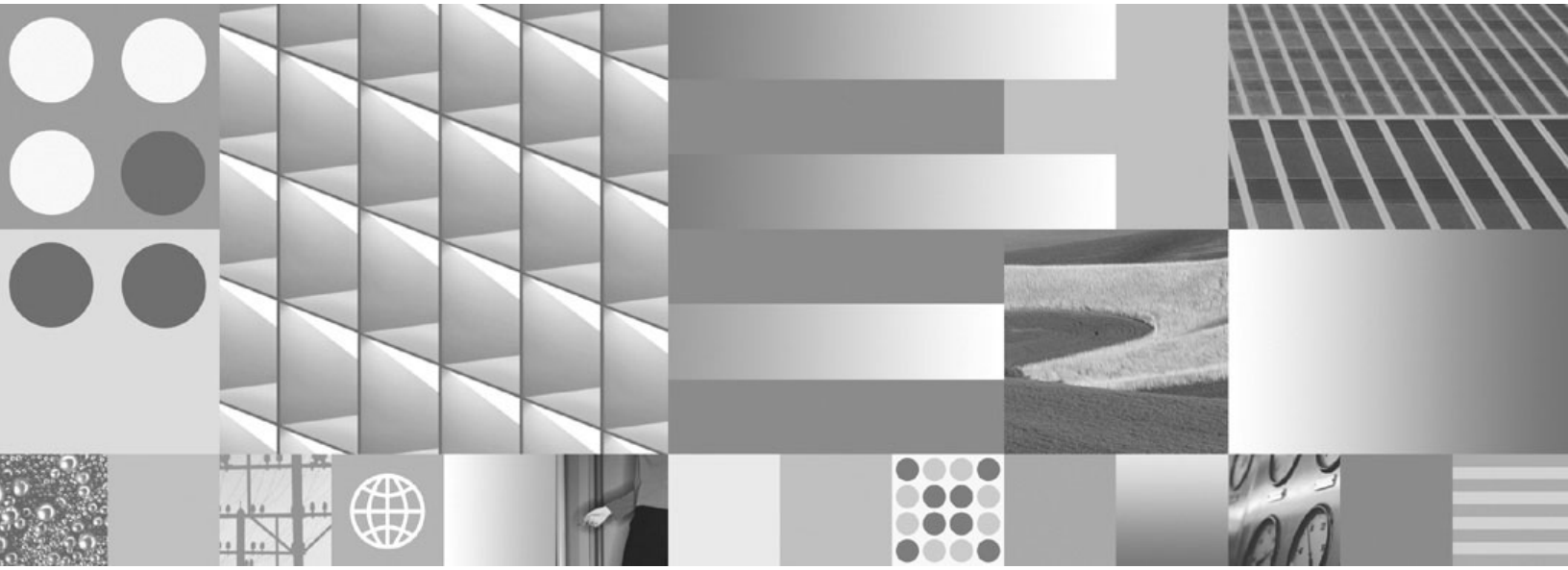
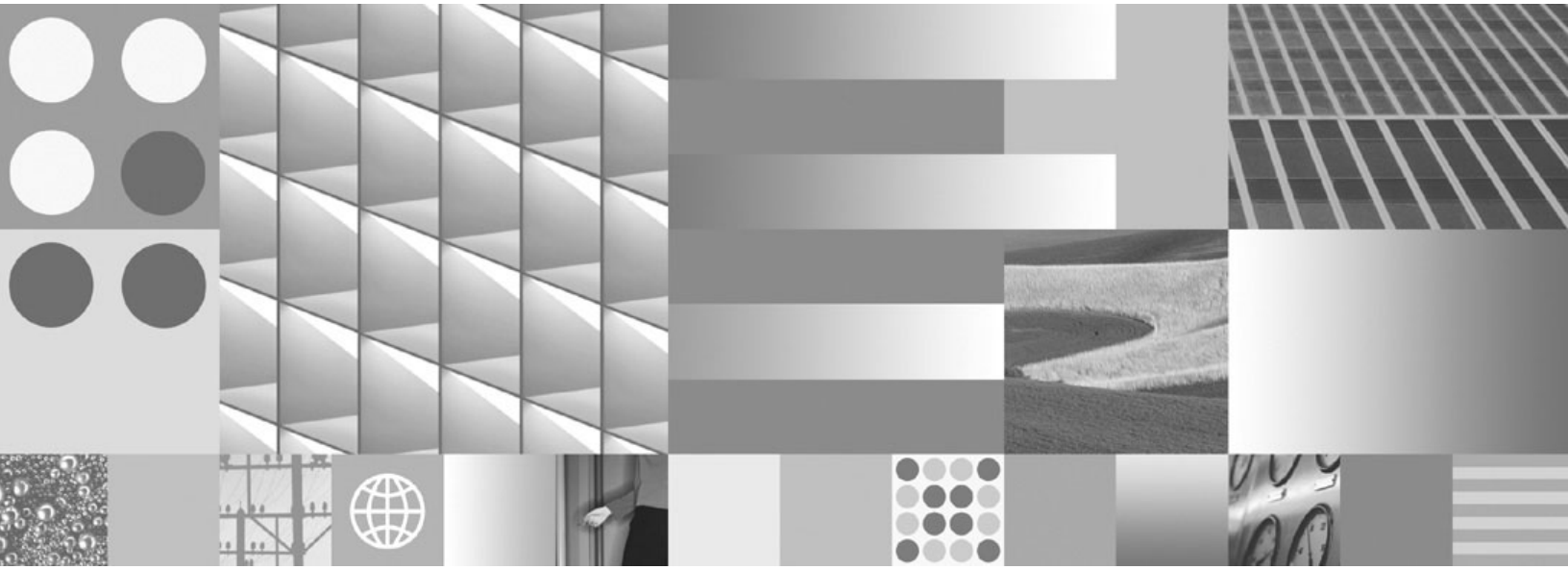
DB2 Version 9.5
for Linux, UNIX, and Windows



組み込み SQL アプリケーションの開発

SC88-4426-01
(英文原典：SC23-5852-01)

DB2 Version 9.5
for Linux, UNIX, and Windows



組み込み SQL アプリケーションの開発

SC88-4426-01
(英文原典：SC23-5852-01)

ご注意

本書および本書で紹介する製品をご使用になる前に、235 ページの『付録 B. 特記事項』に記載されている情報をお読みください。

原　典： SC23-5852-01
DB2 Version 9.5 for Linux, UNIX, and Windows
Developing Embedded SQL Applications

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 2008.3

目次

第 1 部 組み込み SQL の概要

組み込み SQL データベース・アプリケーションは、データベースに接続し、組み込み SQL ステートメントを実行します。組み込み SQL ステートメントは、ホスト言語アプリケーション内に組み込まれます。組み込み SQL データベース・アプリケーションは、静的または動的に実行される SQL ステートメントの組み込みをサポートします。

DB2® 用の組み込み SQL アプリケーションは、ホスト・プログラミング言語の C、C++、COBOL、FORTRAN、および REXX で開発することができます。

注: FORTRAN および REXX での組み込み SQL サポートは使用すべきでなくなっているので、DB2 Universal Database™ バージョン 5.2 レベルのままになっています。

組み込み SQL アプリケーションのビルドでは、アプリケーションのコンパイルおよびリンクの前に、前提条件となる 2 つのステップがあります。

- DB2 プリコンパイラーを使用した、組み込み SQL ステートメントを含むソース・ファイルの準備。

  ソース・コードを読み取り、組み込み SQL ステートメントを解析して DB2 実行時サービス API 呼び出しに変換し、最後に出力を新たな変更済みソース・ファイルに書き出す DB2 プリコンパイラーの起動には、`PREP (PRECOMPILE)` コマンドが使用されます。プリコンパイラーは、SQL ステートメントに対するアクセス・プランを生成し、それらは共にパッケージとしてデータベース内に保管されます。
- アプリケーション内のステートメントのターゲット・データベースへのバインド。

  バインディングはプリコンパイル時 (`PREP` コマンド) にデフォルトで行われます。バインディングを延期する (たとえば、`BIND` コマンドを後で実行する) 場合、バインド・ファイルが生成されるようにするため、`BINDFILE` オプションを `PREP` の実行時に指定する必要があります。

組み込み SQL アプリケーションをプリコンパイルしてバインドすると、ホスト言語固有の開発ツールを使用してコンパイルおよびリンクができるようになります。

組み込み SQL アプリケーションの開発に際しては、 『C での組み込み SQL テンプレート』を参照すると役に立つでしょう。サンプルの組み込み SQL アプリケーションを扱う例は、`%DB2PATH%¥SQLLIB¥samples` ディレクトリーにもあります。

注: %DB2PATH% は DB2 インストール・ディレクトリーを指します。

静的および動的 SQL

SQL ステートメントは、2 つの方法、つまり静的または動的な方法のいずれかで実行できます。

静的に実行される SQL ステートメント

静的に実行される SQL ステートメントの場合、構文はプリコンパイル時に完全に分かっています。 SQL ステートメントの構造は、静的と考えられるステートメントとして完全に指定されていなければなりません。たとえば、ステートメントで参照される列または表の名前は、プリコンパイル時に完全に認識されている必要があります。実行時に指定できる唯一の情報は、ステートメントが参照するホスト変数の値だけです。ただし、データ・タイプなどのホスト変数情報は、プリコンパイルしなければなりません。静的に実行される SQL ステートメントのプリコンパイル、バインド、およびコンパイルは、アプリケーションを実行する前に行います。静的 SQL は、統計が大幅に変更されないデータベースで使用するのに最適です。

動的に実行される SQL ステートメント

動的に実行される SQL ステートメントは、アプリケーションによって実行時にビルドおよび実行されます。エンド・ユーザーに対してプロンプトを出し、検索する表および列の名前など、SQL ステートメントの重要な部分の入力を求める対話式アプリケーションが、動的 SQL に適した状況の良い例です。

第 1 章 ホスト言語における組み込み SQL ステートメント

構造化照会言語 (SQL) は、データベース・オブジェクトおよびその中に含まれるデータの操作に使用できる標準化された言語です。ホスト言語ごとに差異はありますが、組み込み SQL アプリケーションはすべて、 SQL ステートメントのセットアップおよび実行に必要な 3 つの主なエレメントから構成されます。

1. ホスト変数を宣言するための DECLARE SECTION。SQLCA 構造体の宣言は、DECLARE セクション内に存在する必要はありません。
2. アプリケーションのメイン・ボディで、SQL ステートメントをセットアップおよび実行する部分。
3. SQL ステートメントによって行われた変更をコミットあるいはロールバックするロジックの配置。

それぞれのホスト言語について、すべての言語に適用される一般ガイドラインと、個々の言語に固有のルールの間の違いがあります。

C および C++ アプリケーションにおける組み込み SQL ステートメント

組み込み SQL C および C++ アプリケーションは、SQL ステートメントをセットアップおよび実行するための 3 つの主要エレメントから構成されます。

- ホスト変数を宣言するための DECLARE SECTION。SQLCA 構造体の宣言は、DECLARE セクション内に存在する必要はありません。
- アプリケーションのメイン・ボディで、SQL ステートメントをセットアップおよび実行する部分。
- SQL ステートメントによって行われた変更をコミットあるいはロールバックするロジックの配置。

正しい C および C++ エレメント構文

ステートメント初期化指定子
EXEC SQL

ステートメント・ストリング
任意の有効な SQL ステートメント

ステートメント終止符
セミコロン (;)

たとえば、C アプリケーション内で SQL ステートメントを静的に実行するために、アプリケーション・コード内に以下のものを含めることもできます。

```
EXEC SQL SELECT col INTO :hostvar FROM table;
```

以下の例は、ホスト変数 stmt1 を使用して SQL ステートメントを動的に実行する方法を示したものです。

```
strcpy(stmt1, "CREATE TABLE table1(col1 INTEGER)");
EXEC SQL EXECUTE IMMEDIATE :stmt1;
```

以下のガイドラインおよび規則は、C および C++ アプリケーションでの組み込み SQL ステートメントの実行に適用されます。

- SQL ステートメント・ストリングは、EXEC SQL ステートメント初期化指定子と同一の行で開始することができる。
- ただし、対になった EXEC SQL を複数行に分割してはなりません。
- SQL ステートメント終止符を使用しなければならない。使用しない場合、プリコンパイラーはアプリケーション内の次の終止符まで処理を継続します。これにより、不確定のエラーが起こる恐れがあります。
- C および C++ コメントは、ステートメント初期化指定子の前、またはステートメント終止符の後に入れることができます。
- 複数の SQL ステートメントと C または C++ ステートメントは同じ行に置くことができる。以下に例を示します。

  ```
  EXEC SQL OPEN c1; if (SQLCODE >= 0) EXEC SQL FETCH c1 INTO :hv;
  ```

- 引用符付きストリングには、復帰 (CR)、改行 (LF)、 および TAB を含めることができる。 SQL プリコンパイラーはこれらをそのまま残します。
- SQL ステートメントを含んだファイルの組み込みに #include ステートメントは使用できません。 SQL ステートメントはモジュールがコンパイルされる前にプリコンパイルされます。プリコンパイラーは、#include ステートメントを無視します。代わりに SQL INCLUDE ステートメントを使用して、組み込みファイルをインポートしてください。
- SQL コメントは、組み込み SQL ステートメント内であれば、どの行にでも置くことができる。ただし、動的に実行されるステートメントは例外です。
  - SQL コメントの形式は、ダブル・ダッシュ (--) の後に 0 個以上の文字ストリングが続き、行末で終了します。
  - SQL コメントは SQL ステートメント終止符の後に置かないでください。これらの SQL コメントは、コンパイラーによって C または C++ 構文としてインタープリットされるために、コンパイル・エラーが生じます。
  - SQL コメントは、ブランクを使用できる静的ステートメント・ストリング内であればどこでも使用することができます。
  - 静的および動的組み込み SQL ステートメントのいずれにおいても、C および C++ のコメント区切り文字である /* */ の使用が許可されます。
  - // スタイルの C++ コメントの使用は、静的 SQL ステートメント内では許可されません。
- C および C++ アプリケーションでは、 SQL ストリング・リテラルや区切り ID は次の行にわたって続けることができる。このことを行うには、次の行に続けたい行の末尾に円記号 (¥) を使用します。たとえば、社員表 (staff) で NAME 列が 'Sanders' であるという条件で NAME 列からデータを選択する場合、以下のように書きます。

  ```
  EXEC SQL SELECT "NA¥
  ME" INTO :n FROM staff WHERE name='Sa¥
  nders';
  ```

  改行文字 (復帰や改行など) はいずれも、 SQL ステートメントとしてデータベース・マネージャーに渡されるストリングには含まれません。
- 行末文字およびタブ文字などの空白文字の置換は、次のように行われる。

– 引用符の外 (ただし、SQL ステートメント内) では、行末文字または TAB 文字は単一スペースと置き換えられる。
– 引用符内では、ストリングが C プログラムに適合した形で継続されていれば、行末文字は消去される。 TAB は修正されません。

行末および TAB に使用される実際の文字は、プラットフォームごとに異なります。たとえば、UNIX® および Linux® ベースのシステムでは改行 (LF) が使用されます。

FORTRAN アプリケーションにおける組み込み SQL ステートメント

FORTRAN アプリケーションの組み込み SQL ステートメントは次の 3 つのエレメントから成ります。

正しい FORTRAN エレメント構文

ステートメント初期化指定子
EXEC SQL

ステートメント・ストリング
ブランクを区切り文字として使用した有効な SQL ステートメント

ステートメント終止符
ソース行の終わり

ソース行の終端は、ステートメント終止符として機能します。行が継続する場合、ステートメント終止符は連続記入行の最終行の終端に位置します。

以下に例を示します。

```
EXEC SQL SELECT COL INTO :hostvar FROM TABLE
```

FORTRAN アプリケーションの組み込み SQL ステートメントには、以下の規則が適用されます。

- SQL ステートメントは 7 列から 72 列までの間でのみコーディングする。
- 全行の FORTRAN コメント、または SQL コメントを使用する。ただし、SQL ステートメント内で FORTRAN 行末コメント '!' 文字は使用できません。それ以外の場所であれば、このコメント文字はホスト変数宣言を含めてどこでも使用できます。
- 組み込み SQL ステートメントをコーディングする際は、 FORTRAN ステートメントにその必要がない場合でも、ブランクを区切り文字として使用する。
- 各 FORTRAN ソース行に対しては、SQL ステートメントのみを使用する。複数の行が必要なステートメントには、通常の FORTRAN の連結規則が適用されます。ただし、EXEC SQL ステートメントの初期化指定子は複数行に分割しないでください。
- SQL コメントは、組み込み SQL ステートメントの一部となっている行であればどこでも使用することができる。このコメントは、動的に実行するステートメントでは使用できません。 SQL コメントの形式は、ダブル・ダッシュ (--) の後に 0 個以上の文字ストリングが続き、行末で終了します。
- FORTRAN のコメントは、組み込み SQL ステートメント内のほとんど の場所で使用することができる。例外は以下のとおりです。

- コメントは EXEC と SQL との間では使用できない。
- コメントは動的に実行されるステートメントでは使用できない。
- 行末で ! を使用して FORTRAN のコメントをコーディングする拡張は、組み込み SQL ステートメント内部ではサポートされていない。

- SQL ステートメント内で実定数を指定する際には、指数表記法を使用する。データベース・マネージャーは、SQL ステートメント内にある 小数点を持った数字ストリングを、実定数ではなく 10 進定数と解釈します。
- 最初の実行可能 FORTRAN ステートメントよりも前にある SQL ステートメントでは、ステートメント番号は無効になる。 SQL ステートメントにこれと対応するステートメント番号が付けられている場合、プリコンパイラーは、SQL ステートメントの直前に置かれるラベル付き CONTINUE ステートメントを生成します。
- SQL ステートメント内のホスト変数の参照時に宣言されたとおりのホスト変数を使用する。
- 行末文字およびタブ文字などの空白文字の置換は、次のように行われる。
  - 引用符の外 (ただし、SQL ステートメント内) では、行末文字または TAB 文字は単一スペースと置き換えられる。
  - 引用符内では、FORTRAN プログラムに適合した形でストリングが継続されていれば、行末文字は消去される。 TAB は修正されません。

  行末および TAB に使用される実際の文字は、プラットフォームごとに異なります。たとえば Windows ベースのプラットフォームでは、復帰/改行を行末に使用するのに対し、UNIX および Linux ベースのプラットフォームは改行のみを使用します。

COBOL アプリケーションにおける組み込み SQL ステートメント

COBOL アプリケーションの組み込み SQL ステートメントは次の 3 つのエレメントから成ります。

正しい COBOL エレメント構文

ステートメント初期化指定子
EXEC SQL

ステートメント・ストリング
任意の有効な SQL ステートメント

ステートメント終止符
END-EXEC.

以下に例を示します。

```
EXEC SQL SELECT col INTO :hostvar FROM table END-EXEC.
```

COBOL アプリケーションの場合、組み込み SQL ステートメントには、以下の規則が適用されます。

- 実行可能な SQL ステートメントは、PROCEDURE DIVISION セクションになければなりません。SQL ステートメントの前には、COBOL ステートメントとして段落名を付けることができます。

- SQL ステートメントは、**領域 A** (列 8 から 11 まで) または**領域 B** (列 12 から 72) のどちらからも開始することができます。
- 各 SQL ステートメントは、ステートメント初期化指定子 EXEC SQL で開始してステートメント終了指定子 END-EXEC で終了します。各 SQL ステートメントは、修正済みソース・ファイル内のコメントとして、 SQL プリコンパイラーに含まれます。
- SQL ステートメント終止符を使用しなければならない。使用しない場合、プリコンパイラーはアプリケーション内の次の終止符まで処理を継続します。これにより、不確定のエラーが起こる恐れがあります。
- SQL コメントは、組み込み SQL ステートメントの一部となっている行であればどこでも使用することができる。このコメントは、動的に実行するステートメントでは使用できません。 SQL コメントの形式は、ダブル・ダッシュ (**--**) の後に 0 個以上の文字ストリングが続き、行末で終了します。SQL コメントを SQL ステートメントの後に置かないようにしてください。これを行うと、コメントが COBOL 言語の一部のように見えるため、コンパイル・エラーの原因となるからです。
- COBOL コメントはほとんどの場所で許可されます。例外は以下のとおりです。
  - コメントは EXEC と SQL との間では使用できない。
  - コメントは動的に実行されるステートメントでは使用できない。
- SQL ステートメントは、COBOL 言語と同じ行継続規則に従います。ただし、EXEC SQL ステートメント初期化指定子を 2 行に分割しないでください。
- SQL ステートメントを含んだファイルの組み込みに COBOL COPY ステートメントは使用できません。 SQL ステートメントはモジュールがコンパイルされる前にプリコンパイルされます。プリコンパイラーは、COBOL COPY ステートメントを無視します。代わりに SQL INCLUDE ステートメントを使用して、組み込みファイルをインポートしてください。
- ストリング定数を次の行に継続するには、継続行の列 7 に '-' を、また列 12 またはそれ以降にストリング区切り文字を入れなければなりません。
- SQL 算術演算子はブランクで区切らなければなりません。
- 行末文字およびタブ文字などの空白文字の置換は、次のように行われる。
  - 引用符の外 (ただし、SQL ステートメント内) では、行末文字または TAB 文字は単一スペースと置き換えられる。
  - 引用符内では、COBOL プログラムに適合した形でストリングが継続されていれば、行末文字は消去されます。 TAB は修正されません。

  行末および TAB に使用される実際の文字は、プラットフォームごとに異なります。たとえば Windows® ベースのプラットフォームでは、復帰/改行を行末に使用するのに対し、UNIX および Linux ベースのシステムは改行のみを使用します。

REXX アプリケーションにおける組み込み SQL ステートメント

REXX アプリケーションは API を使用することにより、データベース・マネージャーAPI および SQL により提供される機能の大部分を使用できるようになります。 コンパイル言語で作成されたアプリケーションとは異なり、 REXX アプリケーションはプリコンパイルされません。その代わりに、動的 SQL ハンドラーがすべての SQL ステートメントを処理します。 REXX と呼び出し可能な API を組み

合わせることにより、データベース・マネージャー機能の大部分にアクセスできます。組み込み SQL を使用する API の中には REXX が直接サポートしていないものもありますが、DB2 コマンド行プロセッサーを使用すれば、 REXX アプリケーションからこれらの API にアクセスできます。

REXX はインタープリター言語なので、コンパイル済みホスト言語に比べてアプリケーション・プロトタイプの開発やデバッグが容易です。 REXX でコーディングされた データベース・アプリケーションは、コンパイル言語を使用した データベース・アプリケーションほどの性能は持ちませんが、プリコンパイル、コンパイル、リンクを行わず、またはさらに別のソフトウェアを使用せずに データベース・アプリケーションを作成する機能を提供できます。

SQL ステートメントの処理には、SQLEXEC ルーチンを使用してください。
SQLEXEC ルーチンの文字ストリング引数は以下のエレメントから成ります。
- SQL キーワード
- 事前定義 ID
- ステートメント・ホスト変数

有効な SQL ステートメントを SQLEXEC ルーチンに渡すことにより、それぞれの要素を要求します。次の構文を使用してください。

```
CALL SQLEXEC 'statement'
```

SQL ステートメントは複数行に渡って継続が可能です。ステートメントのそれぞれの部分は単一引用符で囲み、以下に示すように、追加ステートメントのテキストとの区切りとしてコンマを使用してください。

```
CALL SQLEXEC 'SQL text',
             'additional text',
                  .
                  .
                  .
             'final text'
```

以下に、REXX での組み込み SQL の例を示します。

```
statement = "UPDATE STAFF SET JOB = 'Clerk' WHERE JOB = 'Mgr'"
CALL SQLEXEC 'EXECUTE IMMEDIATE :statement'
IF ( SQLCA.SQLCODE < 0) THEN
   SAY 'Update Error:  SQLCODE = ' SQLCA.SQLCODE
```

この例では、更新が正常に行われたかどうかを判断するために、 SQLCA 構造体の SQLCODE フィールドがチェックされています。

REXX アプリケーションの場合、組み込み SQL ステートメントには、以下の規則が適用されます。

- 次の組み込み SQL ステートメントは、SQLEXEC ルーチンに直接渡すことができる。
  - CALL
  - CLOSE
  - COMMIT
  - CONNECT
  - CONNECT TO
  - CONNECT RESET
  - DECLARE

- DESCRIBE
- DISCONNECT
- EXECUTE
- EXECUTE IMMEDIATE
- FETCH
- FREE LOCATOR
- OPEN
- PREPARE
- RELEASE
- ROLLBACK
- SET CONNECTION

他の SQL ステートメントは、SQLEXEC ルーチンとともに EXECUTE IMMEDIATE ステートメント、または PREPARE と EXECUTE ステートメントを使用して動的に処理される。

- REXX では、CONNECT および SET CONNECTION ステートメントでホスト変数を使用することはできない。
- カーソル名およびステートメント名は、次のように事前定義される。

  c1 から c100 まで
  WITH HOLD オプションを使用せずに宣言した、範囲が *c1* から *c50* までのカーソルのカーソル名、および WITH HOLD オプションを使用して宣言した、範囲が *c51* から *c100* までのカーソルのカーソル名です。

  カーソル名の ID は、DECLARE、OPEN、FETCH、および CLOSE ステートメントに使用されます。これは、SQL 要求で使用されるカーソルを識別します。

  s1 から s100 まで
  範囲が *s1* から *s100* までのステートメント名です。

  ステートメント名の ID は、DECLARE、DESCRIBE、PREPARE、および EXECUTE ステートメントに使用されます。

  カーソル名およびステートメント名には、事前定義 ID を使用しなければならない。その他の名前は認められません。
- カーソルを宣言する際には、DECLARE ステートメント内のカーソル名とステートメント名が対応していなければならない。たとえば、*c1* をカーソル名として使用する場合、ステートメント名には *s1* を使用しなければなりません。
- SQL ステートメント内ではコメントを使用しない。

注: REXX ではマルチスレッド・データベース・アクセスはサポートされていません。

第 2 章 組み込み SQL アプリケーション用にサポートされている開発ソフトウェア

DB2 データベース・システムは、以下のオペレーティング・システムでの組み込み SQL アプリケーション用に、コンパイラー、インタープリター、および関連開発ソフトウェアをサポートします。

- AIX®
- HP-UX
- Linux
- Solaris
- Windows

組み込み SQL ソース・コードからは 32 ビットおよび 64 ビットの組み込み SQL アプリケーションをビルドできます。

以下のホスト言語は、組み込み SQL アプリケーションを開発するために特定のコンパイラーを必要とします。

- C
- C++
- COBOL
- Fortran
- REXX

第 3 章 組み込み SQL 開発環境のセットアップ

組み込み SQL アプリケーションのビルドを開始するには、その前に、アプリケーションの開発に使用するホスト言語用のサポートされているコンパイラーをインストールし、組み込み SQL 環境をセットアップする必要があります。

- サポートされているプラットフォーム上にインストールされた DB2 データ・サーバー
- DB2 クライアントがインストールされていること
- サポートされる組み込み SQL アプリケーション開発ソフトウェアがインストールされていること - 「*データベース・アプリケーション開発の基礎*」の『サポートされる組み込み SQL アプリケーション開発ソフトウェアがインストール済み』を参照してください。

PREP コマンドおよび BIND コマンドを発行するための権限をユーザーに割り当てます。

組み込み SQL アプリケーション開発環境が正しくセットアップされていることを確認するには、『C における組み込み SQL アプリケーション・テンプレート』のトピックにある組み込み SQL アプリケーション・テンプレートのビルドおよび実行を試行します。

第 4 章 組み込み SQL アプリケーションの設計

組み込み SQL アプリケーションを設計する際には、静的あるいは動的に実行される SQL ステートメントを使用する必要があります。静的 SQL ステートメントは 2 種類に分けることができます。1 つは、ホスト変数を含まないもの (主に初期設定や単純な SQL の例に使用される)、もう 1 つは、ホスト変数を使用するものです。動的 SQL もやはり 2 種類に分けることができます。1 つはパラメーター・マーカーを含まないもの (CLP などのインターフェースがその代表)、もう 1 つは、パラメーター・マーカーを含むもので、後者ではアプリケーションがより柔軟性に富んだものになります。

静的に実行されるステートメントと動的に実行されるステートメントのどちらを選ぶかは、組み込み SQL アプリケーションの移植性、パフォーマンスおよび制約事項など、多数の要因によって決まります。

組み込み SQL における許可に関する考慮事項

許可 を与えられたユーザーまたはグループは、データベースへの接続、表の作成、システムの管理などの一般的なタスクを実行できます。*特権* を付与されたユーザーやグループは、指定された方法で特定のデータベース・オブジェクトにアクセスできます。 DB2® は、保管された情報を保護するために特権のセットを使用します。

大部分の SQL ステートメントは、ステートメントが使用するデータベース・オブジェクトに対する何らかのタイプの特権を必要とします。ほとんどの API 呼び出しは、通常データベース・オブジェクトに対するいかなる特権も必要としませんが、権限を持っていないと呼び出すことができないものが多くあります。 DB2 API を使用すると、アプリケーション・プログラムから DB2 の管理機能を実行できます。たとえば、データベース内にバインド・ファイルの必要ないパッケージを作成するには、 `sqlarbnd` (または REBIND) API を使用できます。

グループは、それぞれのユーザーに個別に特権の付与または取り消しを行うことを必要とせずに、ユーザーの集合に対して許可を実行するための便利な手段を提供します。グループ・メンバーシップは、動的 SQL ステートメントの実行に関して考慮されますが、静的 SQL ステートメントに関しては考慮されません。ただし、`PUBLIC` 特権は静的 SQL ステートメントの実行に関して考慮されます。たとえば、`STAFF` という名前の表に対して静的にバインドされた SQL 照会を持つ、組み込み SQL ストアード・プロシージャーがあるとします。このプロシージャーを `CREATE PROCEDURE` ステートメントで作成しようとすると、開発者のアカウントが `STAFF` 表に対する選択特権を持つグループに属している場合には、 `CREATE` ステートメントは失敗して SQL0551N エラーが出されます。 `CREATE` ステートメントが機能するためには、開発者のアカウントが `STAFF` 表に対する選択特権を直接持っている必要があります。

アプリケーションを設計する際に、ユーザーがアプリケーションを実行するために必要な特権を考慮する必要があります。ユーザーが必要とする特権は、以下によって決まります。

- アプリケーションが動的 SQL (JDBC および DB2 CLI を含む) と静的 SQL のどちらを使用するか。 ステートメントを発行するために必要な特権については、それぞれのステートメントの説明を参照してください。
- アプリケーションがどの API を使用するか。 API 呼び出しに必要な特権と権限については、それぞれの API の説明を参照してください。

グループは、それぞれのユーザーに個別に特権の付与または取り消しを行うことを必要とせずに、ユーザーの集合に対して許可を実行するための便利な手段を提供します。一般的に、グループ・メンバーシップは動的 SQL ステートメントに関して考慮されますが、静的 SQL ステートメントに関しては考慮されません。この一般的なケースに対する例外は、特権が PUBLIC に対して付与されている場合です。この場合、これらの特権は静的 SQL ステートメントが実行されるときに考慮されます。

STAFF 表に対する照会を行う必要のある 2 名のユーザー、PAYROLL と BUDGET があるとします。 PAYROLL は会社の従業員の給与支払いを管理しており、給与支払い小切手を出す際に、さまざまな SELECT ステートメントを発行する必要があります。 PAYROLL は各従業員の給与にアクセスできなければなりません。 BUDGET は、給与の支払いにいくら必要であるかの決定を管理しています。しかし、BUDGET が個々の従業員の給与を見ることはできないようにしておかなければなりません。

PAYROLL がさまざまな SELECT ステートメントを発行しているため、 PAYROLL 用に設計されたアプリケーションは、動的 SQL を使用することになるでしょう。 動的 SQL では、 PAYROLL には STAFF 表に対する SELECT 特権が必要になります。 PAYROLL は表に対するフルアクセス特権が必要なので、このことは問題になりません。

一方 BUDGET は、個々の従業員の給与にアクセスさせてはなりません。つまり、 STAFF 表に対する SELECT 特権を BUDGET に付与してはならないということです。 BUDGET は STAFF 表全体の給与の合計にアクセスする必要があるので、 SELECT SUM (SALARY) FROM STAFF を実行するための静的 SQL アプリケーションを作成してバインドし、このアプリケーションのパッケージに対する EXECUTE 特権を BUDGET に付与することができます。このようにして、見てはならない情報を BUDGET に公開することなく、必要な情報を取得させることができるようになります。

組み込み SQL アプリケーションでの静的および動的 SQL ステートメントの実行

組み込み SQL アプリケーションでは、静的および動的 SQL ステートメント両方の実行がサポートされています。 SQL ステートメントを静的に実行するか動的に実行するかを判断するには、パッケージ、SQL ステートメントが実行時に実行される方法、ホスト変数、パラメーター・マーカーなどについての理解、そしてこれらのことがどのようにアプリケーションのパフォーマンスに関係するかということについての理解も必要になります。

組み込み SQL プログラムにおける静的 SQL

C で書かれた静的に実行されるステートメントの例を以下に示します。

```
/* select values from table into host variables using STATIC SQL and print them*/
EXEC SQL SELECT id, name, dept, salary INTO :id, :name, :dept, :salary
           FROM staff WHERE id = 310;
printf("
```

組み込み SQL プログラムにおける動的 SQL

C で書かれた動的に実行されるステートメントの例を以下に示します。

```
/* Update column in table using DYNAMIC SQL*/
strcpy(hostVarStmtDyn, "UPDATE staff SET salary = salary + 1000  WHERE dept = ?");
EXEC SQL PREPARE StmtDyn FROM :hostVarStmtDyn;
EXEC SQL EXECUTE StmtDyn USING :dept;
```

組み込み SQL 動的ステートメント

動的 SQL ステートメントは、文字ストリング・ホスト変数とステートメント名を引数として受け入れます。ホスト変数には、テキスト形式で動的に処理される SQL ステートメントが入っています。ステートメント・テキストは、アプリケーションのプリコンパイル時には処理されません。実際、ステートメント・テキストは、アプリケーションのプリコンパイル時には必要ありません。その代わりに、SQL ステートメントはプリコンパイルの段階でホスト変数と見なされ、その変数はアプリケーションを実行するときに参照されます。

動的 SQL サポート・ステートメントは、SQL テキストが入っているホスト変数を実行可能形式に変換するために必要です。また動的 SQL サポート・ステートメントは、ステートメント名を参照することによってホスト変数に対して機能します。それらのサポート・ステートメントは以下のとおりです。

EXECUTE IMMEDIATE

ホスト変数を使用しないステートメントを準備し実行します。このステートメントは PREPARE および EXECUTE ステートメントの代用として使います。

例として、以下のような C のステートメントについて考えてみましょう。

```
strcpy (qstring,"INSERT INTO WORK_TABLE SELECT *
   FROM EMP_ACT WHERE ACTNO >= 100");
EXEC SQL  EXECUTE IMMEDIATE :qstring;
```

PREPARE

SQL ステートメントの文字ストリング式をステートメントの実行可能書式に変換し、ステートメント名を割り当て、オプションでそのステートメントに関する情報を SQLDA 構造体に入れます。

EXECUTE

前に準備した SQL ステートメントを実行します。このステートメントは、接続内で繰り返し実行することができます。

DESCRIBE

準備済みステートメントに関する情報を SQLDA に入れます。

例として、以下のような C のステートメントについて考えてみましょう。

```
strcpy(hostVarStmt, "DELETE FROM org WHERE deptnumb = 15");
EXEC SQL PREPARE Stmt FROM :hostVarStmt;
EXEC SQL DESCRIBE Stmt INTO :sqlda;
EXEC SQL EXECUTE Stmt;
```

注: 動的 SQL ステートメントの内容は、静的 SQL ステートメントの場合と同じ構文に従っていますが、以下の点が異なります。

- ステートメントの先頭に EXEC SQL を使用してはならない。
- ステートメントをステートメント終止符で終了してはならない。これに対する例外は CREATE TRIGGER ステートメントで、このステートメントの場合はセミコロン (;) を入れることができます。

組み込み SQL アプリケーションで SQL ステートメントを静的または動的に実行する時点の判別

組み込み SQL アプリケーションで SQL ステートメントを静的に実行するかそれとも動的に実行するかを判別する場合、その前に考慮しなければならない複数の考慮事項があります。考慮する必要のある主要な考慮事項、および静的 SQL と動的 SQL のどちら使用するべきか、あるいはどんな場合にはどちらを使用しても同じように良い結果になるかに関係した推奨事項を、以下の表に示します。

注: この情報はあくまでも一般的な提案に過ぎません。どちらを選ぶにしても、ご使用のアプリケーションの要件、本来の用途、および作業環境を考慮に入れる必要があります。はっきりしない場合は、まずステートメントを静的 SQL としてプロトタイプ化してから、次に動的 SQL としてプロトタイプ化し、その違いを比較することが最善の方法です。

表 1. 静的 SQL と動的 SQL の比較

考慮事項	推奨 SQL
望ましい照会実行時間:	
• 2 秒未満	• 静的
• 2 秒から 10 秒まで	• 両方
• 10 秒を超える	• 動的
SQL ステートメントによって照会または操作されるデータの均一性	
• データ分散が均一	• 静的
• やや不均一	• 両方
• 非常に不均一な分散	• 動的
照会中の範囲述部の数量	
• ごく少数	• 静的
• 多少	• 両方
• 多数	• 動的
SQL ステートメントの実行が繰り返される可能性	
• 何回も実行 (10 回以上)	• 両方
• 少数回実行 (10 回未満)	• 両方
• 1 回だけ実行	• 静的
照会の種類	
• ランダム	• 動的
• 永久的	• 両方

表 1. 静的 SQL と動的 SQL の比較 (続き)

考慮事項	推奨 SQL
SQL ステートメントのタイプ (DML/DDL/DCL)	
• トランザクション処理 (DML のみ)	• 両方
• 混合 (DML および DDL - DDL はパッケージに影響を及ぼす)	• 動的
• 混合 (DML および DDL - DDL はパッケージに影響を及ぼさない)	• 両方
RUNSTATS コマンドが発行される頻度	
• 非常にまれ	• 静的
• 普通	• 両方
• 頻繁	• 動的

SQL ステートメントは、常に実行される前にコンパイルされます。一方動的 SQL ステートメントは実行時にコンパイルされるので、静的 SQL の場合の単一の初期コンパイル段階と比較した場合に、アプリケーション実行時に各動的ステートメントをコンパイルするために発生するオーバーヘッドのために、アプリケーションは多少低速になります。

混合環境では、静的 SQL と動的 SQL との間の選択はパッケージが無効にされる頻度にも影響してきます。 DDL によってパッケージが無効にされる場合は、動的 SQL のほうが便利です。実際に実行する照会だけが次回の使用時に再コンパイルされ、その他の照会は再コンパイルされないからです。静的 SQL の場合は、いったん無効にされるとパッケージ全体が再バインドされます。

静的 SQL または動的 SQL のどちらを使用するかは大きな問題ではない場合もあります。例えば、アプリケーションに含まれるほとんどの参照が動的に実行される SQL ステートメントに対するものである場合に、静的 SQL として実行される方が適切なステートメントが 1 つあるとします。そのような場合、コーディングに一貫性を持たせるため、単にその 1 つのステートメントも動的に実行するのがよいかもしれません。前の表の考慮事項は、重要性の高い順におおまかに掲載してあることに注意してください。

SQL ステートメントの静的 SQL は、動的に処理される同じステートメントよりも自動的に速く実行されるとは考えないでください。動的ステートメントの準備にはオーバーヘッドが必要なため、静的 SQL の方が速く実行されます。それ以外の場合は、オプティマイザーがバインド時以前に使用可能なデータベース統計ではなく、現行のデータベース統計を使用できるので、動的に作成された同じステートメントのほうが速く実行されます。トランザクションの完了にかかる時間が 2 秒未満である場合は、一般に静的 SQL の方が速くなります。使用する方式を選択するために、両方のバインド方式をプロトタイプ化してください。

注: 静的および動的 SQL はそれぞれ、ホスト変数を利用するステートメントと利用しないステートメントの 2 種類に分けられます。それらは以下のとおりです。

1. ホスト変数を含まない静的 SQL ステートメント

   これは、以下の場合にしか見られない、まれな状態です。

   - 初期化コード
   - 非常に単純な SQL ステートメント

ホスト変数のない単純な SQL ステートメントは、実行時パフォーマンスのオーバーヘッドがなく、しかも DB2 オプティマイザーの機能を十分に活用しているため、パフォーマンスの観点からは適切に実行します。

2. ホスト変数を含む静的 SQL

   ホスト変数を利用する静的 SQL ステートメントは、従来型のスタイルによる DB2 アプリケーションと見なされます。静的 SQL ステートメントでは、ステートメントのコンパイル中に獲得する PREPARE およびカタログ・ロックの実行時オーバーヘッドがなくなります。オプティマイザーは SQL ステートメント全体を認識しないため、残念ながら、オプティマイザーの全性能を活用することはできません。高度に均一化されていないデータ分散の場合には、特殊な問題があります。

3. パラメーター・マーカーを含まない動的 SQL

   これは、随時照会の実行に頻繁に使用される CLP などの標準的なインターフェースです。 CLP からは、SQL ステートメントは動的にのみ実行できます。

4. パラメーター・マーカーを含む動的 SQL

   動的 SQL ステートメントの主な利点は、パラメーター・マーカーがあるために、ステートメント (多くの場合、選択または挿入) を繰り返し実行するうちに、ステートメントを準備するコストを償却できるという点です。この償却は、すべての反復性のある動的 SQL アプリケーションに当てはまります。残念なことに、ホスト変数を含む静的 SQL と同様に、 DB2 オプティマイザーの一部は、情報のすべてを利用することができないために作動しません。

   最も効率的な方法としては、ホスト変数を指定して静的 SQL を使用することまたはパラメーター・マーカーなしで動的 SQL を使用することをお勧めします。

組み込み SQL アプリケーションのパフォーマンス

パフォーマンスは、データベース・アプリケーションを開発する際に考慮すべき重要な要素です。組み込み SQL アプリケーションのパフォーマンスは高いですが、それは、静的 SQL ステートメントの実行、そして静的および動的 SQL ステートメントの混合実行をサポートしているからです。静的 SQL ステートメントのコンパイルの方法によっては、組み込み SQL アプリケーションが確実に、長期的に継続して高いパフォーマンスを発揮できるよう、開発者またはデータベース管理者が実行しなければならないいくつかのステップがあります。

組み込み SQL アプリケーションのパフォーマンスには、以下の要因が影響を与えます。

- 時間の経過に伴うデータベース・スキーマの変化
- 時間の経過に伴う表のカーディナリティー (表中の行の数) の変化
- SQL ステートメントにバインドされたホスト変数の値の変化

パッケージはデータベースがある特定の特性を持つある一時点で作成されるので、これらの要因は組み込み SQL アプリケーションのパフォーマンスに影響を与えます。その特性は、データベース・マネージャーが SQL ステートメントをどのように実行すれば最も効率的かを定義する、パッケージ実行時のアクセス・プランの作成にあたって考慮に入れられます。時間の経過に伴い、データベース・スキーマお

よびデータの変化により、実行時のアクセス・プランのレンダリングが最適でないものになる場合があります。これが、アプリケーションのパフォーマンスの低下につながる可能性があるのです。

このことから、使用される情報を定期的にリフレッシュし、パッケージの実行時アクセス・プランをよく管理された状態に保つことは重要です。

表や索引についての最新の統計を収集するには、RUNSTATS コマンドを使用します。特に、前回 RUNSTATS コマンドが実行されてから大幅な改訂作業が行われたときや、新たな索引が作成されたときに、このコマンドを実行します。これにより、オプティマイザーに、最適なアクセス・プランの決定に使用できる、最も正確な情報が提供されます。

組み込み SQL アプリケーションのパフォーマンスはいくつかの方法で改善できます。

- RUNSTATS コマンドを実行してデータベース統計を更新する。
- アプリケーション・パッケージをデータベースに再バインドし、データベースがディスク上のデータを物理的に取り出すのに使用する実行時アクセス・プランを (更新された統計に基づいて) 再生成する。
- 静的および動的プログラムの中で REOPT バインド・オプションを使用する。

組み込み SQL アプリケーションの 32 ビットと 64 ビットのサポート

組み込み SQL アプリケーションのビルドは、32 ビット・プラットフォームと 64 ビット・プラットフォームのどちらでも行えます。ただし、ビルドと実行に関する考慮事項はそれぞれ異なります。ビルド・スクリプトには、ビット幅を判別するためのチェックが含まれます。検出されるビット幅が 64 ビットの場合、必要な変更に対応するための追加のスイッチのセットが設定されます。

DB2 データベース・システムは、以下にリストされる 32 ビットおよび 64 ビット・バージョンのオペレーティング・システムでサポートされます。多くの場合、これらのオペレーティング・システム上の 32 ビットおよび 64 ビット環境での組み込み SQL アプリケーションのビルドはそれぞれ異なります。

- AIX
- HP-UX
- Linux
- Solaris
- Windows

DB2 バージョン 9 でサポートされる 32 ビット・インスタンスは、以下のインスタンスのみです。

- x86 版 Linux
- x86 版 Windows
- X64 版 Windows (x86 版 DB2 for Windows インストール・イメージを使用する場合)

DB2 バージョン 9 でサポートされる 64 ビット・インスタンスは、以下のインスタンスのみです。

- AIX
- Sun
- HP PA-RISC
- HP IPF
- x86_64 版 Linux
- POWER 版 Linux
- zSeries 版 Linux
- X64 版 Windows (Windows for X64 インストール・イメージを使用する場合)
- IPF 版 Windows
- IPF 版 Linux

Linux IA64 および Linux zSeries® を除き、DB2 データベース・システムは、サポートされているすべての 64 ビット・オペレーティング・システム環境での 32 ビット・アプリケーションおよびルーチンの実行をサポートします。

ホスト言語の場合はそれぞれ、32 ビット・プラットフォームまたは 64 ビット・プラットフォームのいずれかあるいはその両方で、使用されるホスト変数が望ましい場合があります。各プログラミング言語ごとにそれぞれ、さまざまなデータ・タイプを調べてください。

組み込み SQL アプリケーションに対する制約事項

サポートされている各ホスト言語には、それぞれ独自の一連の制限事項および仕様があります。 C/C++ は、3 文字表記という一連の 3 つの文字を使用して、特定の特殊文字の表示に対する制限事項に対処しています。 COBOL では、オブジェクト指向の COBOL アプリケーションの使用時に役に立つ一連のルールが定められています。 FORTRAN には、プリコンパイル・プロセスに影響を与える可能性のある対象領域があるのに対して、REXX の場合は、言語サポートなどの特定の領域に制限されます。

C および C++ を使用して組み込み SQL アプリケーションをプログラミングする際の文字セットに関する制約事項

C または C++ 文字セットの文字の中には、すべてのキーボードで使用できないものもあります。これらの文字は、*3 文字表記* と呼ばれる一続きの 3 つの文字を使用して C または C++ のソース・プログラムに入力することができます。3 文字表記は SQL ステートメントでは認識されません。プリコンパイラーは、ホスト変数宣言内で以下の 3 文字表記を認識します。

3 文字表記
定義

??(左大括弧 '['

??) 右大括弧 ']'

??< 左中括弧 '{'

??> 右中括弧 '}'

以下に示すその他の 3 文字表記は、C または C++ ソース・プログラムの別の場所で使用されることがあります。

3 文字表記
定義

??= ハッシュ・マーク '#'

??/ 円記号 '¥'

??' 脱字記号 '^'

??! 縦棒 '|'

??– 波形記号 '~'

COBOL を使用した組み込み SQL アプリケーションのプログラムに関する制約事項

COBOL アプリケーションでの API 呼び出しには以下のような制約事項があります。

- API 呼び出しで値パラメーターとして使用する整数の変数はすべて、 USAGE COMP-5 節で宣言しなければなりません。

オブジェクト指向 COBOL プログラムの場合:

- SQL ステートメントは、1 コンパイル単位内の最初のプログラムまたはクラスだけに使用できます。この制約事項は、プリコンパイラーが参照する最初の 作業用ストレージ・セクションに一時作業データを挿入するためのものです。
- SQL ステートメントが含まれるすべてのクラスには、クラス・レベルの作業用ストレージ・セクションがなければなりません。これは、このセクションが空である場合にも当てはまります。このセクションを使用して、プリコンパイラーが生成したデータ定義を保管します。

FORTRAN を使用した組み込み SQL アプリケーションのプログラミングに関する制約事項

FORTRAN に対する組み込み SQL のサポートは DB2 Universal Database バージョン 5 で確立され、今後拡張される予定はありません。たとえば、FORTRAN プリコンパイラーは、表名などの SQL オブジェクト ID を処理できません。これは 18 バイトより長いからです。 長さが 19 バイトから 128 バイトまでの表名などの、UDB バージョン 5 より後の DB2 データベース・システムに追加された機能を使用する場合には、FORTRAN 以外の言語でアプリケーションを作成する必要があります。

FORTRAN データベース・アプリケーション開発は、Windows または Linux 環境の DB2 インスタンスではサポートされていません。

FORTRAN はマルチスレッド・データベース・アクセスをサポートしていません。

一部の FORTRAN コンパイラーは、 1 列目が 'D' または 'd' である行を条件行と見なします。これらの行は、デバッグのためにコンパイルすることも、コメントとして取り扱うことも可能です。プリコンパイラーは、1 列目が 'D' または 'd' である行を常にコメントと見なします。

API パラメーターの中には、呼び出し変数に値ではなくアドレスを必要とするものもあります。データベース・マネージャーは、それらのパラメーターの指定を単純化する GET ADDRESS、DEREFERENCE ADDRESS、および COPY MEMORY API を提供します。

以下の項目は、プリコンパイル処理に影響を及ぼします。

- プリコンパイラーは、継続行の 1 列目から 5 列目までには数字、ブランク、およびタブ文字しか認めない。
- `.sqf` ソース・ファイルでは、ホレリス定数はサポートされない。

REXX を使用した組み込み SQL アプリケーションのプログラミングに関する制約事項

REXX アプリケーションの組み込み SQL に関する制約事項は以下のとおりです。

- REXX に対する組み込み SQL のサポートは DB2 Universal Database バージョン 5 で確立され、今後拡張される予定はありません。たとえば、REXX は、表名などの SQL オブジェクト ID を処理できません。これは 18 バイトより長いからです。 19 から 128 バイト長の表名など、バージョン 5 より後の DB2 データベース・システムに追加された機能を使用する場合には、REXX 以外の言語でアプリケーションを作成する必要があります。
- REXX/SQL では、コンパウンド SQL はサポートされません。
- REXX では静的 SQL はサポートされていません。
- REXX アプリケーションは、日本語や中国語 (繁体字) の EUC 環境ではサポートされません。

XML および XQuery を使用した組み込み SQL アプリケーションの開発に関する推奨事項

組み込み SQL アプリケーションで XML および XQuery を使用するにあたっては、以下の推奨事項および制約事項が適用されます。

- アプリケーションは、直列化されたストリング・フォーマットですべての XML データにアクセスする必要がある。
  - 数値、日付時間データなどを含め、すべてのデータを、直列化されたストリング・フォーマットで表現する必要がある。
- 外部化される XML データは 2 GB に制限される。
- XML データを含むカーソルはすべて非ブロッキングになる (取り出し操作のたびにデータベース・サーバー要求が出される)。
- 文字ホスト変数に直列化された XML データが含まれる場合は常に、アプリケーション・コード・ページがデータのエンコード方式として使用されるとみなされるため、それが、データ中に存在する内部エンコード方式と一致しなくてはならない。

- XML ホスト変数の基本タイプとしては、LOB データ・タイプを指定しなくてはならない。
- 静的 SQL には以下が適用されます。
  - 文字およびバイナリー・ホスト変数は、SELECT INTO 操作による XML 値の取得には使用できない。
  - XML データ・タイプの入力が予期されているところで CHAR、VARCHAR、CLOB、BLOB ホスト変数を使用すると、それらは、デフォルトの空白処理特性 ('STRIP WHITESPACE') を伴う XMLPARSE 操作の対象になります。他の非 XML ホスト変数・タイプはすべて拒否されます。
  - 静的 XQuery 式のサポートはない。XQuery 式をプリコンパイルしようとすると、エラーが発生して失敗します。XQuery 式は、XMLQUERY 関数によってのみ実行できます。
- XQuery 式は、ストリング「XQUERY」を式の前に付けることで、動的に実行できる。

組み込み SQL アプリケーションにおける並行トランザクションおよびマルチスレッド・データベース・アクセス

いくつかのオペレーティング・システムに共通する特徴は、1 つのプロセスで実行プログラムの複数のスレッドを実行できることです。複数のスレッドにより、アプリケーションが非同期のイベントを処理することができ、ポーリング機能がなくても容易にイベント・ドリブン・アプリケーションを作成できます。以下の情報では、DB2 データベース・マネージャーが複数のスレッドを処理する方法を解説し、留意すべき設計の指針をいくつかリストします。

マルチスレッドのアプリケーション開発に関する用語 (クリティカル・セクションおよびセマフォーなど) について詳しくない方は、ご使用のオペレーティング・システムのプログラミングに関する資料を調べてください。

DB2 組み込み SQL アプリケーションは、コンテキスト を使用して複数のスレッドから SQL ステートメントを実行することができます。コンテキストとは、アプリケーションがすべての SQL ステートメントおよび API 呼び出しを実行する環境のことです。すべての接続、作業単位、および他のデータベース・リソースは、特定のコンテキストに関連付けられています。各コンテキストは、アプリケーション内の 1 つ以上のスレッドに関連付けられています。スレッド・セーフ・コードでのマルチスレッド組み込み SQL アプリケーションの開発は、C および C++ でのみサポートされています。言語によって提供されている機能に加え、並行マルチスレッド・データベース・アクセスも可能にするプリコンパイラーを、ユーザーが独自に作成することもできます。

各実行可能 SQL ステートメントでは、最初のランタイム・サービス呼び出しは常にラッチを取得しようとします。成功すると処理を続行しますが、 (他のスレッドの SQL ステートメントがすでにラッチを取得しているために) 失敗すると、呼び出しは信号セマフォーでこれがポストされるまでブロックされ、それからラッチを取得し処理を続行します。ラッチは SQL ステートメントが処理を終了するまで保持され、その SQL ステートメントに対して生成された最後のランタイム・サービス呼び出しにより解放されます。

最終的な結果として、他のスレッドが SQL ステートメントを同時に実行しようとしても各 SQL ステートメントはアトミック単位で実行されます。これにより内部データ構造は、異なるスレッドによって同時に変更されることがなくなります。API もランタイム・サービスを使用したラッチを使用します。したがって、API には、各コンテキスト内のランタイム・サービス・ルーチンと同じ制限が課されます。

