含まれていて、 DB2 API 呼び出しは含まれない場合には、 pgmname(mixed) コンパイル・オプションを使う必要はありません。 DB2 API 呼び出しを使用する場合には、 pgmname(mixed) コンパイル・オプションを使う必要があります。
- IBM VisualAge COBOL コンパイラーの「システム/390 ホスト・データ型サポート」機能を使用している場合、アプリケーション用の DB2 組み込みファイルは、次のディレクトリーの中にあります。

  ```
  %DB2PATH%¥include¥cobol_i
  ```

  提供されたバッチ・ファイルを使って DB2 サンプル・プログラムを作成している場合、バッチ・ファイルで指定された組み込みファイルのパスは、 cobol_a ディレクトリーではなく、 cobol_i ディレクトリーを指すように変更しなければなりません。

  IBM VisualAge COBOL コンパイラーの「システム/390 ホスト・データ型サポート」機能を使用していない場合、またはこのコンパイラーのそれよりも前のバージョンを使用している場合、アプリケーション用の DB2 組み込みファイルは、次のディレクトリー中にあります。

  ```
  %DB2PATH%¥include¥cobol_a
  ```

COPY ファイル名を .cbl 拡張子を含めて指定するには、次のように入力します。

```
COPY "sql.cbl".
```

DB2 API と組み込み SQL アプリケーション

%DB2PATH%¥samples¥cobol にあるバッチ・ファイル bldapp.bat には、 DB2 アプリケーション・プログラムを作成するためのコマンドが入っています。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を含まないプログラムに必要な唯一のパラメーターです。 組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは %2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは %3 で、データベースのユーザー ID を指定します。そしてもう 1 つが %4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、bldapp は、プリコンパイルおよびバインドのファイル embprep にパラメーターを渡します。 データベース名を指定しない場合は、デフォルトの sample データベースを使用します。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
@echo off
rem bldapp.bat  -- Windows 32-bit operating systems
rem Builds a VisualAge COBOL application program
rem Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
rem If an embedded SQL program, precompile and bind it.
if not exist "%1.sqb" goto compile_step
call embprep %1 %2 %3 %4
:compile_step
rem Compile the error-checking utility.
cob2 -qpgmname(mixed) -c -qlib -I"%DB2PATH%¥include¥cobol_a" checkerr.cbl
rem Compile the program.
cob2 -qpgmname(mixed) -c -qlib -I"%DB2PATH%¥include¥cobol_a" %1.cbl
rem Link the program.
cob2 %1.obj checkerr.obj db2api.lib
@echo on
```

bldapp のコンパイルおよびリンク・オプション
コンパイル・オプション **cob2** IBM VisualAge COBOL コンパイラー。 **-qpgmname(mixed)** コンパイラーに、大文字小文字混合の名前を持つライブラリー入り口点の CALL を許可するように指示します。 **-c** コンパイルのみを実行し、リンクは実行しません。本書では、コンパイルとリンクが別個のステップであることを前提としています。 **-qlib** コンパイラーに COPY ステートメントを処理するように指示します。 **-I***path* DB2 インクルード・ファイルのロケーションを指定します。たとえば、-I"%DB2PATH%¥include¥cobol_a"。 **checkerr.cbl** エラー検査ユーティリティーをコンパイルします。
リンク・オプション **cob2** リンク編集をするコンパイラーを使用します。 **checkerr.obj** エラー検査ユーティリティー・オブジェクト・ファイルを組み込みます。 **db2api.lib** DB2 ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.cbl から組み込み SQL を含まないサンプル・プログラム client を作成するには、次のように入力します。

```
bldapp client
```

結果として、実行可能ファイル client.exe が作成されます。この実行可能ファイルを sample データベースに対して実行するには、次の実行可能名を (拡張子なしで) 入力します。

```
client
```

組み込み SQL アプリケーションの構築と実行

ソース・ファイル updat.sqb から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンス上のサンプル・データベースに接続している場合には、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

```
bldapp updat database
```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

```
bldapp updat database userid password
```

結果として、実行可能ファイル updat が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前を入力します。

```
updat
```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

```
updat database
```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

```
updat database userid password
```

組み込み SQL ストアード・プロシージャー

%DB2PATH%¥samples¥cobol にあるバッチ・ファイル bldsrv.bat には、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが入っています。 バッチ・ファイルは、ストアード・プロシージャーをサーバー上の DLL にコンパイルします。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。第 2 パラメーター %2 には、接続先のデータベースの名前を指定します。 ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

バッチ・ファイルでは、ソース・ファイル名 %1 を DLL 名に使用します。

```
@echo off
rem bldsrv.bat  -- Windows 32-bit operating systems
rem Builds a VisualAge COBOL stored procedure
```

```
rem Usage: bldsrv <prog_name> [ <db_name> ]
rem Precompile and bind the program.
call embprep %1 %2
rem  Compile the stored procedure.
cob2 -qpgmname(mixed) -c -qlib -I"%DB2PATH%¥include¥cobol_a" %1.cbl
rem  Link the stored procedure and create a shared library.
ilib /nol /gi:%1 %1.obj
ilink /free /nol /dll db2api.lib %1.exp %1.obj iwzrwin3.obj
rem Copy stored procedure to the %DB2PATH%¥function directory.
copy %1.dll "%DB2PATH%¥function"
@echo on
```

bldsrv のコンパイルおよびリンク・オプション

コンパイル・オプション

cob2 IBM VisualAge COBOL コンパイラー。

-qpgmname(mixed)
コンパイラーに、大文字小文字混合の名前を持つライブラリー入り口点の CALL を許可するように指示します。

-c コンパイルのみを実行し、リンクは実行しません。このバッチ・ファイルでは、コンパイルとリンクは別個のステップです。

-qlib コンパイラーに COPY ステートメントを処理するように指示します。

-I*path* DB2 インクルード・ファイルのロケーションを指定します。たとえば、-I"%DB2PATH%¥include¥cobol_a"。

リンク・オプション

ilink IBM VisualAge COBOL リンカーを使用します。

/free 自由書式。

/nol ロゴなし。

/dll DLL をソース・プログラム名を使用して作成します。

db2api.lib
DB2 ライブラリーとリンクします。

%1.exp エクスポート・ファイルを組み込みます。

%1.obj プログラム・オブジェクト・ファイルを組み込みます。

iwzrwin3.obj
IBM VisualAge COBOL が提供するオブジェクト・ファイルを組み込みます。

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。

サンプル・データベースに接続している場合、ソース・ファイル outsrv.sqb からサンプル・プログラム outsrv を作成するには、次のように入力します。

```
bldsrv outsrv
```

他のデータベースに接続しているときは、さらにデータベース名も含めます。

```
bldsrv outsrv database
```

このスクリプト・ファイルは、パス sqllib/function 内のサーバーにストアード・プロシージャーをコピーします。

必要であれば、ストアード・プロシージャーにファイル・モードを設定して DB2 インスタンスがそれを実行できるようにしてください。

ストアード・プロシージャー outsrv を構築してしまえば、そのストアード・プロシージャーを呼び出すクライアント・アプリケーション outcli を構築できます。 outcli は、bldapp バッチ・ファイルを使用して構築することができます。詳細については、 404ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

```
outcli database userid password
```

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample またはそのリモート別名、あるいはその他の名前にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションはストアード・プロシージャー・ライブラリー outsrv にアクセスし、ストアード・プロシージャー関数をサーバー・データベース上で実行します。この出力は、クライアント・アプリケーションに戻されます。

Micro Focus COBOL

この節では以下のトピックを取り上げています。

- コンパイラーの使用
- DB2 API と組み込み SQL アプリケーション
- 組み込み SQL ストアード・プロシージャー

コンパイラーの使用法

組み込み SQL および DB2 API 呼び出しを含むアプリケーションを開発しており、 Micro Focus コンパイラーを使用している場合には、以下の点に留意してください。

- コマンド行プロセッサーのコマンド db2 prep を使ってアプリケーションをプリコンパイルする場合は、 target mfcob オプション (デフォルト) を使用してください。
- 以下のように入力して、必ず LIB 環境変数が %DB2PATH%¥lib を指すようにしてください。

  ```
  set LIB="%DB2PATH%¥lib;%LIB%"
  ```

- Micro Focus COBOL 用の DB2 COPY ファイルは、%DB2PATH%¥include¥cobol_mf にあります。 COBCPY 環境変数を、以下のようにディレクトリーを含めて設定してください。

  ```
  set COBCPY="%DB2PATH%¥include¥cobol_mf;%COBCPY%"
  ```

すべての DB2 アプリケーション・プログラミング・インターフェースへの呼び出しは、呼び出し規則 74 を使用して実行しなければなりません。 DB2 COBOL プリコンパイラーは、自動的に CALL-CONVENTION 節を SPECIAL-NAMES 段落に挿入します。 SPECIAL-NAMES 段落が存在しない場合、 DB2 COBOL プリコンパイラーはそれを以下のように作成します。

```
Identification Division
Program-ID. "static".
special-names.
    call-convention 74 is DB2API.
```

さらにプリコンパイラーは、呼び出し規則を識別するために使用する記号 DB2API を、 DB2 API が呼び出されるたびに必ず call キーワードの後に自動的に置きます。 これはたとえば、プリコンパイラーが DB2 API 実行時呼び出しを組み込み SQL ステートメントから生成する場合にも必ず実行されます。

DB2 API への呼び出しをプリコンパイルされていないアプリケーションで実行する場合、前述したものと同様の SPECIAL-NAMES 段落を、手操作で入力してアプリケーションに作成する必要があります。 DB2 API を直接呼び出す場合は、 call キーワードの後に DB2API 記号を手動で追加する必要があります。

DB2 API と組み込み SQL アプリケーション

%DB2PATH%¥samples¥cobol にあるバッチ・ファイル bldapp には、 DB2 アプリケーション・プログラムを作成するためのコマンドが入っています。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を含まないプログラムに必要な唯一のパラメーターです。 組み込み SQL プログラムを作成するためにはデータベースへの接続が必要なため、3 つのパラメーターがオプションとして用意されています。 2 番目のパラメーターは %2 で、接続するデータベースの名前を指定します。 3 番目のパラメーターは %3 で、データベースのユーザー ID を指定します。そしてもう 1 つが %4 で、データベースのパスワードを指定します。

組み込み SQL プログラムの場合、 bldapp は、プリコンパイルおよびバインド・バッチ・ファイル embprep にパラメーターをパスします。 データベース名を指定しない場合は、デフォルトの sample データベースを使用します。なお、ユーザー ID とパスワードのパラメーターは、プログラムを構築するインスタンスとデータベースのあるインスタンスが異なる場合にのみ必要になります。