コンテキストはプロセス内のスレッド間で交換できますが、プロセス間では交換できません。複数のコンテキストの使用方法の 1 つは、並行トランザクションのサポートです。

DB2 データベースに対するスレッド化アプリケーションのデフォルト・インプリメンテーションでは、データベースへのアクセスのシリアライゼーションは、データベース API によって実施されます。 1 つのスレッドがデータベース呼び出しを実行する場合、他のスレッドによる呼び出しは、後続の呼び出しが、最初の呼び出しとは関係のないデータベース・オブジェクトへのアクセスを行うものでも、最初の呼び出しが完了するまでブロックされます。また、1 つのプロセス内のスレッドはすべて同じコミット有効範囲を共有します。データベースへの真の並行アクセスは、個別のプロセスを使用するか、このトピックで解説する API を使用した場合にのみ実現されます。

DB2 データベース・システムは、データベース API および 組み込み SQL を使用するための分離環境 (コンテキスト) の割り振りと操作に使用できる API を提供します。個々のコンテキストは別個のエンティティーで、1 つのコンテキストを使用するあらゆる接続あるいはアタッチメントは、他のすべてのコンテキストから (つまり、1 つのプロセス内の他のすべての接続あるいはアタッチメントからも) 独立しています。処理があるコンテキストで実行されるようにするには、まずスレッドに関連付ける必要があります。スレッドは、データベース API 呼び出しを行う際、あるいは組み込み SQL を使用する際に、必ずコンテキストを持っていなければなりません。

DB2 データベース・システム・アプリケーションはすべて、デフォルトでマルチスレッドであり、複数のコンテキストを使用できます。以下の DB2 API を使用して、複数のコンテキストを使用することができます。具体的に言えば、アプリケーションはスレッド用のコンテキストを作成でき、各スレッド用の別個のコンテキストにアタッチ、またはデタッチができ、スレッド間でコンテキストを渡すことができます。アプリケーションがこれらの API のいずれも 呼び出さない場合は、 DB2 が自動的にアプリケーション用に複数コンテキストを管理します。

- `sqleAttachToCtx` - コンテキストへのアタッチ
- `sqleBeginCtx` - アプリケーション・コンテキストの作成およびアプリケーション・コンテキストへのアタッチ
- `sqleDetachFromCtx` - コンテキストからのデタッチ
- `sqleEndCtx` - アプリケーション・コンテキストからのデタッチおよびアプリケーション・コンテキストの破棄
- `sqleGetCurrentCtx` - 現在のコンテキストの取得
- `sqleInterruptCtx` - コンテキストの中断

これらの API はアプリケーションのスレッド化をサポートしていないプラットフォームでは何の効力も持ちません (つまり、何の操作も行いません)。

コンテキストは、接続またはアタッチメントが継続している間、ずっと特定のスレッドに関連付けられている必要はありません。 1 つのスレッドが 1 つのコンテキストにアタッチし、1 つのデータベースに接続した後、そのコンテキストからデタッチすれば、2 番目のスレッドは同じコンテキストにアタッチできるようになり、すでに存在するデータベース接続を利用して処理を続行できます。コンテキストは、同一プロセス内の複数のスレッド間で受け渡しができますが、複数プロセス間での受け渡しはできません。

たとえ新たな API が使用されても、以下の API は引き続きシリアライズされます。

- `sqlabndx` - バインド
- `sqlaprep` - プログラムのプリコンパイル
- `sqluexpr` - エクスポート
- `db2Import` および `sqluimpr` - インポート

注:

1. DB2 CLI は、マルチスレッド化をサポートしたプラットフォームでは自動的に複数のコンテキストを使用して、スレッド・セーフな並行データベース・アクセスを行います。 DB2 で推奨されてはいませんが、ユーザーは、必要に応じて明示的にこの機能を使用不可にできます。
2. デフォルトでは、AIX は、32 ビットのアプリケーションがプロセスごとに 11 を超える共有メモリー・セグメントへのアタッチを許可しません。DB2 接続に使用できるのは、そのうち最大で 10 までです。

   この制限に到達すると、DB2 は、SQL CONNECT について SQLCODE -1224 を戻します。DB2 Connect™ にも、ローカル・ユーザーが TP モニター (TCP/IP) によって 2 フェーズ・コミットを実行する場合に、同じく 10 接続の制限があります。

   プロセスがアタッチできる共有メモリー・セグメントの数の最大値を増やすには、AIX 環境変数 `EXTSHM` を使用できます。

   `EXTSHM` を DB2 で使用するには、次のようにします。

   クライアント・セッションで:

   ```
   export EXTSHM=ON
   ```

   DB2 サーバーを開始する際:

   ```
   export EXTSHM=ON
   db2set DB2ENVLIST=EXTSHM
   db2start
   ```

   DPF 上では、userprofile あるいは usercshrc ファイルに以下の行を追加することも必要です。

   ```
   EXTSHM=ON
   export EXTSHM
   ```

他には、ローカル・データベースまたは DB2 Connect を別のマシンに移動してそれにリモートでアクセスする方法や、ローカル・データベースまたは DB2 Connect データベースをローカル・マシンの TCP/IP アドレスを持つリモート・ノードとしてカタログすることにより、TCP/IP ループバックによってそれにアクセスするという方法もあります。

マルチスレッドの使用に際しての推奨事項

マルチスレッドのアプリケーションからデータベースをアクセスする際には、これらの指針に従ってください。

連続したデータ構造体の変更。

アプリケーションは、SQL ステートメントまたはデータベース・マネージャー・ルーチンが、あるスレッドで処理されている間に、SQL ステートメントおよびデータベース・マネージャー・ルーチンが使用するユーザー定義のデータ構造体が、別のスレッドによって変更されていないことを確認する必要があります。 たとえば、他のスレッドの SQL ステートメントが SQLDA を使用している場合は、スレッドが SQLDA を再び割り振ることができないようにしてください。

個別のデータ構造体の使用を考える。

繰り返しを避けるため、各スレッドにそれぞれのユーザー定義のデータ構造体を渡す方が容易かもしれません。このガイドラインは特に、SQLCA の場合にそういえます。SQLCA は個々の実行可能 SQL ステートメントばかりでなく、すべてのデータベース・マネージャー・ルーチンによって使用されるからです。 SQLCA に関連したこの問題を避けるための代替手段が 3 つあります。

- 最初のスレッド以外のスレッドが使用するルーチンにはすべて、その先頭に `struct sqlca sqlca` を追加して、 `EXEC SQL INCLUDE SQLCA` を使用する。
- `EXEC SQL INCLUDE SQLCA` を、グローバル有効範囲内に置くのではなく、SQL を含む各ルーチンの内部に置く。
- `EXEC SQL INCLUDE SQLCA` を `#include "sqlca.h"` に置き換え、 SQL を使用するルーチンの先頭に `"struct sqlca sqlca"` を追加する。

マルチスレッド UNIX アプリケーションのコード・ページおよび国別あるいは地域別コードについての考慮事項

このセクションは、C および C++ 組み込み SQL アプリケーションにのみ関係があります。

AIX、Solaris、および HP-UX では、データベース接続で使用されるコード・ページおよび国別あるいは地域別コードを実行時に照会するために使用される関数が現在、スレッド・セーフになっています。しかし、これらの関数は多数の並行データベース接続を使用するマルチスレッド・アプリケーションでは、何らかのロック競合 (およびその結果、パフォーマンスの低下) を起こす可能性があります。

DB2_FORCE_NLS_CACHE 環境変数を使用して、マルチスレッド・アプリケーションでのロック競合の可能性を除去できます。 *DB2_FORCE_NLS_CACHE* が `TRUE` に設定されると、コード・ページおよび国別または地域別コード情報は、スレッドが

最初にアクセスする際に保管されます。その時点から、キャッシュされた情報は、この情報を要求する他のすべてのスレッドで使用されます。この情報を保管するとロック競合は削減され、状況によってはパフォーマンスが向上します。

接続間のロケール設定をアプリケーションが変更する場合には、DB2_FORCE_NLS_CACHE を TRUE に設定するべきではありません。この状態になると、ロケール設定が変更されても、元のロケール情報が戻されます。一般的には、マルチスレッド・アプリケーションはロケール設定を変更しないので、変更しないままにしておけばアプリケーションはスレッド・セーフになります。

マルチスレッド組み込み SQL アプリケーションのトラブルシューティング

以下のセクションでは、マルチスレッド組み込み SQL アプリケーションで発生する問題、およびその回避方法について説明します。

複数のスレッドを使用するアプリケーションは、単一スレッドを使用するアプリケーションよりも複雑です。この余分の複雑さにより、予期しない問題がいくつか生じる可能性が潜んでいます。マルチスレッドのアプリケーションを作成するときには、次の事柄に注意を払ってください。

2 つ以上のコンテキスト間でのデータベースの従属関係

アプリケーション内の各コンテキストには、データベース・オブジェクトに対するロックなど、それぞれ固有のデータベース・リソースがあります。この特性のため、2 つのコンテキストが同じデータベース・オブジェクトにアクセスしている場合、デッドロックを引き起こす可能性があります。データベース・マネージャーはデッドロックを検出し、一方のコンテキストが SQLCODE -911 を受け取ると、その作業単位がロールバックされます。

2 つ以上のコンテキスト間におけるアプリケーションの従属関係

コンテキスト間の従属関係を確立するプログラミング技法に注意してください。ラッチ、セマフォー、およびクリティカル・セクションは、そのような従属関係を確立するプログラミング技法の例です。アプリケーションに 2 つのコンテキストがあり、そのコンテキスト間にはアプリケーションの従属関係とデータベースの従属関係のどちらもが存在する場合、アプリケーションがデッドロックとなる可能性があります。従属関係のあるものがデータベース・マネージャーの管理範囲外にある場合、デッドロックが検出されないので、アプリケーションは中断またはハングします。

複数コンテキストでのデッドロック防止

データベース・マネージャーはスレッド間のデッドロックを検出できないため、デッドロックしないように (または少なくともデッドロックを回避できるように) アプリケーションを設計してコーディングしなければなりません。

データベース・マネージャーが検出しないデッドロックの例として、2 つのコンテキストがあり、そのどちらも共通のデータ構造体にアクセスするアプリケーションを考えてみましょう。両方のコンテキストがそのデータ構造体を同時に変更するという問題を避けるため、データ構造体はセマフォーによって保護されます。コンテキストは次のようになります。

```
context 1
SELECT * FROM TAB1 FOR UPDATE....
UPDATE TAB1 SET....
get semaphore
access data structure
release semaphore
COMMIT

context 2
get semaphore
access data structure
SELECT * FROM TAB1...
release semaphore
COMMIT
```

最初のコンテキストが SELECT および UPDATE ステートメントを正常に実行しているときに、 2 番目のコンテキストがセマフォーを獲得してデータ構造体にアクセスするとします。最初のコンテキストがセマフォーを獲得しようとしますが、2 番目のコンテキストがセマフォーを保持しているため、獲得できません。ここで 2 番目のコンテキストは表 TAB1 から行を読み取ろうとしますが、最初のコンテキストが保持するデータベース・ロックによってその操作が停止してしまいます。アプリケーションは、コンテキスト 1 がコンテキスト 2 の前に完了できず、コンテキスト 2 がコンテキスト 1 の完了を待っている状態になります。アプリケーションはデッドロックしますが、データベース・マネージャーはセマフォーの従属関係を知らないため、コンテキストはロールバックされません。未解決の従属関係のため、アプリケーションは延期状態になってしまいます。

上記の例で起こるデッドロックを回避する方法はいくつかあります。

- セマフォーを獲得する前に保持していたロックをすべて解除する。

  コンテキスト 1 のコードを変更して、セマフォーを獲得する前にコミットを実行するようにします。
- セマフォーによって保護されたセクションの内部に、SQL ステートメントをコーディングしない。

  コンテキスト 2 のコードを変更して、SELECT を実行する前にセマフォーを解除するようにします。
- すべての SQL ステートメントをセマフォー内部にコーディングする。

  コンテキスト 1 のコードを変更して、SELECT ステートメントを実行する前にセマフォーを獲得するようにします。この技法は、動作はしますが、あまりお勧めできません。というのは、セマフォーはデータベース・マネージャーへのアクセスをシリアライズするため、複数のスレッドを使用する効果が発揮されないからです。
- *locktimeout* データベース構成パラメーターを -1 以外の値に設定する。

  -1 以外の値ではデッドロックを防ぐことはできませんが、実行を再開することはできます。コンテキスト 2 は、要求されたロックを獲得できないため、結局はロールバックされます。ロールバックのエラーを処理するときには、コンテキスト 2 はセマフォーを解除しなければなりません。セマフォーを解除すると、コンテキスト 1 が継続でき、コンテキスト 2 が解放されて作動を再試行します。

デッドロックを回避する技法を例を参考にして説明しましたが、この方法はすべてのマルチスレッド・アプリケーションに適用できます。一般に、保護リソースを扱うようにデータベース・マネージャーを扱うならば、マルチスレッド・アプリケーションで問題が生じることはありません。

第 5 章 組み込み SQL アプリケーションのプログラミング

組み込み SQL アプリケーションのプログラミングには、選択された組み込み SQL プログラミング言語でのアプリケーションのアセンブルに必要なあらゆるステップが含まれます。組み込み SQL が自分のプログラミングのニーズに適した API であると判断してから、目的の組み込み SQL アプリケーションの設計を行った後は、組み込み SQL アプリケーションのプログラミングを行うことができます。

前提条件:

- 静的あるいは動的 SQL ステートメントのどちらを使用するかを選択する
- 組み込み SQL アプリケーションの設計

組み込み SQL アプリケーションのプログラミングは以下のサブタスクから構成されます。

- 必要なヘッダー・ファイルの組み込み
- サポートされる組み込み SQL プログラミング言語の選択
- SQL ステートメントに組み込む代表値のためのホスト変数の宣言
- データ・ソースへの接続
- SQL ステートメントの実行
- SQL ステートメントの実行に関連する SQL エラーおよび警告の処理
- データ・ソースからの切断

組み込み SQL アプリケーションが完成したなら、アプリケーションをコンパイルして実行する準備ができたことになります。組み込み SQL アプリケーションの構築。

組み込み SQL のソース・ファイル

組み込み SQL を含むソース・コードを作成する際には、サポートされる各ホスト言語の、特定のファイル命名規則に従う必要があります。

C および C++ の入力および出力ファイル

デフォルトでは、ソース・アプリケーションに以下のような拡張子を付けることができます。

.sqc サポートされているすべてのオペレーティング・システム上の C ファイル

.sqC UNIX および Linux オペレーティング・システム上の C++ ファイル

.sqx Windows オペレーティング・システム上の C++ ファイル

デフォルトでは、対応するプリコンパイラーの出力ファイルは、以下の拡張子が付けられます。

.c サポートされているすべてのオペレーティング・システム上の C ファイル

.C UNIX および Linux オペレーティング・システム上の C++ ファイル

.cxx Windows オペレーティング・システム上の C++ ファイル

OUTPUT プリコンパイル・オプションを使用することにより、修正後の出力ソース・ファイルの名前とパスをオーバーライドにできます。 TARGET C または TARGET CPLUSPLUS プリコンパイル・オプションを使用する場合、入力ファイルに拡張子は必要ありません。

COBOL の入力および出力ファイル

デフォルトでは、ソース・アプリケーションに以下の拡張子が付けられます。

.sqb あらゆるオペレーティング・システム上の COBOL ファイル

ただし、TARGET プリコンパイル・オプション (TARGET ANSI_COBOL、TARGET IBMCOB あるいは TARGET MFCOB) を使用した場合、入力ファイルには任意の拡張子を付けることができます。

デフォルトでは、対応するプリコンパイラーの出力ファイルは、以下の拡張子が付けられます。

.cbl あらゆるオペレーティング・システム上の COBOL ファイル

ただし、OUTPUT プリコンパイル・オプションを使用することで、出力の修正済みソース・ファイルに新たな名前とパスを設定することもできます。

FORTRAN の入力および出力ファイル

デフォルトでは、ソース・アプリケーションに以下の拡張子が付けられます。

.sqf あらゆるオペレーティング・システム上の FORTRAN ファイル

ただし、TARGET プリコンパイル・オプションを FORTRAN オプションとともに使用した場合、入力ファイルには任意の拡張子を付けることができます。

デフォルトでは、対応するプリコンパイラーの出力ファイルは、以下の拡張子が付けられます。

.f UNIX および Linux オペレーティング・システム上の FORTRAN ファイル

.for Windows オペレーティング・システム上の FORTRAN ファイル

ただし、OUTPUT プリコンパイル・オプションを使用することで、出力の修正済みソース・ファイルに新たな名前とパスを設定することもできます。

C における組み込み SQL アプリケーション・テンプレート

これは、組み込み SQL 開発環境のテストに使用するため、および組み込み SQL アプリケーションの基本構造を学習するために提供されている、簡単な組み込み SQL アプリケーションです。

組み込み SQL アプリケーションは、以下によって構成されます。

- 必要なヘッダー・ファイルの組み込み
- SQL ステートメントに組み込む値のホスト変数の宣言
- データベース接続

- SQL ステートメントの実行
- SQL ステートメントの実行に関連する SQL エラーおよび警告の処理
- データベース接続のドロップ

次のソース・コードは、C で記述された組み込み SQL アプリケーションに必要な基本構造を示しています。

サンプル・プログラム: template.sqc

```
 #include <stdio.h>                                                    1
 #include <stdlib.h>
 #include <string.h>
 #include <sqlenv.h>
 #include <sqlutil.h>

EXEC SQL BEGIN DECLARE SECTION;                                        2
  short id;
  char name[10];
  short dept;
  double salary;
  char hostVarStmtDyn[50];
EXEC SQL END DECLARE SECTION;

int main()
 {
   int rc = 0;                                                         3
   EXEC SQL INCLUDE SQLCA;                                             4

   /* connect to the database */
   printf("¥n Connecting to database...");
   EXEC SQL CONNECT TO "sample";                                       5
   if (SQLCODE  <0)                                                    6
   {
      printf("¥nConnect Error:  SQLCODE =
      goto connect_reset;
   }
   else
   {
      printf("¥n Connected to database.¥n");
   }

   /* execute an SQL statement (a query) using static SQL; copy the single row
      of result values into host variables*/
   EXEC SQL SELECT id, name, dept, salary                              7
           INTO :id, :name, :dept, :salary
           FROM staff WHERE id = 310;
   if (SQLCODE  <0)                                                    6
   {
      printf("Select Error:  SQLCODE =
   }
   else
   {
      /* print the host variable values to standard output */
      printf("¥n Executing a static SQL query statement, searching for
         ¥n the id value equal to 310¥n");
      printf("¥n ID   Name        DEPT       Salary¥n");
      printf("
   }

   strcpy(hostVarStmtDyn, "UPDATE staff
                           SET salary = salary + 1000
                           WHERE dept = ?");
   /* execute an SQL statement (an operation) using a host variable
      and DYNAMIC SQL*/
   EXEC SQL PREPARE StmtDyn FROM :hostVarStmtDyn;
```

```
if (SQLCODE  <0)                                                    6
{
   printf("Prepare Error:  SQLCODE =
}
else
{
   EXEC SQL EXECUTE StmtDyn USING :dept;                            8
}
if (SQLCODE  <0)                                                    6
{
   printf("Execute Error:  SQLCODE =
}

/* Read the updated row using STATIC SQL and CURSOR */
EXEC SQL DECLARE posCur1 CURSOR FOR
   SELECT id, name, dept, salary
   FROM staff WHERE id = 310;
if (SQLCODE  <0)                                                        6
{
   printf("Declare Error:  SQLCODE =
}
EXEC SQL OPEN posCur1;
EXEC SQL FETCH posCur1 INTO :id, :name, :dept, :salary ;                9
if (SQLCODE  <0)                                                        6
{
   printf("Fetch Error:  SQLCODE =
}
else
{
   printf(" Executing an dynamic SQL statement, updating the
            ¥n salary value for the id equal to 310¥n");
   printf("¥n ID   Name        DEPT       Salary¥n");
   printf("
}

EXEC SQL CLOSE posCur1;

/* Commit the transaction */
printf("¥n  Commit the transaction.¥n");
EXEC SQL COMMIT;                                                        10
if (SQLCODE  <0)                                                        6
{
   printf("Error:  SQLCODE =
}

/* Disconnect from the database */
connect_reset :
   EXEC SQL CONNECT RESET;                                              11
   if (SQLCODE  <0)                                                     6
   {
      printf("Connection Error:  SQLCODE =
   }
return 0;
} /* end main */
```

サンプル・プログラム: template.sqc の注:

注	説明
1	ファイルの組み込み: このディレクティブは、ファイルをソース・アプリケーションに組み込みます。
2	宣言セクション: C アプリケーションの SQL ステートメントで参照される値を保持するために使用される、ホスト変数の宣言。
3	ローカル変数宣言: このブロックは、アプリケーションで使用されるローカル変数を宣言します。これらはホスト変数ではありません。

注	説明
4	SQLCA 構造体の組み込み: SQLCA 構造体は、各 SQL ステートメントの実行後に更新されます。このテンプレート・アプリケーションは、特定の SQLCA フィールドを使用してエラー処理を行います。
5	データベースへの接続: データベースを処理するための最初のステップは、そのデータベースへの接続を確立することです。ここで、接続は CONNECT SQL ステートメントを実行して確立されます。
6	エラー処理: エラーが生じたかどうかを調べます。
7	照会の実行: この SQL ステートメントを実行すると、表から戻されるデータをホスト変数に割り当てます。SQL ステートメント実行の下にある C コードは、ホスト変数内の値を標準出力に印刷します。
8	操作の実行: この SQL ステートメントを実行すると、部門番号によって識別される表内の行のセットが更新されます。準備 (3 行上で実行される) は、このステートメント内で参照されるものなどのホスト変数値が、実行される SQL ステートメントにバインドされるステップです。
9	操作の実行: この行および直前の行では、アプリケーションが静的 SQL 内のカーソルを使用して、表内の情報を選択してデータを印刷します。カーソルが宣言されてオープンされた後、データがフェッチされて、最後にカーソルがクローズされます。
10	トランザクションのコミット: COMMIT ステートメントは、作業単位内で行われたデータベース変更を終了します。
11	そして最後に、データベース接続をドロップする必要があります。

組み込み SQL アプリケーションに必要な組み込みファイルおよび定義

組み込みファイルは、ライブラリー内で使用される関数およびタイプを提供するために必要です。これらのファイルが組み込まれてからでなければ、プログラムはライブラリー関数を使用できません。デフォルトで、これらのファイルは $HOME/sqllib/include フォルダーにインストールされます。ホスト言語ごとに、ファイルを組み込む独自の方法、およびファイル拡張子を使用する独自の方法があります。指定された言語に応じて、ファイル・パスを指定するなどの、特定の予防措置を講じる必要があります。

C および C++ 組み込み SQL アプリケーションにおける組み込みファイル

C および C++ でのホスト言語固有の組み込みファイル (ヘッダー・ファイル) には、ファイル拡張子の .h が付いています。ファイルの組み込みには、EXEC SQL INCLUDE ステートメントを使用する方法と、#include マクロを使用する方法の 2 つがあります。プリコンパイラーは #include を無視し、 EXEC SQL INCLUDE ステートメントを使用して組み込まれたファイルのみを処理します。 EXEC SQL INCLUDE を使用して組み込んだファイルを探索する際に、 DB2 C プリコンパイラーはまず最初に現行ディレクトリーを検索し、次いで DB2INCLUDE 環境変数に指定されたディレクトリーを検索します。次の例を考えてみてください。

- `EXEC SQL INCLUDE payroll;`

  INCLUDE ステートメントに指定されたファイルが、上のように、単引用符 (') で囲まれていない場合、 C プリコンパイラーは各ディレクトリーで最初に payroll.sqc を検索し、次に payroll.h を検索します。 UNIX および Linux オ

ペレーティング・システムの場合、C++ プリコンパイラーは、検索対象の各ディレクトリー内で payroll.sqC、payroll.sqx、 payroll.hpp、payroll.h の順に検索を行います。Windows 32 ビット・オペレーティング・システムの場合、C++ プリコンパイラーは、検索対象の各ディレクトリー内で payroll.sqx、payroll.hpp、payroll.hの順に検索を行います。

- EXEC SQL INCLUDE 'pay/payroll.h';

  ファイル名が単引用符 (') で囲まれている場合、すでに説明したとおり、名前に拡張子は追加されません。

  単引用符 (') 内のファイル名に完全パスが含まれていない場合、INCLUDE ファイル名に指定されているパスの前に、DB2INCLUDE の内容を付加したものを順次使用してファイルが検索されます。例えば、UNIX および Linux オペレーティング・システムでは、DB2INCLUDE が '/disk2:myfiles/c' に設定された場合、C/C++ プリコンパイラーは './pay/payroll.h'、'/disk2/pay/payroll.h'、'./myfiles/c/pay/payroll.h' の順に検索します。プリコンパイラー・メッセージには、実際にファイルが見つかったパスが表示されます。 Windows オペレーティング・システムでは、上の例のスラッシュを円記号 (¥) に置き換えてください。

注: DB2INCLUDE の設定は、コマンド行プロセッサーによってキャッシュされます。何らかの CLP コマンドを発行した後に DB2INCLUDE の設定を変更する場合は、TERMINATE コマンドを入力してから、データベースに接続し直し、あとは通常どおりプリコンパイルしてください。

プリコンパイラーは、コンパイラー・エラーを元のソースに戻して関連付けるために、出力ファイルに #line マクロを生成します。これにより、コンパイラーはプリコンパイルされた出力ソース・ファイルでの行番号ではなく、ソースまたは組み込まれたソース・ファイルのファイル名と行番号を使用してエラーを報告することができます。

PREPROCESSOR オプションを指定する場合は、プリコンパイラーにより生成されるすべての #line マクロは、外部 C プリプロセッサーからプリプロセスされたファイルを参照します。ソース・コードをオブジェクト・コードに関連付けるデバッガーやその他のツールの中には、#line マクロを使用して常に正常に作動するわけではないものもあります。使用したいツールが期待どおりに動作しない場合は、プリコンパイル時に (DB2 PREP と共に使用される) NOLINEMACRO オプションを使用してください。 このオプションにより、#line マクロは生成されなくなります。

アプリケーションで使用するためのこれらの組み込みファイルについて以下で説明します。

SQLADEF (sqladef.h)
このファイルには、プリコンパイル済みの C および C++ アプリケーションが使用する関数プロトタイプが含まれています。

SQLCA (sqlca.h)
このファイルは SQL 連絡域 (SQLCA) 構造体を定義します。 SQLCA には、データベース・マネージャーが SQL ステートメントおよび API 呼び出しの実行に関するエラー情報をアプリケーションに提供するために使用する変数が含まれています。

SQLCODES (sqlcodes.h)
このファイルは、SQLCA 構造体の SQLCODE フィールドで使用する定数を定義します。

SQLDA (sqlda.h)
このファイルは SQL 記述子域 (SQLDA) 構造体を定義します。 SQLDA はアプリケーションとデータベース・マネージャーとの間でデータをやりとりするために使用されます。

SQLEXT (sqlext.h)
このファイルには、X/Open コール・レベル・インターフェースの仕様の一部ではないこれらの ODBC レベル 1 およびレベル 2 API の関数プロトタイプと定数が含まれており、Microsoft® 社の許可を得て使用します。

SQLE819A (sqle819a.h)
データベースのコード・ページが 819 (ISO ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE819B (sqle819b.h)
データベースのコード・ページが 819 (ISO ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE850A (sqle850a.h)
データベースのコード・ページが 850 (ASCII ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE850B (sqle850b.h)
データベースのコード・ページが 850 (ASCII ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE932A (sqle932a.h)
データベースのコード・ページが 932 (ASCII 日本語) の場合、このシーケンスは、ホスト CCSID 5035 (EBCDIC 日本語) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE932B (sqle932b.h)
データベースのコード・ページが 932 (ASCII 日本語) の場合、このシーケンスは、ホスト CCSID 5026 (EBCDIC 日本語) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLJACB (sqljacb.h)
このファイルは、DB2 Connect インターフェースの定数、構造、および制御ブロックを定義します。

SQLSTATE (sqlstate.h)
このファイルは、SQLCA 構造体の SQLSTATE フィールドで使用する定数を定義します。

SQLSYSTM (sqlsystm.h)
このファイルには、データベース・マネージャー API およびデータ構造が用いるプラットフォーム固有の定義が含まれています。

SQLUDF (sqludf.h)
このファイルは、ユーザー定義関数 (UDF) を作成するための定数およびインターフェース構造を定義します。

SQLUV (sqluv.h)
このファイルは、非同期ログ読み取り API および表のロード/アンロードを行うベンダーが使用する API の構造、定数、プロトタイプなどを定義します。

COBOL 組み込み SQL アプリケーションにおける組み込みファイル

COBOL のホスト言語固有の組み込みファイルには、ファイル拡張子 .cbl が付いています。IBM® COBOL コンパイラーの「システム/390® ホスト・データ・タイプ・サポート」機能を使用する場合、アプリケーション用の DB2 組み込みファイルは以下のディレクトリーにあります。

```
$HOME/sqllib/include/cobol_i
```

DB2 サンプル・プログラムを、提供されたスクリプト・ファイルを使用して構築する場合、スクリプト・ファイルで指定されている組み込みファイルのパスを、cobol_i ディレクトリー (cobol_a ディレクトリーではない) に変更する必要があります。

IBM COBOL コンパイラーの「システム/390 ホスト・データ・タイプ・サポート」機能を使用**しない**場合、またはこのコンパイラーの以前のバージョンを使用する場合、アプリケーション用の DB2 組み込みファイルは以下のディレクトリーにあります。

```
$HOME/sqllib/include/cobol_a
```

INCLUDE ファイルを探索する際、DB2 COBOL プリコンパイラーはまず最初に現行ディレクトリーを検索し、次いで DB2INCLUDE 環境変数に指定されたディレクトリーを検索します。次の例を考えてみてください。

- `EXEC SQL INCLUDE payroll END-EXEC.`

  INCLUDE ステートメントに指定されたファイルが単引用符 (') で囲まれていない場合、上記のとおり、プリコンパイラーは最初 payroll.sqb を検索し、次に payroll.cpy、次に payroll.cbl、最後にそれがロックする各ディレクトリーを検索します。
- `EXEC SQL INCLUDE 'pay/payroll.cbl' END-EXEC.`

  ファイル名が単引用符 (') で囲まれている場合、すでに説明したとおり、名前に拡張子は追加されません。

単引用符 (') 内のファイル名に完全パスが含まれていない場合、INCLUDE ファイル名に指定されているパスの前に、DB2INCLUDE の内容を付加したものを順次使用してファイルが検索されます。例えば、DB2 データベース・システム (AIX 版) で、DB2INCLUDE が '/disk2:myfiles/cobol' に設定されているとすると、プリコンパイラーは './pay/payroll.cbl'、'/disk2/pay/payroll.cbl'、'./myfiles/cobol/pay/payroll.cbl' の順に検索します。プリコンパイラー・メッセージには、実際にファイルが見つかったパスが表示されます。 Windows プラットフォームでは、上の例のスラッシュを円記号 (¥) に置き換えます。

注: DB2INCLUDE の設定は、DB2 コマンド行プロセッサーによってキャッシュされます。 何らかの CLP コマンドを発行した後に DB2INCLUDE の設定を変更する場合は、 TERMINATE コマンドを入力してから、データベースに接続し直し、あとは通常どおりプリコンパイルしてください。

アプリケーションで使用するためのこれらの組み込みファイルについて以下で説明します。

SQLCA (sqlca.cbl)
このファイルは SQL 連絡域 (SQLCA) 構造体を定義します。 SQLCA には、データベース・マネージャーが SQL ステートメントおよび API 呼び出しの実行に関するエラー情報をアプリケーションに提供するために使用する変数が含まれています。

SQLCA_92 (sqlca_92.cbl)
このファイルには、SQL 連絡域 (SQLCA) 構造体の FIPS SQL92 Entry Level 準拠版が入っています。このファイルは、FIPS SQL92 Entry Level standard に準拠する DB2 アプリケーションを作成するときには、ファイル sqlca.cbl の代わりに組み込む必要があります。 LANGLEVEL プリコンパイラー・オプションに SQL92E が設定されていれば、 sqlca_92.cbl ファイルは DB2 プリコンパイラーによって自動的に組み込まれます。

SQLCODES (sqlcodes.cbl)
このファイルは、SQLCA 構造体の SQLCODE フィールドで使用する定数を定義します。

SQLDA (sqlda.cbl)
このファイルは SQL 記述子域 (SQLDA) 構造体を定義します。 SQLDA はアプリケーションとデータベース・マネージャーとの間でデータをやりとりするために使用されます。

SQLEAU (sqleau.cbl)
このファイルには、DB2 セキュリティー監査 API に必要な定数および構造の定義が含まれています。これらの API を使用する場合は、プログラムにこのファイルを組み込む必要があります。このファイルにはまた、監査証跡レコード内のフィールドの定数およびキーワード値の定義も含まれています。これらの定義は、外部または取引先の監査証跡抽出プログラムで使用できます。

SQLETSD (sqletsd.cbl)
このファイルは、表スペース記述子構造 SQLETSDESC を定義します。これはデータベース作成 API である sqlgcrea に渡されます。

SQLE819A (sqle819a.cbl)
データベースのコード・ページが 819 (ISO ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE819B (sqle819b.cbl)
データベースのコード・ページが 819 (ISO ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE850A (sqle850a.cbl)
データベースのコード・ページが 850 (ASCII ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE850B (sqle850b.cbl)
データベースのコード・ページが 850 (ASCII ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE932A (sqle932a.cbl)
データベースのコード・ページが 932 (ASCII 日本語) の場合、このシーケンスは、ホスト CCSID 5035 (EBCDIC 日本語) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE932B (sqle932b.cbl)
データベースのコード・ページが 932 (ASCII 日本語) の場合、このシーケンスは、ホスト CCSID 5026 (EBCDIC 日本語) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQL1252A (sql1252a.cbl)
データベースのコード・ページが 1252 (Windows ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQL1252B (sql1252b.cbl)
データベースのコード・ページが 1252 (Windows ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLSTATE (sqlstate.cbl)
このファイルは、SQLCA 構造体の SQLSTATE フィールドで使用する定数を定義します。

SQLUDF (sqludf.cbl)
このファイルは、ユーザー定義関数 (UDF) を作成するための定数およびインターフェース構造を定義します。

SQLUTBCQ (sqlutbcq.cbl)
このファイルは、表スペース・コンテナー照会データ構造 SQLB-TBSCONTQRY-DATA を定義します。これは、表スペース・コンテナー照会 API である sqlgstsc、sqlgftcq、および sqlgtcq で使用されます。

SQLUTBSQ (sqlutbsq.cbl)
このファイルは、表スペース照会データ構造 SQLB-TBSQRY-DATA を定義します。これは、表スペース照会 API である sqlgstsq、 sqlgftsq、および sqlgtsq で使用されます。

FORTRAN 組み込み SQL アプリケーションにおける組み込みファイル

FORTRAN 用のホスト言語固有の組み込みファイルには、UNIX および Linux オペレーティング・システムの場合にファイル拡張子 `.f` が、Windows オペレーティング・システムの場合に拡張子 `.for` が付きます。ファイルの組み込みには、EXEC SQL INCLUDE ステートメントを使用する方法と、 FORTRAN INCLUDE ステートメントを使用する方法の 2 つがあります。プリコンパイラーは FORTRAN INCLUDE ステートメントを無視し、 EXEC SQL ステートメントを使用して組み込まれたファイルのみを処理します。 INCLUDE ファイルを探索する際、 DB2 FORTRAN プリコンパイラーはまず最初に現行ディレクトリーを検索し、次いで DB2INCLUDE 環境変数に指定されたディレクトリーを検索します。

次の例を考えてみてください。

- `EXEC SQL INCLUDE payroll`

  上の例のように、INCLUDE ステートメントに指定されたファイルが単引用符 (') で囲まれていない場合、プリコンパイラーは検索対象の各ディレクトリーで `payroll.sqf` を検索し、次に `payroll.f` (Windows オペレーティング・システムでは `payroll.for`) を検索します。

- `EXEC SQL INCLUDE 'pay/payroll.f'`

  ファイル名が単引用符 (') で囲まれている場合、すでに説明したとおり、名前に拡張子は追加されません。 (Windows オペレーティング・システムの場合、ファイルは `'pay¥payroll.for'` と指定されます。)

  単引用符 (') 内のファイル名に完全パスが含まれていない場合、INCLUDE ファイル名に指定されているパスの前に、DB2INCLUDE の内容を付加したものを順次使用してファイルが検索されます。例えば、DB2 (UNIX および Linux オペレーティング・システム版) で、DB2INCLUDE が '`/disk2:myfiles/fortran`' に設定されている場合、プリコンパイラーは '`./pay/payroll.f`'、'`/disk2/pay/payroll.f`'、'`./myfiles/cobol/pay/payroll.f`' の順に検索します。プリコンパイラー・メッセージには、実際にファイルが見つかったパスが表示されます。Windows オペレーティング・システムでは、上記の例でスラッシュ (/) の代わりに円記号 (¥) を使用し、拡張子 `'f'` の代わりに `'for'` を使用します。

注: DB2INCLUDE の設定は、DB2 コマンド行プロセッサーによってキャッシュされます。 何らかの CLP コマンドを発行した後に DB2INCLUDE の設定を変更する場合は、 TERMINATE コマンドを入力してから、データベースに接続し直し、あとは通常どおりプリコンパイルしてください。

DB2 データベース・アプリケーションの開発に必要な 32 ビット FORTRAN ヘッダー・ファイルは、これまでは `$INSTHOME/sqllib/include` に置かれていましたが、現在は `$INSTHOME/sqllib/include32` 内にあります。

バージョン 8.1 ではこれらのファイルは、`$INSTDIR/sqllib/include` ディレクトリー内にありました。これは、32 ビット・インスタンスまたは 64 ビット・インスタンスのどちらであるかに応じて、`$DB2DIR/include` または `$DB2DIR/include64` のどちらかのディレクトリーへのシンボリック・リンクでした。

バージョン 9.1 では `$DB2DIR/include` にはすべての組み込みファイル (32 ビットおよび 64 ビット) が入れられ、`$DB2DIR/include32` には 32 ビットの FORTRAN ファイルのみが入れられると共に、FORTRAN を除いて 32 ビットの組み込みファイルは 64 ビットの組み込みファイルと同じであることを示すための README ファイルが入れられます。

`$DB2DIR/include32` ディレクトリーが存在するのは、AIX、Solaris、HP-PA、および HP-IPF のみです。

以下の FORTRAN 組み込みファイルをアプリケーションで使用することができます。

SQLCA (sqlca_cn.f, sqlca_cs.f)
このファイルは SQL 連絡域 (SQLCA) 構造体を定義します。 SQLCA には、データベース・マネージャーが SQL ステートメントおよび API 呼び出しの実行に関するエラー情報をアプリケーションに提供するために使用する変数が含まれています。

FORTRAN アプリケーションには 2 つの SQLCA ファイルが提供されます。 `sqlca_cs.f` はデフォルトで、IBM SQL 互換フォーマットで SQLCA 構造体を定義します。 `SQLCA NONE` オプションを指定して `sqlca_cn.f` ファイルをプリコンパイルすると、定義される SQLCA 構造体のパフォーマンスが向上します。

SQLCA_92 (sqlca_92.f)
このファイルには、SQL 連絡域 (SQLCA) 構造体の FIPS SQL92 Entry Level 準拠版が入っています。このファイルは、FIPS SQL92 Entry Level standard に適合する DB2 アプリケーションを作成する際に、 `sqlca_cn.f` または `sqlca_cs.f` のどちらかのファイルの代わりに組み込まれる必要があります。 LANGLEVEL プリコンパイラー・オプションに SQL92E が設定されていれば、 `sqlca_92.f` ファイルは DB2 プリコンパイラーによって自動的に組み込まれます。

SQLCODES (sqlcodes.f)
このファイルは、SQLCA 構造体の SQLCODE フィールドで使用する定数を定義します。

SQLDA (sqldact.f)

このファイルは SQL 記述子域 (SQLDA) 構造体を定義します。 SQLDA はアプリケーションとデータベース・マネージャーとの間でデータをやりとりするために使用されます。

SQLEAU (sqleau.f)

このファイルには、DB2 セキュリティー監査 API に必要な定数および構造の定義が含まれています。これらの API を使用する場合は、プログラムにこのファイルを組み込む必要があります。このファイルにはまた、監査証跡レコード内のフィールドの定数およびキーワード値の定義も含まれています。これらの定義は、外部または取引先の監査証跡抽出プログラムで使用できます。

SQLE819A (sqle819a.f)

データベースのコード・ページが 819 (ISO ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE819B (sqle819b.f)

データベースのコード・ページが 819 (ISO ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE850A (sqle850a.f)

データベースのコード・ページが 850 (ASCII ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE850B (sqle850b.f)

データベースのコード・ページが 850 (ASCII ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE932A (sqle932a.f)

データベースのコード・ページが 932 (ASCII 日本語) の場合、このシーケンスは、ホスト CCSID 5035 (EBCDIC 日本語) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE932B (sqle932b.f)

データベースのコード・ページが 932 (ASCII 日本語) の場合、このシーケンスは、ホスト CCSID 5026 (EBCDIC 日本語) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQL1252A (sql1252a.f)

データベースのコード・ページが 1252 (Windows ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に

従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQL1252B (sql1252b.f)
データベースのコード・ページが 1252 (Windows ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLSTATE (sqlstate.f)
このファイルは、SQLCA 構造体の SQLSTATE フィールドで使用する定数を定義します。

SQLUDF (sqludf.f)
このファイルは、ユーザー定義関数 (UDF) を作成するための定数およびインターフェース構造を定義します。

エラー処理のための SQLCA の宣言

アプリケーション・プログラムで SQLCA を宣言して、データベース・マネージャーからアプリケーションに情報を戻すようにすることができます。プログラムをプリプロセスする際に、データベース・マネージャーは INCLUDE SQLCA ステートメントの代わりにホスト言語変数宣言を挿入します。システムは、警告標識、エラー・コード、および診断情報の変数を使用してプログラムと連絡します。

各 SQL ステートメントを実行すると、システムは SQLCODE および SQLSTATE の両方の戻りコードを戻します。 SQLCODE はステートメントの実行を要約した整数値で、 SQLSTATE は IBM のリレーショナル・データベース製品に共通のエラー・コードを示す文字フィールドです。 SQLSTATE は ISO/ANS SQL92 標準、および FIPS 127-2 標準にも準拠しています。

注: FIPS 127-2 とは、*Federal Information Processing Standards Publication 127-2 for Database Language SQL* のことです。 ISO/ANS SQL92 とは、 *American National Standard Database Language SQL X3.135-1992* および *International Standard ISO/IEC 9075:1992, Database Language SQL* を指します。

0 未満の SQLCODE は、エラーが発生してステートメントが処理されなかったことを示していることに注意してください。 1 以上の SQLCODE は、警告が出されたものの、ステートメントの処理は継続していることを示します。

C または C++ 言語で作成された DB2 アプリケーションの場合、アプリケーションが複数のソース・ファイルで構成されているなら、 SQLCA の多重定義を回避するため、 EXEC SQL INCLUDE SQLCA ステートメントを組み込むのはその中の 1 つのファイルだけにしてください。それ以外のソース・ファイルには、次の行を組み込みます。

```
#include "sqlca.h"
extern struct sqlca sqlca;
```

SQLCA を宣言するには、プログラムに INCLUDE SQLCA ステートメントを次のようにコーディングします。

- 各言語でのステートメントを以下に示します。 C または C++ アプリケーションの場合、

```
EXEC SQL INCLUDE SQLCA;
```

- Java™ アプリケーションの場合、明示的に SQLCA を使用することはしません。その代わりに、SQLException インスタンス・メソッドを使用して、 SQLSTATE および SQLCODE 値を入手します。
- COBOL アプリケーションの場合、

```
EXEC SQL INCLUDE SQLCA END-EXEC.
```

- FORTRAN アプリケーションの場合、

```
EXEC SQL INCLUDE SQLCA
```

ご使用のアプリケーションが、 ISO/ANS SQL92 または FIPS 127-2 標準に準拠する必要がある場合は、上記のステートメントや INCLUDE SQLCA ステートメントを使用しないでください。

WHENEVER ステートメントを用いたエラー処理

WHENEVER ステートメントがあると、プリコンパイラーは、エラー、警告、または実行中に行が見つからないなどの状態が起こった場合にアプリケーションに指定のラベルまで進むように指示するソース・コードを生成します。 WHENEVER ステートメントは、別の WHENEVER ステートメントが状況を変更するまで、後続の実行可能 SQL ステートメントに影響を与えます。

WHENEVER ステートメントには以下の 3 つの基本形式があります。

```
EXEC SQL WHENEVER SQLERROR   action
EXEC SQL WHENEVER SQLWARNING action
EXEC SQL WHENEVER NOT FOUND  action
```

上記のステートメントについて説明します。

SQLERROR
SQLCODE < 0 である状態です。

SQLWARNING
SQLWARN(0) = W または SQLCODE > 0 ですが、 100 ではない状態です。

NOT FOUND
SQLCODE = 100 である状態です。

いずれの場合も、*action* は以下のどちらかになります。

CONTINUE
アプリケーションの次の命令を続行するよう指示します。

GO TO *label*
GO TO の後に指定されたラベルの直後のステートメントに進むように指示します。 (GO TO は、2 つの語とすることも、GOTO として 1 つの語とすることもできます。)

WHENEVER ステートメントを使用しない場合、実行中にエラー、警告、または例外状態が起こると、デフォルトのアクションとして処理が継続されることになります。

WHENEVER ステートメントは、影響を及ぼす SQL ステートメントの前に指定しなければなりません。そうしないと、プリコンパイラーは実行可能 SQL ステートメント用に余分のエラー処理コードを生成しなければならないことを認識しません。 3 つの基本形式はいつでも任意に組み合わせてアクティブにすることができます。これら 3 つの形式の宣言順序は重要ではありません。

無限ループ状態を避けるには、SQL ステートメントをハンドラー内部で実行する前に、 WHENEVER 処理が取り消されていることを確認してください。 WHENEVER SQLERROR CONTINUE ステートメントを使用すれば、 SQL ステートメントをハンドラー内部で実行することができます。

組み込み SQL アプリケーションにおける DB2 データベースへの接続

データベースを使った作業の前には、そのデータベースへの接続を確立する必要があります。組み込み SQL は、データベース接続を確立するためのコードを組み込む方法を複数提供しています。組み込み SQL ホスト・プログラミング言語によっては、1 つ以上の方法でこれを行える場合があります。

データベース接続は暗黙的にも明示的にも確立できます。暗黙接続とは、ユーザー ID を現在のユーザー ID と想定した接続です。このタイプの接続は、データベース・アプリケーションには推奨されていません。明示的なデータベース接続は、ユーザー ID とパスワードの指定が必要になるため、強く推奨されます。

C および C++ 組み込み SQL アプリケーションにおける DB2 データベースへの接続

C および C++ アプリケーションの場合、データベース接続は、以下のステートメントの実行によって確立できます。

```
EXEC SQL CONNECT TO sample;
```

特定のユーザー ID (herrick) およびパスワード (mypassword) を使用する場合は、以下のステートメントを使用します。

```
EXEC SQL CONNECT TO sample USER herrick USING mypassword;
```

COBOL 組み込み SQL アプリケーションにおける DB2 データベースへの接続

COBOL アプリケーションの場合、データベース接続は、以下のステートメントの実行によって確立できます。このステートメントは、デフォルトのユーザー名を使用して、サンプル・データベースへの接続を作成します。

```
EXEC SQL CONNECT TO sample END-EXEC.
```

特定のユーザー ID (herrick) およびパスワード (mypassword) を使用する場合は、以下のステートメントを使用します。

```
EXEC SQL CONNECT TO sample USER herrick USING mypassword END-EXEC.
```

FORTRAN 組み込み SQL アプリケーションにおける DB2 データベースへの接続

FORTRAN アプリケーションの場合、データベース接続は、以下のステートメントの実行によって確立できます。このステートメントは、デフォルトのユーザー名を使用して、サンプル・データベースへの接続を作成します。

```
EXEC SQL CONNECT TO sample
```

特定のユーザー ID (herrick) およびパスワード (mypassword) を使用する場合は、以下のステートメントを使用します。

```
EXEC SQL CONNECT TO sample USER herrick USING mypassword
```

REXX 組み込み SQL アプリケーションにおける DB2 データベースへの接続

REXX アプリケーションの場合、データベース接続は、以下のステートメントの実行によって確立できます。このステートメントは、デフォルトのユーザー名を使用して、サンプル・データベースへの接続を作成します。

```
CALL SQLEXEC 'CONNECT TO sample'
```

特定のユーザー ID (herrick) およびパスワード (mypassword) を使用する場合は、以下のステートメントを使用します。

```
CALL SQLEXEC 'CONNECT TO sample USER herrick USING mypassword'
```

組み込み SQL アプリケーション内で SQL データ・タイプにマップするデータ・タイプ

アプリケーションとデータベースとの間でデータを交換するには、使用される変数に応じた正しいデータ・タイプ・マッピングを使用してください。プリコンパイラーはホスト変数宣言を検出すると、該当する SQL タイプの値を判別します。ホスト言語ごとに、その特定の言語にだけ固有の特殊なマッピング規則があるので、それに従う必要があります。

C および C++ 組み込み SQL アプリケーションにおいてサポートされる SQL データ・タイプ

特定の事前定義済み C および C++ データ・タイプは、 DB2 データベースの列タイプに対応しています。 ホスト変数として宣言できるのは、その種の C および C++ データ・タイプだけです。

それぞれの列タイプに対応する C および C++ の列タイプを次の表に示します。プリコンパイラーはホスト変数宣言を検出すると、該当する SQL タイプの値を判別します。データベース・マネージャーはこの値を使用して、アプリケーションとの間でやりとりするデータを変換します。

表 2. C および C++ 宣言にマップされた SQL データ・タイプ

SQL 列タイプ[1]	C および C++ データ・タイプ	SQL 列タイプ記述
SMALLINT (500 または 501)	short short int sqlint16	16 ビットの符号付き整数
INTEGER (496 または 497)	long long int sqlint32[2]	32 ビットの符号付き整数
BIGINT (492 または 493)	long long long __int64 sqlint64[3]	64 ビットの符号付き整数
REAL[4] (480 または 481)	float	単精度浮動小数点
DOUBLE[5] (480 または 481)	double	倍精度浮動小数点
DECIMAL(*p*,*s*) (484 または 485)	厳密な対応なし; double を使用	パック 10 進数 (パック 10 進フィールドを文字データとして操作するために CHAR および DECIMAL 関数を使用することを推奨。)
CHAR(1) (452 または 453)	char	単一文字
CHAR(*n*) (452 または 453)	厳密な対応なし; char[*n*+*1*] を使用 (n はデータを収容するのに十分な大きさ) 1<=*n*<=254	固定長文字ストリング

VARCHAR(*n*)
(448 または 449)
	struct tag {
	short int;
	char[*n*]
	}
	1<=*n*<=32 672
		NULL 終了ではない可変長文字ストリング (2 バイトのストリング長指定子を含む)

	代替使用; char[*n*+*1*] を使用 (n はデータを収容するのに十分な大きさ) 1<=*n*<=32 672	NULL 終了可変長文字ストリング **注:** 460/461 の SQL タイプを割り当てられる。

LONG VARCHAR
(456 または 457)
	struct tag {
	short int;
	char[*n*]
	}
	32 673<=*n*<=32 700
		NULL 終了ではない可変長文字ストリング (2 バイトのストリング長指定子を含む)

表 2. C および C++ 宣言にマップされた SQL データ・タイプ (続き)

SQL 列タイプ[1]	C および C++ データ・タイプ	SQL 列タイプ記述
CLOB(*n*)(408 または 409)	sql type is clob(*n*) 1<=*n*<=2 147 483 647	NULL 終了ではない可変長文字ストリング (4 バイトのストリング長指定子を含む)
CLOB ロケーター変数[6] (964 または 965)	sql type is clob_locator	サーバー上の CLOB エンティティーを識別する
CLOB ファイル参照変数[6] (920 または 921)	sql type is clob_file	CLOB データを含むファイルの記述子
BLOB(*n*) (404 または 405)	sql type is blob(*n*) 1<=*n*<=2 147 483 647	NULL 終了ではない可変長バイナリー・ストリング (4 バイトのストリング長標識を含む)
BLOB ロケーター変数[6] (960 または 961)	sql type is blob_locator	サーバー上の BLOB エンティティーを識別する
BLOB ファイル参照変数[6] (916 または 917)	sql type is blob_file	BLOB データを含むファイルの記述子
DATE (384 または 385)	NULL 終了文字形式	NULL 終止符を収容するために最低 11 文字を使用できる。
	VARCHAR 構造書式	最低 10 文字を使用できる。
TIME (388 または 389)	NULL 終了文字形式	NULL 終止符を収容するために最低 9 文字を使用できる。
	VARCHAR 構造書式	最低 8 文字を使用できる。
TIMESTAMP (392 または 393)	NULL 終了文字形式	NULL 終止符を収容するために最低 27 文字を使用できる。
	VARCHAR 構造書式	最低 26 文字が使用できる。

XML[7]
(988 または 989)
	struct {
	sqluint32 length;
	char data[*n*];
	}
	1<=*n*<=2 147 483 647
	SQLUDF_CLOB
		XML 値

BINARY	unsigned char myBinField[4]; 1<= *n* <=255	バイナリー・データ
VARBINARY	struct myVarBinField_t {sqluint16 length;char data[12];} myVarBinField; 1<= *n* <=32704	Varbinary データ

以下のデータ・タイプは、WCHARTYPE NOCONVERT オプションを指定してプリコンパイルした場合に DBCS または EUC 環境でのみ使用できます。

表 3. C および C++ 宣言にマップされた SQL データ・タイプ

SQL 列タイプ[1]	C および C++ データ・タイプ	SQL 列タイプ記述
GRAPHIC(1) (468 または 469)	sqldbchar	単一の 2 バイト文字
GRAPHIC(*n*) (468 または 469)	厳密な対応なし; sqldbchar[*n+1*] を使用 (n はデータを収容するのに十分な大きさ) 1<=*n*<=127	固定長 2 バイト文字ストリング

VARGRAPHIC(*n*)
(464 または 465)
	struct tag {
	short int;
	sqldbchar[*n*]
	}
	1<=*n*<=16 336
		NULL 終了ではない可変長 2 バイト文字ストリング (2 バイトのストリング長指定子を含む)