```
@echo off
rem bldapp.bat  -- Windows 32-bit operating systems
rem Builds a Micro Focus Cobol application program
rem Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
rem If an embedded SQL program, precompile and bind it.
if not exist "%1.sqb" goto compile_step
call embprep %1 %2 %3 %4
:compile_step
rem Compile the error-checking utility.
cobol checkerr.cbl;
rem  Compile the program.
cobol %1.cbl;
rem  Link the program.
cbllink -l %1.obj checkerr.obj db2api.lib
@echo on
```

bldapp のコンパイルおよびリンク・オプション
コンパイル・オプション **cobol**　Micro Focus COBOL コンパイラー。

bldapp のコンパイルおよびリンク・オプション
リンク・オプション **cbllink** リンク編集にリンカーを使用します。 **-l** lcobol ライブラリーとリンクします。 **checkerr.obj** エラー検査ユーティリティー・オブジェクト・ファイルとリンクします。 **db2api.lib** DB2 API ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

ソース・ファイル client.cbl から組み込み SQL を含まないサンプル・プログラム client を作成するには、次のように入力します。

```
bldapp client
```

結果として、実行可能ファイル client.exe が作成されます。この実行可能ファイルを sample データベースに対して実行するには、次の実行可能名を (拡張子なしで) 入力します。

```
client
```

組み込み SQL アプリケーションの構築および実行

ソース・ファイル updat.sqb から組み込み SQL アプリケーション updat を構築する方法には、次の 3 つがあります。

1. 同じインスタンス上のサンプル・データベースに接続している場合には、次のように入力します。

   ```
   bldapp updat
   ```

2. 同じインスタンスにある他のデータベースに接続している場合は、さらにデータベース名も入力します。

   ```
   bldapp updat database
   ```

3. 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

   ```
   bldapp updat database userid password
   ```

結果として、実行可能ファイル updat.exe が作成されます。

この組み込み SQL アプリケーションを実行する方法には次の 3 つがあります。

1. 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前 (拡張子なし) を入力します。

   ```
   updat
   ```

2. 同じインスタンスにある他のデータベースにアクセスする場合は、実行可能ファイル名とデータベース名を入力します。

   ```
   updat database
   ```

3. 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

   ```
   updat database userid password
   ```

組み込み SQL ストアード・プロシージャー

%DB2PATH%¥samples¥cobol_mf にあるバッチ・ファイル bldsrv には、組み込み SQL ストアード・プロシージャーを作成するためのコマンドが入っています。バッチ・ファイルは、ストアード・プロシージャーをサーバー上の DLL にコンパイルします。

第 1 パラメーター %1 には、ソース・ファイルの名前を指定します。第 2 パラメーター %2 には、接続先のデータベースの名前を指定します。 ストアード・プロシージャーは、必ずデータベースが常駐するインスタンスに構築される必要があるため、ユーザー ID やパスワードを指定するパラメーターはありません。

第 1 パラメーター (ソース・ファイル名) だけが必須です。データベース名は任意で指定します。データベース名を指定しない場合、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。

バッチ・ファイルでは、ソース・ファイル名 %1 を DLL 名に使用します。

```
@echo off
rem bldsrv.bat  -- Windows 32-bit operating systems
rem Builds a Micro Focus Cobol stored procedure
rem Usage: bldsrv <prog_name> [ <db_name> ]
rem Precompile and bind the program.
call embprep %1 %2
rem  Compile the stored procedure.
cobol %1.cbl /case;
rem  Link the stored procedure and create a shared library.
cbllink /d %1.obj db2api.lib
rem Copy the stored procedure to the %DB2PATH%¥function directory.
copy %1.dll "%DB2PATH%¥function"
@echo on
```

bldsrv のコンパイルおよびリンク・オプション
コンパイル・オプション **cobol** Micro Focus COBOL コンパイラー。 **/case** 外部記号が大文字に変換されないようにします。
リンク・オプション **cbllink** リンク編集のために Micro Focus COBOL リンカーを使用します。 **/d** .DLL ファイルを作成します。 **db2api.lib** DB2 API ライブラリーとリンクします。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

サンプル・データベースに接続している場合、ソース・ファイル outsrv.sqb からサンプル・プログラム outsrv を作成するには、次のように入力します。

```
bldsrv outsrv
```

他のデータベースに接続しているときは、さらにデータベース名も入力します。

```
bldsrv outsrv database
```

スクリプト・ファイルは、 DLL をサーバー上の sqllib/function というパスにコピーします。

必要であれば、 DLL にファイル・モードを設定してクライアント・プログラムがそれにアクセスできるようにしてください。

DLL outsrv を作成したなら、その DLL を呼び出すクライアント・アプリケーション outcli を構築できます。 outcli は、bldapp バッチ・ファイルを使用して構築することができます。詳細については、 409ページの『DB2 API と組み込み SQL アプリケーション』を参照してください。

ストアード・プロシージャーを呼び出すためには、次のように入力してサンプル・クライアント・アプリケーションを実行します。

```
outcli database userid password
```

ここで、それぞれは次のものを表します。

database
接続先のデータベースの名前です。名前は、sample かその別名、またはその他のデータベース名にすることができます。

userid 有効なユーザー ID です。

password
有効なパスワードです。

クライアント・アプリケーションは、 DLL outsrv にアクセスし、サーバー・データベース上の同じ名前のストアード・プロシージャー関数を実行してから、クライアント・アプリケーションに出力を戻します。

オブジェクト REXX

オブジェクト REXX は、オブジェクト指向バージョンの REXX 言語です。オブジェクト指向拡張子が従来の REXX に追加されていますが、既存の関数および命令には変更はありません。 オブジェクト REXX 解釈プログラムは、以下のサポートが加えられ、前のバージョンの拡張バージョンとなっています。

- クラス、オブジェクト、およびメソッド
- メッセージ交換およびポリモアフィズム
- 単一および複数継承

オブジェクト REXX は、従来の REXX と完全な互換性があります。この節で REXX と述べる場合は、オブジェクト REXX を含むすべての REXX のバージョンのことを言います。

REXX プログラムはプリコンパイルまたはバインドしません。

Windows NT 上では、REXX プログラムはコメントで開始する必要はありません。しかし、移行上の理由から、第 1 行の第 1 列から始まるコメントで各 REXX プログラムを開始することをお勧めします。これによってプログラムを、他のプラットフォームのバッチ・コマンドと区別することができます。

```
/* Any comment will do. */
```

REXX サンプル・プログラムは、ディレクトリー %DB2PATH%¥samples¥rexx にあります。サンプル REXX プログラム updat を実行するには、以下のようにしてください。

1. サーバー上でデータベース・マネージャーをまだ開始していなければ、次のように入力して開始します。

   ```
   db2start
   ```

2. 次のように入力します。

```
rexx updat.cmd
```

REXX および DB2 について詳しくは、 *アプリケーション開発の手引き* の『REXX でのプログラミング』の章を参照してください。

付録A. データベース・マネージャー・インスタンスについて

DB2 は、同じマシン上の複数のデータベース・マネージャー・インスタンスをサポートします。 1 つのデータベース・マネージャー・インスタンスには、それ独自の構成ファイル、ディレクトリー、およびデータベースがあります。

各データベース・マネージャー・インスタンスは、複数のデータベースを管理することができます。しかし、1 つのデータベースが属することができるのは 1 つのインスタンスだけです。 図1 はこの関係を示しています。

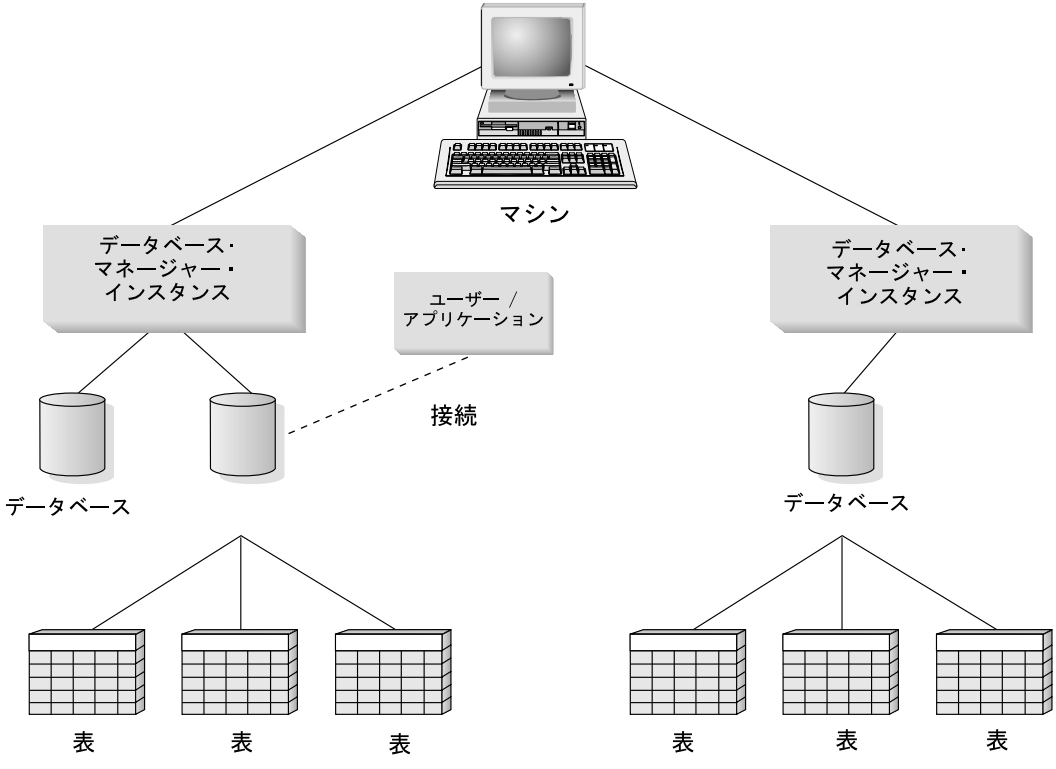


図1. データベース・マネージャー・インスタンス

データベース・マネージャー・インスタンスにより、同じマシン上で複数のデータベース環境を持つという柔軟性が実現されます。たとえば、1 つのデータベース・マネージャー・インスタンスを開発用に、別のインスタンスを実動用にすることができます。

UNIX サーバーを使用すると、異なるデータベース・マネージャー・インスタンス上で別々の DB2 のバージョンを使用することができます。 たとえば、1 つのデータベース・マネージャー・インスタンスで DB2 ユニバーサル・データベース バージョン 6.1 を実行し、別のインスタンスで DB2 ユニバーサル・データベース バージョン 7.1 を実行することができます。 ただし、同一バージョン・レベルでは、 1 つのリリースとモディフィケーション・レベルしかサポートされていません。たとえば、DB2 バージョン 5.0 および DB2 バージョン 5.2 は、 UNIX サーバー上に共存することはできません。

OS/2、Windows NT、および Windows 2000 サーバーでは、各データベース・マネージャー・インスタンスの DB2 は、同じバージョン、リリース、およびモディフィケーション・レベルのものでなければなりません。 1 つのデータベース・マネージャー・インスタンスで DB2 ユニバーサル・データベース バージョン 6.1 を実行し、別のインスタンスで DB2 ユニバーサル・データベース バージョン 7.1 を実行することはできません。

使用するそれぞれのインスタンスについて、以下のことを知っておく必要があります。

インスタンス名

UNIX プラットフォームの場合は、データベース・マネージャー・インスタンスを作成するときに指定する、有効なユーザー名です。

OS/2、Windows NT、および Windows 2000 の場合は、最大 8 文字の英数字ストリングです。 DB2 インスタンスは、インストール時に作成されます。

インスタンス・ディレクトリー

インスタンスがあるホーム・ディレクトリー。

UNIX プラットフォームでは、インスタンス・ディレクトリーは `$HOME/sqllib` です。 $HOME はインスタンス所有者のホーム・ディレクトリーです。

OS/2、Windows NT、および Windows 2000 の場合、インスタンス・ディレクトリーは `%DB2PATH%¥`*instance_name* です。変数 `%DB2PATH%` で、 DB2 のインストール先を判別できます。 DB2 のインストール先のドライブに応じて、 `%DB2PATH%` は *drive*`:¥sqllib` を指します。

OS/2、Windows NT および Windows 2000 のインスタンス・パスは、以下のいずれかに基づいて作成されます。

`%DB2PATH%¥%DB2INSTANCE%` (たとえば、 `C:¥SQLLIB¥DB2`)

または、DB2INSTPROF が定義されている場合、

%DB2INSTPROF%¥%DB2INSTANCE% (たとえば、 C:¥PROFILES¥DB2)

DB2INSTPROF 環境変数は、クライアント・マシンが読み取りアクセスしか持っていないネットワーク・ドライブ上で、 DB2 の実行をサポートするために OS/2、Windows NT および Windows 2000 上で使用されます。 この場合、 DB2 は *drive*:¥sqllib に設定され、また DB2INSTPROF はローカル・パス (C:¥PROFILES など) に設定されます。これには、カタログや構成などのインスタンス固有情報すべてが含まれます。 DB2 はこれらのファイルへの更新アクセスを必要とするためです。

データベース・マネージャー・インスタンスの作成および保守については、使用しているプラットフォーム用の *概説およびインストール* を参照してください。

付録B. アプリケーションの移行

DB2 バージョン 2 以降、 DB2 クライアント・アプリケーション・イネーブラー、または DB2 ソフトウェア開発者キットから DB2 ユニバーサル・データベース バージョン 7.1 にアップグレードする場合、データベースとノード・ディレクトリーは自動的に移行されます。 DB2 バージョン 1 から移行する場合は、まず DB2 ユニバーサル・データベース バージョン 5 に移行する必要があります。その後、バージョン 5 からバージョン 7.1 に移行できます。既存のデータベースを移行するには、 *管理の手引き* で説明されているツールを使用します。

注:

1. **HP-UX**。 HP-UX バージョン 10 または HP-UX バージョン 11 以前から DB2 を移行している場合、 DB2 プログラムは HP-UX バージョン 11 (組み込み SQL がある場合) 上の DB2 で再びプリコンパイルして、再コンパイルしなければなりません。これには、すべての DB2 アプリケーション、ストアード・プロシージャー、ユーザー定義関数、およびユーザー出口プログラムが含まれます。さらに、HP-UX バージョン 11 上でコンパイルされた DB2 プログラムは、 HP-UX バージョン 10 以前の上では実行できない可能性があります。 HP-UX バージョン 10 上でコンパイルされ実行される DB2 プログラムは、 HP-UX バージョン 11 サーバーにリモートで接続することができます。
2. **Linux**。 DB2 は、 DB2 ユニバーサル・データベース (Linux 版) バージョン 5.2 (ベータ) からの移行はサポートしていません。
3. **Micro Focus COBOL**。 DB2 バージョン 2.1.1 またはそれ以前を使ってプリコンパイルし、 Micro Focus COBOL を使ってコンパイルした既存アプリケーションは、現行バージョンの DB2 を使って再プリコンパイルした後、 Micro Focus COBOL を使って再コンパイルする必要があります。 IBM プリコンパイラーの旧バージョンを使って構築したアプリケーションを再プリコンパイルしないと、異常終了時にデータベースが破壊される恐れがあります。

注: 以下、および『質問』と『条件』の節に記載する内容は、 UNIX プラットフォームだけに適用されます。

DB2 バージョン 1、DB2 バージョン 2、DB2 バージョン 5、または DB2 バージョン 6.1 からのアプリケーションがあり、データベース・インスタンスの

以前のバージョンだけでなく、同じマシンの DB2 バージョン 7.1 インスタンスの上でもそれらを実行したい場合、環境にいくらかの変更を行う必要があることがあります。どのような変更を行うかを判別するには、以下の質問に答え、『条件』の節を検討して、現状に当てはまる条件があるかどうかを調べてください。

挙げられた点を説明するために、AIX システムが使われています。同じ概念が他の UNIX プラットフォームにも当てはまりますが、詳細な点 (環境変数と特定のコマンドなど) は異なる可能性があります。使用しているオペレーティング・システムのこれらの詳細な点をあまり知らない場合は、 *管理の手引き* または *DB2 ユニバーサル・データベース (UNIX 版) 概説およびインストール* の『インストールの計画』の章の『以前のバージョンからの移行』の節を参照してください。

質問

質問 1: どのように以前のバージョンのアプリケーションを、 DB2 クライアントの実行時ライブラリー (AIX 上の libdb2.a など) にリンクしましたか ?

実行可能用に組み込み共用ライブラリーの探索パスを判別するには、次のシステム・コマンドの 1 つを使用します。

AIX /usr/bin/dump -H *executable_filename*

HP-UX
/usr/bin/chatr *executable_filename*

Linux /usr/bin/objdump -p *executable_filename*

DYNIX/ptx
/usr/bin/dump -Lv *executable_filename*

Silicon Graphics IRIX
/bin/elfdump -Lv *executable_filename*

Solaris
/usr/bin/dump -Lv *executable_filename*

executable_filename はアプリケーションの実行可能ファイルの名前です。

以下に、 AIX アプリケーション用の DB2 バージョン 1 のサンプル・ダンプ・リストを記載します。