	代替使用; sqldbchar[*n+1*] を使用 (n はデータを収容するのに十分な大きさ) 1<=*n*<=16 336	NULL 終了可変長 2 バイト文字ストリング **注:** 400/401 の SQL タイプを割り当てられる。

LONG VARGRAPHIC
(472 または 473)
	struct tag {
	short int;
	sqldbchar[*n*]
	}
	16 337<=*n*<=16 350
		NULL 終了ではない可変長 2 バイト文字ストリング (2 バイトのストリング長指定子を含む)

以下のデータ・タイプは、WCHARTYPE CONVERT オプションを使用してプリコンパイルする場合の DBCS または EUC 環境でのみ使用できる。

表 4. C および C++ 宣言にマップされた SQL データ・タイプ

SQL 列タイプ[1]	C および C++ データ・タイプ	SQL 列タイプ記述
GRAPHIC(1) (468 または 469)	wchar_t	• 単一のワイド・キャラクター (C タイプの場合) • 単一の 2 バイト文字 (列タイプの場合)
GRAPHIC(*n*) (468 または 469)	厳密な対応なし; wchar_t [*n+1*] を使用 (n はデータを収容するのに十分な大きさ) 1<=*n*<=127	固定長 2 バイト文字ストリング

表 4. C および C++ 宣言にマップされた SQL データ・タイプ (続き)

SQL 列タイプ[1]	C および C++ データ・タイプ	SQL 列タイプ記述

VARGRAPHIC(*n*)
(464 または 465)
	struct tag {
	short int;
	wchar_t [*n*]
	}
	1<=*n*<=16 336
		NULL 終了ではない可変長 2 バイト文字ストリング (2 バイトのストリング長指定子を含む)

	代替使用; char[*n+1*] を使用 (n はデータを収容するのに十分な大きさ) 1<=*n*<=16 336	NULL 終了可変長 2 バイト文字ストリング **注:** 400/401 の SQL タイプを割り当てられる。

LONG VARGRAPHIC
(472 または 473)
	struct tag {
	short int;
	wchar_t [*n*]
	}
	16 337<=*n*<=16 350
		NULL 終了ではない可変長 2 バイト文字ストリング (2 バイトのストリング長指定子を含む)

以下のデータ・タイプは DBCS または EUC 環境でのみ使用できる。

表 5. C および C++ 宣言にマップされた SQL データ・タイプ

SQL 列タイプ[1]	C および C++ データ・タイプ	SQL 列タイプ記述
DBCLOB(*n*) (412 または 413)	sql type is dbclob(*n*) 1<=*n*<=1 073 741 823	NULL 終了ではない可変長 2 バイト文字ストリング (4 バイトのストリング長指定子を含む)
DBCLOB ロケーター変数[6] (968 または 969)	sql type is dbclob_locator	サーバー上の DBCLOB エンティティーを識別する
DBCLOB ファイル参照変数[6] (924 または 925)	sql type is dbclob_file	DBCLOB データを含むファイルの記述子

表 5. C および C++ 宣言にマップされた SQL データ・タイプ (続き)

SQL 列タイプ[1]	C および C++ データ・タイプ	SQL 列タイプ記述

注:

1. **SQL 列タイプ**の欄にある最初の番号は、標識変数が提供されていないこと、2 番目の番号は標識変数が提供されていることを示します。標識変数は、NULL 値を示したり、切り捨てられたストリングの長さを保持するのに必要です。これらの値は、それぞれのデータ・タイプの SQLDA の SQLTYPE フィールドに現れます。
2. プラットフォームと互換性を持たせるには、sqlint32 を使用してください。 64 ビットの UNIX および Linux オペレーティング・システムでは、long は 64 ビット整数です。 64 ビットの Windows オペレーティング・システムおよび 32 ビットの UNIX および Linux オペレーティング・システムでは、long は 32 ビットの整数です。
3. プラットフォームで互換性を持たせるには、sqlint64 を使用してください。 DB2 データベース・システムの sqlsystm.h ヘッダー・ファイルは、Microsoft コンパイラーを使用する場合に、サポートされる Windows オペレーティング・システムで sqlint64 を __int64 とタイプ定義します。また、32 ビットの UNIX および Linux オペレーティング・システムでは long long と、64 ビットの UNIX および Linux オペレーティング・システムでは long とタイプ定義します。
4. FLOAT(*n*)。ここで 0 < *n* < 25 の場合、REAL と同義。 SQLDA での REAL と DOUBLE の違いは長さの値です (4 または 8)。
5. 以下の SQL タイプは、DOUBLE と同義です。
   - FLOAT
   - FLOAT(*n*) (24 < *n* < 54) は DOUBLE と同義。
   - DOUBLE PRECISION
6. これは列タイプではなく、ホスト変数タイプである。
7. SQL_TYP_XML/SQL_TYP_NXML 値は、DESCRIBE 要求によってのみ戻されます。アプリケーション・リソースを XML 値にバインドするために、アプリケーションが直接この値を使用することはできません。

以下に、サポートされている C および C++ データ・タイプに関するその他の規則を示します。

- データ・タイプ `char` は `char` または `unsigned char` と宣言することができる。
- データベース・マネージャーは、 NULL で終了する可変長文字ストリング・データ・タイプ char[*n*] (データ・タイプ 460) を、VARCHAR(*m*) として処理する。
  - LANGLEVEL が SAA1 の場合、ホスト変数の長さ *m* は、char[*n*] 内の文字ストリングの長さ *n* または最初の NULL 終止符 (¥0) の前のバイト数のいずれか小さい方と等しくなる。
  - LANGLEVEL が MIA の場合には、ホスト変数の長さ *m* は最初の NULL 終止符 (¥0) の前のバイト数と等しくなる。
- データベース・マネージャーは、NULL で終了する可変長 GRAPHIC ストリング・データ・タイプ `wchar_t[n]` または `sqldbchar[n]` (データ・タイプ 400) を、VARGRAPHIC(*m*) として処理する。
  - LANGLEVEL が SAA1 の場合、ホスト変数の長さ *m* は、`wchar_t[n]` または `sqldbchar[n]` 内の文字ストリングの長さ *n*、または最初のグラフィック NULL 終止符の前の文字数のいずれか小さい方と等しくなる。
  - LANGLEVEL が MIA の場合には、ホスト変数の長さ *m* は最初のグラフィック NULL 終止符の前の文字数と等しくなる。
- 符号なし数値データ・タイプはサポートされていない。
- C および C++ データ・タイプの `int` は、その内部表現がマシン依存型であるため使用できない。

C および C++ 組み込み SQL アプリケーションにおけるプロシージャー、関数、およびメソッドのデータ・タイプ

次の表は、プロシージャー、UDF、およびメソッドの SQL データ・タイプと C および C++ データ・タイプ間のサポートされるマッピングをリストしています。

表 6. C および C++ 宣言にマップされた SQL データ・タイプ

SQL 列タイプ[1]	C および C++ データ・タイプ	SQL 列タイプ記述
SMALLINT (500 または 501)	short	16 ビットの符号付き整数
INTEGER(496 または 497)	sqlint32	32 ビットの符号付き整数
BIGINT (492 または 493)	sqlint64	64 ビットの符号付き整数
REAL(480 または 481)	float	単精度浮動小数点
DOUBLE (480 または 481)	double	倍精度浮動小数点
DECIMAL(*p*,*s*) (484 または 485)	サポートされていない	10 進数値を渡すには、パラメーターを DECIMAL からキャスト可能なデータ・タイプ (例えば CHAR または DOUBLE) に定義し、明示的に引数をこのタイプにキャストします。
CHAR(*n*) (452 または 453)	char[*n*+*1*] (n はデータを収容するのに十分な大きさ) 1<=*n*<=254	固定長、NULL 終了文字ストリング
CHAR(*n*) FOR BIT DATA (452 または 453)	char[*n*+*1*] (n はデータを収容するのに十分な大きさ) 1<=*n*<=254	固定長文字ストリング
VARCHAR(*n*) (448 または 449) (460 または 461)	char[*n*+*1*] (n はデータを収容するのに十分な大きさ) 1<=*n*<=32 672	NULL 終了可変長ストリング

VARCHAR(*n*) FOR BIT DATA
(448 または 449)
	struct {
	sqluint16 length;
	char[*n*]
	}
	1<=*n*<=32 672
		非 NULL 終了の可変長文字ストリング

LONG VARCHAR
(456 または 457)
	struct {
	sqluint16 length;
	char[*n*]
	}
	32 673<=*n*<=32 700
		非 NULL 終了の可変長文字ストリング

表 6. C および C++ 宣言にマップされた SQL データ・タイプ (続き)

SQL 列タイプ[1]	C および C++ データ・タイプ	SQL 列タイプ記述

CLOB(*n*)
(408 または 409)
	struct {
	sqluint32 length;
	char data[n];
	}
	1<=*n*<=2 147 483 647
		4 バイト・ストリング長標識をもった、非 NULL 終了の可変長文字ストリング

BLOB(*n*)
(404 または 405)
	struct {
	sqluint32 length;
	char data[n];
	}
	1<=*n*<=2 147 483 647
		4 バイト・ストリング長標識をもった、非 NULL 終了の可変バイナリー・ストリング

DATE (384 または 385)	char[11]	NULL 終了文字形式
TIME (388 または 389)	char[9]	NULL 終了文字形式
TIMESTAMP (392 または 393)	char[27]	NULL 終了文字形式
XML (988/989)	サポートされていない	この記述子タイプ値 (988/989) が定義されるのは、SQLDA で記述に使用するため、また XML Data を (シリアライズされた形式で) 示すためです。データの取り出しおよび挿入には、既存の文字タイプおよびバイナリー形式 (LOB および LOB ファイル参照タイプも含む) も使用できます (動的 SQL のみ)。

注: 以下のデータ・タイプは、WCHARTYPE NOCONVERT オプションを指定してプリコンパイルした場合に DBCS または EUC 環境でのみ使用できます。

表 7. C および C++ 宣言にマップされた SQL データ・タイプ

SQL 列タイプ[1]	C および C++ データ・タイプ	SQL 列タイプ記述
GRAPHIC(*n*) (468 または 469)	sqldbchar[*n+1*] (n はデータを収容するのに十分な大きさ) 1<=*n*<=127	固定長、NULL 終了 2 バイト文字ストリング
VARGRAPHIC(*n*) (400 または 401)	sqldbchar[*n+1*] (n はデータを収容するのに十分な大きさ) 1<=*n*<=16 336	非 NULL 終了の可変長 2 バイト文字ストリング

LONG VARGRAPHIC
(472 または 473)
	struct {
	sqluint16 length;
	sqldbchar[*n*]
	}
	16 337<=*n*<=16 350
		非 NULL 終了の可変長 2 バイト文字ストリング

表 7. C および C++ 宣言にマップされた SQL データ・タイプ (続き)

SQL 列タイプ[1]	C および C++ データ・タイプ	SQL 列タイプ記述

DBCLOB(*n*)
(412 または 413)
	struct {
	sqluint32 length;
	sqldbchar data[n];
	}
	1<=*n*<=1 073 741 823
		4 バイト・ストリング長標識をもった、非 NULL 終了の可変長文字ストリング

注:

1. **SQL 列タイプ**の欄にある最初の番号は、標識変数が提供されていないこと、2 番目の番号は標識変数が提供されていることを示します。標識変数は、NULL 値を示したり、切り捨てられたストリングの長さを保持するのに必要です。これらの値は、それぞれのデータ・タイプの SQLDA の SQLTYPE フィールドに現れます。

COBOL 組み込み SQL アプリケーションでサポートされている SQL データ・タイプ

特定の事前定義 COBOL データ・タイプは、DB2 データベースの列タイプに対応しています。ホスト変数として宣言できるのは、その種の COBOL データ・タイプだけです。

以下の表に、各列タイプに対応する COBOL データ・タイプを示します。プリコンパイラーはホスト変数宣言を検出すると、該当する SQL タイプの値を判別します。データベース・マネージャーはこの値を使用して、アプリケーションとの間でやり取りするデータを変換します。

ホスト変数のすべての有効なデータ記述が認識されるわけではありません。COBOL データ項目は、以下の表に示す項目と一致していなければなりません。別のデータ項目を使用すると、エラーになる場合があります。

表 8. COBOL 宣言にマップされる SQL データ・タイプ

SQL 列タイプ[1]	COBOL データ・タイプ	SQL 列タイプ記述
SMALLINT (500 または 501)	01 name PIC S9(4) COMP-5	16 ビットの符号付き整数
INTEGER(496 または 497)	01 name PIC S9(9) COMP-5	32 ビットの符号付き整数
BIGINT (492 または 493)	01 name PIC S9(18) COMP-5	64 ビットの符号付き整数
DECIMAL(*p*,*s*) (484 または 485)	01 name PIC S9(*m*)V9(*n*) COMP-3.	パック 10 進数
REAL[2] (480 または 481)	01 name USAGE IS COMP-1	単精度浮動小数点
DOUBLE[3] (480 または 481)	01 name USAGE IS COMP-2	倍精度浮動小数点

表 8. COBOL 宣言にマップされる SQL データ・タイプ (続き)

SQL 列タイプ[1]	COBOL データ・タイプ	SQL 列タイプ記述
CHAR(*n*) (452 または 453)	01 name PIC X(*n*)	固定長文字ストリング
VARCHAR(*n*) (448 または 449)	01 name 49 length PIC S9(4) COMP-5 49 name PIC X(*n*) 1<=*n*<=32 672	可変長文字ストリング
LONG VARCHAR (456 または 457)	01 name 49 length PIC S9(4) COMP-5 49 data PIC X(*n*) 32 673<=*n*<=32 700	long 可変長文字ストリング
CLOB(*n*) (408 または 409)	01 MY-CLOB USAGE IS SQL TYPE IS CLOB(n) 1<=*n*<=2 147 483 647	ラージ・オブジェクト可変長文字ストリング
CLOB ロケーター変数[4] (964 または 965)	01 MY-CLOB-LOCATOR USAGE IS SQL TYPE IS CLOB-LOCATOR	サーバー上の CLOB エンティティーを識別する
CLOB ファイル参照変数[4] (920 または 921)	01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE	CLOB データを含むファイルの記述子
BLOB(*n*) (404 または 405)	01 MY-BLOB USAGE IS SQL TYPE IS BLOB(n) 1<=*n*<=2 147 483 647	ラージ・オブジェクト可変長バイナリー・ストリング
BLOB ロケーター変数[4] (960 または 961)	01 MY-BLOB-LOCATOR USAGE IS SQL TYPE IS BLOB-LOCATOR	サーバー上の BLOB エンティティーを識別する
BLOB ファイル参照変数[4] (916 または 917)	01 MY-BLOB-FILE USAGE IS SQL TYPE IS BLOB-FILE.	BLOB データを含むファイルの記述子
DATE (384 または 385)	01 identifier PIC X(10)	10 バイトの文字ストリング
TIME (388 または 389)	01 identifier PIC X(8)	8 バイトの文字ストリング
TIMESTAMP (392 または 393)	01 identifier PIC X(26)	26 バイトの文字ストリング
XML[5] (988 または 989)	01 name USAGE IS SQL TYPE IS XML AS CLOB (size).	XML 値

以下のデータ・タイプは、DBCS 環境でのみ使用可能です。

表 9. COBOL 宣言にマップされる SQL データ・タイプ

SQL 列タイプ[1]	COBOL データ・タイプ	SQL 列タイプ記述
GRAPHIC(*n*) (468 または 469)	01 name PIC G(*n*) DISPLAY-1	固定長 2 バイト文字ストリング
VARGRAPHIC(*n*) (464 または 465)	01 name 49 length PIC S9(4) COMP-5 49 name PIC G(*n*) DISPLAY-1 1<=*n*<=16 336	2 バイトのストリング長標識を持つ、可変長 2 バイト文字ストリング
LONG VARGRAPHIC (472 または 473)	01 name 49 length PIC S9(4) COMP-5 49 name PIC G(*n*) DISPLAY-1 16 337<=*n*<=16 350	2 バイトのストリング長標識を持つ、可変長 2 バイト文字ストリング
DBCLOB(*n*) (412 または 413)	01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(n) 1<=*n*<=1 073 741 823	4 バイトのストリング長標識を持つ、ラージ・オブジェクト可変長 2 バイト文字ストリング
DBCLOB ロケーター変数[4] (968 または 969)	01 MY-DBCLOB-LOCATOR USAGE IS SQL TYPE IS DBCLOB-LOCATOR	サーバー上の DBCLOB エンティティーを識別する
DBCLOB ファイル参照変数[4] (924 または 925)	01 MY-DBCLOB-FILE USAGE IS SQL TYPE IS DBCLOB-FILE	DBCLOB データを含むファイルの記述子

注:

1. **SQL 列タイプ**の欄にある最初の番号は、標識変数が提供されていないこと、2 番目の番号は標識変数が提供されていることを示します。標識変数は、NULL 値を示したり、切り捨てられたストリングの長さを保持するのに必要です。これらの値は、それぞれのデータ・タイプの SQLDA の SQLTYPE フィールドに現れます。
2. FLOAT(*n*)。ここで 0 < *n* < 25 の場合、REAL と同義。 SQLDA での REAL と DOUBLE の違いは長さの値です (4 または 8)。
3. 以下の SQL タイプは、DOUBLE と同義です。
   - FLOAT
   - FLOAT(*n*) (24 < *n* < 54) は DOUBLE と同義。
   - DOUBLE PRECISION
4. これは列タイプではなく、ホスト変数タイプである。
5. SQL_TYP_XML/SQL_TYP_NXML 値は、DESCRIBE 要求によってのみ戻されます。アプリケーション・リソースを XML 値にバインドするために、アプリケーションが直接この値を使用することはできません。

サポートされる COBOL データ・タイプについては、さらに次の規則があります。

- PIC S9 と COMP-3/COMP-5 が明示されている場合、これらは必須です。
- VARCHAR、LONG VARCHAR、VARGRAPHIC、LONG VARGRAPHIC、すべての LOB 変数タイプ以外の列タイプについては、レベル番号として 01 の代わりに 77 を使用できます。
- DECIMAL(p,s) 列タイプのホスト変数を宣言する際には、以下の規則を使用します。以下のサンプルを参照してください。

```
01 identifier PIC S9(m)V9(n) COMP-3
```

– 小数点の表記に V を使用します。
– *n* と *m* の値は 1 以上でなければなりません。
– *n* + *m* の値は 31 以下でなければなりません。
– *s* の値は *n* の値と等しくなります。
– *p* の値は *n* + *m* の値と等しくなります。
– 反復因数 *(n)* と *(m)* はオプションです。以下の例はすべて有効です。

```
01 identifier PIC S9(3)V COMP-3
01 identifier PIC SV9(3) COMP-3
01 identifier PIC S9V COMP-3
01 identifier PIC SV9 COMP-3
```

– COMP-3 の代わりに PACKED-DECIMAL を使用できます。

• 配列は、COBOL プリコンパイラーではサポートされていません。

FORTRAN 組み込み SQL アプリケーションでサポートされている SQL データ・タイプ

特定の事前定義 FORTRAN データ・タイプは、DB2 データベースの列タイプに対応しています。ホスト変数として宣言できるのは、その種の FORTRAN データ・タイプだけです。

それぞれの列タイプに対応する FORTRAN データ・タイプを次の表に示します。プリコンパイラーはホスト変数宣言を検出すると、該当する SQL タイプの値を判別します。データベース・マネージャーはこの値を使用して、アプリケーションとの間でやり取りするデータを変換します。

表 10. FORTRAN 宣言にマップされる SQL データ・タイプ

SQL 列タイプ[1]	FORTRAN データ・タイプ	SQL 列タイプ記述
SMALLINT (500 または 501)	INTEGER*2	16 ビットの符号付き整数
INTEGER(496 または 497)	INTEGER*4	32 ビットの符号付き整数
REAL[2] (480 または 481)	REAL*4	単精度浮動小数点
DOUBLE[3] (480 または 481)	REAL*8	倍精度浮動小数点
DECIMAL(*p*,*s*) (484 または 485)	厳密に対応するものがない ; REAL*8 を使用	パック 10 進数
CHAR(*n*) (452 または 453)	CHARACTER**n*	長さが *n* の固定長文字ストリング (*n* の範囲は 1 から 254 まで)
VARCHAR(*n*) (448 または 449)	SQL TYPE IS VARCHAR(*n*) (*n* の範囲は 1 から 32 672 まで)	可変長文字ストリング
LONG VARCHAR (456 または 457)	SQL TYPE IS VARCHAR(*n*) (*n* の範囲は 32 673 から 32 700 まで)	long 可変長文字ストリング

表 10. FORTRAN 宣言にマップされる SQL データ・タイプ (続き)

SQL 列タイプ[1]	FORTRAN データ・タイプ	SQL 列タイプ記述
CLOB(*n*) (408 または 409)	SQL TYPE IS CLOB (n) (*n* の範囲は 1 から 2 147 483 647 まで)	ラージ・オブジェクト可変長文字ストリング
CLOB ロケーター変数[4] (964 または 965)	SQL TYPE IS CLOB_LOCATOR	サーバー上の CLOB エンティティーを識別する
CLOB ファイル参照変数[4] (920 または 921)	SQL TYPE IS CLOB_FILE	CLOB データを含むファイルの記述子
BLOB(*n*) (404 または 405)	SQL TYPE IS BLOB (n) (*n* の範囲は 1 から 2 147 483 647 まで)	ラージ・オブジェクト可変長バイナリー・ストリング
BLOB ロケーター変数[4] (960 または 961)	SQL TYPE IS BLOB_LOCATOR	サーバー上の BLOB エンティティーを識別する
BLOB ファイル参照変数[4] (916 または 917)	SQL TYPE IS BLOB_FILE	BLOB データを含むファイルの記述子
DATE (384 または 385)	CHARACTER*10	10 バイトの文字ストリング
TIME (388 または 389)	CHARACTER*8	8 バイトの文字ストリング
TIMESTAMP (392 または 393)	CHARACTER*26	26 バイトの文字ストリング
XML (988 または 989)	SQL_TYP_XML	FORTRAN 向けの XML サポートはありません。アプリケーションは、戻される記述タイプを受け取ることはできても、それを利用することはできません。

注:

1. **SQL 列タイプ**の欄にある最初の番号は、標識変数が提供されていないこと、2 番目の番号は標識変数が提供されていることを示します。標識変数は、NULL 値を示したり、切り捨てられたストリングの長さを保持するのに必要です。これらの値は、それぞれのデータ・タイプの SQLDA の SQLTYPE フィールドに現れます。
2. FLOAT(*n*)。ここで 0 < *n* < 25 の場合、REAL と同義。 SQLDA での REAL と DOUBLE の違いは長さの値です (4 または 8)。
3. 以下の SQL タイプは、DOUBLE と同義です。
   - FLOAT
   - FLOAT(*n*) (24 < *n* < 54) は DOUBLE と同義。
   - DOUBLE PRECISION
4. これは列タイプではなく、ホスト変数タイプである。

以下に、サポートされる FORTRAN データ・タイプのその他の規則を示します。

- VARCHAR、LONG VARCHAR、または CLOB ホスト変数を使用して、 254 文字よりも長い動的 SQL ステートメントを定義できる。

REXX 組み込み SQL アプリケーションでサポートされている SQL データ・タイプ

特定の事前定義 REXX データ・タイプは、DB2 データベースの列タイプに対応しています。ホスト変数として宣言できるのは、その種の REXX データ・タイプだけです。以下の表は、 SQLEXEC および SQLDBS が REXX 変数をどのように解釈して、その内容を DB2 のデータ・タイプに変換するかを示しています。

表 11. REXX 宣言にマップされる SQL 列タイプ

SQL 列タイプ[1]	REXX データ・タイプ	SQL 列タイプ記述
SMALLINT (500 または 501)	小数点を持たない -32 768 から 32 767 までの数	16 ビットの符号付き整数
INTEGER (496 または 497)	小数点を持たない -2 147 483 648 から 2 147 483 647 までの数	32 ビットの符号付き整数
REAL[2] (480 または 481)	$-3.40282346 \times 10^{38}$ から $3.40282346 \times 10^{38}$ までの浮動小数値	単精度浮動小数点
DOUBLE[3] (480 または 481)	$-1.79769313 \times 10^{308}$ から $1.79769313 \times 10^{308}$ までの浮動小数値	倍精度浮動小数点
DECIMAL(*p*,*s*) (484 または 485)	小数点を持つ数	パック 10 進数
CHAR(*n*) (452 または 453)	前後に引用符 (') を持つストリング。2 つの引用符を除くと、長さが *n* になる。 先行および後続ブランク、または浮動小数の E 以外の非数値文字を持つ、長さ *n* のストリング	長さが *n* の固定長文字ストリング (*n* の範囲は 1 から 254 まで)
VARCHAR(*n*) (448 または 449)	CHAR(*n*) と等しい	長さが *n* の可変長文字ストリング。 *n* の範囲は 1 から 4000 まで
LONG VARCHAR (456 または 457)	CHAR(*n*) と等しい	長さが *n* の可変長文字ストリング。 *n* の範囲は 1 から 32 700 まで
CLOB(*n*)(408 または 409)	CHAR(*n*) と等しい	長さが *n* のラージ・オブジェクト可変長文字ストリング。 *n* の範囲は 1 から 2 147 483 647 まで
CLOB ロケーター変数[4] (964 または 965)	DECLARE *:var_name* LANGUAGE TYPE CLOB LOCATOR	サーバー上の CLOB エンティティーを識別する
CLOB ファイル参照変数[4] (920 または 921)	DECLARE *:var_name* LANGUAGE TYPE CLOB FILE	CLOB データを含むファイルの記述子
BLOB(*n*) (404 または 405)	前後にアポストロフィーを持つストリングで、BIN が先行する。先行の BIN と 2 つのアポストロフィーを除くと、*n* 文字となる	長さが *n* のラージ・オブジェクト可変長バイナリー・ストリング。 *n* の範囲は 1 から 2 147 483 647 まで

表 11. REXX 宣言にマップされる SQL 列タイプ (続き)

SQL 列タイプ[1]	REXX データ・タイプ	SQL 列タイプ記述
BLOB ロケーター変数[4] (960 または 961)	DECLARE *:var_name* LANGUAGE TYPE BLOB LOCATOR	サーバー上の BLOB エンティティーを識別する
BLOB ファイル参照変数[4] (916 または 917)	DECLARE *:var_name* LANGUAGE TYPE BLOB FILE	BLOB データを含むファイルの記述子
DATE (384 または 385)	CHAR(*10*) と等しい	10 バイトの文字ストリング
TIME (388 または 389)	CHAR(*8*) と等しい	8 バイトの文字ストリング
TIMESTAMP (392 または 393)	CHAR(*26*) と等しい	26 バイトの文字ストリング
XML (988 または 989)	SQL_TYP_XML	REXX 向けの XML サポートはありません。アプリケーションは、記述タイプを戻すことはできても、それを利用することはできません。

以下のデータ・タイプは、DBCS 環境でのみ使用可能です。

表 12. REXX 宣言にマップされる SQL 列タイプ

SQL 列タイプ[1]	REXX データ・タイプ	SQL 列タイプ記述
GRAPHIC(*n*) (468 または 469)	前後にアポストロフィーを持つストリングで、G または N が先行する。先行の文字と 2 つのアポストロフィーを除くと、 *n* 個の DBCS 文字となる	長さが *n* の固定長 GRAPHIC ストリング。 *n* の範囲は 1 から 127 まで
VARGRAPHIC(*n*) (464 または 465)	GRAPHIC(*n*) と等しい	長さが *n* の可変長 GRAPHIC ストリング。 *n* の範囲は 1 から 2000 まで
LONG VARGRAPHIC (472 または 473)	GRAPHIC(*n*) と等しい	長さが *n* の長可変長 GRAPHIC ストリング。 *n* の範囲は 1 から 16 350 まで
DBCLOB(*n*) (412 または 413)	GRAPHIC(*n*) と等しい	長さが *n* のラージ・オブジェクト可変長 GRAPHIC ストリング。 *n* の範囲は 1 から 1 073 741 823 まで
DBCLOB ロケーター変数[4] (968 または 969)	DECLARE *:var_name* LANGUAGE TYPE DBCLOB LOCATOR	サーバー上の DBCLOB エンティティーを識別する
DBCLOB ファイル参照変数[4] (924 または 925)	DECLARE *:var_name* LANGUAGE TYPE DBCLOB FILE	DBCLOB データを含むファイルの記述子

表 12. REXX 宣言にマップされる SQL 列タイプ (続き)

SQL 列タイプ[1]	REXX データ・タイプ	SQL 列タイプ記述

注:

1. **列タイプ**の最初の番号は、標識変数が提供されないことを示し、2 番目の番号は標識変数が提供されることを示す。標識変数は、NULL 値を示したり、切り捨てられたストリングの長さを保持するのに必要です。
2. FLOAT(*n*)。ここで 0 < *n* < 25 の場合、REAL と同義。 SQLDA での REAL と DOUBLE の違いは長さの値です (4 または 8)。
3. 以下の SQL タイプは、DOUBLE と同義です。
   - FLOAT
   - FLOAT(*n*) (24 < *n* < 54) は DOUBLE と同義。
   - DOUBLE PRECISION
4. これは列タイプではなく、ホスト変数タイプである。

組み込み SQL アプリケーションにおけるホスト変数

ホスト変数 とは、組み込み SQL ステートメントが参照する変数です。それらは、データベース・サーバーと組み込み SQL アプリケーションの間でデータ値を交換するために使用されます。組み込み SQL アプリケーションには、リレーショナル SQL 照会のためのホスト変数宣言を含めることもできます。さらに、ホスト変数は、実行される XQuery 式を収容するために使用できます。ただし、XQuery 式の中のパラメーターに値を渡すためのメカニズムは存在しません。

ホスト変数の宣言には、ホスト言語に固有の変数宣言構文を宣言セクション内で使用します。

宣言セクションとは、組み込み SQL アプリケーションの一部分です。これは、組み込み SQL ソース・コード・ファイルの最上部近くにあり、以下の 2 つの非実行可能 SQL ステートメントに囲まれています。

- BEGIN DECLARE SECTION
- END DECLARE SECTION

これらのステートメントによって、プリコンパイラーは変数の宣言を検出することができます。個々のホスト変数宣言は、上記の 2 つのステートメントの間になければなりません。そうでない場合、変数は単なる通常の変数とみなされます。

以下の規則は、ホスト変数宣言セクションに当てはまります。

- ホスト変数はすべて、ソース・ファイルの、正しい形式の宣言セクションで宣言してから参照しなければならない。ただし、SQLDA 構造を参照するホスト変数は例外です。
- 複数の宣言セクションを 1 つのソース・ファイルで使用することもできる。
- したがって、ホスト変数の名前はソース・ファイル内で固有でなければなりません。これは、DB2 プリコンパイラーが、ホスト言語に固有の変数範囲規則の根拠とはならないためです。そのため、ホスト変数の有効範囲は 1 つだけになります。

注: これは、DB2 プリコンパイラーが、定義されている有効範囲外でもアクセスできるようにするためにホスト変数の有効範囲をグローバルに変更するということではありません。

次の例を考えてください。

```
foo1(){
  .
  .
  .
  BEGIN SQL DECLARE SECTION;
  int x;
  END SQL DECLARE SECTION;
x=10;
  .
  .
  .
}


foo2(){
  .
  .
  .
  y=x;
  .
  .
  .
}
```

言語によっては、変数 x が foo2() 関数内で宣言されていなかったり、 x の値が foo2() 内で 10 に設定されていないため、どちらの場合も上記の例ではコンパイルが失敗します。 こうした問題を避けるには、グローバル変数として x を宣言するか、 x をパラメーターとして foo2() 関数に渡すかのいずれかにする必要があります。以下のようにします。

```
  foo1(){
  .
  .
  .
    BEGIN SQL DECLARE SECTION;
    int x;
    END SQL DECLARE SECTION;
    x=10;
    foo2(x);
  .
  .
  .
  }


  foo2(int x){
  .
  .
  .
    y=x;
  .
  .
  .
  }
```

組み込み SQL アプリケーションにおけるホスト変数の宣言

データベース・サーバーとアプリケーションの間でデータを転送するには、アプリケーションのソース・コードの中で、リレーショナル SQL 照会や、XQuery 式のホスト変数宣言などに関し、ホスト変数の宣言をする必要があります。

以下の表に、組み込み SQL ホスト言語のホスト変数宣言の例を示します。

表 13. ホスト言語によるホスト変数宣言

言語	ソース・コード例

C および C++	`EXEC SQL BEGIN DECLARE SECTION;`
	`  short     dept=38, age=26;`
	`  double    salary;`
	`  char      CH;`
	`  char      name1[9], NAME2[9];`
	`  short     nul_ind;`
	`EXEC SQL END DECLARE SECTION;`

COBOL	`EXEC SQL BEGIN DECLARE SECTION END-EXEC.`
	`  01 age        PIC S9(4) COMP-5 VALUE 26.`
	`  01 DEPT       PIC S9(9) COMP-5 VALUE 38.`
	`  01 salary     PIC S9(6)V9(3) COMP-3.`
	`  01 CH         PIC X(1).`
	`  01 name1      PIC X(8).`
	`  01 NAME2      PIC X(8).`
	`  01 nul-ind    PIC S9(4) COMP-5.`
	`EXEC SQL END DECLARE SECTION END-EXEC.`

FORTRAN	`EXEC SQL BEGIN DECLARE SECTION`
	`  integer*2     age   /26/`
	`  integer*4     dept  /38/`
	`  real*8        salary`
	`  character     ch`
	`  character*8   name1,NAME2`
	`  integer*2     nul_ind`
	`EXEC SQL END DECLARE SECTION`

db2dclgn 宣言生成プログラムを使用したホスト変数の宣言

宣言生成プログラムを使用して、データベース内の指定された表の宣言を生成することができます。これによって、アプリケーションに簡単に挿入できる組み込み SQL 宣言のソース・ファイルを作成します。 db2dclgn は、C/C++、 Java、 COBOL、 FORTRAN の各言語をサポートします。

宣言ファイルを生成するには、db2dclgn コマンドを次の形式で入力してください。

```
db2dclgn -d database-name -t table-name [options]
```

たとえば、SAMPLE データベース内の STAFF 表の宣言を C 言語で出力ファイル staff.h に生成するには、次のコマンドを入力します。

```
db2dclgn -d sample -t staff -l C
```

生成される staff.h ファイルには以下のものが含まれています。

```
struct
{
  short id;
  struct
  {
    short length;
```

```
  char data[9];
} name;
short dept;
char job[6];
short years;
double salary;
double comm;
} staff;
```

組み込み SQL アプリケーションにおける列データ・タイプおよびホスト変数

DB2 表の各列には、列の作成時に *SQL データ・タイプ* が付与されます。列にデータ・タイプを割り当てる方法については、CREATE TABLE ステートメントを参照してください。

注:

1. サポートされているすべてのデータ・タイプに、NOT NULL 属性を指定できます。これは別のタイプとして扱われます。
2. データ・タイプは、ユーザー定義特殊タイプ (UDT) を定義することにより拡張することができます。 UDT は、組み込み SQL タイプの 1 つの表現を使用する、別個のデータ・タイプです。

サポートされる組み込み SQL ホスト言語には、データベース・マネージャーのデータ・タイプの大多数に対応するデータ・タイプがあります。ホスト変数宣言には、これらのホスト言語のデータ・タイプだけが使用できます。プリコンパイラーは、ホスト変数宣言を検出すると、適切な SQL データ・タイプの値を判別します。データベース・マネージャーはこの値を使用して、アプリケーションとの間でやり取りするデータを変換します。

データベース・マネージャーが異なるデータ・タイプ間の比較と割り当てをどのように処理するかを理解することは、アプリケーション・プログラマーにとって重要です。つまり、データベース・マネージャーが 2 つの SQL 列データ・タイプと 2 つのホスト言語データ・タイプ (あるいはそのいずれか) で処理を行っているとしても、データ・タイプは代入および比較の操作中は互いに互換性がなければなりません。

データ・タイプの互換性に関する*一般的な* 規則とは、サポートされるホスト言語の数値データ・タイプはすべてデータベース・マネージャーの数値データ・タイプと比較して割り当てることができ、ホスト言語の文字タイプはすべてデータベース・マネージャーの文字タイプと互換性があるということです。数値タイプには文字タイプとの互換性がありません。ただし、この一般的な規則には、ホスト言語においてラージ・オブジェクトでの処理時に課される特徴および制限に基づいた例外もあります。

SQL ステートメント内であれば、DB2 では互換性のあるデータ・タイプ同士での変換が可能です。例えば、以下の SELECT ステートメントでは、SALARY と BONUS は DECIMAL 列ですが、各従業員の合計支給額は DOUBLE データとして戻されます。

```
SELECT EMPNO, DOUBLE(SALARY+BONUS) FROM EMPLOYEE
```

上記のステートメントの実行では、 DECIMAL と DOUBLE のデータ・タイプ間で変換が行われることに注目してください。

画面に表示される照会結果をもっと読みやすくするには、次の SELECT ステートメントを使用することができます。

```
SELECT EMPNO, DIGIT(SALARY+BONUS) FROM EMPLOYEE
```

上記の例で使用されている DIGITS 関数は、数値の文字ストリング表現を戻します。

アプリケーション内でデータを変換するには、この変換をサポートするその他のルーチン、クラス、組み込みタイプ、または API があるかどうかコンパイラーのベンダーにお問い合わせください。

アプリケーションのコード・ページがデータベースのコード・ページと異なる場合は、文字データ・タイプは文字変換にも従います。

組み込み SQL アプリケーションにおける XML ホスト変数の宣言

データベース・サーバーと組み込み SQL アプリケーションの間で XML データを交換するには、アプリケーションのソース・コードの中でホスト変数を宣言する必要があります。

DB2 V9.1 では、XML データをツリー形式でノードの構造化セットに保管する XML データ・タイプが導入されています。この XML データ・タイプを持つ列は SQL_TYP_XML 列 SQLTYPE として記述され、アプリケーションは、これらの列またはパラメーターとの間の入力あるいは出力のために、さまざまな言語固有のデータ・タイプをバインドできます。 XML 列には、SQL、SQL/XML 拡張、または XQuery を使用して直接アクセスすることができます。 XML データ・タイプは、単に列に適用されるだけではありません。関数が XML 値の引数を取ったり、XML 値を生成したりすることもできます。同様に、ストアード・プロシージャーは、入力パラメーターおよび出力パラメーターの両方として XML 値を取ることができます。さらに、XQuery 式は、XML 列にアクセスするかどうかに関係なく、XML 値を生成します。

XML データは本来、文字であり、使用される文字セットを定義するエンコード方式を持っています。 XML データのエンコード方式は、 XML 文書をシリアライズされたストリングで表示したものを含む基本アプリケーション・タイプから導出して、外部的に決めることができます。さらに、内部的に決めることもでき、その場合にはデータの解釈が必要になります。 Unicode でエンコードされた文書には、データ・ストリームの先頭の Unicode 文字コードで構成されるバイト・オーダー・マーク (BOM) が推奨されます。 BOM は、バイト・オーダーおよび Unicode エンコード・フォームを定義するシグニチャーとして使用されます。

データの取り出しおよび挿入には、XML ホスト変数に加え、CHAR、VARCHAR、CLOB、BLOB といった、既存の文字およびバイナリー・タイプも使用できます。しかし、こうしたタイプは、XML ホスト変数とは違い、暗黙的な XML 構文解析の対象になることはありません。その代わり、デフォルトで空白文字の除去を伴う明示的な XMLPARSE 関数が挿入され、適用されます。

組み込み SQL アプリケーションの開発における XML および XQuery に関する制約事項

組み込み SQL アプリケーションで XML ホスト変数を宣言するには以下のようにします。

アプリケーションの宣言セクションで、以下のように、XML ホスト変数を LOB データ・タイプとして宣言します。

- `SQL TYPE IS XML AS CLOB(n) <hostvar_name>`

  ここで、<hostvar_name> は、アプリケーションの混合コード・ページでエンコードされる XML データを含む CLOB ホスト変数です。

- `SQL TYPE IS XML AS DBCLOB(n) <hostvar_name>`

  ここで、<hostvar_name> は、アプリケーションのグラフィック・コード・ページでエンコードされる XML データを含む DBCLOB ホスト変数です。

- `SQL TYPE IS XML AS BLOB(n) <hostvar_name>`

  ここで、<hostvar_name> は、内部でエンコードされる XML データを含む BLOB ホスト変数です。[1]

- `SQL TYPE IS XML AS CLOB_FILE <hostvar_name>`

  ここで、<hostvar_name> は、アプリケーションの混合コード・ページでエンコードされる XML データを含む CLOB ファイルです。

- `SQL TYPE IS XML AS DBCLOB_FILE <hostvar_name>`

  ここで、<hostvar_name> は、アプリケーションのグラフィック・コード・ページでエンコードされる XML データを含む DBCLOB ファイルです。

- `SQL TYPE IS XML AS BLOB_FILE <hostvar_name>`

  ここで、<hostvar_name> は、内部でエンコードされる XML データを含む BLOB ファイルです。[1]

注:

1. XML 1.0 仕様によってエンコード方式を決めるためのアルゴリズムを参照してください (`http://www.w3.org/TR/REC-xml/#sec-guessing-no-ext-info`)。

SQLDA 内の XML 値の識別

基本タイプが XML データを保持していることを示すには、SQLVAR の sqlname フィールドは以下のように更新する必要があります。

- `sqlname.length` は 8 でなければならない。

- sqlname.data の最初の 2 バイトは X'0000' でなければならない。
- sqlname.data の 3 番目、4 番目のバイトは X'0000' にするべきである。
- sqlname.data の 5 番目のバイトは X'01' でなければならない (最初の 2 つの条件が満たされた場合にのみ、XML サブタイプ標識として参照される)。
- 残りのバイトは X'000000' にするべきである。

XML サブタイプ標識が、SQLTYPE が非 LOB である SQLVAR で設定された場合、実行時に SQL0804 エラー (rc=115) が戻されます。

注: SQL_TYP_XML は DESCRIBE ステートメントからのみ戻せます。このタイプは、他のどの要求に対しても使用することはできません。アプリケーションは、有効な文字タイプまたはバイナリー・タイプを収容するように SQLDA を変更し、sqlname フィールドを、データが XML であることを示すよう適切に設定する必要があります。

NULL 標識変数によるヌル SQL 値の識別

組み込み SQL アプリケーションは、*NULL 標識変数* を、NULL 値を受け取ることができるホスト変数と関連付けることによって、 NULL 値を受け取れる状態に準備しておく必要があります。NULL 標識変数は、データベース・マネージャー とホスト・アプリケーションにより共用されます。したがって、この変数は、アプリケーションの中で、SQL データ・タイプ SMALLINT に対応するホスト変数として宣言する必要があります。

NULL 標識変数は SQL ステートメントでホスト変数の直後に置かれ、接頭部としてコロンが付けられます。スペースを用いると NULL 標識変数とホスト変数を分けることができますが、このスペースは必須ではありません。ただし、ホスト変数と NULL 標識変数の間にコンマを使用しないでください。また、オプションの INDICATOR キーワードをホスト変数とその NULL 標識の間に置いて NULL 標識変数を指定することもできます。

NULL 標識変数が負の値かどうかを検査します。変数が負の値ではない場合、アプリケーションはホスト変数の戻り値を使用することができます。変数が負の値の場合、取り出される値は NULL なので、ホスト変数は使用すべきではありません。この場合、データベース・マネージャーはホスト変数の値を変更しません。

注: データベースの構成パラメーター *dft_sqlmathwarn* を 'YES' に設定した場合、NULL 標識変数の値は -2 になることがあります。この値は、算術計算エラーのある式を評価したため、または結果数値をホスト変数に変換しようとした時にオーバーフローが起きたために、 NULL になったことを示しています。

データ・タイプが NULL を処理できる場合、アプリケーションは NULL 標識を指定しなければなりません。そうでない場合は、エラーになります。 NULL 標識を使用しない場合、SQLCODE -305 (SQLSTATE 22002) が戻されます。

SQLCA 構造体が切り捨て警告を示す場合、NULL 標識変数を検査して切り捨てが行われているかどうか調べます。 NULL 標識変数が正の値の場合は、切り捨てが行われています。

- TIME データ・タイプの秒の部分が切り捨てられると、 NULL 標識値には切り捨てられたデータの秒の部分が含まれます。
- 他のすべてのストリングのデータ・タイプ (ラージ・オブジェクト (LOB) を除く) の場合、 NULL 標識値は戻されたデータの実際の長さを表します。ユーザー定義特殊タイプ (UDT) は、それらの基本タイプと同じように扱われます。

INSERT または UPDATE ステートメントを処理中に、データベース・マネージャーは NULL 標識変数をチェックします (標識変数がある場合)。標識変数が負の値の場合、データベース・マネージャーはターゲットの列値を NULL に設定します (NULL が使用できる場合)。

NULL 標識変数がゼロ以上の場合、データベース・マネージャーは関連付けられたホスト変数の値を使用します。

ホスト変数に割り当てられる際にストリング列の値が切り捨てられた場合、 SQLCA 構造体の SQLWARN1 フィールドに X または W が入れられる場合があります。 NULL 終止符が切り捨てられた場合、そのフィールドには 'N' が入ります。

下記の条件のすべてに適合した場合にのみ、データベース・マネージャーから X の値が戻されます。

- データベースのコード・ページからアプリケーションのコード・ページに文字ストリング・データを変換するとデータの長さが変わる場合に、混合コード・ページ接続が行われる。
- カーソルがブロック化されている。
- NULL 標識変数がアプリケーションにより指定されている。

NULL 標識変数に戻される値は、アプリケーションのコード・ページでの文字ストリングの長さになります。

それ以外の場合でデータ切り捨てが行われると、 (NULL 終止符の切り捨てとは対照的に) データベース・マネージャーは必ず W を戻します。 この場合、データベース・マネージャーは、選択リスト項目のコード・ページ (アプリケーションのコード・ページであれ、データベースのコード・ページであれ、あるいは何もない場合であれ) にある結果文字ストリングの長さを指す NULL 標識変数の値をアプリケーションに戻します。

ホスト言語の中でヌル標識変数を使用する前には、そのヌル標識変数を宣言する必要があります。たとえば、C、C++プログラム向けには、ヌル標識変数 cmind は以下のように宣言できます。

```
EXEC SQL BEGIN DECLARE SECTION;
  char cm[3];
  short cmind;
EXEC SQL END DECLARE SECTION;
```

次の表にサポートされるホスト言語を例示します。

表 14. ホスト言語による NULL 標識変数

言語	ソース・コード例

C および C++	`EXEC SQL FETCH C1 INTO :cm INDICATOR :cmind;`
	`if ( cmind < 0 )`
	`  printf( "Commission is NULL¥n" );`

COBOL	`EXEC SQL FETCH C1 INTO :cm INDICATOR :cmind END-EXEC`
	`IF cmind LESS THAN 0`
	`  DISPLAY 'Commission is NULL'`

FORTRAN	`EXEC SQL FETCH C1 INTO :cm INDICATOR :cmind`
	`IF ( cmind .LT. 0 ) THEN`
	`  WRITE(*,*) 'Commission is NULL'`
	`ENDIF`

REXX	`  CALL SQLEXEC 'FETCH C1 INTO :cm INDICATOR :cmind'`
	`  IF ( cmind < 0 )`
	`  SAY 'Commission is NULL'`

組み込み SQL アプリケーションにおける SQLSTATE および SQLCODE ホスト変数の組み込み

エラー情報は、SQLCA 構造体の SQLCODE と SQLSTATE のフィールドに戻されます。SQLCA 構造体は、すべての実行可能 SQL ステートメントとほとんどのデータベース・マネージャー API 呼び出しの後に更新されます。ご使用のアプリケーションが FIPS 127-2 標準に準拠している場合、組み込み SQL アプリケーションで明示的に SQLCA 構造体を宣言する代わりに、SQLSTATE および SQLCODE というホスト変数を宣言することができます。

- PREP オプション LANGLEVEL SQL92E の指定が必要。

次の例では、更新が正常に行われたかどうかを判断するために、 SQLCA 構造の SQLCODE フィールドをアプリケーションがチェックします。

表 15. ホスト言語における組み込み SQL ステートメント

言語	サンプル・ソース・コード

C および C++	`EXEC SQL UPDATE staff SET job = 'Clerk' WHERE job = 'Mgr';`
	`if ( SQLCODE < 0 )`
	`  printf( "Update Error:  SQLCODE =`

COBOL	`EXEC SQL UPDATE staff SET job = 'Clerk' WHERE job = 'Mgr' END_EXEC.`
	`IF SQLCODE LESS THAN 0`
	`  DISPLAY 'UPDATE ERROR:  SQLCODE = ', SQLCODE.`

FORTRAN	`EXEC SQL UPDATE staff SET job = 'Clerk' WHERE job = 'Mgr'`
	`if ( sqlcode .lt. 0 ) THEN`
	`  write(*,*) 'Update error:  sqlcode = ', sqlcode`

組み込み SQL アプリケーションでのホスト変数の参照

組み込み SQL アプリケーション・コード内でホスト変数を宣言したなら、その変数をアプリケーション内で後に参照することができます。 SQL ステートメントで

ホスト変数を使用するときは、名前の接頭部にコロン (:) を付けてください。 ホスト言語プログラミング構文でホスト変数を使用するときは、コロンは省略してください。

使用するホスト言語の構文を使用してホスト変数を参照する。次の表に例示します。

表 16. ホスト言語によるホスト変数参照

言語	ソース・コード例
C または C++	`EXEC SQL FETCH C1 INTO :cm;` `printf( "Commission = %f¥n", cm );`
COBOL	`EXEC SQL FETCH C1 INTO :cm END-EXEC` `DISPLAY 'Commission = ' cm`
FORTRAN	`EXEC SQL FETCH C1 INTO :cm` `WRITE(*,*) 'Commission = ', cm`
REXX	`CALL SQLEXEC 'FETCH C1 INTO :cm'` `SAY 'Commission = ' cm`

例: 組み込み SQL アプリケーションでの XML ホスト変数の参照

以下のサンプル・アプリケーションは、C および COBOL で XML ホスト変数を参照する方法を示しています。

例: 組み込み SQL C アプリケーション

```
The following code example has been formatted for clarity:
EXEC SQL BEGIN DECLARE;
  SQL TYPE IS XML AS CLOB( 10K ) xmlBuf;
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
  SQL TYPE IS CLOB( 10K ) clobBuf;
EXEC SQL END DECLARE SECTION;

// as XML AS CLOB
// The XML value written to xmlBuf will be prefixed by an XML declaration
// similar to: <?xml version = "1.0" encoding = "ISO-8859-1" ?>
// Note: The encoding name will depend upon the application codepage
EXEC SQL SELECT xmlCol INTO :xmlBuf
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlBuf
   WHERE id = '001';

// as XML AS BLOB
// The XML value written to xmlblob will be prefixed by an XML declaration
// similar to: <?xml version = "1.0" encoding = "UTF-8"?>
EXEC SQL SELECT xmlCol INTO :xmlblob
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlblob
   WHERE id = '001';

// as CLOB
// The output will be encoded in the application character codepage,
// but will not contain an XML declaration
EXEC SQL SELECT XMLSERIALIZE (xmlCol AS CLOB(10K)) INTO :clobBuf
   FROM myTable
```

```
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = XMLPARSE (:clobBuf PRESERVE WHITESPACE)
   WHERE id = '001';
```

例: 組み込み SQL COBOL アプリケーション

```
The following code example has been formatted for clarity:
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
  01 xmlBuf USAGE IS SQL TYPE IS XML as CLOB(5K).
  01 clobBuf USAGE IS SQL TYPE IS CLOB(5K).
  01 xmlblob  USAGE IS SQL TYPE IS BLOB(5K).
EXEC SQL END DECLARE SECTION END-EXEC.

* as XML
EXEC SQL SELECT xmlCol INTO :xmlBuf
   FROM myTable
   WHERE id = '001'.
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlBuf
   WHERE id = '001'.

* as BLOB
EXEC SQL SELECT xmlCol INTO :xmlblob
   FROM myTable
   WHERE id = '001'.
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlblob
   WHERE id = '001'.

* as CLOB
EXEC SQL SELECT XMLSERIALIZE(xmlCol AS CLOB(10K)) INTO :clobBuf
   FROM myTable
   WHERE id= '001'.
EXEC SQL UPDATE myTable
   SET xmlCol = XMLPARSE(:clobBuf) PRESERVE WHITESPACE
   WHERE id = '001'.
```

C および C++ 組み込み SQL アプリケーションにおけるホスト変数

ホスト変数は、SQL ステートメント内で参照される C または C++ の言語変数です。これにより、アプリケーションがデータベース・マネージャーとデータを交換することができます。アプリケーションのプリコンパイルが行われると、コンパイラーはホスト変数をその他の C または C++ 変数と同様に使用します。ホスト変数の命名、宣言、および使用は、以下の節で述べる規則に従って行ってください。

long 変数に関する考慮事項

手作業で SQLDA を構成するアプリケーションでは、`sqlvar::sqltype==SQL_TYP_INTEGER` のときは long 変数を使用できません。その代わり、`sqlint32` タイプを使用しなければなりません。この問題は、ホスト変数宣言で long 変数を使用する場合と同様で、手動で SQLDA を構成しなければ、プリコンパイラーはこのエラーをカバーできず実行時にエラーが発生します。

sqlvar::sqldata 情報にアクセスするために使用されるどんな long および unsigned long によるキャストも、それぞれ `sqlint32` および `sqluint32` に変更しなければなりません。`sqloptions` および `sqla_option` 構造体の `val` メンバーは `sqluintptr` として宣言されます。それゆえ、ポインター・メンバーを `sqla_option::val` または `sqloptions::val` メンバーに割り当てる場合には、unsigned long でキャストす

るのではなく sqluintptr でキャストしなければなりません。この変更は、64 ビット UNIX および Linux オペレーティング・システムでは実行時の問題を引き起こしませんが、long タイプが 32 ビットでしかない 64 ビット Windows アプリケーションのための準備では変更が必要になります。

マルチバイト・エンコード方式に関する考慮事項

文字のコード化スキームの中には、特に東アジアの国々の文字には 1 つの文字を表すのに複数バイトを必要とするものがあります。このデータの外部表現は文字のマルチバイト文字コード 表現と呼ばれ、2 バイト文字 (2 バイトで表される文字) を含みます。ホスト変数は、DB2 のグラフィック・データが 2 バイト文字から構成されるため、適宜選択されることになります。

2 バイト文字で文字ストリングを扱うためには、アプリケーションでデータの内部表現を使用するのが便利です。この内部表現は、2 バイト文字のワイド・キャラクター・コード 表現と呼ばれており、通常 wchar_t C または C++ データ・タイプで使用される形式です。 ワイド・キャラクター・データの処理やワイド・キャラクター形式データのマルチバイト形式との変換を行うためには、 ANSI C および X/OPEN Portability Guide 4 (XPG4) に準拠するサブルーチンを使用することができます。

アプリケーションでは、文字データをマルチバイト形式またはワイド・キャラクター形式のどちらかで処理できますが、データベース・マネージャーとの対話は、DBCS (マルチバイト) 文字コードでしか行うことができないことに注意してください。 つまり、データの GRAPHIC 列への保管や GRAPHIC 列からの検索は、DBCS 形式で行われます。 WCHARTYPE プリコンパイラー・オプションは、ワイド・キャラクター形式のアプリケーション・データがデータベース・エンジンで交換される際に、これをマルチバイト形式に変換したり元に戻したりするために使用されます。

C および C++ 組み込み SQL アプリケーションにおけるホスト変数名

SQL プリコンパイラーは、宣言された名前によってホスト変数を識別します。以下の規則が適用されます。

- ホスト変数名の長さは 255 文字まででなくてはならない。
- ホスト変数名では、システムで使用する予約語となっている SQL、sql、DB2、db2 という接頭部を使用してはならない。以下に例を示します。

```
EXEC SQL BEGIN DECLARE SECTION;
  char  varsql;     /* allowed */
  char  sqlvar;     /* not allowed */
  char  SQL_VAR;    /* not allowed */
EXEC SQL END DECLARE SECTION;
```

- プリコンパイラーは、ホスト変数名をモジュールに対してグローバルであると見なす。ただし、これは、ホスト変数をグローバル変数として宣言しなければならないという意味ではありません。ホスト変数を関数内でローカル変数として宣言しても全く問題ありません。たとえば、次ページのコーディングは正しいものとして処理されます。

```
void f1(int i)
{
EXEC SQL BEGIN DECLARE SECTION;
```

```
  short host_var_1;
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT COL1 INTO :host_var_1 from TBL1;
}
void f2(int i)
{
EXEC SQL BEGIN DECLARE SECTION;
  short host_var_2;
EXEC SQL END DECLARE SECTION;
EXEC SQL INSERT INTO TBL1 VALUES (:host_var_2);
}
```

複数のローカル・ホスト変数に同じ名前を付けることも可能です。ただしこの場合、それらの変数がすべて同じタイプかつ同じサイズであることが条件です。 このことを行うには、そのようなホスト変数の最初の出現箇所を BEGIN DECLARE SECTION ステートメントと END DECLARE SECTION ステートメントの間でプリコンパイラーに宣言し、残りの変数の宣言については宣言セクション外でそのままにしておきます。以下のコーディング例は、このことを示したものです。

```
void f3(int i)
{
EXEC SQL BEGIN DECLARE SECTION;
  char host_var_3[25];
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT COL2 INTO :host_var_3 FROM TBL2;
}
void f4(int i)
{
char host_var_3[25];
EXEC SQL INSERT INTO TBL2 VALUES (:host_var_3);
}
```

f3 と f4 は同じモジュールにあり、 host_var_3 はどちらの関数でも同じタイプかつ長さなので、プリコンパイラーへの宣言は 1 回で済みます。

C および C++ 組み込み SQL アプリケーションにおけるホスト変数の宣言セクション

ホスト変数宣言の識別には、SQL の宣言セクションを使用しなければなりません。このようにして、それ以降の SQL ステートメントで参照が可能なホスト変数をプリコンパイラーに知らせます。以下に例を示します。

```
EXEC SQL BEGIN DECLARE SECTION;
  char  varsql;     /* allowed */
EXEC SQL END DECLARE SECTION;
```

C または C++ プリコンパイラーは、有効な C または C++ 宣言のサブセットのみを有効なホスト変数宣言として認識します。これらの宣言は、数値変数または文字変数のいずれかを宣言します。ホスト変数タイプのタイプ定義は使用できません。ホスト変数は、グループ化して単一のホスト構造体にすることができます。 C++ クラスのデータ・メンバーは、ホスト変数として宣言できます。

数値ホスト変数は、数値の SQL 入出力値に対する入出力変数として使用することができます。文字ホスト変数は任意の文字、日付、時間またはタイム・スタンプの SQL 入出力値に対する入出力変数として使用できます。アプリケーションでは、出力変数が受け取る値を入れることのできる長さを持つようにしなければなりません。

例: C および C++ 組み込み SQL アプリケーション用の SQL 宣言セクション・テンプレート

以下に、サポートされている SQL データ・タイプのために宣言されたホスト変数を使用したサンプルの SQL 宣言セクションを示します。

```
   EXEC SQL BEGIN DECLARE SECTION;

.
.
.
        short     age = 26;               /* SQL type  500 */
        short     year;                   /* SQL type  500 */
        sqlint32  salary;                 /* SQL type  496 */
        sqlint32  deptno;                 /* SQL type  496 */
        float     bonus;                  /* SQL type  480 */
        double    wage;                   /* SQL type  480 */
        char      mi;                     /* SQL type  452 */
        char      name[6];                /* SQL type  460 */
        struct   {
                  short len;
                  char data[24];
                 } address;               /* SQL type  448 */
        struct   {
                  short len;
                  char data[32695];
                 } voice;                 /* SQL type  456 */
        sql type is clob(1m)
                  chapter;                /* SQL type  408 */
        sql type is clob_locator
                  chapter_locator;        /* SQL type  964 */
        sql type is clob_file
                  chapter_file_ref;       /* SQL type  920 */
        sql type is blob(1m)
                  video;                  /* SQL type  404 */
        sql type is blob_locator
                  video_locator;          /* SQL type  960 */
        sql type is blob_file
                  video_file_ref;         /* SQL type  916 */
        sql type is dbclob(1m)
                  tokyo_phone_dir;        /* SQL type  412 */
        sql type is dbclob_locator
                  tokyo_phone_dir_lctr;   /* SQL type  968 */
        sql type is dbclob_file
                  tokyo_phone_dir_flref;  /* SQL type  924 */
        sql type is varbinary(12)
                  myVarBinField;          /* SQL type  908 */
        sql type is binary(4)
                  myBinField;             /* SQL type  912 */
        struct   {
                  short len;
                  sqldbchar data[100];
                 } vargraphic1;           /* SQL type  464 */
                                          /* Precompiled with
                                          WCHARTYPE NOCONVERT option */
        struct   {
                  short len;
                  wchar_t data[100];
                 } vargraphic2;           /* SQL type  464 */
                                          /* Precompiled with
                                          WCHARTYPE CONVERT option */
        struct   {
                  short len;
                  sqldbchar data[10000];
                 } long_vargraphic1;      /* SQL type  472 */
                                          /* Precompiled with
                                          WCHARTYPE NOCONVERT option */
```

```
        struct   {
                  short len;
                  wchar_t data[10000];
                 } long_vargraphic2;      /* SQL type  472 */
                                          /* Precompiled with
                                          WCHARTYPE CONVERT option */
        sqldbchar graphic1[100];          /* SQL type  468 */
                                          /* Precompiled with
                                          WCHARTYPE NOCONVERT option */
        wchar_t   graphic2[100];          /* SQL type  468 */
                                          /* Precompiled with
                                          WCHARTYPE CONVERT option */
        char      date[11];               /* SQL type  384 */
        char      time[9];                /* SQL type  388 */
        char      timestamp[27];          /* SQL type  392 */
        short     wage_ind;               /* Null indicator */

.
.
.

   EXEC SQL END DECLARE SECTION;
```

C および C++ 組み込み SQL アプリケーションにおける SQLSTATE および SQLCODE 変数

LANGLEVEL プリコンパイル・オプションを SQL92E の値とともに使用すると、次の 2 つの宣言をホスト変数として組み込めます。

```
EXEC SQL BEGIN DECLARE SECTION;
  char      SQLSTATE[6]
  sqlint32  SQLCODE;


EXEC SQL END DECLARE SECTION;
```

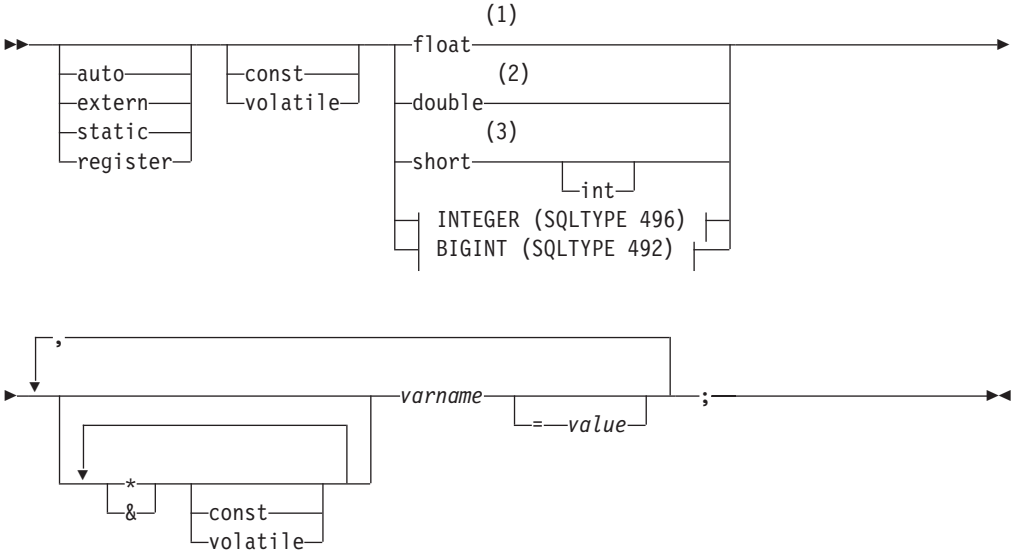
プリコンパイル・ステップの間、SQLCODE 宣言が仮定されます。このオプションを使用するときには、INCLUDE SQLCA ステートメントを指定してはならないことに注意してください。

複数のソース・ファイルから成るアプリケーションでは、上の例のように、最初のソース・ファイルで SQLCODE および SQLSTATE 変数を定義することができます。その後のソース・ファイルは、次のようにその定義を修正する必要があります。

```
extern sqlint32 SQLCODE;
extern char     SQLSTATE[6];
```

C および C++ 組み込み SQL アプリケーションにおける数値ホスト変数の宣言

C または C++ における数値ホスト変数の宣言構文を次に示します。



INTEGER (SQLTYPE 496)

BIGINT (SQLTYPE 492)

注:

1 REAL (SQLTYPE 480)、長さ 4

2 DOUBLE (SQLTYPE 480)、長さ 8

3 SMALLINT (SQLTYPE 500)

4 アプリケーションの移植性を最大限にするには、 INTEGER および BIGINT ホスト変数でそれぞれ `sqlint32` および `sqlint64` を使用してください。デフォルトでは、long のホスト変数を使用すると、long が 64 ビットである 64 BIT UNIX などのプラットフォームでプリコンパイラー・エラー SQL0402 が発生します。 PREP オプション LONGERROR NO を使用して、 DB2 が long 変数を受け入れ可能なホスト変数型として認めるようにしてください。そして、それらを BIGINT 変数として扱ってください。

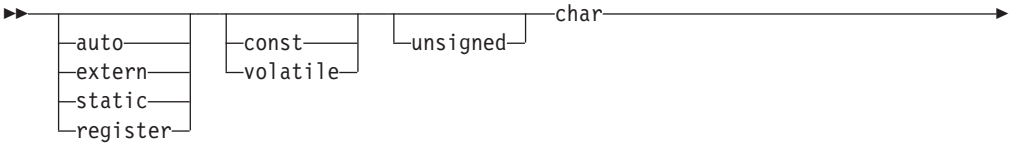
5 アプリケーションの移植性を最大限にするには、 INTEGER および BIGINT ホスト変数でそれぞれ `sqlint32` および `sqlint64` を使用してください。

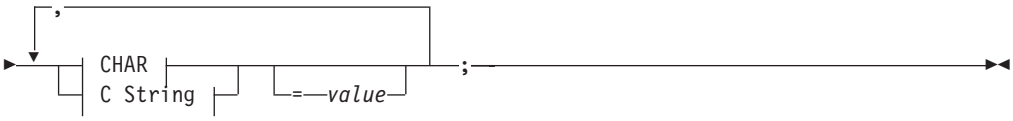
BIGINT データ・タイプを使用するには、プラットフォームで 64 ビットの整数値がサポートされていなければなりません。デフォルトでは、long のホスト変数を使用すると、long が 64 ビットである 64 BIT UNIX などのプラットフォームでプリコンパイラー・エラー SQL0402 が発生します。 PREP オプション LONGERROR NO を使用して、 DB2 が long 変数を受け入れ可能なホスト変数型として認めるようにしてください。そして、それらを BIGINT 変数として扱ってください。

C および C++ 組み込み SQL アプリケーションにおける固定長、ヌル終了、および可変長文字ホスト変数の宣言

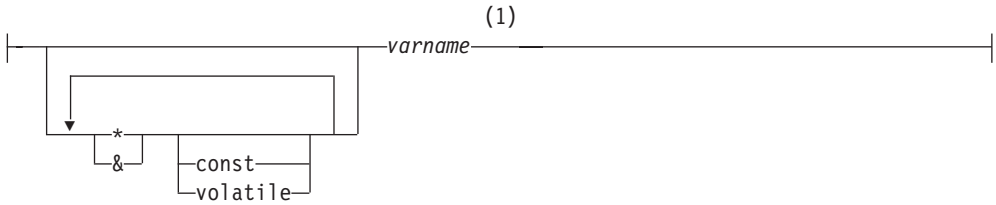
C または C++ における固定、ヌル終了 (書式 1)、および可変長 (書式 2) 文字ホスト変数の宣言構文を次に示します。

書式 1: C または C++ 組み込み SQL アプリケーションにおける固定およびヌル終了文字ホスト変数の構文

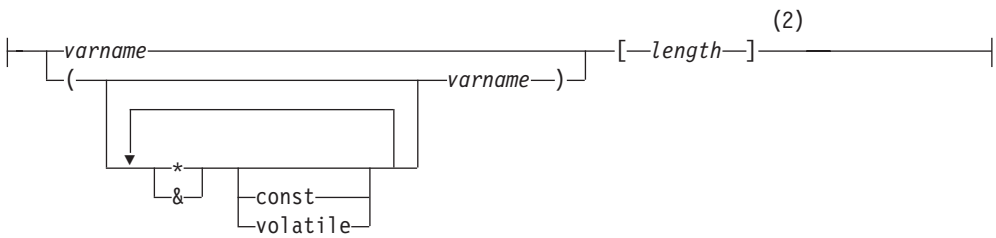




CHAR



C String

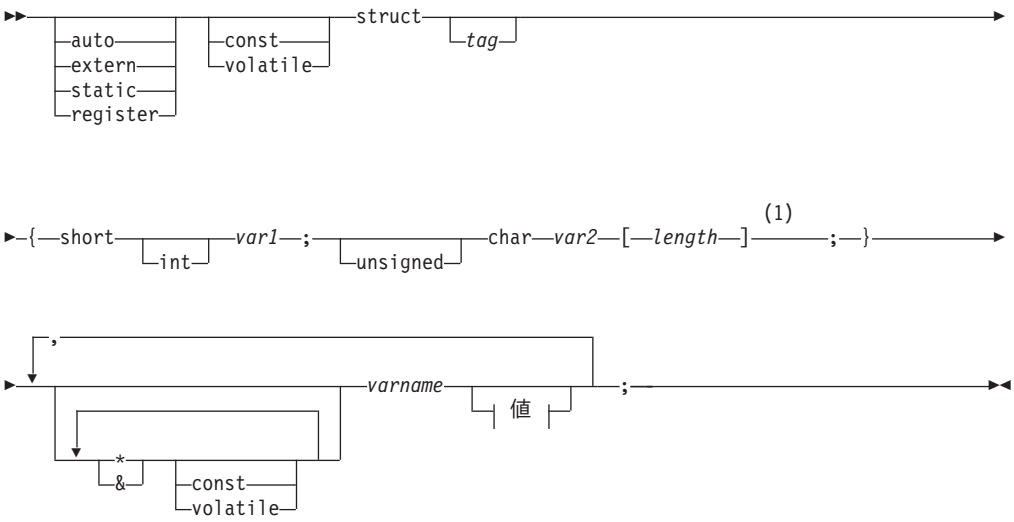


注:

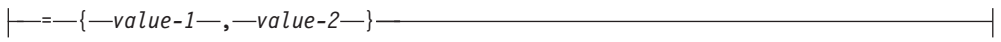
1 CHAR (SQLTYPE 452)、長さ 1

2 NULL 終了 C ストリング (SQLTYPE 460); 長さは任意の有効な定数式

書式 2: C または C++ 組み込み SQL アプリケーションにおける可変長文字ホスト変数の構文



値



注:

1 書式 2 では、length は任意の有効な定数式。評価後の値で、ホスト変数が VARCHAR (SQLTYPE 448) または LONG VARCHAR (SQLTYPE 456) のどちらであるかが判別されます。

可変長文字ホスト変数に関する考慮事項:

1. データベース・マネージャーは、可能な場合には必ず文字データを**書式 1** または**書式 2** に変換するが、**書式 1** が列タイプ CHAR または VARCHAR に対応するのに対し、**書式 2** は列タイプ VARCHAR および LONG VARCHAR に対応する。
2. **書式 1** を長さ指定子の *[n]* と共に使用した場合、評価後の長さ指定子の値は 32 672 より大きくてはならず、変数に含めるストリングは NULL で終わらなければならない。
3. **書式 2** を使用する場合は、評価後の長さ指定子の値は 32 700 以下でなければならない。
4. **書式 2** では、 *var1* および *var2* は単純変数参照 (演算子ではない) でなければならず、ホスト変数として使用することはできない (*varname* がホスト変数)。
5. *varname* には、単純変数名、または **varname* などのように演算子を含むものを指定できる。詳しくは、C および C++ におけるポインター・データ・タイプの説明を参照してください。

6. プリコンパイラーはすべてのホスト変数の SQLTYPE および SQLLEN を判別する。ホスト変数が SQL ステートメント内に標識変数とともにある場合、そのステートメントの持続期間中は、 SQLTYPE は基本の SQLTYPE プラス 1 となるように割り当てられます。
7. プリコンパイラーは、C または C++ において構文的に無効であるものも宣言できる場合がある。特定の宣言構文に疑問がある場合には、ご使用のコンパイラーに関する資料をご覧ください。

C および C++ 組み込み SQL アプリケーションにおけるグラフィック・ホスト変数の宣言

C または C++ で作成されたアプリケーションでグラフィック・データを処理するには、wchar_t C または C++ データ・タイプまたは DB2 提供の sqldbchar データ・タイプに基づくホスト変数を使用してください。これら 2 つのホスト変数は、GRAPHIC、 VARGRAPHIC、または DBCLOB などの表の列に割り当てることができます。たとえば、表の GRAPHIC または VARGRAPHIC 列から、DBCS データを更新したり選択したりすることができます。

グラフィック・ホスト変数には、以下のような 3 つの有効な書式があります。

- 単純グラフィック書式

  単純グラフィック・ホスト変数は、GRAPHIC(1) SQL データ・タイプに相当する 468/469 の SQLTYPE を持っています。
- NULL 終了グラフィック書式

  NULL 終了とは、GRAPHIC ストリングの最後の文字のバイトがすべてバイナリー・ゼロ ('¥0') である状態を言います。これらは SQLTYPE が 400/401 となります。
- VARGRAPHIC 構造書式

  VARGRAPHIC 構造ホスト変数は、長さが 1 から 16 336 バイトの間の場合は SQLTYPE が 464/465 となります。この変数の長さが 2 000 から 16 350 バイトの間の場合は、SQLTYPE が 472/473 となります。

C および C++ 組み込み SQL アプリケーションにおけるグラフィック・データ用の wchar_t および sqldbchar データ・タイプ

DB2 グラフィック・データのサイズおよびエンコードは、特定のコード・ページではどのプラットフォームでも同じですが、 ANSI C または C++ wchar_t データ・タイプのサイズおよび内部書式は、使用するコンパイラーとプラットフォームによって異なります。しかしながら、sqldbchar データ・タイプは、DB2 によってサイズが 2 バイトと定義されており、データベース内で保管されるのと同じ形式で DBCS および UCS-2 データを操作する、移植可能な方法が使用されています。

DB2 C グラフィック・ホスト変数タイプはすべて、wchar_t か sqldbchar によって定義できます。 WCHARTYPE CONVERT プリコンパイル・オプションを使用してアプリケーションを構築する場合には、必ず wchar_t の方を使用してください。

注: Windows オペレーティング・システムで WCHARTYPE CONVERT オプションを指定する場合、Windows オペレーティング・システムの wchar_t が Unicode であるという点にご注意ください。したがって、ご使用の C または C++ コンパイラ

一の wchar_t が Unicode でない場合には、 wcstombs() 関数呼び出しは SQLCODE -1421 (SQLSTATE=22504) を出して失敗することがあります。この場合、WCHARTYPE NOCONVERT オプションを指定したり、ご使用のプログラムから wcstombs() および mbstowcs() 関数を明示的に呼び出したりすることができます。

WCHARTYPE NOCONVERT プリコンパイル・オプションを使用してアプリケーションを構築する場合には、異なる DB2 クライアントとサーバー・プラットフォーム間でも最大限の移植性を利用できるよう、 sqldbchar の方を使用してください。 WCHARTYPE NOCONVERT を使用する場合でも wchar_t は使えますが、 wchar_t の長さが 2 バイトで定義されているプラットフォームだけに限ります。

ホスト変数宣言で wchar_t か sqldbchar を誤って使用すると、プリコンパイル時に SQLCODE 15 (SQLSTATE ではない) が返されます。

C および C++ 組み込み SQL アプリケーションでのグラフィック・データ用の WCHARTYPE プリコンパイラー・オプション

WCHARTYPE プリコンパイラー・オプションを使用すると、 C または C++ アプリケーションでどのグラフィック文字形式を使用するかを指定できます。このオプションにより、グラフィック・データをマルチバイト形式またはワイド・キャラクター形式のどちらにするかを柔軟に選択することができます。 WCHARTYPE オプションには、次の 2 つの値があります。

CONVERT

WCHARTYPE CONVERT オプションを選択した場合、文字コードはグラフィック・ホスト変数とデータベース・マネージャーとの間で変換されます。グラフィック入力ホスト変数の場合、ワイド・キャラクター形式からマルチバイト DBCS 文字形式への文字コード変換は、データがデータベース・マネージャーに送信される前に ANSI C 関数の wcstombs() を使用して行われます。 グラフィック出力ホスト変数の場合には、マルチバイト DBCS 文字形式からワイド・キャラクター形式への文字コード変換は、データベース・マネージャーから受け取られたデータがホスト変数に保管される前に、ANSI C 関数の mbstowcs() を使用して実行されます。

WCHARTYPE CONVERT を使用する利点は、それによってアプリケーションが、データベース・マネージャーと通信する前に、データをマルチバイト形式に明示的に変換しなくても、ワイド・キャラクター・ストリング (L-リテラル、'wc' ストリング関数など) を処理するための ANSI C 機構を十分に利用できることです。欠点としては、暗黙のうちに変換を実行することによって実行時にアプリケーションのパフォーマンスに影響を及ぼすことがあり、さらにメモリー要件が大きくなる恐れがあることが挙げられます。

WCHARTYPE CONVERT を選択した場合は、すべてのグラフィック・ホスト変数を、sqldbchar ではなく、wchar_t を使用して宣言してください。

WCHARTYPE CONVERT 振る舞いは希望するものの、アプリケーションはプリコンパイルする必要がない場合 (たとえば、CLI アプリケーション)、コンパイル時に C プリプロセッサー・マクロ SQL_WCHART_CONVERT を定義してください。 これによって、DB2 ヘッダー・ファイルの特定の定義で、データ・タイプ sqldbchar ではなく wchar_t が使用されます。

NOCONVERT (デフォルト)

WCHARTYPE NOCONVERT オプションを選択した場合、あるいは WCHARTYPE オプションをまったく指定しない場合は、アプリケーションとデータベース・マネージャーの間で暗黙の文字コード変換は行われません。グラフィック・ホスト変数のデータは、変換されない 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。 これにはパフォーマンスを向上させるという利点がありますが、短所としてアプリケーションが wchar_t ホスト変数内のワイド・キャラクター・データの使用をやめるか、またはデータベース・マネージャーとのインターフェースをとる際にデータのマルチバイト形式への変換のために wcstombs() および mbstowcs() 関数を明示的に呼び出さなければならないということがあります。

WCHARTYPE NOCONVERT を選択した場合は、他の DB2 クライアント/サーバー・プラットフォームへの移植性を最大限に得られるようにするため、すべてのグラフィック・ホスト変数を sqldbchar タイプを使用して宣言してください。

注意すべきその他の指針としては以下のものがあります。

- wchar_t または sqldbchar サポートは DBCS データの処理のために使用されるため、これを使用する場合は DBCS または EUC で使用可能なハードウェアとソフトウェアが必要になる。このサポートが使用可能であるのは、DBCS 環境の DB2 Database for Linux, UNIX, and Windows か、または UCS-2 データベースに接続されている任意のアプリケーション (1 バイト・アプリケーションを含む) で GRAPHIC データを処理している場合のみです。
- DBCS 以外の文字と、DBCS 以外の文字に変換できるワイド・キャラクターは、GRAPHIC ストリング内では使用してはならない。 *DBCS 以外の文字* とは、 1 バイト文字と、2 バイト文字以外の文字のことを指します。 GRAPHIC ストリングでは、その値に 2 バイト文字のコード・ポイントのみが含まれているかどうかを確認するための妥当性検査は行われません。グラフィック・ホスト変数には、DBCS データ、または WCHARTYPE CONVERT が有効な場合には、 DBCS データに変換されるワイド・キャラクター・データしか含めることができません。2 バイト文字と 1 バイト文字が混在しているデータは、文字ホスト変数に保管してください。混合データのホスト変数は WCHARTYPE オプションの設定の影響を受けないことに注意してください。
- WCHARTYPE NOCONVERT プリコンパイル・オプションを使用しているアプリケーションでは、 L リテラルをグラフィック・ホスト変数とともに使用しない。これは、L リテラルがワイド・キャラクター形式であるためです。 L リテラルは、L という接頭部を付けた C 言語のワイド・キャラクター・ストリング・リテラルであり、データ・タイプは "array of wchar_t" です。たとえば、L"dbcs-string" は L リテラルです。
- L リテラルを使用した wchar_t ホスト変数の初期化は、WCHARTYPE CONVERT プリコンパイル・オプションを使用しているアプリケーションでは行えるものの、 SQL ステートメントでは使用できない。 SQL ステートメントでは、 L リテラルを使用する代わりに WCHARTYPE の設定から独立している GRAPHIC ストリング定数を使用してください。
- WCHARTYPE オプションの設定は、ホスト変数だけでなく SQLDA 構造体を使用してデータベース・マネージャーとの間で受け渡しするグラフィック・データ

に影響を与える。 WCHARTYPE CONVERT が有効な場合、 SQLDA を介してアプリケーションから受け取られるグラフィック・データはワイド・キャラクター形式と見なされ、 wcstombs() を暗黙のうちに呼び出して DBCS 形式に変換されます。同様に、アプリケーションが受け取るグラフィック出力データは、アプリケーション・ストレージに保管される前にワイド・キャラクター形式に変換されています。

- 境界域が設定されていないストアード・プロシージャーは、 WCHARTYPE NOCONVERT オプションを用いてプリコンパイルしなければならない。通常の境界域が設定されたストアード・プロシージャーは CONVERT または NOCONVERT のいずれのオプションを用いてもプリコンパイルすることができますが、このオプションの指定はストアード・プロシージャーに含まれる SQL ステートメントに操作されるグラフィック・データの形式に影響を及ぼします。ただしどちらの場合も、SQLDA を介してストアード・プロシージャーに渡されるグラフィック・データはすべて DBCS 形式となります。同じように、 SQLDA を介してストアード・プロシージャーから渡されるデータも DBCS 形式でなければなりません。
- アプリケーションがデータベース・アプリケーション・リモート・インターフェース (DARI) のインターフェース (sqleproc() API) を介してストアード・プロシージャーを呼び出す場合、入力 SQLDA 内のグラフィック・データはすべて、呼び出しているアプリケーションの WCHARTYPE 設定に関係なく、DBCS 形式でなければならない。または UCS-2 データベースに接続されている場合は、UCS-2 でなければならない。同じく、出力 SQLDA 内のグラフィック・データはすべて、 WCHARTYPE 設定に関係なく、 DBCS 形式、または UCS-2 データベースに接続されている場合は UCS-2 形式で戻されます。
- アプリケーションが SQL CALL ステートメントを介してストアード・プロシージャーを呼び出す場合は、呼び出しているアプリケーションの WCHARTYPE 設定に従って、 SQLDA でグラフィック・データが変換される。
- ユーザー定義関数 (UDF) に渡されるグラフィック・データは、常に DBCS 形式である。同じように、UDF から戻されるグラフィック・データもすべて、 DBCS データベースでは DBCS 形式、 EUC および UCS-2 データベースでは UCS-2 形式と見なされます。
- DBCLOB ファイル参照変数の使用により DBCLOB ファイルに保管されるデータは、 DBCS 形式か、または UCS-2 データベースの場合には、UCS-2 形式で保管されます。同様に、DBCLOB ファイルからの入力データは、 DBCS 形式か、または UCS-2 データベースの場合には UCS-2 形式のいずれかで検索されます。

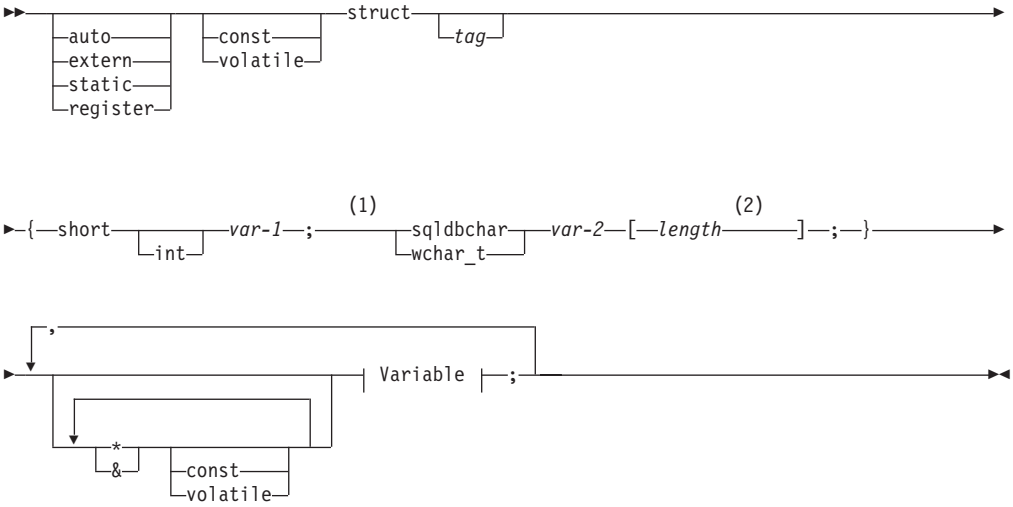
注:

1. DB2 (Windows オペレーティング・システム版) の場合、 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については WCHARTYPE CONVERT オプションがサポートされます。データベース・コード・ページとは異なるコード・ページのデータを DB2 データベースにアプリケーションが挿入する場合は、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 サーバー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 ランタイム環境は、特定の 2 バイト文字の置換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。
2. C 言語アプリケーションを WCHARTYPE CONVERT オプションを使用してプリコンパイルする場合、DB2 は変換関数の間でデータが渡される際に、入出力

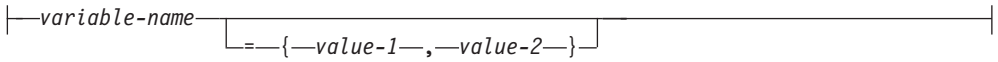
両方のアプリケーションのグラフィック・データを妥当性検査します。
CONVERT オプションを使用しない場合は、グラフィック・データの変換は行われず、したがって検証も行われません。このことが CONVERT/NOCONVERT 混合環境では、無効なデータが NOCONVERT アプリケーションによって挿入され、それを CONVERT アプリケーションが取り出したりする場合に、問題の原因になります。このようなデータの変換は失敗し、CONVERT アプリケーションでの FETCH 時に、SQLCODE -1421 (SQLSTATE 22504) が返されます。

C または C++ 組み込み SQL アプリケーションでの、構造化書式を用いる VARGRAPHIC タイプのホスト変数の宣言

VARGRAPHIC 構造書式を用いるグラフィック・ホスト変数の宣言構文を次に示します。



Variable:



注:

1 2 つのグラフィック・タイプのどちらを使用するかを判別するための基準については、C および C++ における wchar_t および sqldbchar データ・タイプの説明を参照してください。

2 *length* は、任意の有効な定数式。評価後の値で、ホスト変数が VARGRAPHIC (SQLTYPE 464) または LONG VARGRAPHIC (SQLTYPE 472) のどちらであるかが判別されます。 length の値は 1 以上でなければならず、かつ LONG VARGRAPHIC の最大長である 16 350 以下でなければなりません。

グラフィック宣言 (VARGRAPHIC 構造書式) に関する考慮事項:

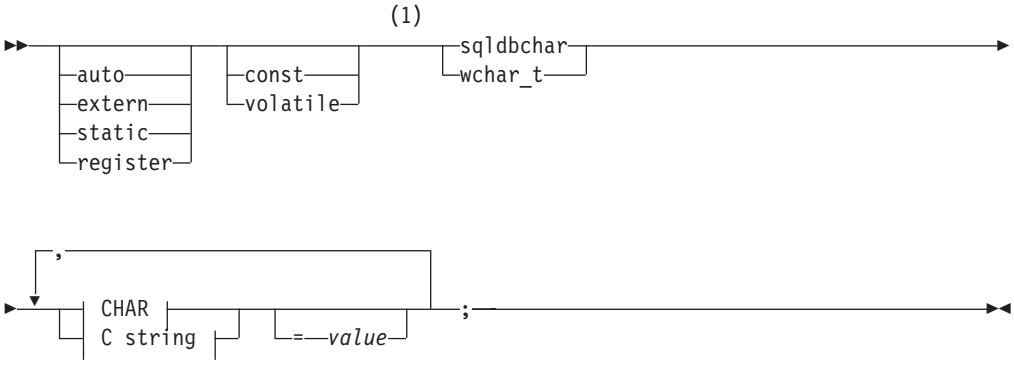
1. *var-1* および *var-2* は単純変数参照 (演算子ではない) でなければならず、ホスト変数として使用することはできない。
2. *value-1* および *value-2* は、 *var-1* と *var-2* に対する初期化指定子である。WCHARTYPE CONVERT プリコンパイラー・オプションを使用している場合、

value-1 は整数でなければならず、 *value-2* はワイド・キャラクター・ストリング・リテラル (L-リテラル) を使用してください。

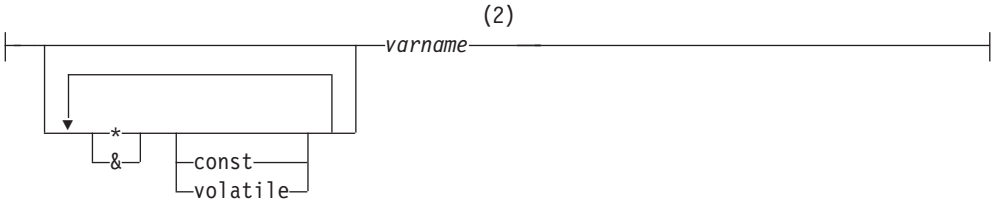
3. struct *tag* は他のデータ域を定義するために使用できるが、それ自体はホスト変数としては使用できない。

C および C++ 組み込み SQL アプリケーションでの、単純グラフィック書式および NULL 終了グラフィック書式を用いる GRAPHIC タイプのホスト変数の宣言

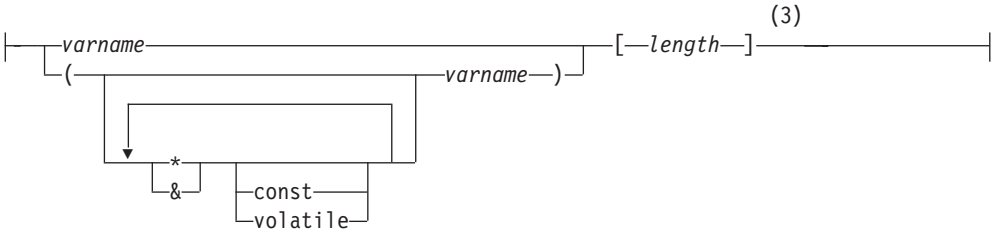
単純グラフィック書式および NULL 終了グラフィック書式を用いるグラフィック・ホスト変数の宣言構文を、次に示します。



CHAR



C string



注:

1 2 つのグラフィック・タイプのどちらを使用するかを判別するための基準については、C および C++ における wchar_t および sqldbchar データ・タイプの説明を参照してください。

2 GRAPHIC (SQLTYPE 468)、長さ 1

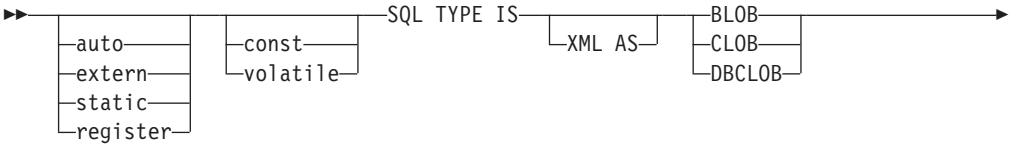
3 NULL 終了 GRAPHIC ストリング (SQLTYPE 400)

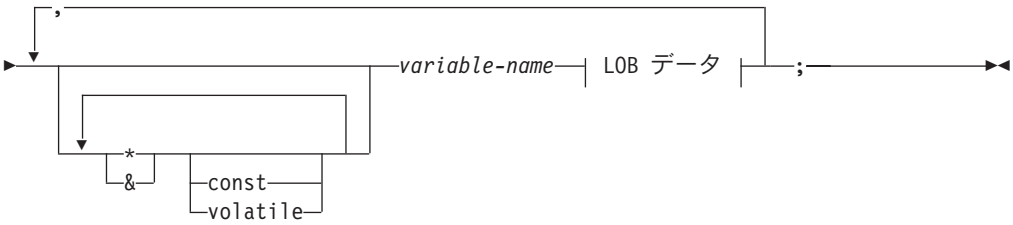
グラフィック・ホスト変数に関する考慮事項:

1. 単純グラフィック書式では、SQLTYPE が 468 または 469 で長さ 1 の固定長の GRAPHIC ストリング・ホスト変数が宣言される。
2. *value* は初期化指定子である。 WCHARTYPE CONVERT プリコンパイラー・オプションを使用している場合は、ワイド・キャラクターのストリング・リテラル (L-リテラル) を使用してください。
3. *length* は任意の有効な定数式にすることができる。評価後の値は、1 以上 16 336 (VARGRAPHIC の最大長) 以下の範囲でなければなりません。
4. NULL 終了 GRAPHIC ストリングは、標準レベルのプリコンパイラー・オプションの値に基づいてさまざまに処理されます。

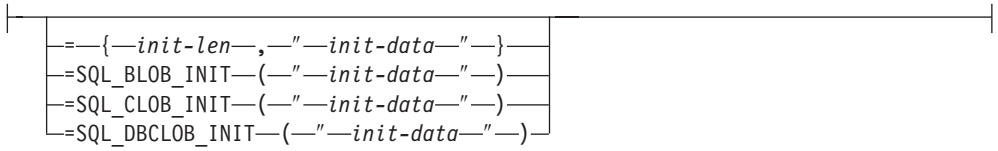
C または C++ 組み込み SQL アプリケーションでのラージ・オブジェクト・タイプのホスト変数の宣言

C または C++ におけるラージ・オブジェクト (LOB) ホスト変数の宣言構文を次に示します。





LOB データ



注:

1 *length* は、任意の有効な定数式。これには定数 K、M、または G を使用できる。 BLOB および CLOB の評価後の length 値は、1 <= length <= 2 147

483 647 でなければなりません。 DBCLOB の評価後の *length* の値は、1 <= length <= 1 073 741 823 でなければなりません。

LOB ホスト変数に関する考慮事項:

1. 関数に渡される LOB タイプのホスト変数に対してタイプ検査と関数分解を実行できるように、 3 タイプの LOB を区別するための SQL TYPE IS 節が必要である。
2. SQL TYPE IS、BLOB、CLOB、DBCLOB、K、M、G は、大文字と小文字が混在してもかまわない。
3. 初期化ストリング *"init-data"* に許可される最大長は、ストリング区切り文字を含めて 32 702 バイトである (プリコンパイラー内の C および C++ ストリングの既存の限界と同じ)。
4. 初期設定長である *init-len* は、数値の定数でなければならない (たとえば、K、M、G は使用できない)。
5. LOB の長さを指定しなければならない。すなわち、次の宣言は無効です。

   ```
   SQL TYPE IS BLOB my_blob;
   ```

6. LOB を宣言内で初期化しないと、プリコンパイラーで生成されたコード内での初期化は行われない。
7. DBCLOB を初期化する場合、ユーザーは、ストリングに 'L' (ワイド・キャラクター・ストリングを表す) という接頭部を付けること。

   注: ワイド・キャラクター・リテラル、たとえば `L"Hello"` は、 WCHARTYPE CONVERT プリコンパイル・オプションを選択した場合に、プリコンパイル済みプログラムでのみ使用すべきである。
8. プリコンパイラーは、ホスト変数のタイプをキャストするために使用できる構造体タグを生成する。

BLOB の例:

宣言:

```
static Sql Type is Blob(2M) my_blob=SQL_BLOB_INIT("mydata");
```

この結果、以下の構造が生成されます。

```
static struct my_blob_t {
      sqluint32        length;
      char             data[2097152];
 } my_blob=SQL_BLOB_INIT("mydata");
```

CLOB の例:

宣言:

```
volatile sql type is clob(125m) *var1, var2 = {10, "data5data5"};
```

この結果、以下の構造が生成されます。

```
volatile struct var1_t {
     sqluint32        length;
     char             data[131072000];
} * var1, var2 = {10, "data5data5"};
```

DBCLOB の例:

宣言:

```
SQL TYPE IS DBCLOB(30000) my_dbclob1;
```

WCHARTYPE NOCONVERT オプション指定でプリコンパイルされ、その結果、以下の構造体が生成されます。

```
struct my_dbclob1_t {
     sqluint32        length;
     sqldbchar        data[30000];
} my_dbclob1;
```

宣言:

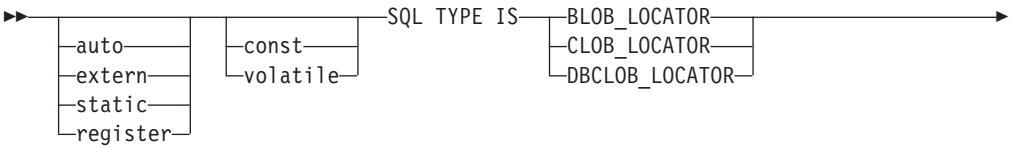
```
SQL TYPE IS DBCLOB(30000) my_dbclob2 = SQL_DBCLOB_INIT(L"mydbdata");
```

WCHARTYPE CONVERT オプション指定でプリコンパイルされ、その結果、以下の構造体が生成されます。

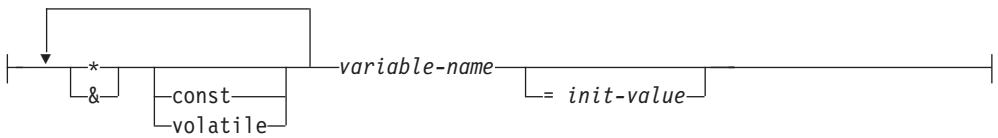
```
struct my_dbclob2_t {
     sqluint32        length;
     wchar_t          data[30000];
} my_dbclob2 = SQL_DBCLOB_INIT(L"mydbdata");
```

C および C++ 組み込み SQL アプリケーションでのラージ・オブジェクト・ロケーター・タイプのホスト変数の宣言

C または C++ におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の宣言構文を次に示します。



Variable



LOB ロケーター・ホスト変数に関する考慮事項:

1. SQL TYPE IS、BLOB-LOCATOR、CLOB-LOCATOR、DBCLOB-LOCATOR は、大文字小文字混合のいずれでもかまわない。
2. *init-value* により、ポインターの初期化およびロケーター変数の参照ができる。他のタイプの初期化は、無意味となる。

CLOB ロケーターの例 (他のタイプの LOB ロケーターの場合も同様):

宣言:

```
SQL TYPE IS CLOB_LOCATOR my_locator;
```

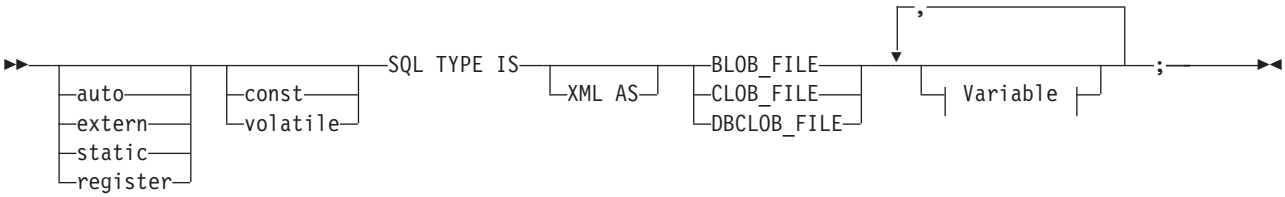
この結果、以下の宣言が生成されます。

```
sqluint32 my_locator;
```

C および C++ 組み込み SQL アプリケーションにおけるファイル参照タイプのホスト変数の宣言

C または C++ におけるファイル参照ホスト変数の宣言構文を次に示します。

C または C++ におけるファイル参照ホスト変数の構文



Variable

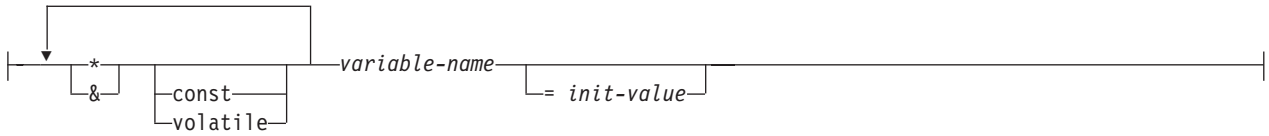


図 1. 構文ダイアグラム

注: SQL TYPE IS、BLOB-FILE、CLOB-FILE、DBCLOB-FILE は、大文字小文字混合のいずれでもかまわない。

CLOB ファイル参照の例 (その他の LOB ファイル参照タイプ宣言も同様):

宣言:

```
static volatile SQL TYPE IS BLOB_FILE my_file;
```

この結果、以下の構造が生成されます。

```
static volatile struct {
     sqluint32        name_length;
     sqluint32        data_length;
     sqluint32        file_options;
              char    name[255];
} my_file;
```

注: 上記の構造体は、sql.h ヘッダーにある sqlfile 構造体と同等です。構文ダイアグラムは、図 1を参照してください。

C および C++ 組み込み SQL アプリケーションにおけるホスト変数のポインターとしての宣言

ホスト変数は、特定のデータ・タイプへのポインターとして、以下のような制限付きで宣言することができます。

- ホスト変数をポインターとして宣言する場合、その他のホスト変数を、同じソース・ファイル内で同じ名前で宣言することはできない。以下の例は無効です。

```
char mystring[20];
char (*mystring)[20];
```

- NULL 終了文字配列へのポインターを宣言する場合は、括弧を使用する。その他のすべての場合は、括弧を使用することはできません。以下に例を示します。

```
EXEC SQL BEGIN DECLARE SECTION;
  char  (*arr)[10];  /* correct   */
  char  *(arr);      /* incorrect */
  char  *arr[10];    /* incorrect */
EXEC SQL END DECLARE SECTION;
```

  この例で、最初の宣言は 10 バイトの文字配列へのポインターです。これは有効なホスト変数となっています。 2 番目は無効な宣言です。文字へのポインターでは括弧は使用できません。 3 番目の宣言はポインターの配列です。このデータ・タイプはサポートされていません。

  以下のようなホスト変数の宣言があるとします。

```
char *ptr;
```

  この宣言は受け入れられますが、*長さが未指定の NULL 終了文字ストリング* を意味するわけではありません。その代わり、これは*固定長の単一文字のホスト変数へのポインター* であることを表します。 これは意図された宣言ではないかもしれません。別の文字ストリングを指示することができるポインター・ホスト変数を定義するには、上記の最初の宣言書式を用いてください。

- SQL ステートメントでポインター・ホスト変数を使用する場合は、以下の例のように、宣言されているのと同じ数のアスタリスクを前に付ける。

```
EXEC SQL BEGIN DECLARE SECTION;
  char (*mychar)[20];    /* Pointer to character array of 20 bytes */
EXEC SQL END DECLARE SECTION;
    EXEC SQL SELECT column INTO :*mychar FROM table;   /* Correct */
```

- ホスト変数名には、アスタリスクだけが演算子として使用できる。
- アスタリスクは名前の一部と見なされないため、ホスト変数名の最大長は指定されたアスタリスクの数の影響を受けない。
- SQL ステートメント内でポインター変数を使用する場合は必ず、最適化レベルのプリコンパイル・オプション (OPTLEVEL) をデフォルト設定の 0 (最適化を行わない) のままにしておく。これは、データベース・マネージャーが SQLDA の最適化を行わないということを意味します。

C++ 組み込み SQL アプリケーションでの、クラス・データ・メンバーのホスト変数としての宣言

クラス・データ・メンバーは、ホスト変数として宣言できます (クラスまたはオブジェクト自身ではなく)。以下は、使用方法を説明する例です。

```
class STAFF
{
    private:
         EXEC SQL BEGIN DECLARE SECTION;
           char        staff_name[20];
           short int   staff_id;
           double      staff_salary;
         EXEC SQL END DECLARE SECTION;
         short       staff_in_db;
    .
    .
};
```

データ・メンバーへは、クラス・メンバー関数内の C++ コンパイラーにより提供される暗黙の *this* ポインターを介して、 SQL ステートメント内で直接アクセスできるだけです。 SQL ステートメント内でオブジェクト・インスタンス (SELECT name INTO :my_obj.staff_name ... など) を明示的に修飾することは**できません**。

SQL ステートメント内でクラス・データ・メンバーを直接参照する場合は、データベース・マネージャーが *this* ポインターを使用して参照を解決します。こうした理由から、最適化レベルのプリコンパイル・オプション (OPTLEVEL) は、デフォルト設定の 0 (最適化を行わない) のままにしておいてください。

次の例は、SQL ステートメント内でホスト変数として宣言したクラス・データ・メンバーを、直接使用する方法を示しています。

```
class STAFF
{
.
.
.
    public:

.
.
.

       short int hire( void )
       {
         EXEC SQL INSERT INTO staff ( name,id,salary )
           VALUES ( :staff_name, :staff_id, :staff_salary );
         staff_in_db = (sqlca.sqlcode == 0);
         return sqlca.sqlcode;
       }
};
```

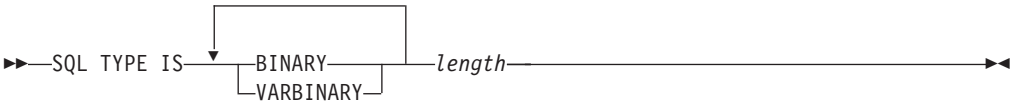
この例では、クラス・データ・メンバーである staff_name、staff_id、および staff_salary が INSERT ステートメント内で直接使用されています。これらはホスト変数として宣言されているため (このセクションの最初の例を参照)、 *this* ポインターを用いて、現行対象に対して暗黙のうちに修飾されています。 SQL ステートメントでは、 *this* ポインターを介してアクセスすることができないデータ・メンバーも参照することができます。 これは、ポインターまたは参照ホスト変数を使用してこれらを間接的に参照することにより行うことができます。

次の例は、2 番目のオブジェクトである *otherGuy* を獲得する、 *asWellPaidAs* という新しい方法を示しています。 この方法では、SQL ステートメント内でメンバーを直接参照できないため、ローカル・ポインターまたは参照ホスト変数を介して間接的にメンバーを参照します。

```
short int STAFF::asWellPaidAs( STAFF otherGuy )
{
    EXEC SQL BEGIN DECLARE SECTION;
      short &otherID = otherGuy.staff_id
      double otherSalary;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL SELECT SALARY INTO :otherSalary
      FROM STAFF WHERE id = :otherID;
      if( sqlca.sqlcode == 0 )
         return staff_salary >= otherSalary;
      else
         return 0;
}
```

C または C++ 組み込み SQL アプリケーションでのバイナリー・タイプのホスト変数の宣言

C、C++ における binary および varbinary ロケーター・ホスト変数の構文を次に示します。



例

宣言:

```
SQL TYPE IS BINARY(4) myBinField;
```

この結果、以下の C コードが生成されます。

```
 unsigned char myBinField[4];

where length  N (1<= N <=255)
```

宣言:

```
SQL TYPE IS VARBINARY(12) myVarBinField;
```

この結果、以下の C コードが生成されます。

```
struct myVarBinField_t { sqluint16 length;
char data[12];
} myVarBinField;

Where length is N (1<= N <=32704)
```

組み込み SQL アプリケーションにおける BINARY および VARBINARY のサポート

BINARY および VARBINARY データ・タイプを組み込みアプリケーションで使用するには、宣言セクションに示されているとおりの適切なデータ・タイプを使用してください。BINARY データの場合、ユーザー定義変数にデータをコピーし、その変数を SQL ステートメントで使います。VARBINARY データの場合、長さを適切な値に設定した後にデータをコピーします。

以下は、これら 2 つのデータ・タイプを組み込みアプリケーションで使用する方法の例です。

```
EXEC SQL BEGIN DECLARE SECTION;
sql type is binary(50)  binary1 ;
sql type is varbinary(100)  binary2 ;
EXEC SQL END DECLARE SECTION;
char  strng1[50];
char  strng2[50];

memset( binary1, 0x00, sizeof(binary1) );
memset( binary2.data, 0x00, sizeof(binary2.data) );
strcpy( strng1, "AAAAAAZZZZZMMMMMMMMMJJJJJJJJJJJJJ" );
strcpy( strng2, "BBBBBBBBBBBBBBBCCCCCCCCCCCCDDDDDDDDEEEEEEEEEEK" );
memcpy( binary1, strng1, strlen(strng1) );
memcpy( binary2.data, strng2, strlen(strng2) );
binary2.length = strlen(binary2.data);

EXEC SQL INSERT INTO test1 VALUES ( :binary1, :binary2 );
```