```
dbcat:
  ***Loader Section***
```

```
                    Loader Header Information
VERSION#         #SYMtableENT     #RELOCent        LENidSTR
0x00000001       0x00000012       0x00000029       0x00000064
#IMPfilID        OFFidSTR         LENstrTBL        OFFstrTBL
0x00000004       0x000003bc       0x00000077       0x00000420
  ***Import File Strings***
INDEX  PATH                          BASE                MEMBER
0      /usr/lpp/db2_01_01_0000/lib:/usr/lpp/xlC/lib:/usr/lib:/lib
1                                    libc.a              shr.o
2                                    libC.a              shr.o
3                                    libdb2.a            shr.o
```

ライン 0 (ゼロ) は、リンクされている共用ライブラリーを検出するために、実行可能が検索するディレクトリー・パスを示します。ライン 1、2、3 は、アプリケーションがリンクされている共用ライブラリーを示しています。

アプリケーションが構築された方法によって、以下のパスを検索できます。/usr/lpp/db2_01_01_0000/lib、 INSTHOME/sqllib/lib (INSTHOME はデータベース・インスタンス所有者のホーム・ディレクトリー)、または /usr/lib:/lib の組み合わせ。

質問 2: どのような方法でシステムの DB2 実行時ライブラリーを構成しましたか ?

DB2 バージョン 1、2、5、6.1、または 7.1 がインストールされている場合、DB2 クライアント実行時ライブラリーを含むシステム・デフォルト共用ライブラリー・パス /usr/lib から、 DB2 クライアント実行時ライブラリーを含む DB2 インストール・パスへの記号リンクを作成するステップ (任意選択) があります。

DB2 の各バージョンのインストール・パスは、以下のとおりです。

バージョン 1

/usr/lpp/db2_01_01_0000/lib

バージョン 2

/usr/lpp/db2_02_01/lib

バージョン 5

/usr/lpp/db2_05_00/lib

バージョン 6.1

/usr/lpp/db2_06_01/lib

バージョン 7.1

/usr/lpp/db2_07_01/lib

すべての場合に、実行時共用ライブラリーの名前は、libdb2.a です。

一度に、これらのライブラリーの 1 つのバージョンだけが、デフォルトになります。 DB2 がこのデフォルトを提供するので、アプリケーションを構築するときに DB2 の特定のバージョンに依存することはありません。

質問 3: 環境の中で異なる検索パスを指定しますか ?

LIBPATH 環境変数 (AIX 上)、SHLIB_PATH (HP-UX 上)、LD_LIBRARY_PATH (Linux、DYNIX/ptx、Silicon Graphics IRIX、および Solaris 上) を使用して、アプリケーション中でコーディングされている共用ライブラリーの検索パスを指定変更できます。

注: Silicon Graphics IRIX 上の n32 オブジェクト・タイプ・アプリケーションの場合には、 LD_LIBRARYN32_PATH 環境変数を使用してください。

ライブラリー検索パスは、質問 1 の答えの中で与えられたプラットフォーム用の適切なシステム・コマンドを使って、調べることができます。

条件

前述の質問に答えたならば、環境を変更する必要があります。下記に挙げた条件をお読みください。いずれかの条件が現状に当てはまるならば、変更が必要です。

条件 1: バージョン 6.1 アプリケーションが、 AIX のデフォルト共用ライブラリー・パス /usr/lib/libdb2.a 以外の共用ライブラリーをロードし、以下の条件が当てはまる場合。

- /usr/lib/libdb2.a から /usr/lpp/db2_06_01/lib/libdb2.a への記号リンクがあり、データベース・サーバーが DB2 ユニバーサル・データベース (AIX 版) バージョン 7.1 の場合、以下のどれかを行います。
  - 以下のものを示す記号リンクを変更します。

    ```
    /usr/lpp/db2_07_01/lib/libdb2.a
    ```

    ライブラリー間のリンクを設定する方法については、 *DB2 ユニバーサル・データベース (UNIX 版) 概説およびインストール* に記載されています。 root で、次のように ″db2ln″ コマンドを使用してリンクを変更できます。

    ```
    /usr/lpp/db2_07_01/cfg/db2ln
    ```

  - LIBPATH 環境変数が /usr/lpp/db2_07_01/lib または INSTHOME/sqllib/lib を指すように設定します。 INSTHOME はバージョン 7.1 インスタンス所有者のホーム・ディレクトリーです。

– アプリケーション (クライアント) からサーバー・インスタンスへの TCP/IP 接続を構成します。 TCP/IP の構成については、*インストールおよび構成 補足* を参照してください。

- /usr/lib/libdb2.a から /usr/lpp/db2_07_01/lib/libdb2.a への記号リンクがあり、データベース・サーバーが DB2 バージョン 6.1 の場合、アプリケーション (クライアント) インスタンスからサーバー・インスタンスへの TCP/IP 接続を構成します。 TCP/IP の構成については、*インストールおよび構成 補足* を参照してください。

条件 2: バージョン 6.1 アプリケーションが、 DB2 バージョン 6.1 インスタンス所有者 ($HOME/sqllib/lib/libdb2.a) の $HOME パス以外の共用ライブラリーをロードし、データベース・サーバーが DB2 ユニバーサル・データベース (AIX 版) バージョン 7.1 の場合、以下のどれかを行います。

- データベース・サーバー・インスタンスとして、アプリケーション・インスタンスを同じバージョンへ移行します。
- LIBPATH 環境変数が /usr/lpp/db2_07_01/lib または INSTHOME/sqllib/lib を示すように設定します。 INSTHOME はバージョン 7.1 インスタンス所有者のホーム・ディレクトリーです。
- アプリケーション (クライアント) からサーバー・インスタンスへの TCP/IP 接続を構成します。 TCP/IP の構成については、*インストールおよび構成 補足* を参照してください。

条件 3: バージョン 6.1 アプリケーションが、 DB2 バージョン 6.1 インストール・パス (/usr/lpp/db2_06_01/lib/libdb2.a) 以外の共用ライブラリーをロードし、データベース・サーバーが DB2 ユニバーサル・データベース (AIX 版) バージョン 7.1 の場合、以下のどちらかを行います。

- LIBPATH 環境変数が /usr/lpp/db2_07_01/lib または INSTHOME/sqllib/lib を示すように設定します。 INSTHOME はデータベース・インスタンス所有者のホーム・ディレクトリーです。
- アプリケーション (クライアント) からサーバー・インスタンスへの TCP/IP 接続を構成します。 TCP/IP の構成については、*インストールおよび構成 補足* を参照してください。

条件 4: バージョン 6.1 アプリケーションが、 DB2 ユニバーサル・データベース (AIX 版) バージョン 7.1 のインストール・パス (/usr/lpp/db2_07_01/lib/libdb2.a) 以外の共用ライブラリーをロードし、データベース・サーバーが DB2 バージョン 6.1 の場合、アプリケーション (ク

ライアント) インスタンスからサーバー・インスタンスへの TCP/IP 接続を構成します。 TCP/IP の構成については、*インストールおよび構成 補足* を参照してください。

移行についての他の考慮事項

アプリケーションを開発するときには、以下の点を考慮してください。これらの情報は、アプリケーションを一層移行性のあるものにするのに役立つでしょう。

- UNIX の場合、アプリケーションでは、デフォルト・パス /usr/lib:/lib だけを使用するようにしてください。 OS/2 および Windows 32 ビット・オペレーティング・システムの場合は、以下のものを使用して、 LIB 環境変数が %DB2PATH%¥lib を指しているかどうか確かめてください。

  ```
  set LIB=%DB2PATH%¥lib;%LIB%
  ```

  また、使用している DB2 のデフォルトのパスとバージョンとの間に記号リンクを作成します。そのリンク先が、アプリケーションが必要とする DB2 の最低レベルであることを確認してください。リンクの設定については、ご使用のプラットフォーム用の*概説およびインストール* を参照してください。
- アプリケーションが特定のバージョンの DB2 を必要とする場合、アプリケーション中で DB2 バージョンを指定するパスをコーディングします。たとえば、AIX アプリケーションが DB2 バージョン 5 を必要とするなら、/usr/lpp/db2_05_00/lib をコーディングしてください。普通は、これを行う必要はありません。
- 内部開発ではなく、実動のためのアプリケーションを構築しているとき、通常、アプリケーション中のパスは、 UNIX 上の sqllib/lib ディレクトリー、または OS/2 および Windows 32 ビット・オペレーティング・システム上の %DB2PATH%¥lib ディレクトリーのインスタンス所有者のコピーを指さないようにしてください。このようにすると、アプリケーションは特定のユーザー名と環境にかなり依存するようになります。
- 通常、特定の環境の検索パスを更新する場合は、 LIBPATH 環境変数または LIB 環境変数を使用しないでください。この変数は、その環境で実行しているアプリケーションで指定した検索パスを指定変更します。アプリケーションが、必要とするライブラリーまたはファイルを検出できなくなる可能性があります。
- DB2 ユニバーサル・データベース バージョン 6.1 および 7.1 では、ストリング意味体系のあるすべての文字配列項目は、 unsigned char などの他のバリエーションの代わりに type char をもっています。 DB2 ユニバーサル・

データベース バージョン 6.1 または 7.1 で作成するどんなアプリケーションであっても、この方式に従ってください。

unsigned char を使う DB2 バージョン 1 アプリケーションがある場合、バージョン 1 アプリケーションの unsigned char とバージョン 6.1 またはバージョン 7.1 関数プロトタイプの char との間でタイプの衝突が起こるので、コンパイラーが警告またはエラーを生成する可能性があります。 これが起こった場合、コンパイラー・オプション `-DSQLOLDCHAR` を使って問題を取り除いてください。

- DB2 ユニバーサル・データベース バージョン 7.1 と以前のバージョンの DB2 の間の非互換性のリストは、 *SQL 解説書* を参照してください。 DB2 ユニバーサル・データベース バージョン 7.1 と以前のバージョンの DB2 の間の、 API 非互換性のリストは、*管理 API 解説書* を参照してください。

付録C. 問題判別

アプリケーションを構築または実行するときには、以下のような問題が発生する可能性があります。

- クライアントまたはサーバーの問題 (構築中またはアプリケーション実行時のデータベースへの接続の失敗など)。
- オペレーティング・システムの問題 (構築中にファイルが見つからないなど)。
- 構築中のコンパイラー・オプションの問題。
- 構築中またはアプリケーション実行時の構文およびコーディングの問題。

これらの問題を解決するために、以下の情報源を使用することができます。

ビルド・ファイル
データベースへの接続、プリコンパイル、コンパイル、リンク、およびバインドなどの作成時の問題の場合、本書で示されているビルド・ファイルを使用してコマンド行プロセッサーのコマンドやコンパイラー・オプションを見ることができます。

コンパイラーの資料
構築スクリプト・ファイルがカバーするコンパイラー・オプションの問題の場合。

アプリケーション開発の手引き
構文およびその他のコーディングの問題の場合は、 *アプリケーション開発の手引き* を参照してください。

コール・レベル・インターフェース (CLI) の手引きおよび解説書
構文、CLI トレース機能、構成キーワード、および CLI プログラムに関連したその他のコーディング問題の場合は、 *コール・レベル・インターフェースの手引きおよび解説書* を参照してください。

SQL 解説書
SQL ステートメントおよび関数の構文については、 *SQL 解説書* を参照してください。

SQLCA データ構造
アプリケーションで、 SQL ステートメントを発行するか、またはデータベース・マネージャー API を呼び出す場合には、 SQLCA データ構造を検査して、エラー条件がないか調べなければなりません。

SQLCA データ構造は、SQLCODE および SQLSTATE フィールドにエラー情報を戻します。データベース・マネージャーは、それぞれの SQL ステートメントを実行した後、および大半のデータベース・マネージャー API 呼び出しの後で、その構造を更新します。

アプリケーションでは、エラー情報を検索および印刷するか、またはそれを画面に表示することができます。詳細については、*アプリケーション開発の手引き* を参照してください。

オンライン・エラー・メッセージ

データベース・マネージャー、データベース管理ユーティリティー、インストールおよび構成プロセス、およびコマンド行プロセッサーを含む、 DB2 の別の構成要素では、オンライン・エラー・メッセージが生成されます。これらの各メッセージには、固有の接頭部があり、接頭部に 4 桁または 5 桁のメッセージ番号が付きます。メッセージ番号の後には、エラーの重大度を示す 1 つの文字が表示されます。

コマンド行プロセッサーで次のように入力して、メッセージのヘルプを表示することができます。