データベースからの取得時、データ長は対応する構造体に適切に設定されます。

C および C++ 組み込み SQL アプリケーションにおける有効範囲解決およびクラス・メンバー演算子

組み込み SQL ステートメント内で、C++ 有効範囲解決演算子 '::' を使用したり、C および C++ メンバー演算子 '.' または '->' を使用したりすることはできません。これと同じことは、ローカル・ポインターまたは参照変数を使用することにより、簡単に行うことができます。ローカル・ポインターや参照変数は SQL ステートメントの外部に設定して使用する効力範囲内の変数を指定し、その後は SQL ステートメント内でこれを参照するために使用されます。以下に、正しい使用方法の例を示します。

```
EXEC SQL BEGIN DECLARE SECTION;
  char (& localName)[20] = ::name;
EXEC SQL END DECLARE SECTION;
EXEC SQL
  SELECT name INTO :localName FROM STAFF
  WHERE name = 'Sanders';
```

C および C++ 組み込み SQL アプリケーションにおける日本語または中国語 (繁体字) EUC、および UCS-2 に関する考慮事項

アプリケーション・コード・ページが日本語または中国語 (繁体字) EUC の場合、またはアプリケーションが UCS-2 データベースと接続されている場合、CONVERT オプションか NOCONVERT オプションのどちらか、および wchar_t または sqldbchar グラフィック・ホスト変数、または入力/出力 SQLDA を使用することにより、データベース・サーバーで GRAPHIC 列にアクセスできます。 このセクションで *DBCS 形式* に言及する場合、それは EUC データ用の UCS-2 コード化スキームを指します。次の 2 つのケースを考えてみてください。

- CONVERT オプションを使用する場合

  DB2 クライアントによって、グラフィック・データがワイド・キャラクター形式からご使用のアプリケーション・コード・ページに変換され、その後、入力 SQLDA をデータベース・サーバーに送信する前に UCS-2 に変換します。グラフィック・データはすべて、UCS-2 コード・ページ ID によってタグ付けされたデータベース・サーバーに送られます。混合文字データは、アプリケーション・コード・ページ ID によってタグ付けされます。 クライアントによってデータベースからグラフィック・データが取り出されると、そのグラフィック・データは UCS-2 コード・ページ ID によってタグ付けされます。 DB2 クライアントが、

データを UCS-2 からクライアント・アプリケーション・コード・ページへ変換し、さらにそれをワイド・キャラクター形式に変換します。 ホスト変数の代わりに入力 SQLDA を使用した場合は、グラフィック・データを必ずワイド・キャラクター形式でエンコードする必要があります。 このデータは UCS-2 に変換され、その後データベース・サーバーに送られます。上記の変換はパフォーマンスに影響を及ぼします。

- NOCONVERT オプションを使用する場合

  グラフィック・データは UCS-2 によってエンコードされ、 UCS-2 コード・ページでタグ付けされたものと DB2 からは見なされます。変換は行われません。 DB2 は、グラフィック・ホスト変数を単にバケットとして使用されるものと見なします。 NOCONVERT オプションを選択した場合、データベース・サーバーから取り出されるグラフィック・データは、 UCS-2 によってエンコードされたアプリケーションに渡されます。アプリケーション・コード・ページから UCS-2、および UCS-2 からアプリケーション・コード・ページへの変換は、すべてユーザーの責任で行うことになります。 UCS-2 としてタグ付けされたデータは、変換や置換なしでデータベース・サーバーに送られます。

変換を最小限に抑えるには、 NOCONVERT オプションを使用してアプリケーション内で変換を処理するか、または GRAPHIC 列を使用しないかのいずれかです。 `wchar_t` エンコード方式が 2 バイト Unicode のクライアント環境 (たとえば Windows 2000® または AIX バージョン 5.1 以上) の場合には、NOCONVERT オプションを使用して直接 UCS-2 で作業できます。この場合、ご使用のアプリケーションはビッグ・エンディアンとリトル・エンディアン・アーキテクチャーとの違いを扱わなければなりません。 DB2 データベース・システムは、NOCONVERT オプションを使用する場合、常に 2 バイト・ビッグ・エンディアンである `sqldbchar` を使用します。

UCS-2 への変換後 (NOCONVERT 指定の場合) や、ワイド・キャラクター形式への変換 (CONVERT 指定の場合) によって、IBM-eucJP/IBM-eucTW CS0 (7 ビット ASCII) データおよび IBM-eucJP CS2 (カタカナ) データをグラフィック・ホスト変数に割り当てることはしないでください。これは、どちらの EUC コード・セットの文字も UCS-2 から PC DBCS へと変換すると 1 バイト文字になってしまうためです。

通常、eucJP および eucTW は GRAPHIC データを UCS-2 として保管しますが、これらのデータベースにある GRAPHIC データは非 ASCII eucJP または eucTW データのままです。 特に、そのような GRAPHIC データに埋め込まれるスペースは、DBCS スペースです (UCS-2、U+3000 では表意文字スペースとも呼ばれます)。しかし、UCS-2 データベースの場合には、 GRAPHIC データに UCS-2 文字を含めることができ、スペースの埋め込みは UCS-2 スペース、U+0020 を使用して実行されます。アプリケーションのコーディングでは、UCS-2 データベースから UCS-2 データを検索する場合と、eucJP および eucTW データベースから UCS-2 データを検索する場合の違いに注意してください。

C および C++ 組み込み SQL アプリケーションでの、FOR BIT DATA 節を使用した変数値のバイナリー保管

標準的な C または C++ のストリング・タイプである 460 は、FOR BIT DATA に指定された列に使用しないでください。データベース・マネージャーは、NULL 文

字が検出されると、このデータ・タイプを切り捨てます。 VARCHAR (SQL タイプ 448) または CLOB (SQL タイプ 408) のどちらかの構造体を使用してください。

C および C++ 組み込み SQL アプリケーションにおけるホスト変数の初期化

C および C++ の宣言セクションでは、単一の行で複数の変数の宣言および初期化ができます。ただし、変数の初期化には、括弧ではなく、″=″ 記号を使用する必要があります。次に、宣言セクション内での初期化の正しい方法と誤った方法の例を示します。

```
EXEC SQL BEGIN DECLARE SECTION;
  short my_short_2 = 5;       /* correct   */
  short my_short_1(5);        /* incorrect */
EXEC SQL END DECLARE SECTION;
```

C および C++ 組み込み SQL アプリケーションのマクロ展開および DECLARE SECTION

C または C++ プリコンパイラーは、宣言セクション内の宣言で使用された C マクロを直接処理できません。代わりに、まず、外部 C プリプロセッサーでソース・ファイルをプリプロセスしなければなりません。これを実行するには、PREPROCESSOR オプションを使って、C プリプロセッサーを起動するためのコマンドをプリコンパイラーに指定します。

PREPROCESSOR オプションを指定すると、プリコンパイラーはまず、 SQL INCLUDE ステートメントで参照されているすべてのファイルの内容をソース・ファイルに結合させることによって、すべての SQL INCLUDE ステートメントを処理します。次にプリコンパイラーは、修正したソース・ファイルを入力として指定するコマンドを使用して、外部 C プリプロセッサーを起動します。プリプロセス済みのファイル (拡張子 .i によってプリコンパイラーは識別) は、プリコンパイルの残りのプロセスでの新しいソース・ファイルとして使用されます。

プリコンパイラーにより生成された任意の #line マクロは、元のソース・ファイルを参照することはありません。代わりに、プリプロセスされたファイルを参照します。コンパイラー・エラーを元のソース・ファイルに関連付けるには、プリプロセスされたファイルにコメントを含めるようにします。これにより、ヘッダー・ファイルを含むオリジナル・ソース・ファイルのあらゆるセクションを位置指定できます。通常、コメントを含めるようにするオプションは C プリプロセッサーで使用でき、 PREPROCESSOR オプションにより、指定するコマンドにこのオプションを含めることができます。 C プリプロセッサーには、#line マクロ自体を出力させてはなりません。これには、プリコンパイラーにより生成されたものが誤って混在している可能性があるためです。

マクロ展開の使用上の注意:

1. PREPROCESSOR オプションを使用して指定するコマンドには、すべての望むオプションを含めることができますが、入力ファイルの名前を含めることはできません。 たとえば、AIX 上の IBM C には、次のオプションを使用できます。

   ```
   xlC -P -DMYMACRO=1
   ```

2. プリコンパイラーは、このコマンドによって、拡張子 .i の付いたプリプロセス済みのファイルが生成されることを予期します。ただし、プリプロセス済みのフ

ファイルを生成するためにリダイレクトを使用することはできません。たとえば、以下のオプションを使用してプリプロセス済みファイルを生成することは**できません**。

```
xlC -E > x.i
```

3. 外部 C プリプロセッサーが検出したエラーは、元のソース・ファイルに対応する名前に拡張子 .err を付けたファイルにレポートされます。

たとえば、以下のようにソース・コード内でマクロ展開を使用することができます。

```
#define SIZE 3

EXEC SQL BEGIN DECLARE SECTION;
  char a[SIZE+1];
  char b[(SIZE+1)*3];
  struct
  {
    short length;
    char  data[SIZE*6];
  } m;
  SQL TYPE IS BLOB(SIZE+1) x;
  SQL TYPE IS CLOB((SIZE+2)*3) y;
  SQL TYPE IS DBCLOB(SIZE*2K) z;
EXEC SQL END DECLARE SECTION;
```

PREPROCESSOR オプションを使用した後は、上記の宣言は以下のように解決します。

```
EXEC SQL BEGIN DECLARE SECTION;
  char a[4];
  char b[12];
  struct
  {
    short length;
    char  data[18];
  } m;
  SQL TYPE IS BLOB(4) x;
  SQL TYPE IS CLOB(15) y;
  SQL TYPE IS DBCLOB(6144) z;
EXEC SQL END DECLARE SECTION;
```

C および C++ 組み込み SQL アプリケーションの宣言セクションにおけるホスト構造のサポート

ホスト構造体サポートを使用すると、C または C++ プリコンパイラーは、複数のホスト変数を単一のホスト構造体にグループ化することができます。この機能により、SQL ステートメントで同じセットのホスト変数を参照するのが簡単になります。たとえば、以下のホスト構造体は、SAMPLE データベース内の STAFF 表のいくつかの列へのアクセスに使用できます。

```
struct tag
   {
     short id;
     struct
     {
       short length;
       char  data[10];
     } name;
     struct
     {
```

```
    short   years;
    double salary;
  } info;
} staff_record;
```

ホスト構造体のフィールドは、有効な任意のホスト変数タイプにすることができます。有効なタイプには、すべての数字、文字、およびラージ・オブジェクト・タイプが含まれます。ネストされるホスト構造体は、25 レベルまでサポートされます。上の例では、フィールド info は下位の構造体であるのに対し、フィールド name は下位の構造体ではなく、VARCHAR フィールドを示しています。 同じ原則は、LONG VARCHAR、 VARGRAPHIC および LONG VARGRAPHIC にも当てはまります。ホスト構造体へのポインターもサポートされます。

SQL ステートメントでホスト構造体にグループ化されるホスト変数を参照するには、以下の 2 つの方法があります。

- SQL ステートメントでホスト構造体名を参照する。

  ```
  EXEC SQL SELECT id, name, years, salary
       INTO :staff_record
       FROM staff
       WHERE id = 10;
  ```

  プリコンパイラーは staff_record の参照を、ホスト構造体で宣言されたすべてのフィールドをコンマで区切ったリストに変換します。他のホスト変数またはフィールドとの名前の重複を避けるために、それぞれのフィールドは、すべてのレベルのホスト構造体名で修飾されます。これは以下の使用法と同じです。

- SQL ステートメントで完全修飾ホスト変数名を参照する。

  ```
  EXEC SQL SELECT id, name, years, salary
       INTO :staff_record.id, :staff_record.name,
            :staff_record.info.years, :staff_record.info.salary
       FROM staff
       WHERE id = 10;
  ```

  同じ名前のホスト変数が他にない場合でも、フィールド名を参照する際には完全修飾しなければなりません。修飾された下位の構造体も参照できます。上の例では、:staff_record.info.years, :staff_record.info.salary を、:staff_record.info に置換することができます。

ホスト構造体への参照 (1 番目の例) は、コンマで区切ったフィールドのリストと等しいため、このタイプの参照はエラーとなる場合があります。 以下に例を示します。

```
EXEC SQL DELETE FROM :staff_record;
```

ここでの DELETE ステートメントは、1 バイト文字ベースのホスト変数を想定しています。代わりにホスト構造体を指定すると、ステートメントはプリコンパイル時にエラーになる可能性があります。

```
SQL0087N  Host variable "staff_record" is a structure used where structure
references are not permitted.
```

SQL0087N エラーの原因となる可能性があるホスト構造体のこの他の使用には、PREPARE、EXECUTE IMMEDIATE、CALL、標識変数、および SQLDA 参照などがあります。このような状況では、個々のフィールドへの参照と同じように (2 番目の例)、フィールドを 1 つしか持たないホスト構造体なら許可されます。

C および C++ 組み込み SQL アプリケーションでの、ヌルまたは切り捨て標識変数および標識表

ヌル値を受け取る可能性があるそれぞれのホスト変数では、**標識変数**を short データ・タイプとして宣言しなければなりません。

標識表は、ホスト構造体で使用される標識変数の集合です。これは、短整数の配列として宣言しなければなりません。以下に例を示します。

```
short ind_tab[10];
```

上の例は、エレメントが 10 個の標識表を宣言します。以下に、これを SQL ステートメントで使用する方法を示します。

```
EXEC SQL SELECT id, name, years, salary
     INTO :staff_record INDICATOR :ind_tab
     FROM staff
     WHERE id = 10;
```

以下の表では、それぞれのホスト構造体フィールドとそれに対応する標識変数をリストしています。

staff_record.id
ind_tab[0]

staff_record.name
ind_tab[1]

staff_record.info.years
ind_tab[2]

staff_record.info.salary
ind_tab[3]

注: 標識表エレメント、例えば ind_tab[1] は、SQL ステートメントで個々に参照することはできません。キーワード INDICATOR はオプションです。構造化フィールドと標識の数が一致している必要はありません。余分の標識が未使用だったり、標識が割り当てられていない余分のフィールドがあってもかまいません。

標識表の代わりにスカラー標識変数を使用して、ホスト構造体の最初のフィールドに標識を提供することもできます。これは、1 つのエレメントだけの標識表を持つことと同じです。 以下に例を示します。

```
short scalar_ind;

EXEC SQL SELECT id, name, years, salary
          INTO :staff_record INDICATOR :scalar_ind
          FROM staff
          WHERE id = 10;
```

ホスト構造体の代わりにホスト変数を指定して標識表を指定すると、標識表の最初のエレメント、例えば ind_tab[0] しか使用されません。

```
EXEC SQL SELECT id
          INTO :staff_record.id INDICATOR :ind_tab
          FROM staff
          WHERE id = 10;
```

短整数の配列がホスト構造体内で宣言される場合、以下のようになります。

```
struct tag
{
  short i[2];
} test_record;
```

SQL ステートメントで test_record が参照されるときに配列がエレメントに展開されると、 :test_record は、:test_record.i[0], :test_record.i[1] と同等になります。

C および C++ 組み込み SQL アプリケーションにおけるヌル終了ストリング

C および C++ のヌル終了ストリングは、独自の SQLTYPE (文字の場合は 460/461 で、グラフィックの場合は 468/469) を持ちます。

C および C++ のヌル終了ストリングは、 LANGLEVEL プリコンパイラー・オプションの値に基づいてさまざまに処理されます。これらの SQLTYPE 値のうちの 1 つのホスト変数と宣言された長さ *n* を SQL ステートメント内に指定し、さらにデータのバイト数 (文字タイプの場合) または 2 バイト文字 (グラフィック・タイプの場合) が *k* である場合を以下に説明します。

- PREP コマンドの LANGLEVEL オプションが SAA1 (デフォルト) の場合:

  出力の場合:

  k と n の関係 ...
  結果 ...

  k **>** *n*
  n 文字はターゲットのホスト変数に移動され、 SQLWARN1 は 'W'、および SQLCODE 0 (SQLSTATE 01004) に設定される。NULL 終止符はストリング内では使用されません。標識変数をホスト変数で指定した場合には、標識変数の値は *k* に設定されます。

  k **=** *n*
  k 文字はターゲットのホスト変数に移動され、 SQLWARN1 は 'N'、および SQLCODE 0 (SQLSTATE 01004) に設定される。NULL 終止符はストリング内では使用されません。標識変数をホスト変数で指定した場合には、標識変数の値は 0 に設定されます。

  k **<** *n*
  k 文字はターゲットのホスト変数に移動され、NULL 文字が *k* + *1* 文字に置かれます。標識変数をホスト変数で指定した場合には、標識変数の値は 0 に設定されます。

  入力の場合:
  データベース・マネージャーは、最後が NULL 終止符ではないこれらの SQLTYPE 値のうちの 1 つの入力ホスト変数を発見すると、文字 *n+1* に NULL 終止符文字が含まれると想定します。

- PREP コマンドの LANGLEVEL オプションが MIA である場合:

  出力の場合:

  k と n の関係 ...
  結果 ...

  k **>=** *n*
  n - 1 文字はターゲットのホスト変数に移動され、 SQLWARN1

は 'W'、および SQLCODE 0 (SQLSTATE 01501) に設定される。 *n* 番目の文字は NULL 終止符に設定されます。標識変数をホスト変数で指定した場合には、標識変数の値は *k* に設定されます。

k + 1 = n
k 文字はターゲットのホスト変数に移動され、 NULL 終止符は文字 *n* に置かれる。標識変数をホスト変数で指定した場合には、標識変数の値は 0 に設定されます。

k + 1 < n
k 文字はターゲットのホスト変数に移動され、文字 *k + 1* で開始している右側に *n - k -1* 個のブランクが追加される。その後、NULL 終止符が文字 *n* に置かれます。標識変数をホスト変数で指定した場合には、標識変数の値は 0 に設定されます。

入力の場合:
データベース・マネージャーが、最後が NULL 文字ではないこれらの SQLTYPE 値のうちの 1 つの入力ホスト変数を発見すると、 SQLCODE -302 (SQLSTATE 22501) が戻されます。

長さが *n* の SQLTYPE 460 のホスト変数を他の SQL コンテキスト内に指定すると、上で定義したように、長さ *n* の VARCHAR データ・タイプとして処理されます。長さが *n* の SQLTYPE 468 のホスト変数をその他の SQL コンテキスト内に指定した場合には、上での定義のように、長さ *n* の VARGRAPHIC データ・タイプとして処理されます。

COBOL のホスト変数

ホスト変数は、SQL ステートメント内で参照される COBOL の言語変数です。これにより、アプリケーションがデータベース・マネージャーとデータを交換することができます。 アプリケーションがプリコンパイルされると、コンパイラーはホスト変数を他の COBOL 変数と同様に使用します。 ホスト変数の命名、宣言、および使用は、以下の節で述べる規則に従って行ってください。

COBOL におけるホスト変数の名前

SQL プリコンパイラーは、宣言された名前によってホスト変数を識別します。以下の規則が適用されます。

- 変数名は 255 文字以内の長さで指定する。
- ホスト変数名は、SQL、sql、DB2、または db2 以外の接頭部で開始する。これらはシステムが使用する予約語です。
- これから説明する宣言構文を使用する FILLER 項目はグループ・ホスト変数宣言では許可されており、プリコンパイラーはその項目を無視します。ただし、SQL DECLARE セクション内で FILLER を複数回使用した場合、プリコンパイラーは失敗します。 VARCHAR、LONG VARCHAR、VARGRAPHIC、または LONG VARGRAPHIC 宣言には、FILLER 項目を組み込むことができません。
- ハイフンは、ホスト変数名として使用できます。

  SQL は、スペースで囲まれたハイフンを減算演算子として解釈します。ハイフンをホスト変数名として使用する場合は、スペースを入れないでください。

- REDEFINES 節は、ホスト変数宣言では許可されています。
- レベル-88 宣言は、ホスト変数宣言セクションでは許可されていますが、無視されます。

COBOL 組み込み SQL アプリケーションにおけるホスト変数の宣言セクション

ホスト変数宣言の識別には、SQL の宣言セクションを使用しなければなりません。このセクションにより、それ以降の SQL ステートメントで参照が可能なホスト変数をプリコンパイラーに知らせます。以下に例を示します。

```
     EXEC SQL BEGIN DECLARE SECTION END-EXEC.
77 dept              pic s9(4) comp-5.
01 userid            pic x(8).
01 passwd.
     EXEC SQL END DECLARE SECTION END-EXEC.
```

COBOL プリコンパイラーは、有効な COBOL 宣言のサブセットのみを認識します。

例: COBOL 組み込み SQL アプリケーション用の SQL 宣言セクション・テンプレート

サポートされている SQL データ・タイプそれぞれについて宣言されたホスト変数を含んだ、 SQL 宣言のサンプルを次に示します。

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
*
  01 age        PIC S9(4) COMP-5.          /* SQL type  500 */
  01 divis      PIC S9(9) COMP-5.          /* SQL type  496 */
  01 salary     PIC S9(6)V9(3) COMP-3.     /* SQL type  484 */
  01 bonus      USAGE IS COMP-1.           /* SQL type  480 */
  01 wage       USAGE IS COMP-2.           /* SQL type  480 */
  01 nm         PIC X(5).                  /* SQL type  452 */
  01 varchar.
     49 leng    PIC S9(4) COMP-5.          /* SQL type  448 */
     49 strg    PIC X(14).                 /* SQL type  448 */
  01 longvchar.
     49 len     PIC S9(4) COMP-5.          /* SQL type  456 */
     49 str     PIC X(6027).               /* SQL type  456 */
  01 MY-CLOB USAGE IS SQL TYPE IS CLOB(1M).                 /* SQL type  408 */
  01 MY-CLOB-LOCATOR USAGE IS SQL TYPE IS CLOB-LOCATOR.     /* SQL type  964 */
  01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE.           /* SQL type  920 */
  01 MY-BLOB USAGE IS SQL TYPE IS BLOB(1M).                 /* SQL type  404 */
  01 MY-BLOB-LOCATOR USAGE IS SQL TYPE IS BLOB-LOCATOR.     /* SQL type  960 */
  01 MY-BLOB-FILE USAGE IS SQL TYPE IS BLOB-FILE.           /* SQL type  916 */
  01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(1M).             /* SQL type  412 */
  01 MY-DBCLOB-LOCATOR USAGE IS SQL TYPE IS DBCLOB-LOCATOR. /* SQL type  968 */
  01 MY-DBCLOB-FILE USAGE IS SQL TYPE IS DBCLOB-FILE.       /* SQL type  924 */
  01 MY-PICTURE PIC G(16000) USAGE IS DISPLAY-1.            /* SQL type  464 */
  01 dt         PIC X(10).                 /* SQL type  384 */
  01 tm         PIC X(8).                  /* SQL type  388 */
  01 tmstmp     PIC X(26).                 /* SQL type  392 */
  01 wage-ind   PIC S9(4) COMP-5.          /* SQL type  464 */
*
EXEC SQL END DECLARE SECTION END-EXEC.
```

COBOL 組み込み SQL アプリケーションにおける BINARY/COMP-4 データ・タイプ

DB2 COBOL プリコンパイラーは、整数ホスト変数および標識が許可されている場所であれば、BINARY、COMP、および COMP-4 データ・タイプの使用をサポート

します。ただしそれは、ターゲット COBOL コンパイラーが BINARY、 COMP、または COMP-4 データ・タイプを、COMP-5 データ・タイプと等しく見なす (または等しく見なすようにできる) 場合に限ります。提供している例では、そのようなホスト変数および標識を、タイプ COMP-5 と示します。COMP、COMP-4、BINARY COMP および COMP-5 を等価として扱う、 DB2 のサポートするターゲット・コンパイラーは、次のとおりです。

- IBM COBOL Set for AIX
- Micro Focus COBOL for AIX

COBOL 組み込み SQL アプリケーションにおける SQLSTATE および SQLCODE 変数

LANGLEVEL プリコンパイル・オプションを SQL92E の値とともに使用すると、次の 2 つの宣言をホスト変数として組み込めます。

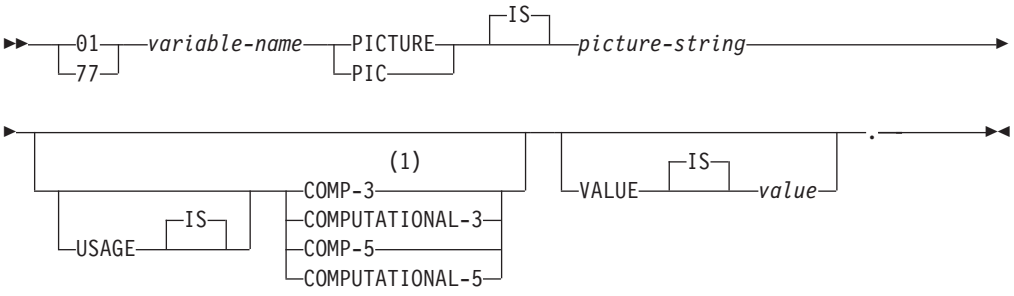
```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 SQLSTATE PIC X(5).
01 SQLCODE  PIC S9(9) USAGE COMP.
.
.
.
EXEC SQL END DECLARE SECTION END-EXEC.
```

これらのいずれも指定しない場合は、SQLCODE 宣言はプリコンパイル中であると見なされます。 SQLCODE および SQLSTATE 変数は、レベル 01 (上の例) またはレベル 77 を使用して宣言できます。このオプションを使用するときには、INCLUDE SQLCA ステートメントを指定してはならないことに注意してください。

複数のソース・ファイルから成るアプリケーションでは、 SQLCODE および SQLSTATE 定義を上の例に示された最初のソース・ファイルで定義することができます。

COBOL 組み込み SQL アプリケーションにおける数値ホスト変数の宣言

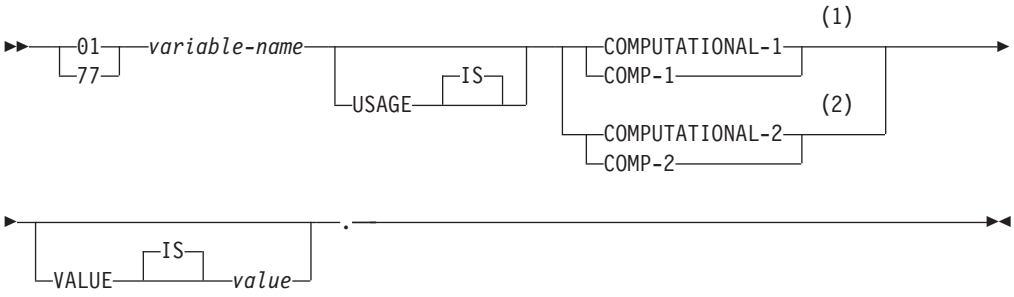
数値ホスト変数の構文を次に示します。



注:

1 COMP-3 の代わりに PACKED-DECIMAL を使用できます。

浮動小数点



注:

1 REAL (SQLTYPE 480)、長さ 4

2 DOUBLE (SQLTYPE 480)、長さ 8

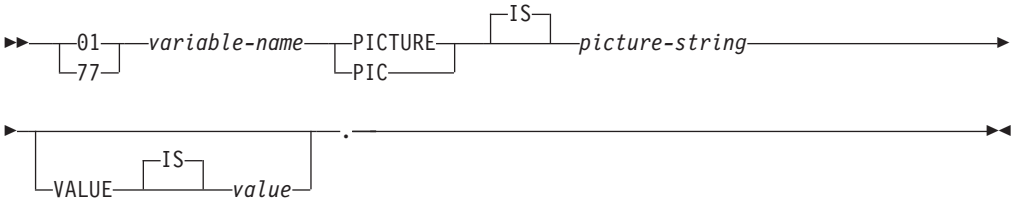
数値ホスト変数に関する考慮事項:

1. *Picture-string* は、以下のいずれかの形式でなければならない。
   - S9(m)V9(n)
   - S9(m)V
   - S9(m)
2. 数字の 9 は拡張することができる (たとえば、″S9(3)″ の代わりに ″S999″ とすることができます)。
3. *m* および *n* は、正の整数でなければならない。

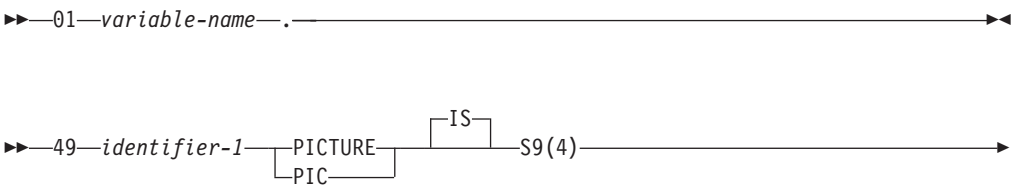
COBOL 組み込み SQL アプリケーションにおける固定長および可変長文字ホスト変数の宣言

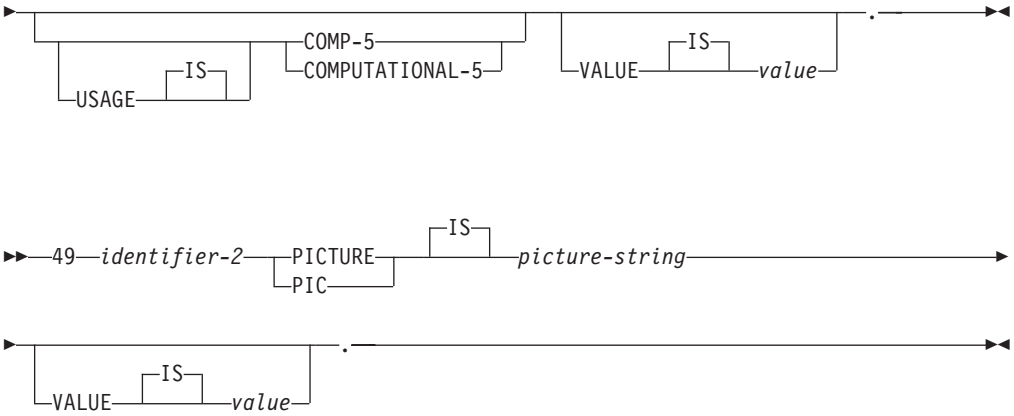
文字ホスト変数の構文を次に示します。

固定長



可変長





文字ホスト変数に関する考慮事項:

1. *Picture-string* の形式は、*X(m)* でなければならない。 X は拡張できます (たとえば、″X(3)″ の代わりに ″XXX″ にできます)。
2. *m* は、1 から 254 までの長さの固定長ストリングである。
3. *m* は、1 から 32 700 までの長さの可変長ストリングである。
4. *m* が 32 672 よりも大きい場合、そのホスト変数は LONG VARCHAR ストリングと見なされ、使用が制限される。
5. X および 9 を、PICTURE 節内のピクチャー文字として使用する。他の文字は使用できません。
6. 可変長ストリングは、長さ項目と値項目とから構成される。適切な COBOL 名を、長さ項目およびストリング項目として使用できます。ただし、SQL ステートメント内の集合名を使用して可変長ストリングを参照してください。
7. 以下の例に示すような CONNECT ステートメントでは、 COBOL 文字ストリング・ホスト変数 `dbname` および `userid` の後続ブランクは、処理前に削除されます。

   ```
   EXEC SQL CONNECT TO :dbname USER :userid USING :p-word
   END-EXEC.
   ```

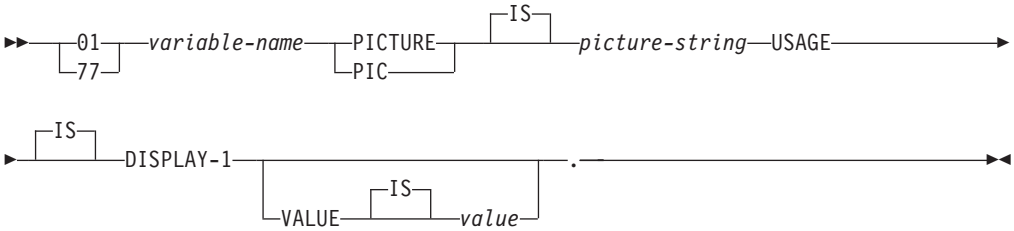
   ただし、パスワードではブランクが有効なため、 `p-word` ホスト変数を VARCHAR データ項目として宣言する必要があります。こうすることにより、アプリケーションは CONNECT ステートメントのパスワードの有効な長さを明示的に指示することができます。以下はその例です。

   ```
   EXEC SQL BEGIN DECLARE SECTION END-EXEC.
   01 dbname PIC X(8).
   01 userid PIC X(8).
   01 p-word.
       49 L PIC S9(4) COMP-5.
       49 D PIC X(18).
   EXEC SQL END DECLARE SECTION END-EXEC.
   PROCEDURE DIVISION.
       MOVE "sample" TO dbname.
       MOVE "userid" TO userid.
       MOVE "password" TO D OF p-word.
       MOVE 8          TO L of p-word.
   EXEC SQL CONNECT TO :dbname USER :userid USING :p-word
   END-EXEC.
   ```

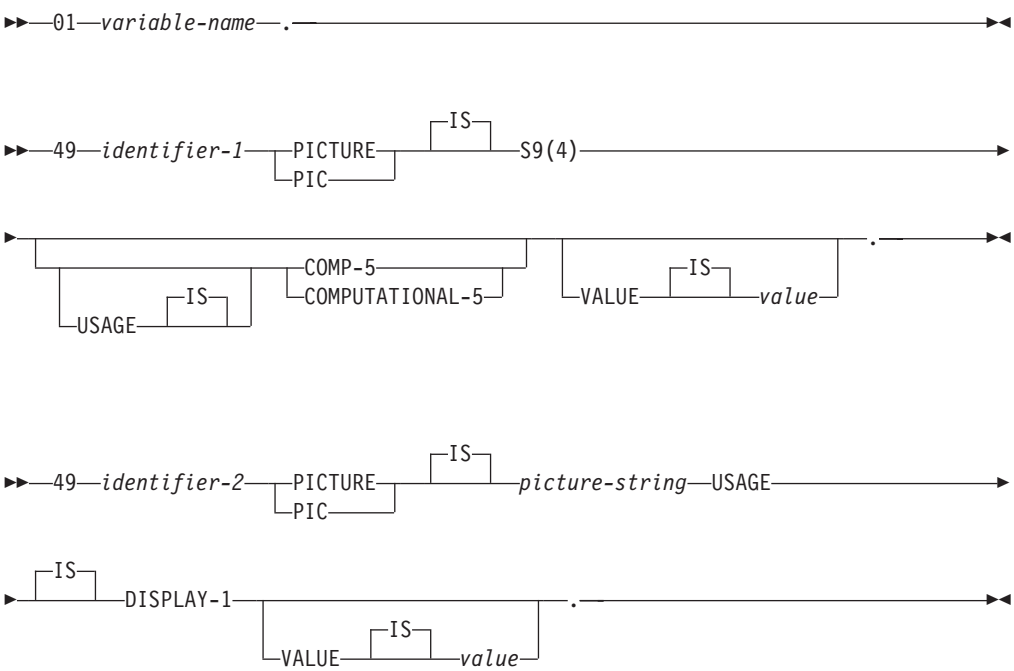
COBOL 組み込み SQL アプリケーションにおける固定長および可変長グラフィック・ホスト変数の宣言

グラフィック・ホスト変数の構文を次に示します。

固定長



可変長

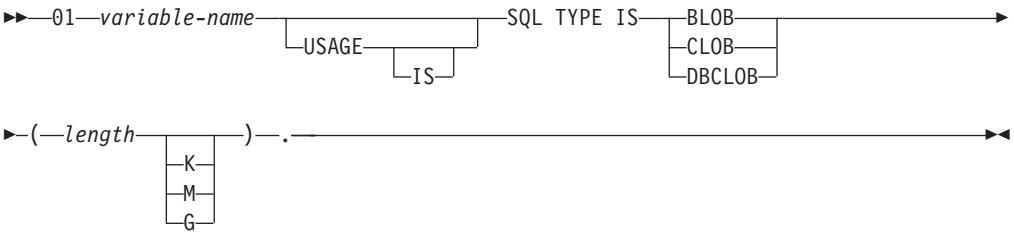


グラフィック・ホスト変数に関する考慮事項:

1. *Picture-string* の形式は、 *G(m)* でなければならない。 G は拡張できます (たとえば、″G(3)″ の代わりに ″GGG″ とできます)。
2. *m* は、1 から 127 までの長さの固定長ストリングである。
3. *m* は、1 から 16 350 までの長さの可変長ストリングである。
4. *m* が 16 336 よりも大きい場合、そのホスト変数は LONG VARGRAPHIC ストリングと見なされ、使用が制限される。

COBOL 組み込み SQL アプリケーションでのラージ・オブジェクト・タイプのホスト変数の宣言

COBOL におけるラージ・オブジェクト (LOB) ホスト変数の宣言構文を次に示します。



LOB ホスト変数に関する考慮事項:

1. BLOB および CLOB の場合は、1 <= lob-length <= 2 147 483 647 である。
2. DBCLOB の場合は、1 <= lob-length <= 1 073 741 823 である。
3. SQL TYPE IS、BLOB、CLOB、DBCLOB、K、M、G は、大文字、小文字、またはその混合のいずれでもかまわない。
4. LOB 宣言内での初期化はできない。
5. ホスト変数名により、プリコンパイラー生成コード内の LENGTH および DATA に接頭語が付けられる。

BLOB の例:

宣言:

```
01 MY-BLOB USAGE IS SQL TYPE IS BLOB(2M).
```

この結果、以下の構造が生成されます。

```
01 MY-BLOB.
   49 MY-BLOB-LENGTH PIC S9(9) COMP-5.
   49 MY-BLOB-DATA PIC X(2097152).
```

CLOB の例:

宣言:

```
01 MY-CLOB USAGE IS SQL TYPE IS CLOB(125M).
```

この結果、以下の構造が生成されます。

```
01 MY-CLOB.
   49 MY-CLOB-LENGTH PIC S9(9) COMP-5.
   49 MY-CLOB-DATA PIC X(131072000).
```

DBCLOB の例:

宣言:

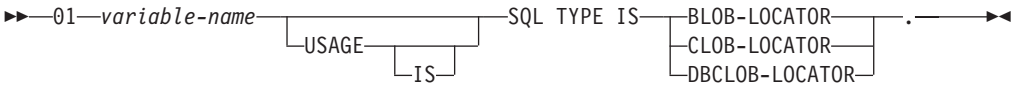
```
01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(30000).
```

この結果、以下の構造が生成されます。

```
01 MY-DBCLOB.
   49 MY-DBCLOB-LENGTH PIC S9(9) COMP-5.
   49 MY-DBCLOB-DATA PIC G(30000) DISPLAY-1.
```

COBOL 組み込み SQL アプリケーションでのラージ・オブジェクト・ロケーター・タイプのホスト変数の宣言

COBOL におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の宣言構文を次に示します。



LOB ロケーター・ホスト変数に関する考慮事項:

1. SQL TYPE IS、BLOB-LOCATOR、CLOB-LOCATOR、DBCLOB-LOCATOR は、大文字、小文字、またはその混合のいずれでもかまわない。
2. ロケーターの初期化はできない。

BLOB ロケーターの例 (他のタイプの LOB ロケーターの場合も同様):

宣言:

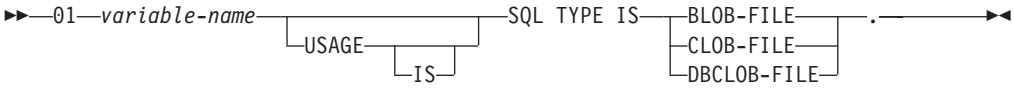
```
01 MY-LOCATOR USAGE SQL TYPE IS BLOB-LOCATOR.
```

この結果、以下の宣言が生成されます。

```
01 MY-LOCATOR PIC S9(9) COMP-5.
```

COBOL 組み込み SQL アプリケーションでのファイル参照タイプのホスト変数の宣言

COBOL におけるファイル参照ホスト変数の宣言構文を次に示します。



- SQL TYPE IS、BLOB-FILE、CLOB-FILE、DBCLOB-FILE は、大文字、小文字、またはその混合のいずれでもかまわない。

BLOB ファイル参照の例 (他のタイプの LOB の場合も同様):

宣言:

```
01 MY-FILE USAGE IS SQL TYPE IS BLOB-FILE.
```

この結果、以下の宣言が生成されます。

```
01 MY-FILE.
   49 MY-FILE-NAME-LENGTH PIC S9(9) COMP-5.
   49 MY-FILE-DATA-LENGTH PIC S9(9) COMP-5.
   49 MY-FILE-FILE-OPTIONS PIC S9(9) COMP-5.
   49 MY-FILE-NAME PIC X(255).
```

COBOL 組み込み SQL アプリケーションにおける REDEFINES を使用したデータ項目のグループ化

ホスト変数の宣言に、REDEFINES 節を使用することができます。 REDEFINES 節を使っていくつかのグループ・データ項目を宣言し、そのグループ・データ項目が SQL ステートメント内で全体として参照される場合、 REDEFINES 節を含む従属項目は展開されません。 以下に例を示します。

```
01 foo.
 10 a pic s9(4) comp-5.
 10 a1 redefines a pic x(2).
 10 b pic x(10).
```

SQL ステートメント内での foo の参照は、次のようになります。

```
... INTO :foo ...
```

上のステートメントは、次のステートメントと等価です。

```
... INTO :foo.a, :foo.b ...
```

つまり、従属項目 a1 は、REDEFINES 節で宣言されていますが、そのような状況では自動的に展開されません。 a1 があいまいでない場合は、次のように、SQL ステートメント内の REDEFINES 節で明示的に従属項目を参照することができます。

```
... INTO :foo.a1 ...
```

または

```
... INTO :a1 ...
```

COBOL 組み込み SQL アプリケーションでの日本語または中国語 (繁体字) EUC、および UCS-2 に関する考慮事項

eucJp または eucTW コード・セットで実行されている、または UCS-2 データベースに接続されているアプリケーションから送られてくるグラフィック・データはすべて、 UCS-2 コード・ページ ID でタグ付けされます。アプリケーションの側では、グラフィック文字ストリングをデータベース・サーバーに送る前に UCS-2 に変換しておく必要があります。同様に、UCS-2 データベースからアプリケーションが取り出すグラフィック・データ、または EUC eucJP または eucTW コード・ページで実行されているアプリケーションがデータベースから取り出すグラフィック・データも、UCS-2 を使用してエンコードされます。このため、アプリケーションの側では、UCS-2 データで表示しようとする場合を除き、内部的に UCS-2 からご使用のアプリケーションのコード・ページに変換する必要があります。

このような変換は、SQLDA へのデータのコピー前、および SQLDA からのデータのコピー後に実行する必要があるため、 UCS-2 への変換および UCS-2 からの変換は、ご使用のアプリケーションが担当することになります。DB2 Database for Linux, UNIX, and Windows では、アプリケーションからアクセス可能な変換ルーチンは提供していません。その代わりに、ご使用のオペレーティング・システムから呼び出し可能なシステム・コールを使用してください。 UCS-2 データベースの場合は、VARCHAR および VARGRAPHIC スカラー関数の使用を考慮することができます。

COBOL 組み込み SQL アプリケーションでの、FOR BIT DATA 節を使用した変数値のバイナリー保管

一定のデータベース列には FOR BIT DATA を宣言できます。通常は文字を含むこれらの列は、バイナリー情報を保持するために使用されます。バイナリー・データを含めることのできる COBOL ホスト変数のタイプは、 CHAR(*n*)、VARCHAR、LONG VARCHAR、および BLOB データ・タイプです。これらのデータ・タイプは、FOR BIT DATA 属性の列を処理する場合に使用してください。

COBOL 組み込み SQL アプリケーションの宣言セクションにおけるホスト構造のサポート

COBOL プリコンパイラーは、ホスト変数宣言セクション内のグループ・データ項目をサポートします。グループ・データ項目は特に、 SQL ステートメント内の基本データ項目の集合を手早く参照する方法を提供します。たとえば、以下のグループ・データ項目は、 SAMPLE データベースの STAFF 表にある列にアクセスするために使用することができます。

```
01 staff-record.
    05 staff-id       pic s9(4) comp-5.
    05 staff-name.
        49 l          pic s9(4) comp-5.
        49 d          pic x(9).
    05 staff-info.
        10 staff-dept pic s9(4) comp-5.
        10 staff-job  pic x(5).
```

宣言セクション内のグループ・データ項目は、上記のホスト変数のタイプを従属データ項目として含むことができます。これには、すべてのタイプのラージ・オブジェクトの他に、数値および文字のすべてのタイプが含まれます。グループ・データ項目は、最高 10 レベルまでネストさせることができます。 上の例のように、レベル 49 の従属項目には VARCHAR 文字タイプを宣言しなければならないことに注意してください。レベルが 49 でない場合は、VARCHAR は 2 つの従属項目を持つグループ・データ項目と見なされ、グループ・データ項目の宣言および使用の規則に従います。上記の例では、`staff-info` はグループ・データ項目であり、`staff-name` は VARCHAR です。同じ原則は、LONG VARCHAR、VARGRAPHIC および LONG VARGRAPHIC にも当てはまります。グループ・データ項目は、02 から 49 までの範囲のどのレベルにおいても宣言することができます。

グループ・データ項目とその従属項目には、以下の 4 とおりの使用法があります。

使用法 1

グループ全体を SQL ステートメント内の単一のホスト変数として参照します。

```
EXEC SQL SELECT id, name, dept, job
  INTO :staff-record
  FROM staff WHERE id = 10 END-EXEC.
```

プリコンパイラーは staff-record の参照を、 `staff-record` 内で宣言されたすべての従属項目をコンマで区切ったリストへと変換します。 他の項目との名前の重複を避けるために、各基本項目はあらゆるレベルのグループ名により修飾されます。これは以下の使用法と同じです。

使用法 2

グループ・データ項目の 2 番目の使用法です。

```
EXEC SQL SELECT id, name, dept, job
  INTO
  :staff-record.staff-id,
  :staff-record.staff-name,
  :staff-record.staff-info.staff-dept,
  :staff-record.staff-info.staff-job
  FROM staff WHERE id = 10 END-EXEC.
```

注: staff-id への参照は、接頭部 staff-record. を使ったグループ名で修飾されており、純粋な COBOL の場合のように staff-record の staff-id によってではありません。

staff-record の従属項目と同じ名前を持つホスト変数がその他にない場合は、上記のステートメントは使用法 3 と同様に、明示的なグループ修飾を取り除いてコーディングできます。

使用法 3

ここでは、特定のグループ項目を修飾しない、通常の COBOL の方式で従属項目が参照されています。

```
EXEC SQL SELECT id, name, dept, job
  INTO
  :staff-id,
  :staff-name,
  :staff-dept,
  :staff-job
  FROM staff WHERE id = 10 END-EXEC.
```

純粋な COBOL と同様に、特定の従属項目が固有に識別できれば、この方法はプリコンパイラーに受け入れられます。たとえば、staff-job が複数のグループに現れるとすると、プリコンパイラーはあいまいな参照であることを示すエラーを出します。

```
SQL0088N Host variable "staff-job" is ambiguous.
```

使用法 4

あいまい参照を解決するために、従属項目の部分修飾を使用することができます。たとえば、以下のようにします。

```
EXEC SQL SELECT id, name, dept, job
  INTO
  :staff-id,
  :staff-name,
  :staff-info.staff-dept,
  :staff-info.staff-job
  FROM staff WHERE id = 10 END-EXEC.
```

使用法 1 のような単一のグループ項目のみの参照は、コンマで区切った従属項目のリストに対応するため、このタイプの参照はエラーとなる場合があります。以下に例を示します。

```
EXEC SQL CONNECT TO :staff-record END-EXEC.
```

ここで、CONNECT ステートメントでは、1 バイト文字ベースのホスト変数が想定されています。 staff-record グループ・データ項目を与えると、このホスト変数は以下のようなプリコンパイル・エラーとなります。

```
SQL0087N Host variable "staff-record" is a structure used where
         structure references are not permitted.
```

この他に SQL0087N を引き起こすグループ項目の使用法には、 PREPARE、 EXECUTE IMMEDIATE、 CALL、標識変数、および SQLDA 参照を含むものがあります。 このような状態では、前述の使用法 2、3 および 4 での個々の従属項目の参照がそうであるように、従属項目を 1 つしか持たないグループを使用することができます。

COBOL 組み込み SQL アプリケーションにおける NULL 標識変数と NULL または切り捨て標識変数の表

NULL 標識変数は、PIC S9(4) COMP-5 データ・タイプとして宣言します。

COBOL プリコンパイラーは、グループ・データ項目での使用に便利な *NULL 標識変数の表* (標識表として知られる) の宣言をサポートします。以下のように宣言します。

```
01 <indicator-table-name>.
    05 <indicator-name> pic s9(4) comp-5
                        occurs <table-size> times.
```

以下に例を示します。

```
01 staff-indicator-table.
    05 staff-indicator pic s9(4) comp-5
                       occurs 7 times.
```

この標識表は、上記のグループ項目の最初の形式で効率的に使用できます。

```
EXEC SQL SELECT id, name, dept, job
  INTO :staff-record :staff-indicator
  FROM staff WHERE id = 10 END-EXEC.
```

ここでは、プリコンパイラーが staff-indicator は標識表として宣言されていることを検出し、 SQL ステートメントの処理の際に、これを個々の標識の参照に拡張します。 staff-indicator(1) は staff-record の staff-id、 staff-indicator(2) は staff-record の staff-name、というように関連付けられます。

注: データ項目内の従属項目よりも k 個多い標識項目が標識表に存在する場合 (たとえば、staff-indicator に項目が 10 個ある場合は、 k=6 となります)、標識表の終端の k 個の余分な項目は無視されます。 同様に、標識項目が従属項目よりも k 個少ない場合は、グループ項目内の最後の k 個の項目は、対応する標識を持ちません。 *SQL ステートメント内の標識表にある個々のエレメントを参照できることに注意してください。*

FORTRAN のホスト変数

ホスト変数は、SQL ステートメント内で参照される FORTRAN 言語変数となります。これにより、アプリケーションがデータベース・マネージャーとデータを交換することができます。アプリケーションがプリコンパイルされると、コンパイラーはホスト変数を他の FORTRAN 変数として使用します。ホスト変数の命名、宣言、および使用は、以下の節で述べる規則に従って行ってください。

FORTRAN 組み込み SQL アプリケーションにおけるホスト変数の名前

SQL プリコンパイラーは、宣言された名前によってホスト変数を識別します。この場合、以下の規則が適用されます。

- 変数名は 255 文字以内の長さで指定する。
- ホスト変数名は、SQL、sql、DB2、または db2 以外の接頭部で開始する。これらはシステムが使用する予約語です。

FORTRAN 組み込み SQL アプリケーションにおけるホスト変数の宣言セクション

ホスト変数宣言の識別には、SQL の宣言セクションを使用しなければなりません。このようにして、それ以降の SQL ステートメントで参照が可能なホスト変数をプリコンパイラーに知らせます。

FORTRAN プリコンパイラーは、有効な FORTRAN 宣言のサブセットのみを有効なホスト変数宣言として認識します。これらの宣言は、数値変数または文字変数のいずれかを宣言します。数値ホスト変数は、数値の SQL 入出力値に対する入出力変数として使用することができます。文字ホスト変数は任意の文字、日付、時間またはタイム・スタンプの SQL 入出力値に対する入出力変数として使用できます。プログラマーは、出力変数が受け取る値を含めることのできる長さを持つようにコーディングしなければなりません。

例: FORTRAN 組み込み SQL アプリケーション用の SQL 宣言セクション・テンプレート

サポートされているデータ・タイプそれぞれについて宣言されたホスト変数を含む、 SQL 宣言のサンプルを以下に示します。

```
EXEC SQL BEGIN DECLARE SECTION
  INTEGER*2     AGE  /26/                       /* SQL type  500 */
  INTEGER*4     DEPT                           /* SQL type  496 */
  REAL*4        BONUS                          /* SQL type  480 */
  REAL*8        SALARY                         /* SQL type  480 */
  CHARACTER     MI                             /* SQL type  452 */
  CHARACTER*112 ADDRESS                        /* SQL type  452 */
  SQL TYPE IS VARCHAR (512) DESCRIPTION        /* SQL type  448 */
  SQL TYPE IS VARCHAR (32000) COMMENTS         /* SQL type  448 */
  SQL TYPE IS CLOB (1M) CHAPTER                /* SQL type  408 */
  SQL TYPE IS CLOB_LOCATOR CHAPLOC             /* SQL type  964 */
  SQL TYPE IS CLOB_FILE  CHAPFL                /* SQL type  920 */
  SQL TYPE IS BLOB (1M) VIDEO                  /* SQL type  404 */
  SQL TYPE IS BLOB_LOCATOR VIDLOC              /* SQL type  960 */
  SQL TYPE IS BLOB_FILE VIDFL                  /* SQL type  916 */
  CHARACTER*10  DATE                           /* SQL type  384 */
  CHARACTER*8   TIME                           /* SQL type  388 */
  CHARACTER*26  TIMESTAMP                      /* SQL type  392 */
  INTEGER*2     WAGE_IND                       /* SQL type  500 */
EXEC SQL END DECLARE SECTION
```

FORTRAN 組み込み SQL アプリケーションにおける SQLSTATE および SQLCODE 変数

LANGLEVEL プリコンパイル・オプションを SQL92E の値とともに使用すると、次の 2 つの宣言をホスト変数として組み込めます。

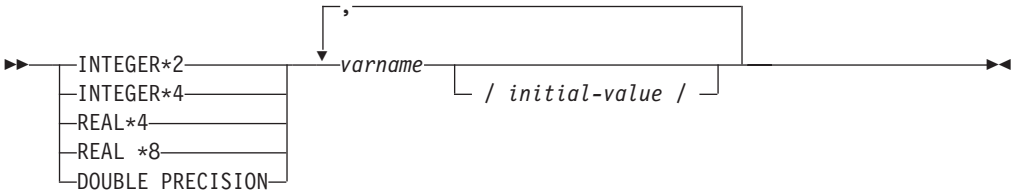
```
EXEC SQL BEGIN DECLARE SECTION;
  CHARACTER*5 SQLSTATE
  INTEGER     SQLCOD
  .
  .
  .
EXEC SQL END DECLARE SECTION
```

プリコンパイル・ステップの間、SQLCOD 宣言が仮定されます。変数には SQLSTATE または SQLSTA と名前が付けられます。 このオプションを使用するときには、INCLUDE SQLCA ステートメントを指定してはならないことに注意してください。

複数のソース・ファイルがあるアプリケーションの場合、各ソース・ファイルに SQLCOD と SQLSTATE の宣言が上記のように組み込まれることがあります。

FORTRAN 組み込み SQL アプリケーションにおける数値ホスト変数の宣言

FORTRAN における数値ホスト変数の構文を次に示します。



数値ホスト変数に関する考慮事項:

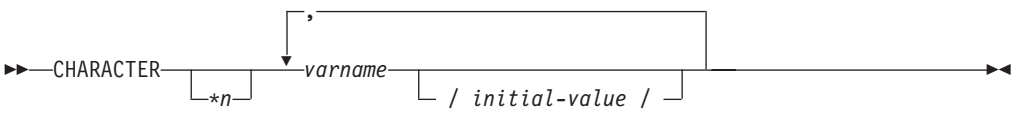
1. REAL*8 と DOUBLE PRECISION は同値である。
2. REAL*8 定数のインディケーターには、D ではなく E を使用する。

FORTRAN 組み込み SQL アプリケーションにおける固定長および可変長文字ホスト変数の宣言

固定長文字ホスト変数の構文を次に示します。

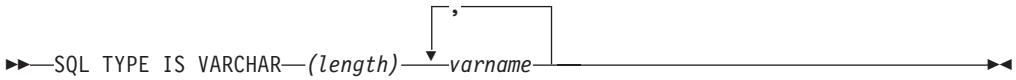
固定長

FORTRAN における文字ホスト変数の構文: 固定長



可変長文字ホスト変数の構文を次に示します。

可変長



文字ホスト変数に関する考慮事項:

1. *n* の最大値は 254。
2. 長さ (length) が 1 と 32 672 の間である場合、ホスト変数のタイプは VARCHAR(SQLTYPE 448)。
3. 長さ (length) が 32 673 と 32 700 の間である場合、ホスト変数のタイプは LONG VARCHAR(SQLTYPE 456)。
4. 宣言内で VARCHAR ホスト変数および LONG VARCHAR ホスト変数を初期設定することは許可されていない。

VARCHAR の例:

宣言:

```
sql type is varchar(1000) my_varchar
```

この結果、以下の構造が生成されます。

```
character    my_varchar(1000+2)
integer*2    my_varchar_length
character    my_varchar_data(1000)
equivalence( my_varchar(1),
+            my_varchar_length )
equivalence( my_varchar(3),
+            my_varchar_data )
```

アプリケーションは、たとえば、ホスト変数の内容の設定や検査のために、my_varchar_length および my_varchar_data の両方を処理できます。 SQL ステートメントでベース名 (ここでは、my_varchar) を使用して、 VARCHAR 全体を参照します。

LONG VARCHAR の例:

宣言:

```
sql type is varchar(10000) my_lvarchar
```

この結果、以下の構造が生成されます。

```
character    my_lvarchar(10000+2)
integer*2    my_lvarchar_length
character    my_lvarchar_data(10000)
equivalence( my_lvarchar(1),
+            my_lvarchar_length )
equivalence( my_lvarchar(3),
+            my_lvarchar_data )
```

アプリケーションは、たとえば、ホスト変数の内容の設定や検査のために、my_lvarchar_length および my_lvarchar_data の両方を処理できます。 SQL ステートメントでベース名 (ここでは、my_lvarchar) を使用して、 LONG VARCHAR 全体を参照します。

注: 以下の例に示すような CONNECT ステートメントでは、 FORTRAN 文字ストリング・ホスト変数 dbname および userid の後続ブランクは、処理前に削除されます。

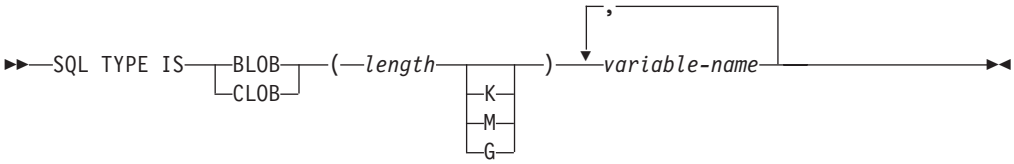
```
EXEC SQL CONNECT TO :dbname USER :userid USING :passwd
```

しかし、パスワードにはブランクも有効であるため、パスワードのホスト変数を VARCHAR として宣言し、実際のパスワードの長さを反映するように長さ (length) フィールドを設定しなければなりません。

```
EXEC SQL BEGIN DECLARE SECTION
  character*8 dbname, userid
  sql type is varchar(18) passwd
EXEC SQL END DECLARE SECTION
character*18 passwd_string
equivalence(passwd_data,passwd_string)
dbname = 'sample'
userid = 'userid'
passwd_length= 8
passwd_string = 'password'
EXEC SQL CONNECT TO :dbname USER :userid USING :passwd
```

FORTRAN 組み込み SQL アプリケーションでのラージ・オブジェクト・タイプのホスト変数の宣言

FORTRAN におけるラージ・オブジェクト (LOB) ホスト変数の宣言構文を次に示します。



LOB ホスト変数に関する考慮事項:

1. GRAPHIC タイプは、FORTRAN ではサポートされていない。
2. SQL TYPE IS、BLOB、CLOB、K、M、G は、大文字、小文字、またはその混合のいずれでもかまわない。
3. BLOB および CLOB の場合は、1 <= lob-length <= 2 147 483 647 である。
4. LOB 宣言内での LOB の初期化はできない。
5. ホスト変数名には、プリコンパイラー生成コードにおいて 'length' と 'data' という接頭部がある。

BLOB の例:

宣言:

```
sql type is blob(2m) my_blob
```

この結果、以下の構造が生成されます。

```
character    my_blob(2097152+4)
integer*4    my_blob_length
character    my_blob_data(2097152)
```

```
 equivalence( my_blob(1),
+             my_blob_length )
 equivalence( my_blob(5),
+             my_blob_data )
```

CLOB の例:

宣言:

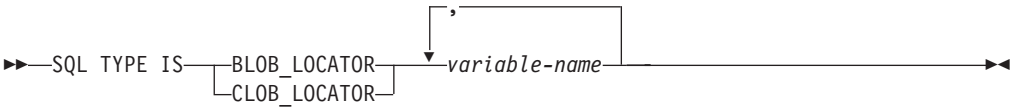
```
sql type is clob(125m) my_clob
```

この結果、以下の構造が生成されます。

```
 character    my_clob(131072000+4)
 integer*4    my_clob_length
 character    my_clob_data(131072000)
 equivalence( my_clob(1),
+             my_clob_length )
 equivalence( my_clob(5),
+             my_clob_data )
```

FORTRAN 組み込み SQL アプリケーションでのラージ・オブジェクト・ロケーター・タイプのホスト変数の宣言

FORTRAN におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の宣言構文を次に示します。



LOB ロケーター・ホスト変数に関する考慮事項:

1. GRAPHIC タイプは、FORTRAN ではサポートされていない。
2. SQL TYPE IS、BLOB_LOCATOR、CLOB_LOCATOR は、大文字、小文字、またはその混合のいずれでもかまわない。
3. ロケーターの初期化はできない。

CLOB ロケーターの例 (BLOB ロケーターと同様):

宣言:

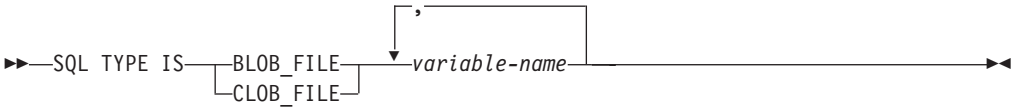
```
SQL TYPE IS CLOB_LOCATOR my_locator
```

この結果、以下の宣言が生成されます。

```
integer*4 my_locator
```

FORTRAN 組み込み SQL アプリケーションでのファイル参照タイプのホスト変数の宣言

FORTRAN におけるファイル参照ホスト変数の宣言構文を次に示します。



ファイル参照ホスト変数に関する考慮事項:

1. GRAPHIC タイプは、FORTRAN ではサポートされていない。
2. SQL TYPE IS、BLOB_FILE、CLOB_FILE は、大文字、小文字、またはその混合のいずれでもかまわない。

BLOB ファイル参照変数の例 (CLOB ファイル参照変数と同様):

```
SQL TYPE IS BLOB_FILE my_file
```

この結果、以下の宣言が生成されます。

```
 character     my_file(267)
 integer*4     my_file_name_length
 integer*4     my_file_data_length
 integer*4     my_file_file_options
 character*255 my_file_name
 equivalence(  my_file(1),
+              my_file_name_length )
 equivalence(  my_file(5),
+              my_file_data_length )
 equivalence(  my_file(9),
+              my_file_file_options )
 equivalence(  my_file(13),
+              my_file_name )
```

FORTRAN 組み込み SQL アプリケーションでのグラフィック (マルチバイト) 文字セットに関する考慮事項

FORTRAN では、グラフィック (マルチバイト) ホスト変数データ・タイプはサポートされていません。 character データ・タイプによって、混合文字ホスト変数だけがサポートされています。 ただし、グラフィック・データを含むユーザー SQLDA を作成することは可能です。

FORTRAN 組み込み SQL アプリケーションでの日本語または中国語 (繁体字) EUC、および UCS-2 に関する考慮事項

eucJp または eucTW コード・セットで実行されている、または UCS-2 データベースに接続されているアプリケーションから送られてくるグラフィック・データはすべて、 UCS-2 コード・ページ ID でタグ付けされます。アプリケーションの側では、グラフィック文字ストリングをデータベース・サーバーに送る前に UCS-2 に変換しておく必要があります。同様に、UCS-2 データベースからアプリケーションが取り出すグラフィック・データ、または EUC eucJP または eucTW コード・ページで実行されているアプリケーションがデータベースから取り出すグラフィック・データも、UCS-2 を使用してエンコードされます。このため、アプリケーションの側

では、UCS-2 データで表示しようとする場合を除き、内部的に UCS-2 からご使用のアプリケーションのコード・ページに変換する必要があります。

このような変換は、SQLDA へのデータのコピー前、および SQLDA からのデータのコピー後に実行する必要があるため、 UCS-2 への変換および UCS-2 からの変換は、ご使用のアプリケーションが担当することになります。DB2 データベース・システムでは、アプリケーションからアクセス可能な変換ルーチンは提供していません。その代わりに、ご使用のオペレーティング・システムから呼び出し可能なシステム・コールを使用してください。 UCS-2 データベースの場合は、VARCHAR および VARGRAPHIC スカラー関数の使用を考慮することができます。

FORTRAN 組み込み SQL アプリケーションでの NULL または切り捨て標識変数

標識変数は、INTEGER*2 データ・タイプとして宣言します。

REXX のホスト変数

ホスト変数は、SQL ステートメント内で参照される REXX の言語変数です。これにより、アプリケーションがデータベース・マネージャーとデータを交換することができます。アプリケーションがプリコンパイルされると、コンパイラーはホスト変数を他の REXX 変数と同様に使用します。ホスト変数の命名、宣言、および使用は、以下の節で述べる規則に従って行ってください。

REXX 組み込み SQL アプリケーションにおけるホスト変数の名前

正しく命名された REXX 変数は、すべてホスト変数として使用できます。変数名の長さは 64 文字までです。ピリオドを変数名の最後の文字として使用しないでください。ホスト変数名には、数字、英字、および @、_、!、.、?、$ といった文字を使用できます。

REXX 組み込み SQL アプリケーションでのホスト変数の参照

REXX インタープリターは、プロシージャー内のストリングで引用符で囲まれていないものをすべて検査します。ストリングが現行の REXX 変数プール内の変数を表している場合には、 REXX がそのストリングを現行値と置き換えます。以下に、REXX におけるホスト変数の参照方法を示します。

```
CALL SQLEXEC 'FETCH C1 INTO :cm'
SAY 'Commission = ' cm
```

文字ストリングが数値データ・タイプに変換されないようにするため、ストリングを以下の例のように単一引用符で囲んでください。

```
VAR = '100'
```

REXX は、3 バイトの文字ストリング 100 に変数 *VAR* をセットします。単一引用符がストリングの一部になっている場合は、次の例に従ってください。

```
VAR = "'100'"
```

CHARACTER フィールドに数値データを挿入する場合、REXX インタープリターは、数値データを整数データと見なします。したがって、数値ストリングを明示的に連結して、単一引用符で囲む必要があります。

事前定義された REXX 変数

SQLEXEC、SQLDBS および SQLDB2 は一定の操作の結果として、事前定義 REXX 変数をセットします。 それらの変数は以下のとおりです。

RESULT

各操作により、戻りコードがセットされます。可能な値は以下のとおりです。

n *n* は、フォーマットされたメッセージのバイト数を示す正の値です。この値を戻すのは GET ERROR MESSAGE API のみです。

0 API が実行されています。REXX 変数 SQLCA には、API の完了状況が含まれます。 SQLCA.SQLCODE がゼロでない場合は、その値に関連したテキスト・メッセージが SQLMSG に含まれます。

–1 API を完了するために十分なメモリーがありません。要求されたメッセージは戻されません。

–2 SQLCA.SQLCODE が 0 にセットされます。メッセージは戻されません。

–3 SQLCA.SQLCODE に無効な SQLCODE が含まれています。メッセージは戻されません。

–6 SQLCA REXX 変数が作成できません。これは、十分なメモリーがないか、または何らかの理由で REXX 変数プールが使用できないということを示します。

–7 SQLMSG REXX 変数が作成できません。これは、十分なメモリーがないか、または何らかの理由で REXX 変数プールが使用できないということを示します。

–8 REXX 変数プールから SQLCA.SQLCODE REXX 変数を取り出すことができません。

–9 取り出しの際に、SQLCA.SQLCODE REXX 変数の切り捨てが行われました。この変数の長さは最大 5 バイトまでです。

–10 SQLCA.SQLCODE REXX 変数を、ASCII から有効な長整数に変換できません。

–11 REXX 変数プールから SQLCA.SQLERRML REXX 変数を取り出すことができませんでした。

–12 取り出しの際に、SQLCA.SQLERRML REXX 変数の切り捨てが行われました。この変数の長さは最大 2 バイトまでです。

–13 SQLCA.SQLERRML REXX 変数を、ASCII から有効な短整数に変換できません。

–14 REXX 変数プールから SQLCA.SQLERRMC REXX 変数を取り出すことができません。

–15 取り出しの際に、SQLCA.SQLERRMC REXX 変数の切り捨てが行われました。この変数の長さは最大 70 バイトまでです。

–16 エラー・テキストに指定された REXX 変数をセットできません。

–17 REXX 変数プールから SQLCA.SQLSTATE REXX 変数を取り出すことができません。

–18 取り出しの際に、SQLCA.SQLSTATE REXX 変数の切り捨てが行われました。この変数の長さは最大 2 バイトまでです。

注: -8 から -18 までの値を戻すのは、GET ERROR MESSAGE API のみです。

SQLMSG

SQLCA.SQLCODE が 0 でない場合、この変数にはエラー・コードに関連したテキスト・メッセージが含まれます。

SQLISL

分離レベル。可能な値は以下のとおりです。

RR 反復可能読み取り。

RS 読み取り固定。

CS カーソル固定。これはデフォルトです。

UR 非コミット読み取り。

NC コミットなし。(NC は、一部のホストまたは System i® サーバーでしかサポートされていません。)

SQLCA

SQL ステートメントの処理の後に更新された SQLCA 構造体と、DB2 API が呼び出されます。

SQLRODA

CALL ステートメントを使用して呼び出される、ストアード・プロシージャーの入出力 SQLDA 構造体です。データベース・アプリケーション・リモート・インターフェース (DARI) API を使用して呼び出されるストアード・プロシージャーの出力 SQLDA 構造体でもあります。

SQLRIDA

データベース・アプリケーション・リモート・インターフェース (DARI) API を使用して呼び出される、ストアード・プロシージャーの入力 SQLDA 構造体です。

SQLRDAT

データベース・アプリケーション・リモート・インターフェース (DARI) API を使用して呼び出される、サーバー・プロシージャーの SQLCHAR 構造体です。

REXX 組み込み SQL アプリケーションをプログラミングする際の考慮事項

REXX はインタープリター言語です。したがって、プリコンパイラー、コンパイラー、またはリンカーを使用しません。その代わり、3 つの DB2 API を使用して、REXX の DB2 アプリケーションを作成します。 DB2 のさまざまなエレメントをアクセスするには、以下の API を使用してください。

SQLEXEC

SQL 言語をサポートします。

SQLDBS
DB2 API のコマンド形式のバージョンをサポートします。

SQLDB2
コマンド行プロセッサーへの REXX 特定のインターフェースをサポートします。このインターフェースを使用する際の制限および詳細については、REXX の API 構文の説明を参照してください。

アプリケーションで、DB2 API のいずれかを使用する前、または SQL ステートメントを発行する前に、SQLDBS、SQLDB2、および SQLEXEC ルーチンを登録しなければなりません。この登録は、REXX インタープリターに REXX/SQL の入り口点を知らせます。 Windows ベースのプラットフォームと AIX プラットフォームでは、登録のための方法が少し異なります。

正しい構文でそれぞれのルーチンを登録するには、以下の例を使用します。

Windows オペレーティング・システムでの登録のサンプル

```
/* ------------ Register SQLDBS with REXX  -------------------------*/
If Rxfuncquery('SQLDBS')  <> 0 then
    rcy = Rxfuncadd('SQLDBS','DB2AR','SQLDBS')
If rcy ¥= 0 then
    do
      say 'SQLDBS was not successfully added to the REXX environment'
      signal rxx_exit
    end

/* ------------ Register SQLDB2 with REXX  -------------------------*/
If Rxfuncquery('SQLDB2')  <> 0 then
    rcy = Rxfuncadd('SQLDB2','DB2AR','SQLDB2')
If rcy ¥= 0 then
    do
      say 'SQLDB2 was not successfully added to the REXX environment'
      signal rxx_exit
    end

/* ----------------- Register SQLEXEC with REXX  --------------------*/
If Rxfuncquery('SQLEXEC') <> 0 then
    rcy = Rxfuncadd('SQLEXEC','DB2AR','SQLEXEC')
If rcy ¥= 0 then
    do
      say 'SQLEXEC was not successfully added to the REXX environment'
      signal rxx_exit
    end
```

AIX での登録のサンプル

```
  /* ------------ Register SQLDBS, SQLDB2 and SQLEXEC with REXX --------*/
 rcy = SysAddFuncPkg("db2rexx")
 If rcy ¥= 0 then
   do
     say 'db2rexx was not successfully added to the REXX environment'
     signal rxx_exit
   end
```

Windows ベースのプラットフォームでは、RxFuncAdd コマンドはすべてのセッションにつき 1 回だけ実行する必要があります。

AIX では、SysAddFuncPkg をすべての REXX/SQL アプリケーションで実行しなければなりません。

Rxfuncadd API および SysAddFuncPkg API の詳細については、それぞれ Windows ベースのプラットフォームおよび AIX 版の REXX の資料に記載されています。

ステートメントまたはコマンドにトークンを含めることができます。それは、REXX 変数に対応できる SQLEXEC、SQLDBS、および SQLDB2 ルーチンに渡されます。この場合、REXX インタープリターは SQLEXEC、SQLDBS、または SQLDB2 を呼び出す前に、変数の値を置換します。

この状態を避けるには、ステートメントのストリングを、引用符 (' ' または " ") で囲んでください。引用符で囲まない場合、変数名と同じトークンがあると、そのトークンは REXX インタープリターによって解決され、SQLEXEC、SQLDBS、または SQLDB2 ルーチンには渡されません。

REXX 組み込み SQL アプリケーションでのラージ・オブジェクト・タイプのホスト変数の宣言

REXX ホスト変数に LOB 列を取り出してくる場合、この列は単純 (つまり、カウントはされない) ストリングとして保管されます。これは、文字ベースの SQL タイプ (たとえば、CHAR、VARCHAR、GRAPHIC、LONG など) すべてと同じ方法で処理されます。入力では、ホスト変数の内容のサイズが 32K を超える場合、または以下に説明するその他の基準を満たしている場合には、適切な LOB タイプが割り当てられます。

REXX SQL では、以下に示すホスト変数のストリングの内容により、LOB タイプが決定されます。

ホスト変数のストリングの内容	使用される LOB タイプ
:hv1='通常の引用符付きストリングが 32K を超えている'	CLOB
:hv2="'組み込み区切り引用符 "," 付きのストリングが 32K を超えている'"	CLOB
:hv3="G'G で開始する組み込み区切り単一引用符 "," 付きの DBCS が 32K を超えている'"	DBCLOB
:hv4="BIN'BIN で開始する組み込み区切り単一引用符 "," 付きのストリングが任意の長さである'"	BLOB

REXX 組み込み SQL アプリケーションでのラージ・オブジェクト・ロケーター・タイプのホスト変数の宣言

以下の図は、REXX における LOB ロケーター・ホスト変数の宣言の構文を示しています。

アプリケーション内で LOB ロケーター・ホスト変数を宣言しなければなりません。これらの宣言が出てくると、REXX/SQL はプログラムのそれ以降の部分で、宣言されたホスト変数をロケーターとして取り扱います。ロケーターの値は、内部形式で REXX 変数に保管されます。

以下に例を示します。

```
CALL SQLEXEC 'DECLARE :hv1, :hv2 LANGUAGE TYPE CLOB LOCATOR'
```

エンジンから戻された LOB により表現されるデータは、以下に示す形式の FREE LOCATOR ステートメントを使用して、 REXX/SQL 内で解放することができます。

FREE LOCATOR ステートメントの構文

以下に例を示します。

```
CALL SQLEXEC 'FREE LOCATOR :hv1, :hv2'
```

REXX 組み込み SQL アプリケーションでのファイル参照タイプのホスト変数の宣言

アプリケーション内で LOB ファイル参照ホスト変数を宣言しなければなりません。これらの宣言が出てくると、REXX/SQL はプログラムのそれ以降の部分で、宣言されたホスト変数を LOB ファイル参照として取り扱います。

以下の図は、REXX における LOB ファイル参照ホスト変数の宣言の構文を示しています。

以下に例を示します。

```
CALL SQLEXEC 'DECLARE :hv3, :hv4 LANGUAGE TYPE CLOB FILE'
```

REXX におけるファイル参照変数には、3 つのフィールドが含まれます。上記の例では、以下のものがその 3 つのフィールドに相当します。

hv3.FILE_OPTIONS.
アプリケーションによりセットされ、ファイルの使用法を指示します。

hv3.DATA_LENGTH.
DB2 によりセットされ、ファイルのサイズを指示します。

hv3.NAME.
アプリケーションにより、LOB ファイルの名前に設定されます。

FILE_OPTIONS の場合は、アプリケーションが以下のキーワードを設定します。

キーワード (整数値)
意味

READ (2)
ファイルが入力に使用されます。オープン、読み取り、クローズできるのは、正規のファイルです。ファイル内のデータの長さ (バイト数) は、(アプリケーションの要求側のコードにより) ファイルのオープン時に計算されます。

CREATE (8)
出力において、新しいファイルを作成します。ファイルがすでに存在している場合はエラーとなります。ファイルの長さ (バイト単位) は、ファイル参照変数構造の DATA_LENGTH フィールドに戻されます。

OVERWRITE (16)
出力において、ファイルがすでに存在する場合はそれを上書きし、そうでない場合は新しいファイルを作成します。ファイルの長さ (バイト単位) は、ファイル参照変数構造の DATA_LENGTH フィールドに戻されます。

APPEND (32)
ファイルがすでに存在する場合はそこに出力が追加され、そうでない場合は新しいファイルが作成されます。ファイルに追加されるデータ (ファイル全体の長さではない) の長さ (バイト単位) は、ファイル参照変数構造の DATA_LENGTH フィールドに戻されます。

注: REXX では、ファイル参照ホスト変数はコンパウンド変数です。したがって、NAME、NAME_LENGTH、および FILE_OPTIONS フィールドは、宣言するだけでなく、値も設定しなければなりません。

REXX 組み込み SQL アプリケーションでの LOB ホスト変数のクリア

Windows ベースのプラットフォームでは、アプリケーション・プログラムの終了後も効力を持つ REXX SQL LOB ロケーターおよびファイル参照ホスト変数宣言を、明示的にクリアしなければならない場合があります。これは、実行中のセッションがクローズされるまでアプリケーション・プロセスが終了しないためです。 REXX SQL LOB 宣言がクリアされないと、 LOB アプリケーションの実行後に同一セッション内で実行されている他のアプリケーションの妨げになります。

宣言をクリアする構文を示します。

```
CALL SQLEXEC "CLEAR SQL VARIABLE DECLARATIONS"
```

このステートメントは、LOB アプリケーションの終端にコーディングしなければなりません。直前のアプリケーションで宣言がクリアされていない場合があるので、宣言をクリアするための回避的な手段として、このステートメントを任意の場所にコーディングできます (たとえば、REXX SQL アプリケーションの最初)。

REXX 組み込み SQL アプリケーションでの NULL または切り捨て標識変数

REXX における標識変数のデータ・タイプは、小数点を持たない数です。以下に、INDICATOR キーワードを使用した REXX における標識変数の例を示します。

```
CALL SQLEXEC 'FETCH C1 INTO :cm INDICATOR :cmind'
IF ( cmind < 0 )
   SAY 'Commission is NULL'
```

上記の例では、cmind が負の値かどうか検査されます。負の値ではない場合、アプリケーションは cm の戻り値を使用することができます。 負の値の場合、取り出される値は NULL で、cm は使用されません。この場合、データベース・マネージャーはホスト変数の値を変更しません。

組み込み SQL アプリケーションにおける XQuery 式の実行

表に XML データを保管し、XQuery 式を使用して、組み込み SQL アプリケーションで XML 列にアクセスすることができます。 XML データへのアクセスには、データを文字またはバイナリー・データ・タイプにキャストする代わりに、 XML ホスト変数を使用します。 XML ホスト変数を使用しない場合、XML データにアクセスするのに最適な代替方法は、コード・ページ変換を避けるために FOR BIT DATA または BLOB データ・タイプを使用することです。

- 組み込み SQL アプリケーション内で XML ホスト変数を宣言する。
- 静的 SQL SELECT INTO ステートメントで XML 値を検索するには、XML タイプの使用が必要。
- XML 値が予期されているところで CHAR、VARCHAR、CLOB あるいは BLOB ホスト変数を入力に使用すると、その値は、デフォルトの空白処理 (STRIP) を伴う XMLPARSE 関数操作の対象になります。そうでない場合、XML ホスト変数は必須です。

XQuery 式を組み込み SQL アプリケーション内で直接実行するには、「XQUERY」キーワードを式の前に付加します。静的 SQL の場合は、XMLQUERY 関数を使用します。 XMLQUERY 関数が呼び出される場合、XQuery 式に「XQUERY」という接頭部は付きません。

例 1: C および C++ 動的 SQL で、「XQUERY」キーワードを前に付加して直接 XQuery 式を実行する

C および C++ アプリケーションでは、XQuery 式は、以下のような方法で実行できます。

```
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
  char stmt[16384];
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
EXEC SQL END DECLARE SECTION;

sprintf( stmt, "XQUERY (10, xs:integer(1) to xs:integer(4))" );

EXEC SQL PREPARE s1 FROM :stmt;
EXEC SQL DECLARE c1 CURSOR FOR s1;
EXEC SQL OPEN c1;

while( sqlca.sqlcode == SQL_RC_OK )
{
  EXEC SQL FETCH c1 INTO :xmlblob;
  /* Display results */
```

```
}

EXEC SQL CLOSE c1;
EXEC SQL COMMIT;
```

例 2: XMLQUERY 関数を使用して、静的 SQL で XQuery 式を実行する

XMLQUERY 関数を含む SQL ステートメントは、以下のように静的に準備できます。

```
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE C1 CURSOR FOR SELECT XMLQUERY( '(10, xs:integer(1) to
 xs:integer(4))' RETURNING SEQUENCE BY REF) from SYSIBM.SYSDUMMY1;

EXEC SQL OPEN c1;

while( sqlca.sqlcode == SQL_RC_OK )
{
  EXEC SQL FETCH c1 INTO :xmlblob;
  /* Display results */
}

EXEC SQL CLOSE c1;
EXEC SQL COMMIT;
```

例 3: COBOL 組み込み SQL アプリケーションで XQuery 式を実行する

COBOL アプリケーションでは、XQuery 式は、以下のような方法で実行できます。

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
     01 stmt pic x(80).
   01 xmlBuff USAGE IS SQL TYPE IS XML AS BLOB (10K).
EXEC SQL END DECLARE SECTION END-EXEC.

MOVE "XQUERY (10, xs:integer(1) to xs:integer(4))" TO stmt.
EXEC SQL PREPARE s1 FROM :stmt END-EXEC.
EXEC SQL DECLARE c1 CURSOR FOR s1 END-EXEC.
EXEC SQL OPEN c1 USING :host-var END-EXEC.

*Call the FETCH and UPDATE loop.
Perform Fetch-Loop through End-Fetch-Loop
   until SQLCODE does not equal 0.

EXEC SQL CLOSE c1 END-EXEC.
    EXEC SQL COMMIT END-EXEC.

Fetch-Loop Section.
    EXEC SQL FETCH c1 INTO :xmlBuff END-EXEC.
    if SQLCODE not equal 0
       go to End-Fetch-Loop.
* Display results
End-Fetch-Loop. exit.
```

組み込み SQL アプリケーションにおける SQL ステートメントの実行

組み込み SQL アプリケーション内で SQL ステートメントを実行する場合、静的に実行されるステートメントと動的に実行されるステートメントとではどちらも EXEC SQL コマンドを使用しますが、それぞれの実行方法は異なります。静的ステートメントは、組み込み SQL アプリケーションのソース・コード内にハードコーディングされています。動的ステートメントは、実行時にコンパイルされ、入力パラメー

ターによって準備されることがある点で、静的ステートメントとは異なります。読み取られる情報はカーソルと呼ばれるメディアに保管されて、ユーザーがそのデータを自由にスクロールして適切な更新を行うことができるようになります。
SQLCODE、SQLSTATE、および SQLWARN からのエラー情報は、アプリケーションのトラブルシューティングを支援するために役立つツールとなります。

組み込み SQL アプリケーションにおけるコメント

コメントは、どのようなアプリケーションでも、コードを見て理解しやすくするために重要です。このセクションでは、組み込み SQL アプリケーションへのコメントの追加について解説します。

C および C++ 組み込み SQL アプリケーションにおけるコメント

C および C++ アプリケーションの場合、SQL コメントは、EXEC SQL ブロックに挿入できます。以下に例を示します。

```
/* Only C or C++ comments allowed here */
 EXEC SQL
    -- SQL comments or
    /* C comments or */
    // C++ comments allowed here
    DECLARE C1 CURSOR FOR sname;
 /* Only C or C++ comments allowed here */
```

COBOL 組み込み SQL アプリケーションにおけるコメント

COBOL アプリケーションの場合、SQL コメントは、EXEC SQL ブロックに挿入できます。以下に例を示します。

```
*   See COBOL documentation for comment rules
*   Only COBOL comments are allowed here
    EXEC SQL
    -- SQL comments or
*   full-line COBOL comments are allowed here
    DECLARE C1 CURSOR FOR sname END-EXEC.
*   Only COBOL comments are allowed here
```

FORTRAN 組み込み SQL アプリケーションにおけるコメント

FORTRAN アプリケーションの場合、SQL コメントは、EXEC SQL ブロックに挿入できます。以下に例を示します。

```
C     Only FORTRAN comments are allowed here
      EXEC SQL
      + -- SQL comments, and
C     full-line FORTRAN comment are allowed here
      + DECLARE C1 CURSOR FOR sname
      I=7 ! End of line FORTRAN comments allowed here
C     Only FORTRAN comments are allowed here
```

REXX 組み込み SQL アプリケーションにおけるコメント

REXX アプリケーションでは、SQL コメントはサポートされていません。

組み込み SQL アプリケーションにおける静的 SQL ステートメントの実行

SQL ステートメントは、以下の方法を使用して、ホスト言語で静的に実行できます。

- C または C++ (**tut_mod.sqc/tut_mod.sqC**)

  次の 3 つの例は、サンプル **tut_mod** からの例です。C または C++ において表データを変更する方法を示している完全なプログラムについては、このサンプルを参照してください。

  以下に、表にデータを挿入する方法の例を示します。

```
EXEC SQL INSERT INTO staff(id, name, dept, job, salary)
  VALUES(380, 'Pearce', 38, 'Clerk', 13217.50),
        (390, 'Hachey', 38, 'Mgr', 21270.00),
        (400, 'Wagland', 38, 'Clerk', 14575.00);
```

  以下に、表のデータを更新する方法の例を示します。

```
EXEC SQL UPDATE staff
  SET salary = salary + 10000
  WHERE id >= 310 AND dept = 84;
```

  以下に、表からデータを削除する方法の例を示します。

```
EXEC SQL DELETE
  FROM staff
  WHERE id >= 310 AND salary > 20000;
```

- COBOL (**updat.sqb**)

  次の 3 つの例は、サンプル **updat** からの例です。COBOL において表データを変更する方法を示している完全なプログラムについては、このサンプルを参照してください。

  以下に、表にデータを挿入する方法の例を示します。

```
EXEC SQL INSERT INTO staff
    VALUES (999, 'Testing', 99, :job-update, 0, 0, 0)
    END-EXEC.
```

  以下に示すのは、表のデータを更新する方法の例で、job-update はソース・コードの宣言セクションで宣言されたホスト変数への参照です。

```
EXEC SQL UPDATE staff
    SET job=:job-update
    WHERE job='Mgr'
    END-EXEC.
```

  以下に、表からデータを削除する方法の例を示します。

```
EXEC SQL DELETE
    FROM staff
    WHERE job=:job-update
    END-EXEC.
```

組み込み SQL アプリケーションにおける SQLDA 構造体からのホスト変数情報の検索

静的 SQL を使用すると、組み込み SQL ステートメントで使用されているホスト変数は、アプリケーションのコンパイル時に認識されます。動的 SQL を使用した場合には、組み込み SQL ステートメントとその結果としてのホスト変数は、アプリケーションを実行するまで認識されません。このように、動的 SQL アプリケーションの場合は、アプリケーションで使用するホスト変数のリストを扱う必要があります。 (PREPARE を使用して) 準備された SELECT ステートメントのホスト変数

情報を得るためには、 DESCRIBE ステートメントを使用し、その情報を SQL 記述子域 (SQLDA) に保管することができます。

アプリケーション内で DESCRIBE ステートメントを実行すると、データベース・マネージャーはホスト変数を SQLDA に定義します。ホスト変数を SQLDA に定義すると、 FETCH ステートメントにより、カーソルを使用してホスト変数に値を割り当てることができます。

動的に実行される SQL プログラムにおける SQLDA 構造体の宣言

SQLDA には不定数の SQLVAR 項目のオカレンスがあり、次の図に示されているように、各 SQLVAR 項目は 1 データ行に 1 つの列を記述するフィールドの集まりで構成されます。 SQLVAR 項目には、基本 SQLVAR 項目と 2 次 SQLVAR 項目という 2 つのタイプがあります。

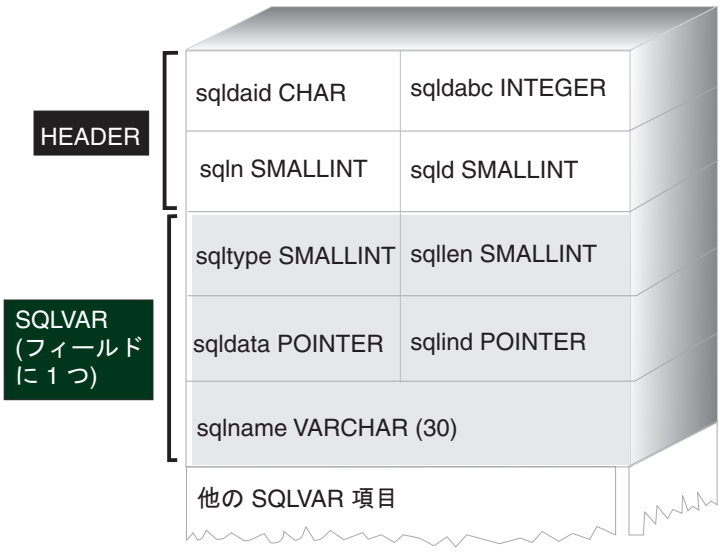


図 2. SQL 記述子域 (SQLDA)

必要とされる SQLVAR 項目の数は結果表の列の数によって決まるため、アプリケーションは必要に応じて適切な数の SQLVAR エレメントを割り振ることができなければなりません。 次のいずれかの方式を使用してください。

- 必要とされる最大の SQLDA (つまり、SQLVAR 項目の最大数を指定した SQLDA) を与える。結果表に戻すことができる列の数は最大 255 です。戻される列のいずれかが LOB タイプか特殊タイプの列である場合、SQLN の値は 2 倍になり、情報を入れるために必要な SQLVAR 項目の数も 2 倍の 510 になります。しかし、255 列も戻す SELECT ステートメントはほとんどないので、割り当てられたスペースの大部分が未使用となります。
- SQLVAR 項目を少なめに指定して、小さめの SQLDA を与える。この場合、結果表に SQLDA で使用できる SQLVAR 項目より多い列が入っていると、記述は戻されません。代わりに、データベース・マネージャーは SELECT ステートメントで検出される選択リスト項目の数を戻します。アプリケーションは SQLDA に必要な数の SQLVAR 項目を割り当ててから、DESCRIBE ステートメントを使用して列記述を入手します。

- 戻された列のいくつかが LOB またはユーザー定義タイプである場合は、SQLDA に SQLVAR 項目の正確な数を設定する。

3 つの方式のすべてについて、最初にいくつの SQLVAR 項目を割り当てればよいかという疑問が生じます。各 SQLVAR エレメントは、最高 44 バイトのストレージを使用します (SQLDATA および SQLIND フィールドに割り振られるストレージはカウントしていません)。メモリーに十分余裕があれば、 SQLDA の最大サイズを割り振るという最初の方法を実行するのが簡単です。

より小さい SQLDA を割り当てるという 2 番目の方法は、メモリーの動的割り振りをサポートする C および C++ のようなプログラミング言語にしか適用できません。メモリーの動的割り振りをサポートしない COBOL や FORTRAN のような言語の場合、最初の方法を使用します。

最小の SQLDA 構造体を用いて動的に実行される SQL ステートメントの準備

ここにある情報を、最小の SQLDA 構造体をステートメント用に割り振る例として用いてください。

メモリーの動的割り振りをサポートする C および C++ のようなプログラミング言語でしか、より小さい SQLDA 構造体を割り当てることはできません。

アプリケーションが、 SQLVAR 項目の入っていない `minsqlda` という名前の SQLDA 構造体を宣言する場面を例にとって考えてみましょう。 SQLDA の SQLN フィールドは割り振られる SQLVAR 項目の数を記述します。この場合、SQLN は 0 にセットされていなければなりません。次に、文字ストリング `dstring` から 1 つのステートメントを準備し、そしてその記述を `minsqlda` の中に入力するには、次の SQL ステートメントを発行します。 (C 構文を使用し、 `minsqlda` が SQLDA 構造体へのポインターとして宣言されているものとします。)

```
EXEC SQL
  PREPARE STMT INTO :*minsqlda FROM :dstring;
```

`dstring` に含まれるステートメントが、各行に 20 列を戻す SELECT ステートメントであるとします。 PREPARE ステートメント (または DESCRIBE ステートメント) の後の SQLDA の SQLD フィールドには、準備済み SELECT ステートメントの結果表の列数が入っています。

SQLDA の SQLVAR 項目は、以下の場合に設定されます。

- SQLN >= SQLD であり、かつどの列も LOB または特殊タイプではない場合

  最初の SQLD SQLVAR 項目が設定され、SQLDOUBLED はブランクに設定されます。

- SQLN >= 2*SQLD であり、かつ少なくとも 1 つの列が LOB または特殊タイプである場合

  2* SQLD SQLVAR 項目が設定され、SQLDOUBLED は 2 に設定されます。

- SQLD <= SQLN < 2*SQLD であり、少なくとも 1 つの列が特殊タイプであり、かつ LOB の列はない場合

最初の SQLD SQLVAR 項目が設定され、SQLDOUBLED はブランクに設定されます。 SQLWARN BIND オプションが YES の場合は、警告 SQLCODE +237 (SQLSTATE 01594) が発行されます。

以下の場合には、SQLDA 内の SQLVAR 項目は設定されません (追加スペースを割り振り、 DESCRIBE をもう一度指定するよう要求されます)。

- SQLN < SQLD であり、どの列も LOB または特殊タイプではない場合

  SQLVAR 項目は設定されず、SQLDOUBLED はブランクに設定されます。SQLWARN BIND オプションが YES の場合は、警告 SQLCODE +236 (SQLSTATE 01005) が発行されます。

  DESCRIBE が正常に実行される場合には、SQLD 個の SQLVAR 項目が割り振られます。

- SQLN < SQLD であり、少なくとも 1 つの列が特殊タイプであり、かつ LOB の列はない場合

  SQLVAR 項目は設定されず、SQLDOUBLED はブランクに設定されます。SQLWARN BIND オプションが YES の場合は、警告 SQLCODE +239 (SQLSTATE 01005) が発行されます。

  特殊タイプの名前を含む DESCRIBE を正常に実行するためには、 2*SQLD SQLVAR 項目を割り振ってください。

- SQLN < 2*SQLD であり、かつ少なくとも 1 つの列が LOB である場合

  SQLVAR 項目は設定されず、SQLDOUBLED はブランクに設定されます。警告 SQLCODE +238 (SQLSTATE 01005) が発行されます (SQLWARN BIND オプションの設定に関係なく)。

  DESCRIBE を正常に実行するためには、2*SQLD SQLVAR 項目を割り振ってください。

BIND コマンドの SQLWARN オプションは、 DESCRIBE (または PREPARE...INTO) が以下の警告を戻すかどうかを制御します。

- SQLCODE +236 (SQLSTATE 01005)
- SQLCODE +237 (SQLSTATE 01594)
- SQLCODE +239 (SQLSTATE 01005)

アプリケーション・コードでは、これらの SQLCODE 値が戻される可能性のあることを常に考慮に入れておいてください。警告 SQLCODE +238 (SQLSTATE 01005) は、選択リストに LOB 列があり SQLDA に不適当な SQLVAR 項目がある場合に、必ず戻されます。これは、結果セット内に LOB 列があるために SQLVAR 項目の数が 2 倍になっていなければならないということをアプリケーションが認識できる唯一の方法です。

十分な数の SQLVAR 項目を指定した、動的に実行される SQL ステートメント用の SQLDA 構造体の割り振り

結果表の列数が決まったら、ストレージを 2 番目の、フルサイズの SQLDA に割り振ってください。最初の SQLDA は入力パラメーターに使用され、2 番目のフルサイズの SQLDA は出力パラメーターに使用されます。

結果表の列が 20 あるとします (すべて LOB 列ではないとします)。この場合、 2 番目の SQLDA 構造体である fulsqlda には、少なくとも 20 の SQLVAR エレメント (または結果表に LOB または特殊タイプがある場合には 40 エレメント) を割り振らなければなりません。この例のその他の部分では、LOB または特殊タイプは結果表に含まれないものとします。

SQLDA 構造体のストレージ要件を計算するときには、以下のものを含めてください。

- 長さ 16 バイトの固定長ヘッダー (SQLN および SQLD などのフィールドを含む)。
- SQLVAR 項目の可変長配列、それぞれのエレメントは長さが 44 バイト (32 ビット・プラットフォームの場合) または 56 バイト (64 ビット・プラットフォームの場合)。

fulsqlda に必要な SQLVAR 項目の数は、 minsqlda という SQLD フィールドに指定されます。 その値が 20 であるとします。そのため、fulsqlda に必要なストレージ割り振りは、以下のようになります。

```
16 + (20 * sizeof(struct sqlvar))
```

この値は、ヘッダーのサイズに各 SQLVAR 項目のサイズの 20 倍を加えて、合計 896 バイトであることを表しています。

SQLDASIZE マクロを使用することにより、自分で計算をしないようにすれば、バージョン間の違いをすべて回避することができます。

動的に実行される SQL プログラムにおける SELECT ステートメントの記述

十分な量のスペースを 2 番目の SQLDA (この例の場合 fulsqlda と呼ばれる) に割り振ってから、アプリケーションをコーディングして SELECT ステートメントを記述しなければなりません。

次のステップを実行するようにアプリケーションをコーディングしてください。

1. fulsqlda の SQLN フィールドの値を 20 にする (この例では結果表の列は 20 であり、すべての列は LOB 列ではないものとしています)。
2. 2 番目の SQLDA 構造体 fulsqlda を用いて SELECT ステートメントに関する情報を入手する。 これには、次の 2 とおりの方法があります。
   - minsqlda の代わりに fulsqlda を指定する別の PREPARE ステートメントを使用する。
   - fulsqlda を指定する DESCRIBE ステートメントを使用する。

DESCRIBE ステートメントを使用するとステートメントを 2 回準備するコストが省けるため、この方法のほうが好んで使用されます。 DESCRIBE ステートメントは、準備操作中に入手した前の情報を再度使用するだけで、その情報を新規の SQLDA 構造体に入れます。次のステートメントを発行することができます。

```
EXEC SQL DESCRIBE STMT INTO :fulsqlda
```

このステートメントを実行すると、それぞれの SQLVAR エレメントには結果表の 1 つの列の記述が含まれるようになります。

行を保持するためのストレージの獲得

先にアプリケーションが行のためのストレージを割り振ってからでなければ、SQLDA 構造体を使用して結果表からアプリケーション行を取り出すことはできません。

次のステップを実行するようにアプリケーションをコーディングしてください。

1. それぞれの SQLVAR 記述を分析して、その列の値に必要なスペースの量を判別する。

   SELECT が記述されている場合、LOB の値について SQLVAR に指定されるデータ・タイプは SQL_TYP_*x*LOB であることに注意してください。このデータ・タイプは一般的な LOB ホスト変数と同じであり、すべての LOB は 1 回でメモリーに保管されます。これは (数 MB までの) 小さい LOB の場合に有効ですが、スタックが十分なメモリーを割り振ることができないため、このデータ・タイプを大きい LOB (1 GB など) に使用することはできません。 SQLVAR 内のアプリケーションの列定義を変更して、 SQL_TYP_*x*LOB_LOCATOR または SQL_TYPE_*x*LOB_FILE のいずれかにすることが必要になります。 (SQLVAR の SQLTYPE フィールドを変更する場合には、 SQLLEN フィールドも変更する必要があるので注意してください。) SQLVAR 内の列定義を変更すると、アプリケーションではその新しいタイプに対して正しい容量のストレージを割り振ることができます。
2. その列の値にストレージを割り振る。
3. 割り振ったストレージのアドレスを SQLDA 構造体の SQLDATA フィールドに保管する。

これらのステップは、各列の記述を分析し、それぞれの SQLDATA フィールドの内容をその列の値を保持するだけの大きさのストレージ域のアドレスと置き換えることによってなされます。長さ属性は、LOB タイプでないデータ項目に対する各 SQLVAR 項目の SQLLEN フィールドから判別されます。タイプが BLOB、CLOB、または DBCLOB の項目の場合、長さ属性は 2 番目の SQLVAR 項目の SQLLONGLEN フィールドから判別されます。

さらに、指定した列に NULL を使用できる場合、アプリケーションは SQLIND フィールドの内容を列の標識変数のアドレスと置き換えなければなりません。

動的に実行される SQL プログラムにおけるカーソルの処理

SQLDA 構造の割り振りが適切に行われると、 SELECT ステートメントに関連するカーソルをオープンし、行を取り出すことができます。

SELECT ステートメントに関連したカーソルを処理するには、最初にカーソルをオープンし、次に FETCH ステートメントの USING DESCRIPTOR 節を指定して行を取り出します。たとえば、C アプリケーションでは次のようにします。

```
EXEC SQL OPEN pcurs
EMB_SQL_CHECK( "OPEN" ) ;
EXEC SQL FETCH pcurs USING DESCRIPTOR :*sqldaPointer
EMB_SQL_CHECK( "FETCH" ) ;
```

FETCH が成功したならば SQLDA からデータを獲得し列見出しを表示するようなアプリケーションを書けるでしょう。以下に例を示します。

```
display_col_titles( sqldaPointer ) ;
```

データを表示したなら、カーソルをクローズし動的に割り振ったメモリーを解放しなければなりません。以下に例を示します。

```
EXEC SQL CLOSE pcurs ;
EMB_SQL_CHECK( "CLOSE CURSOR" ) ;
```

動的に実行される SQL プログラムのための SQLDA 構造体の割り振り

アプリケーションとのデータのやり取りで使用できるようにするため、アプリケーションで使用する SQLDA 構造体を割り振ってください。

C 言語で SQLDA 構造体を作成するには、ホスト言語で INCLUDE SQLDA ステートメントを組み込むか、または SQLDA 組み込みファイルを組み込んで、構造体定義を入手してください。次に、SQLDA のサイズは固定されていないため、アプリケーションは SQLDA 構造体へのポインターを宣言し、それにストレージを割り振らなければなりません。 SQLDA 構造体の実際のサイズは、SQLDA を用いて渡される個別データ項目の数によって決まります。

C および C++ プログラム言語 では、SQLDA の割り振りを簡単に行うためにマクロが提供されています。このマクロの形式は以下のとおりです。

```
#define SQLDASIZE(n) (offsetof(struct sqlda, sqlvar) ¥
        + (n) × sizeof(struct sqlvar))
```

このマクロを使用することによって、 n 個の SQLVAR エレメントに必要なストレージを計算することができます。

COBOL で SQLDA 構造体を作成するには、 INCLUDE SQLDA ステートメントを組み込むか、または COPY ステートメントを使用します。 SQLVAR 項目の最大数を制御して SQLDA が使用するストレージの容量を制御したい場合は、 COPY ステートメントを使用してください。たとえば、SQLVAR 項目のデフォルトの数を 1489 から 1 に変更するには、以下の COPY ステートメントを使用します。

```
COPY "sqlda.cbl"
  replacing --1489--
  by --1--.
```

FORTRAN 言語では、自己定義データ構造または動的割り振りは直接にはサポートされていません。 SQLDA 組み込みファイルは FORTRAN では使用できません。これは、FORTRAN では SQLDA をデータ構造としてサポートできないためです。FORTRAN プログラムでは、プリコンパイラーは INCLUDE SQLDA ステートメントを無視します。

ただし、FORTRAN プログラムで静的 SQLDA 構造体に似た構造体を作成し、これを SQLDA を使用できる任意の場所で使用することができます。 sqldact.f ファイルには、FORTRAN で SQLDA 構造体を宣言するのに役立つ定数が含まれています。

ポインター値を必要とする SQLDA エレメントに値を割り当てるには、SQLGADDR の呼び出しを実行してください。

次の表は、SQLVAR エレメントを 1 つ持つ SQLDA 構造体の宣言および使用方法を示しています。

言語 **ソース・コード例**

C および C++

```
#include
struct sqlda *outda = (struct sqlda *)malloc(SQLDASIZE(1));

/* DECLARE LOCAL VARIABLES FOR HOLDING ACTUAL DATA */
double sal = 0;
short salind = 0;

/* INITIALIZE ONE ELEMENT OF SQLDA */
memcpy( outda->sqldaid,"SQLDA   ",sizeof(outda->sqldaid));
outda->sqln = outda->sqld = 1;
outda->sqlvar[0].sqltype = SQL_TYP_NFLOAT;
outda->sqlvar[0].sqllen  = sizeof( double );.
outda->sqlvar[0].sqldata = (unsigned char *)&sal;
outda->sqlvar[0].sqlind  = (short *)&salind;
```

COBOL

```
       WORKING-STORAGE SECTION.
       77 SALARY          PIC S99999V99 COMP-3.
       77 SAL-IND         PIC S9(4)     COMP-5.

          EXEC SQL INCLUDE SQLDA END-EXEC

      * Or code a useful way to save unused SQLVAR entries.
      * COPY "sqlda.cbl" REPLACING --1489-- BY --1--.

            01 decimal-sqllen pic s9(4) comp-5.
            01 decimal-parts redefines decimal-sqllen.
               05 precision pic x.
               05 scale pic x.

      * Initialize one element of output SQLDA
            MOVE 1 TO SQLN
            MOVE 1 TO SQLD
            MOVE SQL-TYP-NDECIMAL TO SQLTYPE(1)

      * Length = 7 digits precision and 2 digits scale

            MOVE x"07" TO PRECISION.
            MOVE x"02" TO SCALE.
            MOVE DECIMAL-SQLLEN TO O-SQLLEN(1).
            SET SQLDATA(1) TO ADDRESS OF SALARY
            SET SQLIND(1)  TO ADDRESS OF SAL-IND
```

言語	ソース・コード例

FORTRAN

```
      include 'sqldact.f'

      integer*2  sqlvar1
      parameter ( sqlvar1 = sqlda_header_sz + 0*sqlvar_struct_sz )

C     Declare an Output SQLDA -- 1 Variable
      character    out_sqlda(sqlda_header_sz + 1*sqlvar_struct_sz)

      character*8  out_sqldaid     ! Header
      integer*4    out_sqldabc
      integer*2    out_sqln
      integer*2    out_sqld

      integer*2    out_sqltype1    ! First Variable
      integer*2    out_sqllen1
      integer*4    out_sqldata1
      integer*4    out_sqlind1
      integer*2    out_sqlnamel1
      character*30 out_sqlnamec1

      equivalence( out_sqlda(sqlda_sqldaid_ofs), out_sqldaid )
      equivalence( out_sqlda(sqlda_sqldabc_ofs), out_sqldabc )
      equivalence( out_sqlda(sqlda_sqln_ofs), out_sqln       )
      equivalence( out_sqlda(sqlda_sqld_ofs), out_sqld       )
      equivalence( out_sqlda(sqlvar1+sqlvar_type_ofs), out_sqltype1 )
      equivalence( out_sqlda(sqlvar1+sqlvar_len_ofs), out_sqllen1   )
      equivalence( out_sqlda(sqlvar1+sqlvar_data_ofs), out_sqldata1 )
      equivalence( out_sqlda(sqlvar1+sqlvar_ind_ofs), out_sqlind1   )
      equivalence( out_sqlda(sqlvar1+sqlvar_name_length_ofs),
     +             out_sqlnamel1                                   )
      equivalence( out_sqlda(sqlvar1+sqlvar_name_data_ofs),
     +             out_sqlnamec1                                   )

C     Declare Local Variables for Holding Returned Data.
      real*8       salary
      integer*2    sal_ind

C     Initialize the Output SQLDA (Header)
      out_sqldaid  = 'OUT_SQLDA'
      out_sqldabc  = sqlda_header_sz + 1*sqlvar_struct_sz
      out_sqln     = 1
      out_sqld     = 1
C     Initialize VAR1
      out_sqltype1 = SQL_TYP_NFLOAT
      out_sqllen1  = 8
      rc = sqlgaddr( %ref(salary), %ref(out_sqldata1) )
      rc = sqlgaddr( %ref(sal_ind), %ref(out_sqlind1) )
```