```
db2 "? xxxnnnn"
```

ここで、xxx はメッセージ接頭部で、 nnnn はメッセージ番号です。引用符を含めてください。

DB2 エラー・メッセージの完全なリストと説明は、 *メッセージ解説書* を参照してください。

診断ツールおよびエラー・ログ

他の情報源を用いて解決できない、構築または実行時問題の場合。診断ツールには、トレース機能、システム・ログ、およびメッセージ・ログなどがあります。 DB2 は、エラーおよび警告状態を優先度および発生源に基づいてエラー・ログに入れます。詳細については、*問題判別の手引き* を参照してください。 CLI プログラムのデバッグに特に役立つ、 CLI トレース機能もあります。詳細については、*コール・レベル・インターフェースの手引きおよび解説書* を参照してください。

付録D. DB2 ライブラリーの使用法

DB2 ユニバーサル・データベース ライブラリーは、オンライン・ヘルプ、資料 (PDF および HTML)、および HTML 形式のサンプル・プログラムから成っています。このセクションでは、ユーザーに提供される情報について紹介し、その入手方法を示します。

オンライン製品情報をご利用になるには、インフォメーション・センターを使用することができます。詳細については、447ページの『インフォメーション・センターを使用した情報へのアクセス』を参照してください。ここではタスク情報、DB2 資料、トラブルシューティング情報、サンプル・プログラム、および Web の DB2 情報を見ることができます。

DB2 PDF ファイルおよびハードコピー版資料

DB2 情報

以下に示す表では、DB2 資料を 4 つのカテゴリーに分類しています。

DB2 の手引きおよび解説書
これらの資料は、すべてのプラットフォームに共通の DB2 情報を含んでいます。

DB2 のインストールおよび構成の情報
これらの資料は、特定のプラットフォーム上の DB2 ごとに用意されています。たとえば、 OS/2、Windows、および UNIX ベースのプラットフォームで稼働するそれぞれの DB2 用に、別個の*概説*および*インストール* 資料が用意されています。

プラットフォーム共通のサンプル・プログラム (HTML 形式)
これらのサンプルは、アプリケーション開発クライアントとともにインストールされるサンプル・プログラムの HTML 版です。これらのサンプルは参考用であり、実際のプログラムに代わるものではありません。

リリース情報
これらのファイルには、DB2 資料には含められなかった最新の情報が記載されています。

インストール情報、リリース情報、およびチュートリアルは、製品 CD-ROM から HTML 形式で参照することができます。ほとんどの資料は、製品

CD-ROM から HTML 形式で表示できますし、 DB2 の資料 CD-ROM から Adobe Acrobat (PDF) 形式で表示し印刷することができます。 IBM にハードコピー版の資料を注文したい場合は、 443ページの『印刷資料の注文方法』を参照してください。注文可能な資料については、以下の表をご覧ください。

OS/2 および Windows プラットフォームの場合、 HTML ファイルは sqllib¥doc¥html ディレクトリーにインストールできます。 DB2 情報はいくつかの言語で提供されています。しかし、すべての言語に翻訳されているわけではありません。ある言語で情報が提供されていない場合は、英語版の情報が提供されます。

UNIX プラットフォームの場合、言語ごとに異なる複数の HTML ファイルを doc/%*L*/html ディレクトリーにインストールできます。ここで、*%L* は地域を表しています。詳細については、適切な「概説およびインストールの手引き」を参照してください。

DB2 資料を入手して情報を利用するには、次のようなさまざまな方法があります。

- 446ページの『オンライン情報の表示』
- 451ページの『オンライン情報の検索』
- 443ページの『印刷資料の注文方法』
- 443ページの『PDF 資料の印刷』

表 18. DB2 情報

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
	DB2 の手引きおよび解説書情報		
管理の手引き	*管理の手引き: 計画* は、データベース概念について概説し、設計 (たとえば、論理および物理データベース設計) に関する情報を提供し、高い可用性について解説しています。	第 1 巻 SC88-8513 db2d1x70	db2d0
	管理の手引き: インプリメンテーション は、設計、データベースへのアクセス、監査、バックアップ、および回復などのインプリメンテーションについて説明しています。	第 2 巻 SC88-8511 db2d2x70	
	管理の手引き: パフォーマンス は、データベース環境について解説し、さらにアプリケーションのパフォーマンスの評価と調整の方法について説明しています。	第 3 巻 SC88-8512 db2d3x70	
管理 API 解説書	データベースの管理に使用できる DB2 アプリケーション・プログラミング・インターフェース (API) およびデータ構造について説明します。また、この資料は、アプリケーションから API を呼び出す方法も示します。	SC88-8514 db2b0x70	db2b0
アプリケーション構築の手引き	環境設定に関する情報を提供し、Windows、OS/2、および UNIX ベースのプラットフォームでの DB2 アプリケーションのコンパイル、リンク、実行の各ステップについて説明します。	SC88-8515 db2axx70	db2ax
APPC, CPI-C, and SNA Sense Codes	DB2 ユニバーサル・データベース製品をご使用中に発生する可能性のあるセンス・コード APPC、CPI-C、および SNA についての一般情報を提供します。 HTML 形式でのみご利用いただけます。	資料番号なし db2apx70	db2ap

表 18. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
アプリケーション開発の手引き	DB2 データベースにアクセスするアプリケーションを、組み込み SQL または Java (JDBC および SQLJ) を使用して開発する方法について説明します。さらに、ストアード・プロシージャーの作成方法、ユーザー定義関数の作成方法、ユーザー定義タイプの作成方法、トリガーの使用方法、区画化されている環境または統合されているシステムでのアプリケーションの開発方法などについて解説されています。	SC88-8516 db2a0x70	db2a0
コール・レベル・インターフェースの手引きおよび解説書	DB2 データベースにアクセスするアプリケーションを、 DB2 コール・レベル・インターフェース (Microsoft ODBC 仕様互換の呼び出し可能 SQL) を使用して開発する方法について説明します。	SC88-8517 db2l0x70	db2l0
コマンド解説書	コマンド行プロセッサーの使用法について説明し、データベースの管理に使用できる DB2 コマンドについて解説しています。	SC88-8518 db2n0x70	db2n0
コネクティビティー 補足	DB2 (AS/400 版)、DB2 (OS/390 版)、DB2 (MVS 版)、または DB2 (VM 版) を DRDA アプリケーション・リクエスターとして DB2 ユニバーサル・データベースとともに使用するためのセットアップ情報および参照情報を提供します。また、この資料は DRDA アプリケーション・サーバーを DB2 コネクト アプリケーション・リクエスターとともに使用する方法の詳細を示します。 HTML と PDF でのみ利用可能	資料番号なし db2h1x70	db2h1
データ移動ユーティリティー 手引きおよび解説書	データの移動を行う DB2 ユーティリティー (インポート、エクスポート、ロード、AutoLoader、および DPROP など) の使用法について説明しています。	SC88-8522 db2dmx70	db2dm

表 18. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
データウェアハウスセンター 管理の手引き	データウェアハウスセンターを使用してデータウェアハウスを構築および保守する方法を説明します。	SC88-8545 db2ddx70	db2dd
データウェアハウスセンター アプリケーション統合の手引き	プログラマーがアプリケーションをデータウェアハウスセンターおよび情報カタログ・マネージャーと統合するのに役立つ情報を提供します。	SC88-8546 db2adx70	db2ad
DB2 コネクト 使用者の手引き	DB2 コネクト製品の概念、プログラミング、および一般的な使用方法に関する情報を提供します。	SC88-8521 db2c0x70	db2c0
DB2 クエリー・パトローラー 管理の手引き	DB2 クエリー・パトローラー・システムの運用の概説を行い、運用および管理に関する詳細情報、および管理用グラフィカル・ユーザー・インターフェース・ユーティリティーについてのタスク情報を提供します。	SC88-8525 db2dwx70	db2dw
DB2 クエリー・パトローラー 使用者の手引き	DB2 クエリー・パトローラーのツールや関数の使用方法を説明します。	SC88-8527 db2wwx70	db2ww
用語集	DB2 およびその構成要素で使用される用語の定義を示します。 HTML 形式と SQL 解説書 で利用可能	資料番号なし db2t0x70	db2t0
イメージ、オーディオ、およびビデオ・エクステンダー 管理およびプログラミングの手引き	DB2 エクステンダーの一般情報について提供し、画像、音声、およびビデオ (IAV) エクステンダーの管理と構成について、および IAV エクステンダーを使用したプログラミングについて説明しています。さらに、参照情報、診断情報 (メッセージ解説)、およびサンプルも収録されています。	SC88-8609 dmbu7x70	dmbu7
情報カタログ・マネージャー 管理の手引き	情報カタログを管理するためのガイドです。	SC88-8547 db2dix70	db2di
情報カタログ・マネージャー プログラミングの手引きおよび解説書	情報カタログ・マネージャー用の体系化されたインターフェースの定義を示します。	SC88-8549 db2bix70	db2bi

表 18. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
情報カタログ・マネージャー 使用者の手引き	情報カタログ・マネージャー・ユーザー・インターフェースの使用に関する情報を提供します。	SC88-8548 db2aix70	db2ai
インストールおよび構成補足	プラットフォーム固有の DB2 クライアントの計画、インストール、およびセットアップのガイドです。この補足資料には、バインド、クライアント / サーバー通信の設定、 DB2 GUI ツール、DRDA AS、分散インストール、分散要求の構成、および異種データ・ソースへのアクセスについても説明されています。	GC88-8524 db2iyx70	db2iy
メッセージ解説書	DB2、情報カタログ・マネージャー、およびデータウェアハウスセンターから出されるメッセージとコードをリストし、取るべき処置を解説しています。	第 1 巻 GC88-8543 db2m1x70 第 2 巻 GC88-8544 db2m2x70	db2m0
OLAP Integration Server Administration Guide	OLAP Integration Server の Administration Manager 構成要素の使用方法を説明します。	SC27-0782 db2dpx70	n/a
OLAP Integration Server Metaoutline User's Guide	標準の OLAP Metaoutline インターフェースを使用して (Metaoutline Assistant を使用するのではなく) OLAP metaoutline を作成しデータを取り込む方法を説明しています。	SC27-0784 db2upx70	n/a
OLAP Integration Server Model User's Guide	(Model Assistant ではなく) 標準的な OLAP Model Interface を使用して OLAP モデルを作成する方法を説明します。	SC27-0783 db2lpx70	n/a
OLAP Setup and User's Guide	OLAP Starter Kit の構成およびセットアップに関する情報を提供します。	SC27-0702 db2ipx70	db2ip

表 18. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
OLAP Spreadsheet Add-in User's Guide for Excel	Excel 作表計算プログラムを使用して OLAP データを分析する方法を説明します。	SC27-0786 db2epx70	db2ep
OLAP Spreadsheet Add-in User's Guide for Lotus 1-2-3	ロータス 1-2-3 作表計算プログラムを使用して OLAP データを分析する方法を説明します。	SC27-0785 db2tpx70	db2tp