注: 上記の例は 32 ビット FORTRAN 向けに書かれています。

動的なメモリー割り振りをサポートしない言語では、 SQLVAR エレメントの希望数を指定した SQLDA をホスト言語で明示的に宣言しなければなりません。SQLVAR のエレメントには、アプリケーションの必要に応じて決定されたとおりの十分な数を必ず宣言してください。

動的に実行される SQL プログラムにおける SQLDA 構造体を使用したデータ転送

ホスト変数のリストを使用してデータを転送するよりも、 SQLDA を使用してデータを転送するほうが、より高い柔軟性が得られます。たとえば、SQLDA を用いて、ホスト言語に対応するものがないデータ (C 言語における DECIMAL データなど) でも転送することができます。

次の表を、数値とシンボル名がどのように関連付けられるかを示す相互参照リストとして使用してください。

表 17. DB2 SQLDA SQL タイプ: 数値および対応するシンボル名

SQL 列名	SQLTYPE 数値	SQLTYPE シンボル名[1]
DATE	384/385	SQL_TYP_DATE / SQL_TYP_NDATE
TIME	388/389	SQL_TYP_TIME / SQL_TYP_NTIME
TIMESTAMP	392/393	SQL_TYP_STAMP / SQL_TYP_NSTAMP
n/a[2]	400/401	SQL_TYP_CGSTR / SQL_TYP_NCGSTR
BLOB	404/405	SQL_TYP_BLOB / SQL_TYP_NBLOB
CLOB	408/409	SQL_TYP_CLOB / SQL_TYP_NCLOB
DBCLOB	412/413	SQL_TYP_DBCLOB / SQL_TYP_NDBCLOB
VARCHAR	448/449	SQL_TYP_VARCHAR / SQL_TYP_NVARCHAR
CHAR	452/453	SQL_TYP_CHAR / SQL_TYP_NCHAR
LONG VARCHAR	456/457	SQL_TYP_LONG / SQL_TYP_NLONG
n/a[3]	460/461	SQL_TYP_CSTR / SQL_TYP_NCSTR
VARGRAPHIC	464/465	SQL_TYP_VARGRAPH / SQL_TYP_NVARGRAPH
GRAPHIC	468/469	SQL_TYP_GRAPHIC / SQL_TYP_NGRAPHIC
LONG VARGRAPHIC	472/473	SQL_TYP_LONGRAPH / SQL_TYP_NLONGRAPH
FLOAT	480/481	SQL_TYP_FLOAT / SQL_TYP_NFLOAT
REAL[4]	480/481	SQL_TYP_FLOAT / SQL_TYP_NFLOAT
DECIMAL[5]	484/485	SQL_TYP_DECIMAL / SQL_TYP_DECIMAL
INTEGER	496/497	SQL_TYP_INTEGER / SQL_TYP_NINTEGER
SMALLINT	500/501	SQL_TYP_SMALL / SQL_TYP_NSMALL
n/a	804/805	SQL_TYP_BLOB_FILE / SQL_TYPE_NBLOB_FILE
n/a	808/809	SQL_TYP_CLOB_FILE / SQL_TYPE_NCLOB_FILE
n/a	812/813	SQL_TYP_DBCLOB_FILE / SQL_TYPE_NDBCLOB_FILE
n/a	960/961	SQL_TYP_BLOB_LOCATOR / SQL_TYP_NBLOB_LOCATOR
n/a	964/965	SQL_TYP_CLOB_LOCATOR / SQL_TYP_NCLOB_LOCATOR
n/a	968/969	SQL_TYP_DBCLOB_LOCATOR / SQL_TYP_NDBCLOB_LOCATOR
XML	988/989	SQL_TYP_XML / SQL_TYP_XML

表 17. DB2 SQLDA SQL タイプ (続き): 数値および対応するシンボル名

SQL 列名	SQLTYPE 数値	SQLTYPE シンボル名[1]

注: これらの定義タイプは sql.h 組み込みファイルにあり、組み込みファイル自体は、 sqllib ディレクトリーの include サブディレクトリーにあります。 (例えば、C プログラミング言語の場合は sqllib/include/sql.h となります。)

1. COBOL プログラミング言語の場合、SQLTYPE 名には下線 (_) を使用しませんが、その代わりにハイフン (-) を使用します。
2. これは NULL 終了 GRAPHIC ストリングです。
3. これは NULL 終了文字ストリングです。
4. SQLDA での REAL と DOUBLE の違いは長さの値です (4 または 8)。
5. 精度は最初のバイトにあります。位取りは 2 番目のバイトにあります。

動的に実行される SQL プログラムにおける対話式 SQL ステートメントの処理

動的 SQL を使用するアプリケーションを作成し、任意の SQL ステートメントを処理することができます。たとえば、アプリケーションがユーザーから SQL ステートメントを受け入れる場合、アプリケーションはステートメントについて事前にわかっていなくても、そのステートメントを実行できなければなりません。実行時までわからない値は、疑問符 (?) で表示されるパラメーター・マークによって表すことができます。パラメーター・マークは、ユーザーとアプリケーションの間の対話を可能にするもので、静的 SQL ステートメントにとってのホスト変数に似ています。

PREPARE および DESCRIBE ステートメントを SQLDA 構造体で使用して、アプリケーションは実行される SQL ステートメントのタイプを判別し、それに応じて処理することができるようにしてください。

動的に実行される SQL プログラムにおけるステートメント・タイプの判別

SQL ステートメントを準備する場合、ステートメントのタイプに関する情報は SQLDA 構造体を調べて判別することができます。この情報はステートメントの準備時に INTO 節を指定して SQLDA 構造体に入れるか、または事前に準備されたステートメントに対して DESCRIBE ステートメントを発行することによって、SQLDA 構造体に入れることができます。

いずれの場合でも、データベース・マネージャーは SQLDA 構造体の SQLD フィールドに SQL ステートメントにより生成された結果表の列数を示す値を入れます。 SQLD フィールドにゼロ (0) が入っている場合、このステートメントは SELECT ステートメントではありません。ステートメントはすでに準備されているため、 EXECUTE ステートメントを使用してただちに実行することができます。

ステートメントにパラメーター・マーカーが含まれている場合、USING 節を指定しなければなりません。 USING 節は、ホスト変数のリストか SQLDA 構造体のどちらかを指定することができます。

SQLD フィールドが 0 より大きい場合、ステートメントは SELECT ステートメントであるため、次の節での説明に従って処理しなければなりません。

動的に実行される SQL プログラムにおける可変リスト SELECT ステートメントの処理

可変リスト SELECT ステートメントとは、戻される列の数およびタイプがプリコンパイル時にはわからないステートメントのことです。この場合、アプリケーションには、結果表の行を保持するために宣言しなければならない正確なホスト変数がわかりません。

可変リスト SELECT ステートメントを処理するには、次のステップを実行するようにアプリケーションをコーディングしてください。

1. SQLDA を宣言する。

   可変リスト SELECT ステートメントを処理するには、 SQLDA 構造体を必ず使用します。

2. INTO 節を使用してステートメントを PREPARE (準備) する。

   アプリケーションは、宣言した SQLDA 構造体に十分な SQLVAR エレメントがあるかどうかを判別します。十分なエレメントがない場合、アプリケーションは必要な数の SQLVAR エレメントを持つ別の SQLDA 構造体を割り振り、新規の SQLDA を用いて追加の DESCRIBE ステートメントを発行します。

3. SQLVAR エレメントを割り振る。

   各 SQLVAR に必要なホスト変数および標識に、ストレージを割り振ります。このステップでは、それぞれの SQLVAR エレメントにデータの割り振りアドレスおよび標識変数を入れます。

4. SELECT ステートメントを処理する。

   カーソルは準備済みステートメントに関連付けられ、オープンされます。行は適切に割り振られた SQLDA 構造体を用いて取り出されます。

エンド・ユーザーからの SQL 要求の保存

アプリケーションのユーザーがアプリケーションから SQL 要求を発行することができる場合、これらの要求を保管しておきたいかもしれません。

アプリケーションで任意の SQL ステートメントを保管できる場合、これらをデータ・タイプが VARCHAR、LONG VARCHAR、CLOB、VARGRAPHIC、 LONG VARGRAPHIC、または DBCLOB の列を持つ表に保管することができます。VARGRAPHIC、LONG VARGRAPHIC、および DBCLOB データ・タイプは、 2 バイト文字セット (DBCS) および拡張 UNIX コード (EUC) 環境でしか使用できないことに注意してください。

ユーザーは、準備済みのバージョンの SQL ステートメントではなく、そのソースを保管しなければなりません。これは、表に保管されているバージョンを実行する前に、各ステートメントを検索して準備しなければならないことを意味します。つまり、アプリケーションは、文字ストリングから SQL ステートメントを準備し、このステートメントを動的に実行します。

動的に実行される SQL ステートメントへのパラメーター・マーカーを使用した変数入力の提供

動的 SQL ステートメントにはホスト変数を入れることができません。それは、ホスト変数情報 (データ・タイプおよび長さ) がアプリケーションのプリコンパイルの間しか使用できないためです。実行時には、ホスト変数情報は使用できません。

動的 SQL では、ホスト変数の代わりにパラメーター・マーカーを使用します。パラメーター・マーカーは疑問符 (?) で示されます。そして、ホスト変数が SQL ステートメントの内部で置換される位置を示します。

アプリケーションで動的 SQL を使用していて、DELETE を実行できるようにしたいとしましょう。パラメーター・マーカーが入っている文字ストリングは、次のような形となります。

```
DELETE FROM TEMPL WHERE EMPNO = ?
```

このステートメントが実行されると、 EXECUTE ステートメントの USING 節によってホスト変数つまり SQLDA 構造体が指定されます。ステートメントを実行する際に、ホスト変数の内容が使用されます。

パラメーター・マーカーは、 SQL ステートメント内部で使用するコンテキストによって想定されたデータ・タイプおよび長さを持っています。パラメーター・マーカーのデータ・タイプが、これを使用しているステートメントの内容からはっきり判別できない場合は、 CAST を使用してタイプを指定することができます。このようなパラメーター・マーカーは、*型付きパラメーター・マーカー* と見なされます。タイプ付きパラメーター・マーカーは、指定されたタイプのホスト変数と同様に扱われます。たとえば、ステートメント `SELECT ? FROM SYSCAT.TABLES` は、結果列のタイプが DB2 には認識されないため無効です。ただし、`SELECT CAST(? AS INTEGER) FROM SYSCAT.TABLES` は、パラメーター・マーカーが INTEGER を表すことがキャストによって示されているため、 DB2 で結果列のタイプが認識されます。

SQL ステートメントにパラメーター・マーカーが複数個含まれている場合、EXECUTE ステートメントの USING 節は、ホスト変数 (各パラメーター・マーカーに 1 つずつ) のリストを指定するか、または各パラメーター・マーカーの SQLVAR 項目を持つ SQLDA を識別しなければなりません。 (LOB の場合は、各パラメーター・マーカーに SQLVAR 項目が 2 つずつあることに注意してください。) ホスト変数リストまたは SQLVAR 項目は、ステートメント内のパラメーター・マーカーの順序に従って突き合わせが行われます。また、これらのデータ・タイプには互換性がなければなりません。

注: 動的 SQL でのパラメーター・マーカーの使用は、静的 SQL でのホスト変数の使用に似ています。いずれの場合も、オプティマイザーは分散統計を使用せず、最適なアクセス・プランを選択することはありえません。

パラメーター・マーカーに適用される規則は、 PREPARE ステートメントで説明されています。

動的に実行される SQL プログラムにおけるパラメーター・マーカーの例

次の例は、動的 SQL プログラムにおけるパラメーター・マーカーの使用方法を示しています。

- C および C++ (**dbuse.sqc/dbuse.sqC**)

  C 言語サンプル **dbuse.sqc** 中の関数 DynamicStmtWithMarkersEXECUTEusingHostVars() は、パラメーター・マーカーとホスト変数を使用して削除を実行する仕方を示しています。

```
EXEC SQL BEGIN DECLARE SECTION;
  char hostVarStmt1[50];
  short hostVarDeptnumb;
EXEC SQL END DECLARE SECTION;

/* prepare the statement with a parameter marker */
strcpy(hostVarStmt1, "DELETE FROM org WHERE deptnumb = ?");
EXEC SQL PREPARE Stmt1 FROM :hostVarStmt1;

/* execute the statement for hostVarDeptnumb = 15 */
hostVarDeptnumb = 15;
EXEC SQL EXECUTE Stmt1 USING :hostVarDeptnumb;
```

- COBOL (**varinp.sqb**)

  次の例は COBOL サンプル **varinp.sqb** からのもので、検索および更新条件におけるパラメーター・マーカーの使用方法を示しています。

```
   EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 pname             pic x(10).
01 dept              pic s9(4) comp-5.
01 st                pic x(127).
01 parm-var          pic x(5).
   EXEC SQL END DECLARE SECTION END-EXEC.

   move "SELECT name, dept FROM staff
-         "   WHERE job = ? FOR UPDATE OF job" to st.
   EXEC SQL PREPARE s1 FROM :st END-EXEC.

   EXEC SQL DECLARE c1 CURSOR FOR s1 END-EXEC.

   move "Mgr" to parm-var.
   EXEC SQL OPEN c1 USING :parm-var END-EXEC

   move "Clerk" to parm-var.
   move "UPDATE staff SET job = ? WHERE CURRENT OF c1" to st.
   EXEC SQL PREPARE s2 from :st END-EXEC.

* call the FETCH and UPDATE loop.
   perform Fetch-Loop thru End-Fetch-Loop
      until SQLCODE not equal 0.

   EXEC SQL CLOSE c1 END-EXEC.
```

組み込み SQL アプリケーションにおけるストアード・プロシージャーの呼び出し

ストアード・プロシージャーは、CALL ステートメントを適切なプロシージャー参照とパラメーターによって編成し、実行することによって、組み込み SQL アプリケーションから呼び出すことができます。 CALL ステートメントは組み込み SQL アプリケーション内で静的あるいは動的に実行できます。ただし、プログラミング言

語ごとに、このコマンドを実行するための方法は異なります。どのホスト言語でも、ストアード・プロシージャーで使用される各ホスト変数は、必要なデータ・タイプに一致するよう宣言する必要があります。

クライアント・アプリケーションとルーチンの呼び出しは、パラメーターと結果セットを介してプロシージャーと情報を交換します。プロシージャーのパラメーターは、データの送信の向きによっても定義されます (パラメーター・モード)。

プロシージャーのパラメーター・タイプには、以下の 3 種類があります。

- IN パラメーター: プロシージャーに渡されるデータ。
- OUT パラメーター: プロシージャーから戻されるデータ。
- INOUT パラメーター: プロシージャーに渡されて、そのプロシージャーの実行中に、プロシージャーから戻される予定のデータに置き換えられるデータ。

パラメーターのモードとそのデータ・タイプは、プロシージャーが CREATE PROCEDURE ステートメントに登録されるときに定義されます。

C および C++ 組み込み SQL アプリケーションにおけるストアード・プロシージャーの呼び出し

C および C++ 組み込み SQL アプリケーションにおけるストアード・プロシージャーの呼び出し

DB2 では、 SQL プロシージャーでの入力パラメーター、出力パラメーター、入出力パラメーターの使用をサポートしています。パラメーターのモードまたは用途を指定するには、 CREATE PROCEDURE ステートメントで IN、OUT、INOUT のいずれかのキーワードを使用します。 IN パラメーターと OUT パラメーターは値による受け渡し、 INOUT パラメーターは参照による受け渡しになります。

C および C++ アプリケーションの場合、ストアード・プロシージャー、INOUT_PARAM は、以下のステートメントを使用して呼び出せます。

```
EXEC SQL CALL INOUT_PARAM(:inout_median:medianind, :out_sqlcode:codeind,
                          :out_buffer:bufferind);
```

上記で、inout_median、out_sqlcode および out_buffer はホスト変数、medianind、codeind および bufferind は NULL 標識変数です。

注: ストアード・プロシージャーは、CALL ステートメントを準備することにより、動的に呼び出すことも可能です。

REXX からのストアード・プロシージャーの呼び出し

ストアード・プロシージャーは、AIX システム上の REXX を除き、そのサーバー上でサポートされる任意の言語で作成することができます。 (クライアント・アプリケーションは AIX 上の REXX で作成できますが、他の言語と同様、AIX システム上の REXX で作成されたストアード・プロシージャーを呼び出すことはできません。)

組み込み SQL アプリケーションでの結果セットの読み取りおよびスクロール

組み込み SQL アプリケーション・プログラムの最も一般的なタスクの 1 つはデータの検索です。このタスクは、*SELECT* ステートメント を使用して実行されます。SELECT ステートメントは、データベース内の表の行のうち指定された検索条件を満たすものを検索する照会の形式です。条件に該当する行が存在すると、そのデータは検索されてホスト・プログラムの指定された変数に入れられ、設計時に意図された用途に応じて使用できます。

注: 組み込み SQL アプリケーションは、サポートされるストアード・プロシージャー・インプリメンテーションのいずれかを使用して、ストアード・プロシージャーを呼び出し、出力および入出力パラメーター値を検索することができます。しかし、組み込み SQL アプリケーションは、ストアード・プロシージャーによって戻された結果セットの読み取りおよびスクロールを行うことはできません。

SELECT ステートメントのコーディングが終わったら、アプリケーションに渡される情報を定義する SQL ステートメントをコーディングします。

SELECT ステートメントの結果は、データベース内の表の場合のように、行と列による表と見なすことができます。 1 行だけが戻された場合、その結果は SELECT INTO ステートメントで指定したホスト変数に直接渡されます。

複数行が戻された場合は、*カーソル* を使ってそれらの行を一度に 1 行ずつ取り出さなければなりません。 カーソルは、順序付きの行ブロック内の特定の行を指し示す、アプリケーション・プログラムで用いられる名前付き制御構造です。

組み込み SQL アプリケーションでの以前に検索したデータのスクロール

アプリケーションがデータベースからデータを検索するとき、 FETCH ステートメントを使うとデータを前方にスクロールすることができます。しかし、結果セットを後方へスクロールできる SQL ステートメント (逆方向 FETCH に相当) はありません。一方、DB2 CLIと DB2 Universal JDBC Driver では読み取り専用の両方向スクロール・カーソルによる逆方向 FETCH をサポートしています。

組み込み SQL アプリケーションの場合、すでに検索されたデータをスクロールするには以下の技法を使うことができます。

- 取り出されたデータのコピーをアプリケーション・メモリーに保管しておき、何らかのプログラム手法を用いてそれをスクロールする方法。
- SQL を用いて、一般的には 2 番目の SELECT ステートメントを使用して、データを再び検索する方法。

組み込み SQL アプリケーションでの、取り出されたデータのコピーの保持

ある状況では、アプリケーションにより取り出されたデータのコピーを保持しておくと便利な場合があります。

データのコピーを保持するには、アプリケーションで次のようにします。

- 取り出されたデータを仮想ストレージに保管します。

- データを一時ファイルに書き込みます (仮想ストレージにデータを入れられない場合)。この方法の利点は、データベース内のデータがトランザクションによって一時的に変更された場合でさえも、ユーザーは、取り出されたデータとまったく同じものを、逆方向スクロールによって常に見ることができるという点です。
- 反復可能読み取りの分離レベルを使用すると、カーソルをクローズしたりオープンすることにより、トランザクションから検索したデータをもう一度検索することができます。検索結果のデータは、他のアプリケーションにより更新されることはありません。データの更新方法は、分離レベルおよびロックにより左右されます。

組み込み SQL アプリケーションでの取り出したデータの 2 度目の検索

データを 2 度検索するために使用する技法は、データをもう一度見ようとする順序により異なります。

次のいずれかのメソッドを使用して、データを 2 度検索できます。

- 先頭からデータを検索する

  結果表の先頭からデータを再び検索するには、アクティブ・カーソルをクローズしてそれを再オープンします。このアクションによってカーソルは結果表の先頭に置かれます。ただし、アプリケーションがその表に対してロックを保持していない限り、他のユーザーがその表に変更を加える可能性があるので、以前に結果表の最初の行であったものが、最初の行でなくなるということもあり得ます。
- 途中からデータを検索する

  結果表の中ほどから 2 度目のデータ検索を行うには、 2 度目の SELECT ステートメントを実行し、そのステートメント上で 2 つ目のカーソルを宣言してください。たとえば、最初の SELECT ステートメントが次のものであるとします。

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    ORDER BY DEPTNO
  ```

  今度は DEPTNO = 'M95' から始まる行に戻って、その場所から順番に行を取り出すとします。この場合は、次のようにコーディングしてください。

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    AND DEPTNO >= 'M95'
    ORDER BY DEPTNO
  ```

  このステートメントによって、カーソルは希望する場所に置かれます。
- 逆順にデータを検索する

  行の昇順がデフォルトの設定です。 DEPTNO のおのおのの値に対する行が 1 つしかない場合、次のステートメントは行をユニークな昇順に指定します。

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    ORDER BY DEPTNO
  ```

  同じ行を逆順に検索するには、次のステートメントのように順序を降順として指定してください。

```
SELECT * FROM DEPARTMENT
  WHERE LOCATION = 'CALIFORNIA'
  ORDER BY DEPTNO DESC
```

2 番目のステートメント上のカーソルは、最初のステートメント上のカーソルからの順番とはまったく逆の順番に行を検索します。検索の順序は、最初のステートメントがユニークな順序を指定している場合にのみ保証されます。

行を逆順で検索する場合、DEPTNO 列に、 1 つは昇順で、もう 1 つは降順の 2 つの索引を持つと便利です。

結果表における行の順序の差異

同一の SELECT ステートメントに対して複数の結果表がある場合、それらの表の行は同じ順序で表示されない可能性があります。データベース・マネージャーは、SELECT ステートメントが ORDER BY 機能を使用していない場合、行の順序を重要視しません。そのため、同じ DEPTNO 値を持つ行がいくつかある場合には、 2 番目の SELECT ステートメントが最初のものとは違う順序で行を検索する場合があります。保証されているのは、ORDER BY DEPTNO 節での要求に従って、それらすべてが部門番号の順に並べられるということだけです。

同じ SQL ステートメントを同じホスト変数を指定して 2 度実行したとしても、順序付けが異なる場合があります。たとえば、2 度目の実行がなされるまでの間にカタログの統計が更新されたり、索引が作成されるかドロップされる場合もあります。その後で SELECT ステートメントをもう一度実行することも考えられます。

最初の SELECT が持っていなかった述部を 2 番目の SELECT が持っている場合、配列が変更することがあります。それはデータベース・マネージャーが新しい述部に対して索引を使用するということがあり得るからです。たとえば、この例で、データベース・マネージャーが最初のステートメントに対しては LOCATION 上の索引を選び、 2 番目のものに対しては DEPTNO 上の索引を選ぶ場合があります。行は索引キーの順に従って取り出されるため、 2 番目の順序は最初の順序と同じとは限りません。

また、2 つの同様な SELECT ステートメントを実行したときに、統計が変更されず、索引の作成もドロップも行われなかったにもかかわらず、行の順序が異なる場合があります。例では、LOCATION の異なる値が多数ある場合、データベース・マネージャーは両方のステートメント用に LOCATION 上で 1 つの索引を選択することができます。 しかし、2 番目のステートメントの DEPTNO の値を次のように変えると、データベース・マネージャーは DEPTNO 上の索引を選ぶことがあります。

```
SELECT * FROM DEPARTMENT
  WHERE LOCATION = 'CALIFORNIA'
  AND DEPTNO >= 'Z98'
  ORDER BY DEPTNO
```

SQL ステートメントの形式とこのステートメントの値との間にはわずかな関係しかないため、順序が ORDER BY 節で固有のものとして定められているのでない限り、 2 つの異なった SQL ステートメントが同じ順序で行を戻してくるとは考えないでください。

組み込み SQL アプリケーションでの以前に検索したデータの更新

逆方向にスクロールして以前に検索されたデータを更新するには、以前に検索されたデータをスクロールする技法と検索されたデータを更新する技法を組み合わせて使用することができます。

以前に検索されたデータを更新するには、以下の 2 つの技法のいずれかを行うことができます。

- 更新するデータ上に 2 番目のカーソルがあり、 SELECT ステートメントが制限されたエレメントをまったく使用していない場合には、カーソル制御 UPDATE ステートメントを使用できる。 2 番目のカーソルを、WHERE CURRENT OF 節の中で指名してください。
- それ以外の場合は、行の中のすべての値を指名するか、あるいは表の主キーを指定する WHERE 節を伴った UPDATE を使用する。 1 つのステートメントを、変数の異なった値について何度でも実行することができます。

組み込み SQL アプリケーションでのカーソルを使用した複数行の選択

SQL では、アプリケーションが行のセットを取り出すことができるようにするため、*カーソル* という手法を用います。

カーソルの概念を理解しやすくするために、データベース・マネージャーが*結果表*を作成し、そこに SELECT ステートメントを実行して検索されたすべての行を保持する場合を考えてみてください。カーソルを用いて結果表の*現在行* を識別して指示することにより、その表からの行をアプリケーションで使用できるようにします。カーソルを使用するとアプリケーションは、データの終わり状態、すなわち NOT FOUND 状態、SQLCODE +100 (SQLSTATE 02000) になるまで結果表から各行を順次取り出すことができます。 SELECT ステートメントを実行した結果取り出された行のセットは、0 行、または 1 行以上で構成されます。これは検索条件を満たす行数によって決まります。

カーソル処理に必要な手順は以下のとおりです。

1. DECLARE CURSOR ステートメントを用いてカーソルを指定する。
2. OPEN ステートメントを用いて照会を実行し、結果表を作成する。
3. FETCH ステートメントを用いて行を一度に 1 行ずつ取り出す。
4. DELETE または UPDATE ステートメントを用いて行を処理する (必要な場合)。
5. CLOSE ステートメントを使用してカーソルを終了する。

アプリケーションは同時に複数のカーソルを使用することができます。各カーソルには DECLARE CURSOR、OPEN、CLOSE、および FETCH ステートメントのセットが必要です。

静的に実行される SQL アプリケーションにおける検索データの更新と削除

カーソルによって参照された行は、更新したり削除したりできます。更新可能な行の場合、カーソルに対応する照会は読み取り専用であってはなりません。

カーソルを用いて更新を行うためには、 UPDATE ステートメントで WHERE CURRENT OF 節を使用してください。結果表の列のうちどれを更新したいのかをシステムに指示するには、FOR UPDATE 節を使用します。 FOR UPDATE での列の指定は全部を選択しなくてもよいため、カーソルで明確に検索されない列でも更新することができます。 FOR UPDATE 節を列名を使わずに指定すると、表の中のすべての列や、外部で全選択された最初の FROM 節で識別されたビューは更新可能であると見なされます。 FOR UPDATE 節では、必要以上の列を指定しないでください。 FOR UPDATE 節に余分な列の名前を指定すると、 DB2 がデータにアクセスする能率を低下させる場合もあります。

カーソルを用いた削除は、DELETE ステートメントで WHERE CURRENT OF 節を使用して行います。一般に、FOR UPDATE 節はカーソルの現在行の削除には必要ありません。 SAA1 に設定された LANGLEVEL でプリコンパイルされ、BLOCKING ALL でバインドされたアプリケーション内の SELECT ステートメントまたは DELETE ステートメントのいずれかに対して動的 SQL を使った場合だけは例外です。この場合、SELECT ステートメントで FOR UPDATE 節を指定する必要があります。

DELETE ステートメントを使用すると、カーソルで参照される行を削除することができます。削除では、カーソルは次の 行の前に置かれたままになるため、カーソルに対して WHERE CURRENT OF 操作をさらに実行する前に、 FETCH ステートメントを発行する必要があります。

静的に実行される SQL プログラムにおける取り出しの例

次の例では、カーソルを使用して表から選択し、カーソルをオープンし、その表から行を取り出します。そして、取り出したそれぞれの行に対して、プログラムは単純な基準に基づいて削除すべきか更新すべきかを判別します。

REXX 言語は静的 SQL をサポートしないため、サンプルはありません。

- C および C++ (**tut_mod.sqc/tut_mod.sqC**)

  次の例は、サンプル **tut_mod** からの例です。この例では、カーソルを使用して表から選択し、カーソルをオープンし、その表から行を取り出し、取り出した行の更新または削除を行い、その後カーソルをクローズします。

```
EXEC SQL DECLARE c1 CURSOR FOR SELECT * FROM staff WHERE id >= 310;
  EXEC SQL OPEN c1;
  EXEC SQL FETCH c1 INTO :id, :name, :dept, :job:jobInd, :years:yearsInd, :salary,
     :comm:commInd;
```

  サンプル **tbmod** は、サンプル **tut_mod** の長いもので、表データの変更に関するほとんどすべてのケースを示しています。

- COBOL (**openftch.sqb**)

  次の例は、サンプル **openftch** からの例です。この例では、カーソルを使用して表から選択し、カーソルをオープンし、その表から行を取り出します。

```
EXEC SQL DECLARE c1 CURSOR FOR
  SELECT name, dept FROM staff
  WHERE job='Mgr'
  FOR UPDATE OF job END-EXEC.

EXEC SQL OPEN c1 END-EXEC
```

```
* call the FETCH and UPDATE/DELETE loop.
  perform Fetch-Loop thru End-Fetch-Loop
  until SQLCODE not equal 0.

 EXEC SQL CLOSE c1 END-EXEC.
```

組み込み SQL アプリケーションでのエラー・メッセージ検索

アプリケーションが書かれている言語により、エラー情報を検索するメソッドが異なります。

- C、C++、および COBOL アプリケーションでは渡される SQLCA に関する情報を GET ERROR MESSAGE API を使って得られます。

```
C Example: The SqlInfoPrint procedure from UTILAPI.C
/******************************************************************************
** 1.1 - SqlInfoPrint - prints diagnostic information to the screen.
**
******************************************************************************/
int SqlInfoPrint( char * appMsg,
    struct sqlca * pSqlca,
    int line,
    char * file )
{   int  rc = 0;
    char sqlInfo[1024];
    char sqlInfoToken[1024];
    char sqlstateMsg[1024];
    char errorMsg[1024];
    if (pSqlca->sqlcode != 0 && pSqlca->sqlcode != 100)
    {  strcpy(sqlInfo, "");
       if( pSqlca->sqlcode < 0)
       {   sprintf( sqlInfoToken, "¥n---- error report ----¥n");
           strcat( sqlInfo, sqlInfoToken);
       }
       else
       {   sprintf( sqlInfoToken, "¥n---- warning report ----¥n");
           strcat( sqlInfo, sqlInfoToken);
       } /* endif */

       sprintf( sqlInfoToken, " app. message =
       strcat( sqlInfo, sqlInfoToken);
       sprintf( sqlInfoToken, " line =
       strcat( sqlInfo, sqlInfoToken);
       sprintf( sqlInfoToken, " file =
       strcat( sqlInfo, sqlInfoToken);
       sprintf( sqlInfoToken, " SQLCODE = sqlcode);
       strcat( sqlInfo, sqlInfoToken);

       /* get error message */
       rc = sqlaintp( errorMsg, 1024, 80, pSqlca);
       /* return code is the length of the errorMsg string */
       if( rc > 0)
       { sprintf( sqlInfoToken, "
         strcat( sqlInfo, sqlInfoToken);
       }

       /* get SQLSTATE message */
       rc = sqlogstt( sqlstateMsg, 1024, 80, pSqlca->sqlstate);
       if (rc == 0)
       { sprintf( sqlInfoToken, "
         strcat( sqlInfo, sqlInfoToken);
       }

       if( pSqlca->sqlcode < 0)
       { sprintf( sqlInfoToken, "--- end error report ---¥n");
```

```
        strcat( sqlInfo, sqlInfoToken);
        printf("
        return 1;
      }
      else
      { sprintf( sqlInfoToken, "--- end warning report ---¥n");
        strcat( sqlInfo, sqlInfoToken);

        printf("
        return 0;
      } /* endif */
    } /* endif */
    return 0;
}

COBOL Example: From CHECKERR.CBL
    *********************************
    * GET ERROR MESSAGE API called *
    *********************************
      call "sqlgintp" using
           by value buffer-size
           by value line-width
           by reference sqlca
           by reference error-buffer
       returning error-rc.
    ************************
    * GET SQLSTATE MESSAGE *
    ************************
      call "sqlggstt" using
           by value buffer-size
           by value line-width
           by reference sqlstate
           by reference state-buffer
       returning state-rc.
    if error-rc is greater than 0
      display error-buffer.

    if state-rc is greater than 0
      display state-buffer.

    if state-rc is less than 0
      display "return code from GET SQLSTATE =" state-rc.

    if SQLCODE is less than 0
      display "--- end error report ---"
      go to End-Prog.

    display "--- end error report ---"
    display "CONTINUING PROGRAM WITH WARNINGS!".
```

- REXX アプリケーションは CHECKERR プロシージャーを使用します。

```
/******   CHECKERR - Check SQLCODE *****/
CHECKERR:
  arg errloc

  if  ( SQLCA.SQLCODE = 0 ) then
    return 0
  else do
    say '--- error report ---'
    say 'ERROR occurred :' errloc
    say 'SQLCODE :' SQLCA.SQLCODE

   /*********************¥
   * GET ERROR MESSAGE  *
   ¥*********************/
    call SQLDBS 'GET MESSAGE INTO :errmsg LINEWIDTH 80'
    say errmsg
```

```
      say '--- end error report ---'

      if (SQLCA.SQLCODE < 0 ) then
        exit
      else do
        say 'WARNING - CONTINUING PROGRAM WITH ERRORS'
        return 0
      end
    end
  return 0
```

SQLCODE、SQLSTATE、および SQLWARN フィールドのエラー情報

エラー情報は、SQLCA 構造体の SQLCODE と SQLSTATE のフィールドに戻されます。SQLCA 構造体は、すべての実行可能 SQL ステートメントとほとんどのデータベース・マネージャー API 呼び出しの実行後に更新されます。

実行可能 SQL ステートメントが入っているソース・ファイルには、 `sqlca` という名前を持つ SQLCA 構造体が少なくとも 1 つあります。 SQLCA 構造体は SQLCA 組み込みファイルで定義されます。組み込み SQL ステートメントはないがデータベース・マネージャー API を呼び出すソース・ファイルには、1 つ以上の SQLCA 構造体を組み込むことができますが、各構造体の名前は任意に付けられます。

ご使用のアプリケーションが FIPS 127-2 標準に準拠している場合、 SQLCA 構造体の代わりに SQLSTATE および SQLCODE を C、C++、COBOL、および FORTRAN アプリケーション用のホスト変数として宣言することができます。

SQLCODE 値が 0 である場合は、正常に実行されたことを示します (SQLWARN 警告状態を伴うこともあります)。正の値は、ステートメントは正常に実行されたが、ホスト変数の切り捨てなどの警告を伴うことを意味します。負の値は、エラー条件が発生したことを意味します。

追加のフィールド SQLSTATE には、他の IBM データベース製品および SQL92 準拠のデータベース・マネージャーと一貫性がある標準化エラー・コードが含まれています。実際には、SQLSTATE 値は多くのデータベース・マネージャーと共通であるため、移植性を考慮する場合は SQLSTATE 値を使用します。

SQLWARN フィールドには、SQLCODE がゼロの場合でも警告標識の配列が含まれます。 SQLWARN 配列の第 1 エレメントである SQLWARN0 には、その他のエレメントがすべてブランクである場合はブランクが入ります。その他のエレメントの少なくとも 1 つに警告文字が含まれている場合は、 SQLWARN0 には `W` が入ります。

注: さまざまな IBM RDBMS サーバーにアクセスするアプリケーションを開発する場合は、以下のようにしてください。

- 可能な時点で、アプリケーションが SQLCODE ではなく SQLSTATE をチェックするようにする。
- アプリケーションが DB2 Connect を使用する場合は、 DB2 Connect に付属しているマッピング機能を用いて、異なるデータベース間の SQLCODE 変換をマップする。

出口リスト・ルーチンに関する考慮事項

出口リスト・ルーチンでは、SQL や DB2 API 呼び出しを使用しないでください。出口ルーチンの中ではデータベースから切断することはできないことに注意してください。

例外、シグナル、および割り込みハンドラーについての考慮事項

例外、シグナル、または割り込みハンドラーは、例外が起きたときに制御を獲得するルーチンです。ここで適用されるハンドラーのタイプは、オペレーティング環境によって異なります。以下のとおりです。

Windows オペレーティング・システム
: Ctrl-C または Ctrl-Break を押すことにより、割り込みが生成されます。

UNIX オペレーティング・システム
: 通常、Ctrl-C を押すと SIGINT 割り込みシグナルが生成されます。キーボードは簡単に再定義できるため、 SIGINT はマシン上のさまざまなキー・シーケンスで生成される場合があることに注意してください。

例外、シグナル、および割り込みハンドラーの中には SQL ステートメントを置かないでください (COMMIT と ROLLBACK は例外)。これらのエラー状態が起きた場合には、データの矛盾を避けるために、 ROLLBACK するのが普通です。

例外/シグナル/割り込みハンドラーでの COMMIT および ROLLBACK のコーディングは、慎重に行ってください。これらのステートメントのいずれかをそれだけで呼び出す場合、 COMMIT または ROLLBACK は現行の SQL ステートメントが完了するまで実行されません (SQL ステートメントが実行中の場合)。これは、Ctrl-C ハンドラーから実行するには望ましい動作ではありません。

ROLLBACK を発行する前に、 INTERRUPT API (sqleintr/sqlgintr) を呼び出すことで解決できます。 この API は、現行の SQL 照会を中断させ (アプリケーションが SQL 照会を実行中の場合)、 ROLLBACK が即時に開始されるようにします。 ROLLBACK よりも COMMIT を実行する場合、現行のコマンドを中断する必要はありません。

さまざまなハンドラーに特有の詳細な考慮事項については、ご使用のプラットフォームの資料を参照してください。

組み込み SQL アプリケーションからの切断

切断ステートメントは、データベースでの作業の最終ステップです。このトピックでは、サポートされているホスト言語での切断ステートメントの例を提供します。

C および C++ 組み込み SQL アプリケーションにおける DB2 データベースからの切断

C および C++ アプリケーションの場合、データベース接続は、以下のステートメントの実行によってクローズできます。

```
EXEC SQL CONNECT RESET;
```

COBOL 組み込み SQL アプリケーションにおける DB2 データベースからの切断

COBOL アプリケーションの場合、データベース接続は、以下のステートメントの実行によってクローズできます。

```
EXEC SQL CONNECT RESET END-EXEC.
```

REXX 組み込み SQL アプリケーションにおける DB2 データベースからの切断

REXX アプリケーションの場合、データベース接続は、以下のステートメントの実行によってクローズできます。

```
CALL SQLEXEC 'CONNECT RESET'
```

FORTRAN アプリケーションの場合、データベース接続は、以下のステートメントの実行によってクローズできます。

```
EXEC SQL CONNECT RESET
```

第 6 章 組み込み SQL アプリケーションの構築

組み込み SQL アプリケーションのソース・コードを作成したなら、追加のステップを実行してビルドを行う必要があります。新しい組み込み SQL データベース・アプリケーションを開発する場合は、64 ビットの実行可能プログラムのビルドを検討してください。プログラムのコンパイルとリンクに加えて、*プリコンパイル* および*バインド* を行わなくてはなりません。

プリコンパイルの処理では、組み込み SQL ステートメントをホスト言語コンパイラーが処理できる DB2 実行時 API 呼び出しに変換します。デフォルトでは、パッケージはプリコンパイル時に作成されます。オプションで、バインド・ファイルがプリコンパイル時に作成されるようにすることもできます。バインド・ファイルには、アプリケーション・プログラム内の SQL ステートメントに関する情報が入ります。バインド・ファイルは後に `BIND` コマンドとともに使用して、アプリケーションのパッケージを作成することができます。

バインドとは、バインド・ファイルからパッケージ を作成し、それをデータベースに保管する処理です。バインド・ファイルは、アプリケーションによるアクセスを必要とするデータベースそれぞれにバインドする必要があります。アプリケーションが複数のデータベースにアクセスする場合、パッケージはデータベースごとに作成する必要があります。

コンパイルされるホスト言語で記述されたアプリケーションを実行するには、データベース・マネージャーが実行時に必要とするパッケージを作成しなければなりません。次の図は、上記のステップの順序とともに、一般的なコンパイルされる DB2 アプリケーションのさまざまなモジュールを示しています。

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し、組み込み SQL ソース・ステートメントを、データベース・マネージャーが使用できる形式に変換する。

   アプリケーションに置かれる SQL ステートメントは、ホスト言語に固有のものではないため、データベース・マネージャーには、SQL 構文をホスト言語が処理できるように変換する機能があります。 C、C++、COBOL または FORTRAN 言語では、この変換は、`PRECOMPILE` (あるいは `PREP`) コマンドを使用して呼び出される DB2 プリコンパイラーによって扱われます。プリコンパイラーは、組み込み SQL ステートメントを DB2 ランタイム・サービス API 呼び出しに直接変換します。プリコンパイラーはソース・ファイルを処理する際に、特に SQL ステートメントを探して処理し、非 SQL ホスト言語は無視します。
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。

コンパイルとリンク (ステップ 3 および 4) では、必要なオブジェクト・モジュールが作成されます。

5. バインド・ファイルをバインドし、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)。バインディング (ステップ 5) では、プログラムの実行時にデータベース・マネージャーによって使用されるパッケージが作成されます。
6. アプリケーションを実行する。アプリケーションが、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。

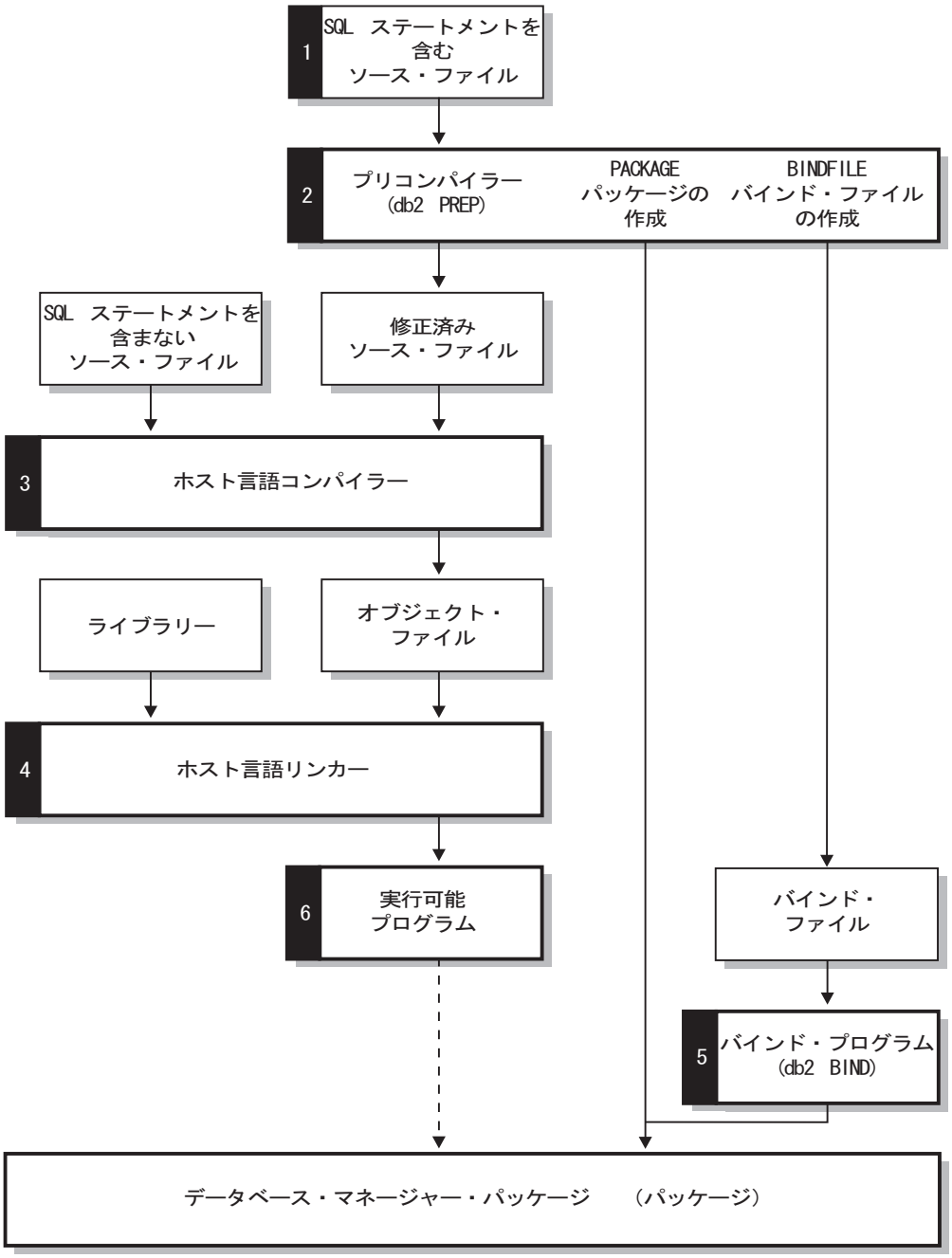


図 3. コンパイルされるホスト言語で記述されたプログラムの作成

PRECOMPILE コマンドによる組み込み SQL アプリケーションのプリコンパイル

組み込み SQL アプリケーションのソース・ファイルを作成してから、 SQL ステートメントが入っているそれぞれのホスト言語ファイルを、個々のホスト言語に固有のオプションを使い、PREP コマンドを使ってプリコンパイルする必要があります。プリコンパイラーはソース・ファイルに含まれている SQL ステートメントをコメントに変換し、そのステートメントについて DB2 実行時 API 呼び出しを生成します。

ソース・ファイルは必ず、特定のデータベースに対してプリコンパイルしなければなりません。そのデータベースは、最終的にそのアプリケーションとともに使用されることのないデータベースであってもかまいません。実際に、テスト・データベースを開発用に使用することができます。そして、アプリケーションを十分にテストしてから、バインド・ファイルを 1 つ以上の実動データベースにバインドすることができます。これは、*据え置きバインディング* として知られています。

注: プリコンパイルが行われたクライアントより古いバージョンのクライアントにおける組み込みアプリケーションの実行は、(そのアプリケーションがどこでコンパイルされたかにかかわらず) サポートされていません。例えば、DB2 V9.5 クライアント上で組み込みアプリケーションをプリコンパイルし、次いで DB2 V9.1 クライアント上でそのアプリケーションを実行する操作はサポートされていません。アプリケーションが使用しているコード・ページがデータベースのコード・ページと異なる場合、プリコンパイル時にどのコード・ページを使用するのかを考慮する必要があります。

アプリケーションでユーザー定義関数 (UDF) またはユーザー定義特殊タイプ (UDT) を使用している場合は、アプリケーションをコンパイルする際に FUNCPATH オプションを使用する必要があります。このオプションは、静的 SQL を含むアプリケーションで UDF と UDT を使用できるようにするための関数パスを指定します。 FUNCPATH を指定しない場合、デフォルトの関数パスは *SYSIBM*、*SYSFUN*、*USER* になります (*USER* は現行のユーザー ID のことです)。

アプリケーションをプリコンパイルする前に、明示的または暗黙に、サーバーに接続する必要があります。アプリケーション・プログラムをクライアント・ワークステーションでプリコンパイルして、プリコンパイラーが変更されたソースとメッセージをクライアント上で生成しても、プリコンパイラーはサーバー接続を使用していくらかの妥当性検査を実行します。

さらに、プリコンパイラーは、データベース・マネージャーがデータベースに対する SQL ステートメントを処理する上で必要な情報も作成します。この情報は、選択したプリコンパイラー・オプションによって、パッケージ、バインド・ファイル、またはその両方に保管されます。

プリコンパイラーの使用の一般的な例を以下に示します。 *filename.sqc* という C 組み込み SQL ソース・ファイルをプリコンパイルするために、以下のコマンドを発行して、デフォルト名 filename.c の C ソース・ファイルと、デフォルト名 filename.bnd のバインド・ファイルを作成することができます。

```
DB2 PREP filename.sqc BINDFILE
```

プリコンパイラーは、最大で次の 4 つのタイプの出力を生成します。

修正済みソース

このファイルは、プリコンパイラーが SQL ステートメントを DB2 実行時 API 呼び出しに変換した後の、元のソース・ファイルの新バージョンです。適切なホスト言語拡張子が与えられます。

パッケージ

PACKAGE オプション (デフォルト) が使用されている場合、または BINDFILE、SYNTAX、SQLFLAG オプションのいずれも指定されていない場合は、パッケージは接続しているデータベースに保管されます。このパッケージには、このデータベースだけに対して特定のソース・ファイルの静的 SQL ステートメントを実行するために必要なすべての情報が入ります。PACKAGE USING オプションを使って別の名前を指定した場合以外は、プリコンパイラーはパッケージ名を、ソース・ファイル名の最初の 8 文字から作成します。

PACKAGE オプションを SQLERROR CONTINUE を指定せずに使用する場合、プリコンパイル処理中に使用するデータベースには、ソース・ファイル内の静的 SQL が参照するデータベース・オブジェクトがすべて含まれていなければなりません。たとえば、SELECT ステートメントは、参照する表がデータベースに入っていない限り、プリコンパイルすることはできません。

VERSION オプションを指定すると、バインド・ファイル (BINDFILE オプションが使用された場合) とパッケージ (PREP 実行時にバインドされるか個別にバインドされる場合) には特定のバージョン ID が付けられます。名前と作成者が同じ複数のバージョンのパッケージが同時に存在可能です。

バインド・ファイル

BINDFILE オプションを使用すると、プリコンパイラーはパッケージの作成に必要なデータを含むバインド・ファイル (拡張子は .bnd) を作成します。後で BIND コマンドでこのファイルを使用することにより、アプリケーションを 1 つ以上のデータベースにバインドできます。 BINDFILE を指定していても PACKAGE オプションを指定していない場合、BIND コマンドを呼び出さない限りバインドは行われません。コマンド行プロセッサー (CLP) の場合は、 PREP のデフォルトは BINDFILE オプションを指定しません。したがって、CLP の使用中にバインドを延期するときは、 BINDFILE オプションを指定する必要があります。

SQLERROR CONTINUE を指定すると、SQL ステートメントをバインドするときにエラーが発生してもパッケージが作成されます。VALIDATE RUN も指定されている場合、許可やファイルの存在の問題でバインドが失敗したそれらのステートメントを、実行時に追加バインドすることができます。実行時にそれらのステートメントを実行しようとするとエラーが発生します。

メッセージ・ファイル

MESSAGES オプションを使用している場合、プリコンパイラーはメッセージを指定のファイルに転送します。このメッセージには、警告およびプリコンパイル中に発生した問題を示しているエラー・メッセージが含まれます。ソース・ファイルが正常にプリコンパイルされない場合は、警告およびエラー・メッセージを使用して問題を判別し、ソース・ファイルを訂正してか

ら、再度プリコンパイルしてみてください。 MESSAGES オプションを使用していない場合は、プリコンパイルのメッセージは標準出力に書き込まれます。

複数のデータベース・サーバーにアクセスする組み込み SQL アプリケーションのプリコンパイル

複数のサーバーにアクセスするアプリケーション・プログラムをプリコンパイルするには、以下のいずれかを行ってください。

- データベースごとに SQL ステートメントを個別のソース・ファイルに分割する。異なるデータベースのための SQL ステートメントを同じファイルに混合しないでください。それぞれのソース・ファイルを、適切なデータベースに対してプリコンパイルすることができます。この方法で行うことをお勧めします。
- 動的 SQL のみを使用してアプリケーションをコーディングし、プログラムがアクセスするそれぞれのデータベースにバインドする。
- すべてのデータベースが同じであると思われる場合、つまり定義が同じである場合は、 SQL ステートメントを 1 つのソース・ファイルにグループ化することができる。

アプリケーションが DB2 Connect を介してホスト・アプリケーション・サーバーへアクセスするとしても、同じプロシージャーが適用されます。サーバーで使用可能な PREP オプションを使用して、接続する予定のサーバーに対してアプリケーションをプリコンパイルしてください。

組み込み SQL アプリケーション・パッケージとアクセス・プラン

プリコンパイラーはデータベース内にパッケージを作成し、作成するようユーザーが指定をすれば、オプションでバインド・ファイルも作成します。

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 オプティマイザー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、オプティマイザー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、オプティマイザーがアクセス・プランを作成します。

データベースに保管されたパッケージは、単一のソース・ファイル内の特定の SQL ステートメントを実行するために必要な情報を含んでいます。データベース・アプリケーションは、そのアプリケーションを作成するのに使用するプリコンパイルされたすべてのソース・ファイルに対して、 1 つのパッケージを用います。それぞれのパッケージは個別のエンティティーであり、同じアプリケーションまたは他のアプリケーションで使用される別のパッケージとは関係ありません。パッケージを作成するには、バインドを実行可能にしてソース・ファイルに対してプリコンパイラーを実行するか、1 つ以上のバインド・ファイルで後からバインド・プログラムを実行します。

バインド・ファイルには、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

て最適化されたパッケージが作成されます。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。

CURRENT PACKAGE PATH 特殊レジスターを使用したパッケージ・スキーマ修飾

パッケージ・スキーマは、パッケージを論理的にグループ化する手段を提供します。パッケージをスキーマにグループ化するための他の方法もあります。一部のインプリメンテーションでは、環境ごとに 1 つのスキーマを使用します (たとえば production と test スキーマ)。他のインプリメンテーションでは、ビジネス領域ごとに 1 つのスキーマを使用するか (たとえば `stocktrd` と `onlinebnk` スキーマ)、またはアプリケーションごとに 1 つのスキーマを使用します (たとえば `stocktrdAddUser` と `onlinebnkAddUser`)。一般の管理目的で、またはパッケージのバリエーションを提供するために (たとえば、アプリケーションのバックアップ・バリエーションの保守、またはアプリケーションの新規バリエーションのテスト)、パッケージをグループ化することもできます。

複数のスキーマがパッケージに使用される場合、データベース・マネージャーは、どのスキーマからパッケージを探すのかを決定する必要があります。このタスクを実行するために、データベース・マネージャーは CURRENT PACKAGESET 特殊レジスターの値を使用します。この特殊レジスターを単一のスキーマ名に設定して、呼び出されるどのパッケージもそのスキーマに属していることを示すことができます。アプリケーションが異なるスキーマのパッケージを使用する場合、パッケージのスキーマが前のパッケージのスキーマと異なるなら、各パッケージが呼び出される前に SET CURRENT PACKAGESET ステートメントを発行する必要があります。

注: DB2 for z/OS® (DB2 for z/OS) バージョン 9.1 だけが CURRENT PACKAGESET 特殊レジスターを持っており、これによって値 (単一のスキーマ名) を、SET CURRENT PACKAGESET ステートメントに対応させて明示的に設定することができます。 DB2 Database for Linux, UNIX, and Windows は SET CURRENT PACKAGESET ステートメントを持っていますが、CURRENT PACKAGESET 特殊レジスターは持っていません。これは、DB2 Database for Linux, UNIX, and Windows の他のコンテキスト (SELECT ステートメント内など) では CURRENT PACKAGESET を参照できないことを意味します。 DB2 for i5/OS® は CURRENT PACKAGESET のサポートを提供していません。

DB2 データベース・サーバーは、パッケージ解決中にスキーマのリストを考慮できるときは、さらに柔軟性が増します。スキーマのリストは、CURRENT PATH 特殊レジスターによって提供される SQL パスに似ています。スキーマ・リストは、ユーザー定義関数、プロシージャー、メソッド、および特殊タイプに使用されます。

注: SQL パスとは、非修飾関数、プロシージャー、メソッド、または特殊タイプ名のスキーマを決定する場合に、DB2 が考慮するスキーマ名のリストです。

パッケージの複数のバリエーション (パッケージ用の BIND オプションの複数セット) を、単一のコンパイル済みプログラムに関連付けたい場合には、SQL オブジェクトに使用されるスキーマのパスを、パッケージに使用されるスキーマのパスから分離することを考慮してください。

CURRENT PACKAGE PATH 特殊レジスターによって、パッケージ・スキーマのリストを指定することができます。他の DB2 ファミリー製品は、CURRENT PATH および CURRENT PACKAGESET などの特殊レジスターと同様の機能を提供します。それらは、ネストされたプロシージャーおよびユーザー定義関数に対して、アプリケーションを呼び出すランタイム環境を破壊することなく、プッシュまたはポップ されます。 CURRENT PACKAGE PATH 特殊レジスターは、パッケージ・スキーマ解決用に、この機能を提供しています。

多くのインストールでは、パッケージ用に複数のスキーマを使用します。パッケージ・スキーマのリストを指定しない場合、異なるスキーマからパッケージを要求するごとに、SET CURRENT PACKAGESET ステートメント (多くても 1 つのスキーマ名を含むことができる) を発行する必要があります。ただし、スキーマ名のリストを指定するためにアプリケーションの開始時点で SET CURRENT PACKAGE PATH ステートメントを発行する場合、異なるスキーマでパッケージが必要となるごとに、SET CURRENT PACKAGESET ステートメントを発行する必要はありません。

たとえば、以下のパッケージが存在し、以下のリストを使用して、サーバー上に存在している最初のものを呼び出すと想定します。
SCHEMA1.PKG1、SCHEMA2.PKG2、SCHEMA3.PKG3、SCHEMA.PKG、および SCHEMA5.PKG5。 DB2 Database for Linux, UNIX, and Windows (単一のスキーマ名を受け入れる) での SET CURRENT PACKAGESET ステートメントに対する現行のサポートを想定すると、特定のスキーマを指定する各パッケージの呼び出しを試行する前に、SET CURRENT PACKAGESET ステートメントを発行する必要があります。たとえば、5 つの SET CURRENT PACKAGESET ステートメントが発行される必要があります。ただし、CURRENT PACKAGE PATH 特殊レジスターを使用すると、単一の SET ステートメントで十分です。以下に例を示します。