レプリケーションの手引きおよび解説書	DB2 に付属の IBM レプリケーション・ツールの計画、構成、管理、および使用方法に関する情報を提供します。	SC88-8550 db2e0x70	db2e0
地理情報エクステンダー使用者の手引きおよび解説書	地理情報エクステンダーのインストール、構成、管理、プログラミング、およびトラブルシューティングに関する情報を提供します。また、地理情報データの概念についての重要事項を示し、地理情報エクステンダー固有の参照情報 (メッセージおよび SQL) を提供します。	SC88-8624 db2sbx70	db2sb
SQL 概説	SQL の概念を紹介し、構造体とタスクの例を多数提供しています。	SC88-8539 db2y0x70	db2y0
SQL 解説書	SQL の構文、セマンティクス、および言語規則について説明します。また、この資料には、各リリース間の互換性、製品の制限事項、およびカタログ・ビューも含まれます。	第 1 巻 SC88-8540 db2s1x70 第 2 巻 SC88-8657 db2s2x70	db2s0
システム・モニター 手引きおよび解説書	データベースおよびデータベース・マネージャーに関連したさまざまな情報を収集する方法を示します。この資料は、この情報を利用して、データベース活動の把握、パフォーマンス向上、および問題原因の判別を行う方法を説明しています。	SC88-8523 db2f0x70	db2f0

表 18. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
テキスト・エクステンダー *管理およびプログラミング*	DB2 エクステンダーの一般情報、テキスト・エクステンダーの管理および構成情報、およびテキスト・エクステンダーを使用したプログラミングの方法について解説します。この資料には、参照情報、診断情報 (メッセージ解説)、およびサンプルが含まれています。	SC88-8610 desu9x70	desu9
問題判別の手引き	エラーの原因の判別、問題からの回復、および DB2 カスタマー・サービスの支援の下での診断ツールの使用法を記載しています。	GD88-7271 db2p0x70	db2p0
新機能	DB2 ユニバーサル・データベース バージョン 7 の新しい機能および拡張機能について説明します。	SC88-8541 db2q0x70	db2q0
DB2 のインストールおよび構成の情報			
DB2 コネクト エンタープライズ・エディション (OS/2 および Windows 版) 概説およびインストール	OS/2 および Windows 32 ビット オペレーティング・システム版の DB2 コネクト エンタープライズ・エディションで、計画、移行、インストール、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8520 db2c6x70	db2c6
DB2 コネクト エンタープライズ・エディション (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 コネクト エンタープライズ・エディションの計画、移行、インストール、構成、およびタスクに関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8519 db2cyx70	db2cy

表 18. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
DB2 コネクト パーソナル・エディション 概説およびインストール	OS/2 および Windows 32 ビット オペレーティング・システムの DB2 コネクト パーソナル・エディションで、計画、移行、インストール、および構成を行う場合のタスク情報を提供します。また、この資料はサポートされているすべてのクライアントのインストールおよびセットアップについても説明します。	GC88-8533 db2c1x70	db2c1
DB2 コネクト パーソナル・エディション (Linux 版) 概説およびインストール	サポートされる Linux 配布プログラムの DB2 コネクト パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8528 db2c4x70	db2c4
DB2 データ・リンク・マネージャー (Windows 版) 概説およびインストール	AIX および Windows 32 ビット・オペレーティング・システムの DB2 データ・リンク・マネージャーで、計画、インストール、構成を行う場合の情報を提供します。	GC88-8532 db2z6x70	db2z6
DB2 エンタープライズ拡張エディション (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 エンタープライズ拡張エディションの計画、インストール、および構成に関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8530 db2v3x70	db2v3
DB2 エンタープライズ拡張エディション (Windows 版) 概説およびインストール	Windows 32 ビット・オペレーティング・システムの DB2 エンタープライズ拡張エディションで、計画、インストール、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8529 db2v6x70	db2v6

表 18. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
DB2 ユニバーサル・データベース (OS/2 版) 概説およびインストール	OS/2 オペレーティング・システムでの DB2 ユニバーサル・データベースの計画、インストール、移行、および構成に関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8534 db2i2x70	db2i2
DB2 ユニバーサル・データベース (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 ユニバーサル・データベースの計画、インストール、移行、および構成に関する情報を提供します。 また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8536 db2ixx70	db2ix
DB2 ユニバーサル・データベース (Windows 版) 概説およびインストール	Windows 32 ビット オペレーティング・システムの DB2 ユニバーサル・データベースで、計画、インストール、移行、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8537 db2i6x70	db2i6
DB2 パーソナル・エディション 概説およびインストール	OS/2 および Windows 32 ビット オペレーティング・システム 版の DB2 ユニバーサル・データベース パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8535 db2i1x70	db2i1
DB2 パーソナル・エディション (Linux 版) 概説およびインストール	サポートされる Linux 配布プログラムの DB2 ユニバーサル・データベース・パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8538 db2i4x70	db2i4
DB2 クエリー・パトローラー インストールの手引き	DB2 クエリー・パトローラーのインストール情報を提供します。	GC88-8526 db2iwx70	db2iw

表 18. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
ウェアハウス・マネージャー *インストールの手引き*	ウェアハウス・エージェント、ウェアハウス・トランスフォーマー、および情報カタログ・マネージャーのインストール情報を提供します。	GC88-8572 db2idx70	db2id
プラットフォーム共通のサンプル・プログラム (HTML 形式)			
サンプル・プログラム (HTML)	DB2 のサポートするすべてのプラットフォームでのプログラム言語用に、サンプル・プログラム (HTML 形式) を提供します。これらのサンプル・プログラムは、参照用としてのみ提供されています。サンプルは、すべてのプログラミング言語で利用できるわけではありません。HTML サンプルが利用できるのは、 DB2 アプリケーション開発クライアントがインストールされている場合だけです。 プログラムの詳細については、*アプリケーション構築の手引き* を参照してください。	資料番号なし	db2hs
リリース情報			
DB2 コネクト リリース情報	DB2 コネクトの資料には含められなかった最新の情報が収録されています。	注 #2 を参照してください。	db2cr
DB2 インストール情報	DB2 資料には含められなかったインストールに関する最新の情報が収録されています。	製品 CD-ROM からのみ利用できます。	
DB2 リリース情報	DB2 資料には含められなかった DB2 製品とその機能に関する最新の情報が収録されています。	注 #2 を参照してください。	db2ir

注:

1. ファイル名の 6 桁目の文字 *x* は、その資料の言語を表します。たとえば、ファイル名 `db2d0e70` は、*管理の手引き* の英語版であることを示し、ファイル名 `db2d0f70` は同じ資料のフランス語版を示します。資料の言語を表すためにファイル名の 6 桁目で使用されている文字は以下のとおりです。

言語	識別子
ブラジル・ポルトガル語	b
ブルガリア語	u
チェコ語	x
デンマーク語	d
オランダ語	q
英語	e
フィンランド語	y
フランス語	f
ドイツ語	g
ギリシャ語	a
ハンガリー語	h
イタリア語	i
日本語	j
韓国語	k
ノルウェー語	n
ポーランド語	p
ポルトガル語	v
ロシア語	r
簡体字中国語	c
スロベニア語	l
スペイン語	z
スウェーデン語	s
繁体字中国語	t
トルコ語	m

2. DB2 資料には含められなかった最新の情報が、「リリース情報」で HTML 形式および ASCII ファイルとして利用できます。 HTML 版は、インフォメーション・センターおよび製品 CD-ROM からご利用になれます。
   ASCII ファイルの参照方法:
   - UNIX ベースのプラットフォームでは、ファイル Release.Notes を参照してください。このファイルは DB2DIR/Readme/*%L* ディレクトリーにあります。ここで *%L* は地域名を、 DB2DIR は以下のものを表します。
     - /usr/lpp/db2_07_01 (AIX の場合)
     - /opt/IBMdb2/V7.1 (HP-UX、DYNIX/ptx、Solaris、および Silicon Graphics IRIX の場合)
     - /usr/IBMdb2/V7.1 (Linux の場合)
   - これ以外のプラットフォームでは、ファイル RELEASE.TXT を参照してください。このファイルは、製品がインストールされているディレクトリーにあります。 OS/2 プラットフォームでは、 **IBM DB2** フォルダーをダブルクリックし、 **Release Notes** アイコンをダブルクリックすることもできます。

PDF 資料の印刷

資料のハードコピー版が必要な場合、 DB2 の資料 CD-ROM にある PDF ファイルを印刷することができます。 Adobe Acrobat Reader を使用すれば、資料全体または特定のページを印刷することができます。ライブラリー内の各資料のファイルについては、433ページの表18 を参照してください。

Adobe Acrobat Reader の最新版は、Adobe の Web サイト http://www.adobe.com から入手できます。

PDF ファイルは、DB2 の資料 CD-ROM に収録されており、ファイル拡張子 PDF が付いています。 PDF ファイルにアクセスするには以下のようにします。

1. DB2 の資料 CD-ROM を挿入します。 UNIX ベースのプラットフォームの場合は、 DB2 資料 CD-ROM をマウントします。マウントの手順については、*概説およびインストール* を参照してください。
2. Acrobat Reader を起動します。
3. 以下に示すいずれかの位置から必要な PDF ファイルを開きます。
   - OS/2 および Windows プラットフォームでは:

     x:¥doc¥*language* ディレクトリー。ここで、*x* は CD-ROM ドライブを、*language* は 2 桁の言語を表す国コード (たとえば、EN は英語) を示します。
   - UNIX ベースのプラットフォームでは:

     CD-ROM の */cdrom*/doc/*%L* ディレクトリー。ここで、*/cdrom* は CD-ROM のマウント・ポイントを、 *%L* は地域名を表します。

さらに、PDF ファイルを CD-ROM からローカル・ドライブまたはネットワーク・ドライブにコピーし、そこから参照することもできます。

印刷資料の注文方法

ハードコピー版の DB2 資料は、個別に注文することができます。資料を注文するには、IBM 承認の販売業者または営業担当員に連絡してください。

DB2 オンライン文書

オンライン・ヘルプへのアクセス

すべての DB2 構成要素で、オンライン・ヘルプを利用できます。以下の表に、さまざまな種類のヘルプを示します。

ヘルプの種類	内容	利用方法
コマンド・ヘルプ	コマンド行プロセッサーのコマンド構文について説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。 `? command` ここで *command* はキーワードまたはコマンド全体を表します。 たとえば、`? catalog` と入力すると、すべての CATALOG コマンドに関するヘルプが表示され、`? catalog database` と入力すると、 CATALOG DATABASE コマンドのヘルプが表示されます。
クライアント構成アシスタントのヘルプ コマンド・センターのヘルプ コントロール・センターのヘルプ データウェアハウスセンターのヘルプ イベント・アナライザーのヘルプ 情報カタログ・マネージャーのヘルプ サテライト管理センターのヘルプ スクリプト・センターのヘルプ	そのウィンドウまたはノートブックで実行できるタスクについて説明します。このヘルプは、知っておく必要のある概説および前提条件に関する情報を含みます。また、ウィンドウやノートブックの制御の使用方法を示します。	ウィンドウまたはノートブックから、**「ヘルプ (Help)」**押しボタンをクリックするか、または **F1** キーを押します。

ヘルプの種類	内容	利用方法

メッセージ・ヘルプ	メッセージの原因、および取るべき処置を説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。
		`? XXXnnnnn`
		ここで、*XXXnnnnn* は有効なメッセージ識別子を表します。
		たとえば、`? SQL30081` と入力すると、メッセージ SQL30081 に関するヘルプを表示します。
		一度に 1 画面分のメッセージ・ヘルプを表示させるには、次のように入力します。
		`? XXXnnnnn | more`
		メッセージ・ヘルプをファイルに保管するには、次のように入力します。
		`? XXXnnnnn > filename.ext`
		ここで、*filename.ext* はメッセージ・ヘルプを保管するファイルを表します。

SQL ヘルプ	SQL ステートメントの構文について説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。 `help statement` ここで、*statement* は SQL ステートメントを表します。 たとえば、help SELECT と入力すると、 SELECT ステートメントのヘルプが表示されます。 **注:** UNIX ベースのプラットフォームでは、SQL ヘルプを利用できません。
SQLSTATE ヘルプ	SQL 状態およびクラス・コードについて説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。 `? sqlstate or ? class code` ここで、*sqlstate* は有効な 5 桁の SQL 状態を、*class code* は SQL 状態の最初の 2 桁を表します。 たとえば、`? 08003` によって SQL 状態 08003 のヘルプが表示され、 `? 08` によってクラス・コード 08 のヘルプが表示されます。

オンライン情報の表示

この製品に付属の資料は、ハイパーテキスト・マークアップ言語 (HTML) ソフトコピー形式です。ソフトコピー形式では情報を検索または表示したり、ハイパーテキスト・リンクを利用して関連情報に移動したりすることができます。また、1 つの端末を超えてライブラリーを容易に共用することができます。

オンライン資料やサンプル・プログラムは、 HTML バージョン 3.2 仕様に準拠するすべてのブラウザーを使って表示できます。

オンライン資料またはサンプル・プログラムは、次のようにして表示します。

- DB2 管理ツールを実行している場合、インフォメーション・センターを使用します。
- ブラウザーで、**ファイル (File) —>ページを開く (Open Page)** をクリックします。次のようなページを開いて、DB2 情報に関する説明とリンクを表示してください。
  - UNIX ベースのプラットフォームでは、以下のページを開きます。

    INSTHOME/sqllib/doc/*%L*/html/index.htm

    ここで *%L* はロケール名です。
  - その他のプラットフォームでは、以下のページを開きます。

    sqllib¥doc¥html¥index.htm

    パスは DB2 がインストールされているドライブです。

    インフォメーション・センターをインストールしていない場合、 **DB2 Information** アイコンをダブルクリックしてページを開くことができます。このアイコンは、ご使用のシステムに応じて、製品のメイン・フォルダー内または Windows 「スタート」メニューにあります。

Netscape ブラウザーのインストール

システムに Web ブラウザーがインストールされていない場合、製品の箱の中にある Netscape CD-ROM から Netscape をインストールすることができます。インストールに関する詳細な説明については、以下を参照してください。

1. Netscape CD-ROM を挿入します。
2. UNIX ベースのプラットフォームでは、CD-ROM をマウントします。マウントの手順については、*概説およびインストール* を参照してください。

3. インストールの手順については、 CDNAV*nn*.txt ファイルを参照します。ここで、*nn* は 2 桁の言語識別子を表します。ファイルは CD-ROM のルート・ディレクトリーにあります。

インフォメーション・センターを使用した情報へのアクセス

インフォメーション・センターを使用すると、DB2 製品情報にすばやくアクセスすることができます。インフォメーション・センターは、DB2 管理ツールを使用できるすべてのプラットフォームで利用できます。

インフォメーション・センターは「インフォメーション・センター (Information Center)」アイコンをダブルクリックすることによってオープンできます。このアイコンのある場所はシステムによって異なります。メイン・プロダクト・フォルダーか Windows の**「スタート」**メニューのどちらかです。

Windows プラットフォームの DB2 では、ツールバーおよび**ヘルプ**・メニューを使用して、インフォメーション・センターにアクセスすることもできます。

インフォメーション・センターは 6 種類の情報を提供します。適切なタブをクリックすると、種類ごとに提供されているトピックが表示されます。

タスク (Tasks)
DB2 を使用して実行できる主要なタスク。

参照 (Reference)
DB2 参照情報 (キーワード、コマンド、API など)。

資料 (Books) DB2 資料。

トラブルシューティング (Troubleshooting)
エラー・メッセージのカテゴリーと、メッセージに対する回復処置。

サンプル・プログラム (Sample Programs)
DB2 アプリケーション開発クライアントに付属のサンプル・プログラム。 DB2 アプリケーション開発クライアントをインストールしていない場合、このタブは表示されません。

Web WWW 上にある DB2 情報。この情報にアクセスするには、ご使用のシステムから Web への接続が必要です。

リストから項目を 1 つ選択すると、インフォメーション・センターはビューアーを立ち上げて情報を表示します。選択した情報の種類に応じて、ビューアーはシステム・ヘルプ・ビューアー、エディター、または Web ブラウザーです。

インフォメーション・センターには検索機能が備わっており、リストを参照せずに特定のトピックを探すことができます。

テキストの全検索を行うには、インフォメーション・センター内のハイパーテキスト・リンク「**DB2 オンライン情報の検索 (Search DB2 Online Information)**」検索フォームに従います。

通常、HTML 検索サーバーは自動的に始動します。 HTML 情報の検索がうまくいかない場合は、以下の方法の 1 つを使用して、検索サーバーを始動しなければならない場合もあります。

Windows では

「**スタート**」をクリックし、「**プログラム**」—>「**IBM DB2**」—>「**Information**」—>「**Start HTML Search Server**」を選択します。

OS/2 では

「**DB2 (OS/2 版)**」フォルダーをダブルクリックして、「**Start HTML Search Server**」アイコンをダブルクリックします。

HTML 情報の検索でこの他の問題が発生した場合は、リリース情報を参照してください。

注: 検索機能は、Linux、DYNIX/ptx、および Silicon Graphics IRIX 環境では利用できません。

DB2 ウィザードの使用

ウィザードを使用すると、各タスクをステップごとに進めることによって、さまざまな管理タスクを遂行することができます。ウィザードは、コントロール・センターおよびクライアント構成アシスタントを通して使用できます。以下の表では、ウィザードとその目的をリストしています。

注: データベース作成、索引作成、複数サイト更新の構成、およびパフォーマンス構成ウィザードは、区分データベース環境で使用できます。

ウィザード	内容	利用方法
データベース追加 (Add Database)	クライアント・ワークステーション上にデータベースのカタログを作成します。	クライアント構成アシスタントから、「**追加 (Add)**」をクリックします。

ウィザード	内容	利用方法
データベース・バックアップ (Back up Database)	バックアップ計画を決定、作成、およびスケジュールします。	「コントロール・センター (Control Center)」からバックアップするデータベースを右クリックし、**「バックアップ (Backup)」—>「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。
複数サイト更新の構成 (Configure Multisite Update)	複数サイト更新、分散トランザクション、または 2 フェーズ・コミットを構成します。	「コントロール・センター (Control Center)」から、**「データベース (Databases)」**フォルダーを右クリックして、**「複数サイト更新 (Multisite Update)」**を選択します。
データベース作成 (Create Database)	データベースを作成し、いくつかの基本的な構成タスクを実行します。	「コントロール・センター (Control Center)」から、**「データベース (Databases)」**フォルダーを右クリックして、**「作成 (Create)」—> 「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。
表作成 (Create Table)	基本的なデータ・タイプを選択して、表の基本キーを作成します。	「コントロール・センター (Control Center)」から、**「表 (Tables)」**アイコンを右クリックして、**「作成 (Create)」—>「ウィザードを使用する表 (Table Using Wizard)」**を選択します。
表スペース作成 (Create Table Space)	新しい表スペースを作成します。	「コントロール・センター (Control Center)」から、**「表スペース (Table Spaces)」**アイコンを右クリックして、**「作成 (Create)」—> 「ウィザードを使用する表スペース (Table Space Using Wizard)」**を選択します。
索引作成 (Create Index)	すべての照会について、作成すべき索引および除去すべき索引を提案します。	「コントロール・センター (Control Center)」から、**「索引 (Index)」**アイコンを右クリックして、**「作成 (Create)」—>「ウィザードを使用する索引 (Index Using Wizard)」**を選択します。

ウィザード	内容	利用方法
パフォーマンス構成 (Performance Configuration)	ビジネス要件に適合するように構成パラメーターを更新して、データベースのパフォーマンスを調整します。	「コントロール・センター (Control Center)」から、調整したいデータベースを右クリックして、**「ウィザードを使用するパフォーマンスの構成 (Configure Performance Using Wizard)」**を選択します。 区分データベース環境では、「Database Partitions」視点から、調整したい最初のデータベース区画を右クリックして、**「ウィザードを使用するパフォーマンスの構成 (Configure Performance Using Wizard)」**を選択します。
データベース復元 (Restore Database)	障害の後、データベースを回復します。どのバックアップを使用し、どのログを再生するかの判別を支援します。	「コントロール・センター (Control Center)」から復元するデータベースを右クリックし、**「復元 (Restore)」—>「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。

文書サーバーのセットアップ

デフォルトでは、DB2 情報はローカル・システムにインストールされます。つまり、DB2 情報にアクセスする必要のある各担当者が同じファイルをインストールする必要があります。 DB2 情報を 1 か所に格納するには、次のようにします。

1. ¥sqllib¥doc¥html のすべてのファイルとサブディレクトリーを、ローカル・システムから Web サーバーにコピーします。各資料には独自のサブディレクトリーがあり、その資料を構成する必要な HTML および GIF ファイルが入っています。ディレクトリー構造は常に同じ状態に保つ必要があります。
2. Web サーバーを構成して、ファイルを新しい場所で検索するようにします。さらに詳しい情報については、 *インストールおよび構成 補足* の NetQuestion 付録を参照してください。
3. インフォメーション・センターの Java バージョンをご使用の場合は、すべての HTML ファイルのベース URL を指定できます。この URL は資料のリストに使用してください。

4. 資料ファイルが表示されるようになったなら、よく使うトピックにはブックマークを付けておいてください。ブックマークを付けるページは、たとえば以下のものがあります。
   - 資料のリスト
   - 頻繁に使用される資料の目次
   - 頻繁に参照する情報 (たとえば、ALTER TABLE トピックなど)
   - 検索フォーム

中央のマシンから DB2 ユニバーサル・データベース オンライン文書ファイルを提供する方法については、 *インストールおよび構成 補足* の NetQuestion 付録を参照してください。

オンライン情報の検索

HTML ファイルの情報を検索するには、以下の方法のどれか 1 つを使用してください。

- 最上部にある**「検索 (Search)」**をクリックします。検索フォームを使用して特定のトピックを見つけます。この機能は、 Linux、DYNIX/ptx、または Silicon Graphics IRIX 環境ではご利用になれません。
- 最上部にある**「索引 (Index)」**をクリックします。索引を使用して、資料内の特定のトピックを見つけます。
- HTML 資料またはヘルプの目次あるいは索引を表示してから、 Web ブラウザーの検索機能を利用して資料内の特定のトピックを見つけます。
- Web ブラウザーのブックマーク機能を使用して、特定のトピックにすばやく戻ります。
- インフォメーション・センターの検索機能を使用して、特定のトピックを検索します。詳しくは、447ページの『インフォメーション・センターを使用した情報へのアクセス』を参照してください。

付録E. 特記事項

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミングまたはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミングまたはサービスを、日本で発表する意図があることを必ずしも示すものではありません。本書で、 IBM ライセンス・プログラムまたは他の IBM 製品に言及している部分があっても、このことは当該プログラムまたは製品のみが使用可能であることを意味するものではありません。これらのプログラムまたは製品に代えて、 IBM の知的所有権を侵害することのない機能的に同等な他社のプログラム、製品またはサービスを使用することができます。ただし、IBM によって明示的に指定されたものを除き、これらのプログラムまたは製品に関連する稼働の評価および検証はお客様の責任で行っていただきます。

IBM および他社は、本書で説明する主題に関する特許権 (特許出願を含む)、商標権、または著作権を所有している場合があります。本書は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用権等を許諾することを意味するものではありません。実施権、使用権等の許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
AP 事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

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。

本書に含まれる情報には、技術的に不正確なもの、または誤植が含まれる場合があります。これらに対する変更は、定期的に行われます。これらの変更は、資料の改訂版に含まれます。 IBM は、本書で説明している製品、プログラムに対して、予告なく改良、変更を加える場合があります。

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、お客様になんら義務も負わせない適切な方法で、使用もしくは配布することがあります。

本プログラムのライセンス保持者で、 (i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Limited
Office of the Lab Director
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

本プログラムに関する上記の情報は、適切な条件の下で使用することができますが、有償の場合もあります。

本書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。

本書に含まれるパフォーマンス・データは、制御された環境下で決定されています。したがいまして、その他の稼働環境で得られる結果とは、かなり異なる可能性もあります。一部の測定値は、開発中のシステムを使用している場合があり、これらの測定値が一般的に提供可能なシステムで同様の数値になることを保証するものではありません。さらに、一部の測定値が推定されたものもあります。実測値と異なる場合があります。本書のユーザーは、使用される特定の環境での該当データを確認してください。

IBM 以外の製品については、当該製品の提供者から直接、出版されている資料または一般公開されている情報から入手しました。 IBM は、これらの製品についてはテストを行っておらず、これらの IBM 以外の製品に関する性能、互換性またはその他の主張について確認することはできません。 IBM 以外の製品の機能に対する質問は、それぞれの製品提供者にお問い合わせください。

IBM の将来の方向性または意図については、予告なしに変更または中止する場合があり、 IBM の目的および目標のみ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が、これは説明に具体性を与えるために記載されたものであり、それらの例には、個人、企業、ブランドの、あるいは製品などの名前が含まれている場合があります。それらの名前はすべて架空のものであり、また名称や住所が類似する企業が実在しても、それは偶然に過ぎません。

著作権：

本書に含まれる情報には、サンプル・アプリケーション・プログラムがソース言語の形式で含まれており、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技法を示しています。お客様は、これらのサンプル・プログラムが書かれているオペレーティング・プラットフォームでアプリケーション・プログラミング・インターフェースが実行可能となるためのアプリケーション・プログラムを開発、使用、販売または配布もしくは転送する目的のためにのみ、サンプル・プログラムを、IBM に対する別途料金を支払うことなく、複製、変更、配布または転送することができます。これらのサンプルは、すべての条件下で十分にテストを行っていません。したがいまして、IBM は、これらのプログラムの信頼性、実用性または機能について、いかなる保証も負いません。

サンプル・プログラムまたはその改変版の複製物には、全部複製か部分複製かを問わず、次の著作権表示を必ず行うものとします。

© (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_. All rights reserved.

商標

次のものは、 IBM Corporation の米国およびその他の国における商標です。

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eNetwork
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
LAN DistanceMVS
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
PowerPC
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/DS
SQL/400
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2

次のものは、他社の商標または登録商標です。

Tivoli および NetView は、米国およびその他の国における Tivoli Systems Inc. の商標です。

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。

Java およびすべての Java 関連の商標およびロゴは Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標または登録商標です。

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］

［ワ行］

A

B

C

D

E

F

H

J

L

M

N

O

P

R

S

U

W

IBM と連絡をとる

技術上の問題がある場合は、時間をとって「問題判別の手引き」に定義されている処置を検討し、それらの提案を実行した後で、DB2 顧客サービスに連絡をとってください。この資料には、DB2 顧客サービスがお客さまを支援するために必要とする情報が説明されています。

製品情報

以下の情報は英語で提供されます。内容は英語版製品に関する情報です。

http://www.ibm.com/software/data/

DB2 World Wide Web ページには、ニュース、製品説明、研修スケジュールなどの DB2 に関する最新情報が提供されています。ただし、提供されている情報は英語です。

http://www.ibm.com/software/data/db2/library/

「DB2 Product and Service Technical Library」では、よくされる質問 (FAQ)、修正内容、資料、および最新の DB2 技術情報などの情報へのアクセスが提供されています。

注: この情報のご提供は英語のみとなりますのでご注意ください。

http://www.elink.ibmlink.ibm.com/pbl/pbl/

「International Publications」注文用 Web サイトでは、マニュアルの注文方法についての情報を提供しています。ただし、提供されている情報は英語です。

http://www.ibm.com/education/certify/

IBM の「Professional Certification Program」Web サイトでは、 DB2 を含むさまざまな IBM 製品の認証テストの情報を提供しています。ただし、提供されている情報は英語です。

ftp.software.ibm.com

匿名でログオンしてください。ディレクトリー `/ps/products/db2` には、 DB2 および多数の他製品に関連したデモ、修正プログラム、情報、およびツールがあります。ただし、提供されている情報は英語です。

comp.databases.ibm-db2, bit.listserv.db2-l

これらのインターネット・ニュースグループは、ユーザーが DB2 製品に関する自分の経験について話し合うために利用できます。ただし、提供されている情報は英語です。

Compuserve: GO IBMDB2

このコマンドを入力すると、IBM DB2 Family forum にアクセスできます。すべての DB2 製品が、このフォーラムでサポートされています。ただし、提供されている情報は英語です。

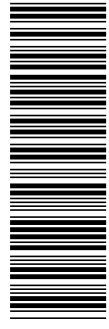
米国以外の国で IBM に連絡する方法については、 IBM Software Support Handbook の Appendix A を参照してください。この資料にアクセスするには、 Web ページ: http://www.ibm.com/support/ にアクセスし、ページの最下部にある「IBM Software Support Handbook」リンク・ボタンを選択します。

注: 国によっては、 IBM が承認している販売業者が、 IBM サポート・センターの代わりにそれら販売業者のサポート・センターに連絡する場合があります。

部品番号: CT7XXJA

Printed in Japan

CT7XXJA



SC88-8515-00

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12