```
SET CURRENT PACKAGE PATH = SCHEMA1, SCHEMA2, SCHEMA3, SCHEMA, SCHEMA5;
```

注: DB2 Database for Linux, UNIX, and Windows では、 SQLE-CLIENT-INFO 構造内の SQLSetConnectAttr API を使用することによって、および組み込み SQL プログラム内の SET CURRENT PACKAGE PATH ステートメントを組み込むことによって、db2cli.ini ファイル内に CURRENT PACKAGE PATH 特殊レジスターを設定することができます。DB2 for z/OS® のみが、バージョン 8 またはそれ以降で、SET CURRENT PACKAGE PATH ステートメントをサポートしています。DB2 Database for Linux, UNIX, and Windows サーバーまたは DB2 for i5/OS に対してこのステートメントを発行した場合、-30005 が戻されます。

複数のスキーマを使用して、パッケージのいくつかのバリエーションを保守することができます。これらのバリエーションは、実稼働環境で加えられた変更を制御するために非常に役立ちます。パッケージの異なるバリエーションを使用して、パッケージのバックアップ・バージョン、またはパッケージのテスト・バージョン (たとえば、新規索引の影響を評価するための) を保持することもできます。パッケージの前のバージョンは、バックアップ・アプリケーション (ロード・モジュールまたは実行可能ファイル) と同じ方法で使用され、特に前のバージョンに復帰する機能を備えることができます。

たとえば、PROD スキーマが実働アプリケーションによって使用される現行のパッケージを組み込み、BACKUP スキーマがそれらのパッケージのバックアップ・コピ

ーを保管すると想定します。アプリケーションの新規バージョン (つまりパッケージ) は、 PROD スキーマを使用してバインディングすることによって、実動にプロモートされます。バックアップ・スキーマ (BACKUP) を使用するアプリケーションの現行バージョンをバインディングすることによって、パッケージのバックアップ・コピーが作成されます。それから、実行時に SET CURRENT PACKAGE PATH ステートメントを使用して、スキーマがパッケージについてチェックされる順序を指定することができます。 MYAPPL アプリケーションのバックアップ・コピーは、BACKUP スキーマを使用してバインドされ、現在実動中のアプリケーションの現行バージョンは、 PROD.MYAPPL パッケージを作成する PROD スキーマにバインドされていると想定します。使用可能な場合には PROD スキーマのパッケージのバリエーションを使用することを指定するには (不可能な場合には BACKUP スキーマのバリエーションが使用される)、特殊レジスターのために次の SET ステートメントを発行します。

```
SET CURRENT PACKAGE PATH = PROD, BACKUP;
```

パッケージの前のバージョンに復帰することが必要な場合、アプリケーションの実働バージョンは DROP PACKAGE ステートメントでドロップすることができます。これは代わりに、BACKUP スキーマを使用してバインドされたアプリケーション (ロード・モジュールまたは実行可能ファイル) の旧バージョンを呼び出します (各オペレーティング・システム・プラットフォームに固有の、アプリケーション・パス技法がここで使用できます)。

注: この例では、パッケージのバージョン間の相違は、パッケージを作成するために使用された BIND オプションだけであると想定しています (つまり、実行可能コードには相違はないということです)。

アプリケーションは、使用するスキーマを選択するために SET CURRENT PACKAGESET ステートメントを使用しません。その代わりこれによって、DB2 は CURRENT PACKAGE PATH 特殊レジスターにリストされたスキーマからチェックして、パッケージを選ぶことができます。

注: DB2 for z/OS のプリコンパイル・プロセスは、DBRM (LEVEL オプションを使用して設定できる) に整合性トークンを保管し、パッケージ解決中に、プログラム内の整合性トークンがパッケージと一致することを確認するためのチェックが行われます。同様に、DB2 Database for Linux, UNIX, and Windows のバインド・プロセスは、バインド・ファイル内にタイム・スタンプを保管します。 DB2 Database for Linux, UNIX, and Windows は LEVEL オプションもサポートします。

異なるスキーマでパッケージの複数のバージョンを作成する別の理由は、さまざまな BIND オプションを有効にできるということです。たとえば、パッケージ内の非修飾名のリファレンスにはさまざまな修飾子を使用できます。

アプリケーションは、非修飾表名で作成されることもよくあります。これは、表名および構造は同じだが、さまざまなインスタンスを識別するためのさまざまな修飾子の付けられた複数の表をサポートしています。たとえば、テスト・システムと実動システムはそれぞれの中で同一のオブジェクトを作成できますが、それらは異なる修飾子を持つことがあります (たとえば、PROD および TEST)。別の例は、異なる DB2 システム間の複数の表にデータを配分するアプリケーションですが、それぞれの表は異なる修飾子を持っています (たとえば、
EAST、WEST、NORTH、SOUTH や、COMPANYA、COMPANYB や、

Y1999、Y2000、Y2001 など)。 DB2 for z/OS では、BIND コマンドの QUALIFIER オプションを使用して表の修飾子を指定します。 QUALIFIER オプションを使用する場合には、複数のプログラムを保守する必要はなく、それぞれは非修飾表にアクセスする必要がある完全修飾名を指定します。その代わりに、アプリケーションからの SET CURRENT PACKAGESET ステートメントを発行し、単一のスキーマ名を指定することによって、訂正したパッケージに実行時にアクセスすることができます。ただし、SET CURRENT PACKAGESET を使用する場合、複数のアプリケーションは依然として保持されて変更される必要があります。要求されたパッケージにアクセスするために、それぞれのアプリケーションが独自の SET CURRENT PACKAGESET ステートメントを持ちます。代わりに SET CURRENT PACKAGE PATH ステートメントを発行する場合、すべてのスキーマをリストすることができます。実行時に、DB2 は訂正したパッケージを選択することができます。

注: DB2 Database for Linux, UNIX, and Windows は QUALIFIER BIND オプションもサポートしています。ただし、QUALIFIER BIND オプションは、 BIND コマンドの DYNAMICRULES オプションを使用する静的 SQL またはパッケージだけに影響を与えます。

プリコンパイラー生成タイム・スタンプ

バインドを実行可能にしてアプリケーションをプリコンパイルすると、タイム・スタンプが一致するパッケージと修正済みソース・ファイルが生成されます。これらのタイム・スタンプは、それぞれ整合性トークンとして認識されます。複数のバージョンのパッケージが (PRECOMPILE VERSION オプションを使用したために) 存在する場合、それぞれのバージョンが関連したタイム・スタンプを持ちます。アプリケーションが実行されるとき、パッケージ名、作成者およびタイム・スタンプがデータベース・マネージャーに送信され、アプリケーションから送られたパッケージの名前、作成者、およびタイム・スタンプと一致するものが存在するかどうかがチェックされます。一致するものがない場合、次の 2 つの SQL エラー・コードのうちのどちらかがアプリケーションに戻されます。

- SQL0818N (タイム・スタンプの矛盾)。名前と作成者が一致する (しかし整合性トークンは一致しない) パッケージは 1 つしか見つからなかったものの、パッケージのバージョンが空ストリング (″″) だった場合、このエラーが戻されます。
- SQL0805N (パッケージが見つからない)。このエラーは、上記以外のすべてのエラーで戻されます。

アプリケーションをデータベースにバインドする場合、 *PREP* コマンドの *PACKAGE USING* オプションを使用してデフォルトを指定変更しない限り、アプリケーション名の最初の 8 文字がパッケージ名として使用されることを覚えておいてください。 PREP コマンドの VERSION オプションを指定しないかぎり、バージョン ID は空ストリング (″″) になります。つまり、同じ名前の 2 つのプログラムをバージョン ID を変えずにプリコンパイルしてバインドすると、最初のパッケージは 2 番目のパッケージによって置き換えられます。最初のプログラムを実行すると、そのプログラムでは修正済みソース・ファイルのタイム・スタンプとデータベースのパッケージのタイム・スタンプとが一致していないため、タイム・スタンプ・エラーまたはパッケージが見つからないというエラーになります。パッケージが見つからないというエラーは、以下の例のようにプリコンパイルまたはバインドの ACTION REPLACE REPLVER オプションを使用した場合にも発生します。

1. パッケージ SCHEMA1.PKG を VERSION VER1 を指定してプリコンパイルする。そして、関連したアプリケーション A1 を生成する。
2. パッケージ SCHEMA1.PKG を VERSION VER2 ACTION REPLACE REPLVER VER1 を指定してプリコンパイルおよびバインドする。そして、関連したアプリケーション A2 を生成する。

   2 番目のプリコンパイルとバインドにより VER2 という VERSION を持つパッケージ SCHEMA1.PKG が生成され、ACTION REPLACE REPLVER VER1 が指定されているので VER1 という VERSION を持っていた SCHEMA1.PKG パッケージは削除されます。

   最初のアプリケーションを実行しようとすると、パッケージのミスマッチが発生し失敗します。

同じような症状が次の例でも発生します。

1. パッケージ SCHEMA1.PKG を VERSION VER1 を指定してプリコンパイルする。そして、関連したアプリケーション A1 を生成する。
2. パッケージ SCHEMA1.PKG を VERSION VER2 を指定してプリコンパイルする。そして、関連したアプリケーション A2 を生成する。

   この時点では、アプリケーション A1 も A2 も実行可能であり、パッケージ SCHEMA1.PKG からバージョン VER1 と VER2 をそれぞれ実行します。たとえば、最初のパッケージが (DROP PACKAGE SCHEMA1.PKG VERSION VER1 SQL ステートメントを使用して) ドロップされている場合、アプリケーション A1 を実行しようとすると、パッケージが見つからないというエラーになります。

ソース・ファイルがプリコンパイルされているもののパッケージが作成されていない場合、タイム・スタンプの一致するバインド・ファイルと修正済みソース・ファイルが生成されます。アプリケーションを実行するためには、パッケージを作成するためにバインド・ファイルを個別に BIND ステップでバインドし、修正済みソース・ファイルをコンパイルしてリンクします。複数のソース・モジュールを必要とするアプリケーションでは、バインディング・プロセスをそれぞれのバインド・ファイルごとに行わなければなりません。

この実行据え置きバインディングのシナリオでは、バインド・ファイルにはプリコンパイル中に修正済みソース・ファイルに保管されたタイム・スタンプと同じものが入るため、アプリケーションとパッケージのタイム・スタンプは一致することになります。

組み込み SQL アプリケーションのプリコンパイルからのエラーおよび警告

組み込み SQL のプリコンパイル時のエラーは、組み込み SQL プリコンパイラーによって検出されます。組み込み SQL プリコンパイラーは、SQL ステートメント中でのセミコロンの欠落や、ホスト変数が宣言されていないことなどの構文エラーを検出します。こうしたエラーのそれぞれについて、適切なエラー・メッセージが生成されます。

組み込み SQL を含むソース・ファイルのコンパイルとリンク

組み込み SQL ソース・ファイルのプリコンパイルの際には、PRECOMPILE コマンドで、プログラミング言語に適合するファイル拡張子の付いた修正済みソース・ファイルを生成します。

適切なホスト言語コンパイラーを使用して、修正済みソース・ファイル (および SQL ステートメントを含まないあらゆる追加ソース・ファイル) をコンパイルしてください。言語コンパイラーは、それぞれの修正済みソース・ファイルをオブジェクト・モジュール に変換します。

デフォルトのコンパイラー・オプションに対する例外については、ご使用のオペレーティング・プラットフォーム用のプログラミング資料を参照してください。使用可能なコンパイラー・オプションの完全な説明については、コンパイラーの資料を参照してください。

ホスト言語リンカーは実行可能アプリケーションを作成します。以下に例を示します。

- Windows オペレーティング・システム上では、アプリケーションは実行可能ファイルまたはダイナミック・リンク・ライブラリー (DLL) にすることができます。
- UNIX および Linux ベースのオペレーティング・システム上では、アプリケーションは実行可能ロード・モジュールまたは共用ライブラリーにすることができます。

注: アプリケーションは、Windows オペレーティング・システム上では DLL にすることができますが、DLL は、DB2 データベース・マネージャーによってではなく、アプリケーションによって直接ロードされます。 Windows オペレーティング・システムでは、データベース・マネージャーは組み込み SQL ストアード・プロシージャーとユーザー定義関数を DLL としてロードします。

実行可能ファイルを作成するには、以下のものにリンクします。

- ユーザー・オブジェクト・モジュール。修正済みソース・ファイル、および SQL ステートメントが入っていないその他のファイルから、言語コンパイラーによって生成されます。
- ホスト言語ライブラリー API。言語コンパイラーによって提供されます。
- データベース・マネージャー・ライブラリー。ご使用のオペレーティング環境用のデータベース・マネージャー API が入っています。使用するデータベース・マネージャー API に必要なデータベース・マネージャー・ライブラリーの特定の名前については、ご使用のオペレーティング・プラットフォーム用のプログラミング資料を参照してください。

組み込み SQL パッケージのデータベースへのバインド

バインドとは、バインド・ファイルからパッケージを作成し、それをデータベースに保管する処理です。

アプリケーション、バインド・ファイル、およびパッケージの関係

データベース・アプリケーションはパフォーマンスの向上とサイズを小さくするためにパッケージを使用しますが、これはアプリケーションをコンパイルする理由と同じものです。 SQL ステートメントをプリコンパイルすることによって、このステートメントはアプリケーションの実行時にではなく作成時にコンパイルされてパッケージとなります。それぞれのステートメントが構文解析され、さらに効率的に解釈されたオペランド・ストリングがパッケージに保管されます。実行時に、プリコンパイラーにより生成されるコードにより、入出力データに必要な変数情報を指定したランタイム・サービス・データベース・マネージャー API が呼び出され、パッケージに保管されている情報が実行されます。

プリコンパイルの利点は静的 SQL ステートメントにだけ当てはまります。動的に実行される SQL ステートメント (PREPARE と、EXECUTE か EXECUTE IMMEDIATE を用いる) はプリコンパイルされません。したがって、一連のステップの処理全体を実行時に処理する必要があります。

DB2 バインド・ファイル記述 (db2bfd) ユーティリティーを使用することにより、バインド・ファイルを作成するのに使用されたプリコンパイル・オプションを表示するだけでなく、バインド・ファイルの内容を簡単に表示して、ファイル内部の SQL ステートメントを検査および検証することができます。これは、アプリケーションのバインド・ファイルに関連する問題の判別に役立ちます。

動的 SQL における DYNAMICRULES BIND オプションの影響

PRECOMPILE コマンドおよび BIND コマンドの DYNAMICRULES オプションにより、次の動的 SQL 属性に実行時に適用される値を決定できます。

- 許可検査中に使用される許可 ID。
- 非修飾オブジェクトの修飾に使用される修飾子。
- GRANT、REVOKE、ALTER、CREATE、DROP、COMMENT ON、RENAME、SET INTEGRITY および SET EVENT MONITOR STATE ステートメントを動的に準備するためにパッケージを使用できるかどうか。

DYNAMICRULES 値に加えてパッケージのランタイム環境が、動的 SQL ステートメントが実行時にどのように振る舞うかを制御します。次の 2 つのランタイム環境が考えられます。

- パッケージはスタンドアロン・プログラムの一部として実行される。
- パッケージはルーチン・コンテキスト内で実行される。

DYNAMICRULES 値とランタイム環境の組み合わせで動的 SQL 属性の値が決まります。その属性値の集合が、動的 SQL ステートメントの振る舞いと呼ばれます。次のような 4 つの振る舞いがあります。

実行動作

動的 SQL ステートメントの許可検査に使用する値および動的 SQL ステートメント内の非修飾オブジェクト参照の暗黙修飾に使用される初期値として、DB2 はパッケージを実行しているユーザーの許可 ID (最初に DB2 へ接続した ID) を使用します。

バインド動作

実行時に DB2 は、静的 SQL に適用されるすべての規則を許可と修飾に使用します。つまり、パッケージ所有者の許可 ID を動的 SQL ステートメントの許可検査で使用される値とし、動的 SQL ステートメント内の非修飾オブジェクト参照の暗黙修飾に使用されるパッケージ・デフォルト修飾子にするということです。

定義動作

定義振る舞いは、動的 SQL ステートメントがルーチン・コンテキスト内で実行されるパッケージに存在し、そのパッケージが DYNAMICRULES DEFINEBIND または DYNAMICRULES DEFINERUN でバインドされている場合にのみ適用されます。 DB2 は、(ルーチンのパッケージ・バインド・プログラムではなく) ルーチンの定義者の許可 ID を、動的 SQL ステートメントの許可検査で使用される値として使用し、そのルーチンの中で動的 SQL ステートメント内の非修飾オブジェクト参照の暗黙修飾の値として使用します。

起動動作

起動動作は、動的 SQL ステートメントがルーチン・コンテキスト内で実行されるパッケージに存在し、そのパッケージが DYNAMICRULES INVOKEBIND または DYNAMICRULES INVOKERUN でバインドされている場合にのみ適用されます。 DB2 は、ルーチンが呼び出されたときに有効であった現行ステートメント許可 ID を、動的 SQL の許可検査で使用される値として使用し、そのルーチン内の動的 SQL 内で非修飾オブジェクト参照の暗黙修飾として使用します。これらのことを次の表にまとめます。

起動環境	使用される ID
任意の静的 SQL	ルーチンを呼び出した SQL の入っているパッケージの OWNER の暗黙または明示的な値。
ビューまたはトリガー定義で使用	ビューまたはトリガーの定義者。
実行動作パッケージの動的 SQL	DB2 への初期接続を行うために使用された ID。
定義動作パッケージの動的 SQL	ルーチンを呼び出した SQL の入っているパッケージを使用しているルーチンの定義者。
起動動作パッケージの動的 SQL	ルーチンを呼び出すカレント許可 ID。

次の表は、それぞれの動的 SQL 振る舞いを決定する DYNAMICRULES 値とランタイム環境の組み合わせを示します。

表 18. 動的 SQL ステートメントの振る舞いを決定する DYNAMICRULES とランタイム環境

DYNAMICRULES 値	スタンドアロン・プログラム環境における動的 SQL ステートメントの振る舞い	ルーチン環境における動的 SQL ステートメントの振る舞い
BIND	バインド動作	バインド動作
RUN	実行動作	実行動作
DEFINEBIND	バインド動作	定義動作
DEFINERUN	実行動作	定義動作

表 18. 動的 SQL ステートメントの振る舞いを決定する DYNAMICRULES とランタイム環境 (続き)

DYNAMICRULES 値	スタンドアロン・プログラム環境における動的 SQL ステートメントの振る舞い	ルーチン環境における動的 SQL ステートメントの振る舞い
INVOKEBIND	バインド動作	起動動作
INVOKERUN	実行動作	起動動作

次の表に、動的 SQL 振る舞いの各タイプ用の動的 SQL 属性値を示します。

表 19. 動的 SQL ステートメント振る舞いの定義

動的 SQL 属性	バインド振る舞いの動的 SQL 属性の設定	実行振る舞いの動的 SQL 属性の設定	定義振る舞いの動的 SQL 属性の設定	起動動作の動的 SQL 属性の設定
許可 ID	OWNER BIND オプションの暗黙または明示的な値	パッケージを実行するユーザーの ID	ルーチン定義者 (ルーチンのパッケージ所有者ではない)	ルーチンが呼び出されたときの現行ステートメント許可 ID
非修飾オブジェクトのデフォルト修飾子	QUALIFIER BIND オプションの暗黙または明示的な値	CURRENT SCHEMA 特殊レジスター	ルーチン定義者 (ルーチンのパッケージ所有者ではない)	ルーチンが呼び出されたときの現行ステートメント許可 ID
GRANT、REVOKE、ALTER、CREATE、DROP、COMMENT ON、RENAME、 SET INTEGRITY および SET EVENT MONITOR STATE の実行	いいえ	はい	いいえ	いいえ

特殊レジスターを使用したステートメント・コンパイル環境の制御

動的に作成されたステートメントについては、いくつかの特殊レジスターの値によってステートメントのコンパイル環境が決定されます。

- CURRENT QUERY OPTIMIZATION 特殊レジスターは、使用される最適化クラスを決定します。
- CURRENT PATH 特殊レジスターは、 UDF および UDT の解決に使用される関数パスを決定します。
- CURRENT EXPLAIN SNAPSHOT レジスターは、 Explain スナップショット情報を収集するかどうかを決定します。
- CURRENT EXPLAIN MODE レジスターは、適格な動的 SQL ステートメントについての Explain 表情報を収集するかどうかを決定します。これらの特殊レジスターのデフォルト値は、関連する BIND オプションで使用されるデフォルトと同じです。

BIND コマンドと既存のバインド・ファイルを使用したパッケージの再作成

バインドとは、データベース・マネージャーがアプリケーションの実行時にデータベースをアクセスするために必要とするパッケージを作成する処理です。デフォルトでは、PRECOMPILE コマンドによってパッケージの作成が行われます。バインディングは、 BINDFILE オプションが指定されない場合、プリコンパイル時に暗黙的に行われます。PACKAGE オプションにより、プリコンパイル時に作成されるパッケージのパッケージ名の指定が可能です。

BIND コマンドの使用の一般的な例を以下に示します。 *filename.bnd* という名前のバインド・ファイルをデータベースにバインドするには、以下のコマンドを発行します。

```
BIND filename.bnd
```

個別にプリコンパイルされたソース・コード・モジュールごとに 1 つのパッケージが作成されます。アプリケーションに 5 つのソース・ファイルがあって、そのうちの 3 ファイルにプリコンパイルが必要な場合、 3 つのパッケージまたはバインド・ファイルが作成されます。デフォルトの解釈では、それぞれのパッケージには .bnd ファイルの作成元となったソース・モジュールの名前と同じ名前が付けられますが、8 文字で切り捨てられます。異なるパッケージ名を明示的に指定するには、PREP コマンドの PACKAGE USING オプションを使用しなければなりません。パッケージのバージョンは、VERSION プリコンパイル・オプションを指定すれば与えられますが、デフォルトでは空ストリングになっています。新しく作成されたパッケージの名前とスキーマが、ターゲット・データベースに現在存在しているパッケージの名前と同じであるもののバージョン ID が異なる場合、新規パッケージが作成されて既存パッケージに代わって新規のパッケージが使用されます。しかしながら、バインドされるパッケージの名前とスキーマとバージョンが同じパッケージが存在する場合、そのパッケージはドロップされバインドされる新規パッケージと置き換えられます (バインドに ACTION ADD が指定されていれば、こうしたことは起こらず、代わりにエラー (SQL0719) が戻されます)。

REBIND コマンドによる既存パッケージの再バインド

再バインド は、以前にバインドされたアプリケーション・プログラムのパッケージを再作成する処理です。パッケージが無効、または作動不能と示されている場合、あるいは最終バインディング時以降にデータベースの統計に変更が加えられている場合は、再バインドしなければなりません。しかし、パッケージが有効であっても再バインドが必要な場合もあります。たとえば、新規に作成された索引を利用する場合、または RUNSTATS コマンドの実行後の更新統計を利用する場合です。

パッケージは、表、ビュー、別名、索引、トリガー、参照制約、および表チェック制約など、データベース・オブジェクトの一定のタイプに従属させることができます。パッケージがデータベース・オブジェクト (表、ビュー、トリガーなど) に従属している場合にそのオブジェクトがドロップされると、パッケージは*無効* な状態になります。ドロップされたオブジェクトが UDF である場合、パッケージは*作動不能* 状態になります。

無効なパッケージは、実行される際にデータベース・マネージャーによって暗黙に (つまり自動的に) 再バインドされます。作動不能パッケージは、 BIND コマンドま

たは REBIND コマンドのいずれかを実行して、明示的に再バインドしなければなりません。暗黙の再バインドに失敗すると、予期しないエラーが生じる場合があることに気を付けてください。つまり、暗黙の再バインドのエラーは、実際にエラーのあるステートメントではなく実行中のステートメントに戻される場合もあります。作動不能パッケージを実行しようとすると、エラーが発生します。無効なパッケージをシステムで自動的に再バインドするのではなく、これらを明示的に再バインドすることができます。これにより、いつ再バインドを行うかを制御できるようになります。

パッケージを明示的に再バインドするために使用するコマンドは、状況により異なります。 SQL ステートメントの数を変更するか、または変更された SQL ステートメントを含めるために修正されたプログラムのパッケージを再バインドするには、BIND コマンドを使用しなければなりません。 BIND オプションを、パッケージが最初にバインドされた時点での値から別の値に変更する必要がある場合にも、BIND コマンドを使用します。その他の場合には、 BIND コマンドと REBIND コマンドのいずれかを使用してください。パフォーマンスの面では BIND よりも REBIND の方が優れているので、特に BIND を使用する必要がないときは REBIND を使用してください。

同じパッケージ名の複数のバージョンが同時にカタログに存在する場合、1 回に 1 つのバージョンしか再バインドできません。

バインドに関する考慮事項

アプリケーションのコード・ページがデータベースのコード・ページと異なる場合、バインド時にどちらのコード・ページを使用するかを考慮する必要があります。

アプリケーションが IMPORT または EXPORT のようなデータベース・マネージャー・ユーティリティー API に呼び出しを発行する場合、提供されるユーティリティー・バインド・ファイルをデータベースにバインドしておくことが必要です。

BIND オプションを使用して、バインド中に発生する一定の操作を制御することができます。

- QUERYOPT BIND オプションは、バインド時に特定の最適化クラスを利用します。
- EXPLSNAP BIND オプションは、 Explain 表内の適格な SQL ステートメント用の Explain スナップショット情報を保管します。
- FUNCPATH BIND オプションは、静的 SQL のユーザー定義特殊タイプおよびユーザー定義関数を正しく解決します。

バインド処理を開始しても応答がない場合、データベースに接続している他のアプリケーションが、必要なロックを保持している可能性があります。この場合には、どのアプリケーションもデータベースに接続されていないことを確認してください。接続されていることがわかったなら、サーバー上のすべてのアプリケーションを切り離して、バインド処理を継続します。

アプリケーションが DB2 Connect を使用してサーバーにアクセスする場合、そのサーバーで使用可能な BIND オプションを使用できます。

バインド・ファイルには、以前のバージョンの DB2 Database for Linux, UNIX, and Windows との後方互換性はありません。レベルが混合した環境では、 DB2 は最下位レベルのデータベース環境で使用できる機能しか使用できません。たとえば、バージョン 8 クライアントがバージョン 7.2 サーバーと接続している場合、そのクライアントは バージョン 7.2 の機能しか使用できません。バインド・ファイルはデータベースの機能を伝えるので、それらは混合レベルの制限に従います。

下位レベルのシステムで、上位レベルのバインド・ファイルを再バインドする必要がある場合は、以下のようにすることができます。

- 下位レベルの DB2 Client を使用して上位レベルのサーバーに接続し、下位レベルの DB2 Database for Linux, UNIX, and Windows 環境に搬出してバインドできるバインド・ファイルを作成します。
- 上位レベルの DB2 クライアントを下位レベルの実稼働環境で使用して、テスト環境で作成された上位レベルのバインド・ファイルをバインドします。上位レベルのクライアントは、下位レベルのサーバーに適用されるオプションだけを渡します。

ブロッキングに関する考慮事項

組み込み SQL アプリケーションにおいてブロッキングを不使用にする際、ソース・コードが入手できない場合には、BIND コマンドを使用し、BLOCKING NO 節を設定して、アプリケーションを再バインドする必要があります。

ブロッキングを要求するには、BIND コマンドを使用し、BLOCKING ALL または BLOCKING UNAMBIGUOUS 節を設定して、既存の組み込み SQL アプリケーションを再バインドする必要があります (まだこのような方法でバインドしていない場合)。サーバーから複数行のブロックが取り出されると、組み込みアプリケーションはサーバーから LOB 値を 1 度に 1 行ずつ取り出します。

据え置きバインドの利点

バインドを実行可能にしてプリコンパイルを行うと、アプリケーションはプリコンパイル処理中に使用されたデータベースだけにアクセスできます。バインドを実行据え置きにしてプリコンパイルすると、BIND ファイルを各データベースにバインドできるので、アプリケーションから多数のデータベースにアクセスできます。このアプリケーション開発方法は、アプリケーションを 1 回だけプリコンパイルするという点で本来柔軟性のあるものですが、アプリケーションはデータベースにいつでもバインドすることができます。

実行中に BIND API を使用すると、アプリケーションはそれ自体をバインドすることができます。これはおそらくインストール手順の一部として、または関連モジュールが実行される前に行われます。たとえば、アプリケーションは複数のタスクを実行することができますが、そのうちで SQL ステートメントを使用する必要があるのは 1 つだけです。アプリケーションは、SQL ステートメントを必要とするタスクが呼び出されるとき、および関連パッケージが存在しない場合にだけ、アプリケーション自体をデータベースにバインドできるように設計することができます。

実行据え置きバインドのもう 1 つの利点は、エンド・ユーザーにソース・コードを提供しなくてもパッケージを作成できるという点です。関連したバインド・ファイルをアプリケーションと共に出荷することができます。

BIND コマンドの REOPT オプションを使用した場合のパフォーマンスの改善

BIND オプション REOPT は、組み込み SQL アプリケーションのパフォーマンスを大幅に改善できます。以下は、静的 SQL および動的 SQL の両方のための説明です。

静的 SQL における REOPT の影響

BIND オプション REOPT は、ホスト変数、グローバル変数、または特殊レジスターを含む静的 SQL ステートメントを、増分バインド・ステートメントのように動作させることができます。これは、これらのステートメントが、バインド時ではなく、 EXECUTE または OPEN の時にコンパイルされることを意味します。このコンパイル時に、これらの変数の実際の値に基づいて、アクセス・プランが選択されます。

REOPT ONCE の場合は、最初の OPEN または EXECUTE 要求の後にアクセス・プランがキャッシュされ、このステートメントの後の実行で使用されます。 REOPT ALWAYS の場合は、OPEN および EXECUTE 要求のたびにアクセス・プランが再生成され、現行のホスト変数、パラメーター・マーカー、グローバル変数、および特殊レジスターの値のセットでこのプランが作成されます。

動的 SQL における REOPT の影響

REOPT ALWAYS オプションを指定している場合、 DB2 は、ホスト変数、パラメーター・マーカー、グローバル変数、または特殊レジスターを含むステートメントについて、OPEN または EXECUTE ステートメントを検出するまで (つまり、これらの変数の値が分かるときまで)、準備を延期します。その時点で、これらの値を使用してアクセス・プランが生成されます。同じステートメントに対するその後の OPEN または EXECUTE 要求では、ステートメントが再コンパイルされ、変数の現行の値のセットを使用して照会プランが再最適化され、新しく生成された照会プランが実行されます。

REOPT ONCE オプションにも同様の効果がありますが、異なる点として、プランがホスト変数、パラメーター・マーカー、グローバル変数、および特殊レジスターの値で 1 回だけ最適化されます。このプランはキャッシュされ、その後の要求で使用されます。

パッケージの保管および保守

パッケージはアプリケーション・プログラムのプリコンパイル/バインディングによって作成されます。パッケージには、アプリケーション内のすべての SQL ステートメントの実行を監督する最適化されたアクセス・プランが含まれます。パッケージに関わる特権には、CONTROL、EXECUTE、および BIND 特権という 3 つのタイプがあり、これらは、許可されるアクセスのレベルをフィルタリングするのに使用されます。コンパイル時に VERSION オプションを指定すると、同じパッケージの複数バージョンを作成できます。このオプションは、タイム・スタンプの不一致エラーを防止するのに役立ち、これによってアプリケーションの複数バージョンを同時に実行することが可能になります。

パッケージにバージョンを付ける

アプリケーションの複数のバージョンを作成する必要がある場合は、PRECOMPILE コマンドの VERSION オプションを使用できます。このオプションにより、同じパッケージ名 (つまり、パッケージ名と作成者名) の複数のバージョンを同時に存在させることができます。たとえば、foo と呼ばれるアプリケーションがあり、foo.sqc からコンパイルされたとしましょう。パッケージ foo をプリコンパイルしてデータベースにバインドし、アプリケーションをユーザーに配布したとします。ユーザーはアプリケーションを実行させることができます。その後アプリケーションに変更を加え、foo.sqc を更新し、再コンパイル、バインディング、およびユーザーへのアプリケーションの配布を繰り返したとします。 foo.sqc の最初のプリコンパイルまたは 2 番目のプリコンパイルのどちらかで VERSION オプションを指定していなかったとすると、最初のパッケージは 2 番目のパッケージで置き換えられてしまいます。古いバージョンのアプリケーションを実行しようとすると、タイム・スタンプがミスマッチしているというエラーを示す SQLCODE -818 を受け取ります。

タイム・スタンプのミスマッチのエラーを避け、同時に両方のバージョンのアプリケーションを実行できるようにするためには、パッケージにバージョンを付けるようにしてください。たとえば、foo の最初のバージョンの作成時に、次のように VERSION オプションを使用してプリコンパイルします。

```
DB2 PREP FOO.SQC VERSION V1.1
```

プログラムの最初のバージョンが実行されているとします。 foo の新しいバージョンの作成時に、次のコマンドを使用してプリコンパイルします。

```
DB2 PREP FOO.SQC VERSION V1.2
```

この時点で、アプリケーションの新規バージョンも実行できますが、最初のアプリケーションのインスタンスも実行されたままです。最初のパッケージのパッケージ・バージョンは V1.1 で 2 番目のパッケージ・バージョンは V1.2 なので、名前の衝突は発生しません。両方のパッケージがデータベースに存在し両方のバージョンのアプリケーションが使用できます。

PRECOMPILE または BIND コマンドで、PRECOMPILE コマンドの VERSION オプションといっしょに ACTION オプションを使用できます。ACTION オプションを使用して、異なるバージョンのパッケージを追加したり置き換える方法を制御できます。

パッケージ特権は、バージョン・レベルではきめ細かに制御できません。つまり、パッケージ特権の GRANT または REVOKE は、名前と作成者を共用するすべてのパッケージのバージョンに適用されるということです。それで、パッケージ foo に対するパッケージ特権がバージョン V1.1 が作成された後ユーザーまたはグループに付与された場合、バージョン V1.2 が配布されるときにはユーザーまたはグループは同じ特権をバージョン V1.2 に対しても持つことになります。一般的に、同じユーザーおよびグループはパッケージのすべてのバージョンに対して同じ特権を持つので、この振る舞いは普通必要とされます。アプリケーションのすべてのバージョンが同じパッケージ特権になることを望まない場合は、パッケージにバージョンを付ける際に PRECOMPILE VERSION オプションを使用しないようにしてください。その代わり、(更新済みソース・ファイルをリネームするか、または PACKAGE USING オプションを使用して明示的にパッケージをリネームして) 異なるパッケージ名を使用するようにすべきです。

非修飾表名の解決

以下の方法の一つを使用すると、アプリケーション内で非修飾表名を処理することができます。

- 各ユーザーは、以下のコマンドを使用することにより、異なる許可 ID を使用して、さまざまな COLLECTION パラメーターでパッケージをバインドできます。

  ```
  CONNECT TO db_name USER user_name
  BIND file_name COLLECTION schema_name
  ```

  上記の例では、*db_name* はデータベース名、*user_name* はユーザー名、そして *file_name* はバインドされるアプリケーション名を表しています。 *user_name* と *schema_name* は普通は同じ値です。その後、SET CURRENT PACKAGESET ステートメントを使用して、使用するパッケージ、すなわち使用する修飾子を指定します。 COLLECTION が指定されていない場合、デフォルト修飾子が、パッケージをバインドするときに使用される許可 ID になります。 COLLECTION が指定された場合、指定されている *schema_name* が非修飾オブジェクトに使用される修飾子になります。
- 各ユーザーにその表と同じ名前のビューを作成して、非修飾表名が正しく解決されるようにします。
- ユーザーごとに 1 つの別名を作成し、希望する表を指すようにします。

サンプル・ビルド・スクリプトを使用した組み込み SQL アプリケーションの構築

サンプル・プログラムの作成の例を示すのに使用するファイルは、UNIX および Linuxではスクリプト・ファイル、Windows ではバッチ・ファイルと呼ばれます。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。このファイルには、サポートされているプラットフォーム・コンパイラー用に推奨されるコンパイルとリンクのコマンドが入っています。

ビルド・ファイルは、サポートされているプラットフォームに対応するホスト言語用に、DB2 により提供されています。ビルド・ファイルは、その言語用のサンプルが含まれるのと同一のディレクトリーで入手できます。以下の表に、さまざまなタイプのプログラムを構築するためのさまざまなタイプのビルド・ファイルを一覧で示してあります。他に明記されていない限り、これらのビルド・ファイルは、サポートされているすべてのプラットフォーム上のサポートされている言語用のファイルです。 Windows ではこのビルド・ファイルには .bat (バッチ) の拡張子が付いていますが、この表では付けられていません。 UNIX プラットフォームの場合には、拡張子はありません。

表 20. 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
bldapp	アプリケーション・プログラム
bldrtn	ルーチン (ストアード・プロシージャーと UDF)
bldmc	C/C++ 複数接続アプリケーション
bldmt	C/C++ マルチスレッド・アプリケーション
bldcli	sqlproc サンプル・サブディレクトリー内の SQL プロシージャー用の CLI クライアント・アプリケーション

注: デフォルトでは、ソース・コードから実行可能ファイルを構築する bldapp サンプル・スクリプトは、64 ビットの実行可能ファイルを構築します。

以下の表は、プラットフォーム別およびプログラム言語別のビルド・ファイルと、それが置かれているディレクトリーを一覧で示しています。オンライン文書では、ビルド・ファイルの名前が HTML のソース・ファイルにホット・リンクされています。該当するサンプル・ディレクトリー内のテキスト・ファイルにアクセスすることもできます。

表 21. 言語別およびプラットフォーム別のビルド・ファイル

プラットフォーム -> 言語	**AIX**	**HP-UX**	**Linux**	**Solaris**	**Windows**
C samples/c	bldapp bldrtn bldmt bldmc	bldapp bldrtn bldmt bldmc	bldapp bldrtn bldmt bldmc	bldapp bldrtn bldmt bldmc	bldapp.bat bldrtn.bat bldmt.bat bldmc.bat
C++ samples/cpp	bldapp bldrtn bldmt bldmc	bldapp bldrtn bldmt bldmc	bldapp bldrtn bldmt bldmc	bldapp bldrtn bldmt bldmc	bldapp.bat bldrtn.bat bldmt.bat bldmc.bat
IBM COBOL samples/cobol	bldapp bldrtn	n/a	n/a	n/a	bldapp.bat bldrtn.bat
Micro Focus COBOL samples/cobol_mf	bldapp bldrtn	bldapp bldrtn	bldapp bldrtn	bldapp bldrtn	bldapp.bat bldrtn.bat

本書でアプリケーションとルーチンの構築にビルド・ファイルが使われているのは、 DB2 がサポートされているコンパイラーに対しお勧めする、コンパイルおよびリンクのオプションが、それによって明らかに実体として示されるからです。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・ファイルを使用してサンプル・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、アプリケーション開発に役立てることができます。

都合のよいことに、コンパイラーで許容される任意のファイル名でソース・ファイルを構築できるようにビルド・ファイルは設計されています。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。 makefile は、作成したプログラムのコンパイルとリンクのためにビルド・ファイルにアクセスします。ビルド・ファイルは、UNIX および Linux の場合には $1 変数、Windows オペレーティング・システムの場合には %1 変数を使用して、プログラム名を内部的に置き換えます。このような変数名中の数字は、順に大きくなって、他の引数が必要となるごとに入れ替わります。

たとえばスタンドアロン・アプリケーション、ルーチン (ストアード・プロシージャーおよび UDF)、またはマルチ接続あるいはマルチスレッド・プログラムなどの

もっと特殊なプログラム・タイプといった、個々の種類のプログラム・ビルドに対して個々のビルド・ファイルがそれぞれ適応しているため、迅速かつ簡単にビルド・ファイルを試してみることができます。ビルド・ファイルの設計目的に沿った種類のプログラムがコンパイラーでサポートされてさえいれば、どこででもすべてのタイプのビルド・ファイルを利用することができます。

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、プログラムがビルドされるたびに自動的に上書きされます。これは、makefile を使用する場合には当てはまりません。 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オーバーライド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 プログラムでは、Windows で IBM COBOL プリコンパイラーを使用する場合を除き、ビルド・ファイルは、組み込み SQLプログラムのためのプリコンパイルとバインドのステップが入っている、別のファイル `embprep` を呼び出します。これらのステップでは、組み込み SQL プログラムをどこに構築するかによって、ユーザー ID とパスワード用のオプション・パラメーターが必要になる場合があります。

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。

エラー・チェック・ユーティリティー

DB2 Client には、ユーティリティー・ファイルがいくつかあります。これらのファイルには、エラー・チェックとエラー情報の印刷出力を行う関数があります。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。このエラー・チェック・ユーティリティー・ファイル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・チェック・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` (sqlogstt) および `GETERROR MESSAGE` (sqlaintp) を使います。 DB2 CLI ユーティリティー・ファイルである `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・チェック・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。ルーチンなどの一部

の DB2 プログラム (ストアード・プロシージャーやユーザー定義関数など) では、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。

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・チェック・ユーティリティー・ファイルです。

表 22. 言語別のエラー・チェック・ユーティリティー・ファイル

言語	非組み込み SQL ソース・ファイル	非組み込み SQL ヘッダー・ファイル	組み込み SQL ソース・ファイル	組み込み SQL ヘッダー・ファイル
C samples/c	utilapi.c	utilapi.h	utilemb.sqc	utilemb.h
C++ samples/cpp	utilapi.C	utilapi.h	utilemb.sqC	utilemb.h
IBM COBOL samples/cobol	checkerr.cbl	n/a	n/a	n/a
Micro Focus COBOL samples/cobol_mf	checkerr.cbl	n/a	n/a	n/a

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・チェック・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・チェック・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() 用の EMB_SQL_CHECK マクロを定義します。

```
/* macro for embedded SQL checking */
#define EMB_SQL_CHECK(MSG_STR)                          ¥
SqlInfoPrint(MSG_STR, &sqlca, __LINE__, __FILE__);      ¥
if (sqlca.sqlcode < 0)                                  ¥
{                                                       ¥
  TransRollback();                                      ¥
  return 1;                                             ¥
}
```

SqlInfoPrint() は SQLCODE を検査し、発生した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。これには、組み込み SQL ステートメント EXEC SQL ROLLBACK が使用されます。以下に、 C プログラム dbuse がマクロを使用して、 SqlInfoPrint() 関数の MSG_STR パラメーターに値 "Delete with host variables -- Execute" 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。

```
EXEC SQL DELETE FROM org
  WHERE deptnumb = :hostVar1 AND
        division = :hostVar2;
EMB_SQL_CHECK("Delete with host variables -- Execute");
```

EMB_SQL_CHECK マクロは、 DELETE 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・チェック・ユーティリティーを使用および拡張することをお勧めします。

C および C++ で作成されたアプリケーションおよびルーチンのビルド

C および C++ の組み込み SQL アプリケーションのビルドを可能にする、さまざまなオペレーティング・システム・プラットフォーム向けのビルド・スクリプトが製品とともに提供されています。アプリケーションのビルドに使用されるビルド・スクリプトに加え、ルーチン (ストアード・プロシージャーおよびユーザー定義関数) をビルドするために特に使用される bldrtn スクリプトも提供されています。VisualAge® で作成されたアプリケーションおよびルーチンの場合、アプリケーションのビルドには構成ファイルが使用されます。提供されている C アプリケーションのサンプルは、チュートリアルからクライアント・レベルあるいはインスタンス・レベルの例まで多岐にわたっています。それらは、UNIX 用には sqllib/samples/c ディレクトリーに、Windows 用には sqllib¥samples¥c ディレクトリーにあります。

C および C++ のコンパイルおよびリンクのオプション

AIX C 組み込み SQL および DB2 API アプリケーションのコンパイルおよびリンク・オプション:

以下は、bldapp ビルド・スクリプトに示されているように、AIX IBM C コンパイラーを使用して、C 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

コンパイル・オプション:

xlc
IBM XL C/C++ コンパイラー。

$EXTRA_CFLAG
64 ビット・サポートが使用可能なインスタンスの場合は「-q64」が入り、それ以外の場合は値は入りません。

-I$DB2PATH/include
DB2 組み込みファイルのロケーションを指定します。たとえば、$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:

xlc　コンパイラーをリンカーのフロントエンドとして使用します。

$EXTRA_CFLAG
64 ビット・サポートが使用可能なインスタンスの場合は「-q64」が入り、それ以外の場合は値は入りません。

-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　DB2 ライブラリーとリンクします。

-L$DB2PATH/$LIB
DB2 ランタイム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。

AIX C++ 組み込み SQL および DB2 管理 API アプリケーションのコンパイルおよびリンク・オプション:

以下は、bldapp ビルド・スクリプトに示されているように、AIX IBM XL C++ コンパイラーを使用して、C++ 組み込み SQL および DB2 管理 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

コンパイル・オプション:

xlC　IBM XL C/C++ コンパイラー。

EXTRA_CFLAG
64 ビット・サポートが使用可能なインスタンスの場合は「-q64」が入り、それ以外の場合は値は入りません。

-I$DB2PATH/include
DB2 組み込みファイルのロケーションを指定します。たとえば、$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:

xlC　コンパイラーをリンカーのフロントエンドとして使用します。

EXTRA_CFLAG
64 ビット・サポートが使用可能なインスタンスの場合は「-q64」が入り、それ以外の場合は値は入りません。

-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　DB2 ライブラリーとリンクします。

-L$DB2PATH/$LIB
DB2 ランタイム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。

HP-UX C アプリケーションのコンパイルとリンクのオプション:

以下は、`bldapp` ビルド・スクリプトに示されているように、HP-UX C コンパイラーを使用して、 C 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

コンパイル・オプション:

cc　C コンパイラー。

$EXTRA_CFLAG

HP-UX プラットフォームが IA64 で、64 ビット・サポートが使用可能な場合は、このフラグには値 **+DD64** が入り、 32 ビット・サポートが使用可能な場合は、値 **+DD32** が入ります。 HP-UX プラットフォームが PA-RISC で、64 ビット・サポートが使用可能な場合は、これには値 **+DA2.0W** が入ります。 PA-RISC プラットフォームでの 32 ビット・サポートの場合は、このフラグには値 **+DA2.0N** が入ります。

+DD64　IA64 版の HP-UX 用の 64 ビット・コードを生成する場合に使用する必要があります。

+DD32　IA64 版の HP-UX 用の 32 ビット・コードを生成する場合に使用する必要があります。

+DA2.0W

PA-RISC 版の HP-UX 用の 64 ビット・コードを生成する場合に使用する必要があります。

+DA2.0N

PA-RISC 版の HP-UX 用の 32 ビット・コードを生成する場合に使用する必要があります。

-Ae　HP ANSI 拡張モードを使用可能にします。

-I$DB2PATH/include

DB2 組み込みファイルのロケーションを指定します。

-c　コンパイルのみを実行し、リンクは実行しません。コンパイルとリンクは別個のステップです。

リンク・オプション:

cc コンパイラーをリンカーのフロントエンドとして使用します。

$EXTRA_CFLAG
HP-UX プラットフォームが IA64 で、64 ビット・サポートが使用可能な場合は、このフラグには値 **+DD64** が入り、 32 ビット・サポートが使用可能な場合は、値 **+DD32** が入ります。 HP-UX プラットフォームが PA-RISC で、64 ビット・サポートが使用可能な場合は、これには値 **+DA2.0W** が入ります。 PA-RISC プラットフォームでの 32 ビット・サポートの場合は、このフラグには値 **+DA2.0N** が入ります。

+DD64 IA64 版の HP-UX 用の 64 ビット・コードを生成する場合に使用する必要があります。

+DD32 IA64 版の HP-UX 用の 32 ビット・コードを生成する場合に使用する必要があります。

+DA2.0W
PA-RISC 版の HP-UX 用の 64 ビット・コードを生成する場合に使用する必要があります。

+DA2.0N
PA-RISC 版の HP-UX 用の 32 ビット・コードを生成する場合に使用する必要があります。

-o $1 実行可能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・ファイルを組み込みます。

$EXTRA_LFLAG
ランタイム・パスを指定します。設定する場合、32 ビットの場合は値 `-Wl,+b$HOME/sqllib/lib32`、 64 ビットの場合は `-Wl,+b$HOME/sqllib/lib64` が入ります。設定しない場合は、これには値が入りません。

-L$DB2PATH/$LIB
DB2 ランタイム共用ライブラリーのロケーションを指定します。 32 ビットの場合は `$HOME/sqllib/lib32`、 64 ビットの場合は `$HOME/sqllib/lib64` です。

-ldb2 DB2 ライブラリーとリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

HP-UX C++ アプリケーションのコンパイルとリンクのオプション:

以下は、`bldapp` ビルド・スクリプトに示されているように、HP-UX C++ コンパイラーを使用して、C++ 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

コンパイル・オプション:

aCC
HP aC++ コンパイラー。

$EXTRA_CFLAG
HP-UX プラットフォームが IA64 で、64 ビット・サポートが使用可能な場合は、このフラグには値 **+DD64** が入り、 32 ビット・サポートが使用可能な場合は、値 **+DD32** が入ります。 HP-UX プラットフォームが PA-RISC で、64 ビット・サポートが使用可能な場合は、これには値 **+DA2.0W** が入ります。 PA-RISC プラットフォームでの 32 ビット・サポートの場合は、このフラグには値 **+DA2.0N** が入ります。

+DD64
IA64 版の HP-UX 用の 64 ビット・コードを生成する場合に使用する必要があります。

+DD32
IA64 版の HP-UX 用の 32 ビット・コードを生成する場合に使用する必要があります。

+DA2.0W
PA-RISC 版の HP-UX 用の 64 ビット・コードを生成する場合に使用する必要があります。

+DA2.0N
PA-RISC 版の HP-UX 用の 32 ビット・コードを生成する場合に使用する必要があります。

-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$DB2PATH/include
DB2 組み込みファイルのロケーションを指定します。たとえば、`$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:

aCC　HP aC++ コンパイラーをリンカーのフロントエンドとして使用します。

$EXTRA_CFLAG
HP-UX プラットフォームが IA64 で、64 ビット・サポートが使用可能な場合は、このフラグには値 **+DD64** が入り、 32 ビット・サポートが使用可能な場合は、値 **+DD32** が入ります。 HP-UX プラットフォームが PA-RISC で、64 ビット・サポートが使用可能な場合は、これには値 **+DA2.0W** が入ります。 PA-RISC プラットフォームでの 32 ビット・サポートの場合は、このフラグには値 **+DA2.0N** が入ります。

+DD64　IA64 版の HP-UX 用の 64 ビット・コードを生成する場合に使用する必要があります。

+DD32　IA64 版の HP-UX 用の 32 ビット・コードを生成する場合に使用する必要があります。

+DA2.0W
PA-RISC 版の HP-UX 用の 64 ビット・コードを生成する場合に使用する必要があります。

+DA2.0N
PA-RISC 版の HP-UX 用の 64 ビット・コードを生成する場合に使用する必要があります。

-o $1　実行可能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・ファイルを組み込みます。

$EXTRA_LFLAG
ランタイム・パスを指定します。設定する場合、32 ビットの場合は値 ″-Wl,+b$HOME/sqllib/lib32″、 64 ビットの場合は ″-Wl,+b$HOME/sqllib/lib64″ が入ります。設定しない場合は、これには値が入りません。

-L$DB2PATH/$LIB
DB2 ランタイム共用ライブラリーのロケーションを指定します。 32 ビットの場合は $HOME/sqllib/lib32、 64 ビットの場合は $HOME/sqllib/lib64 です。

-ldb2　DB2 ライブラリーとリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

Linux C アプリケーションのコンパイルとリンクのオプション:

以下は、bldapp ビルド・スクリプトに示されているように、Linux C コンパイラーを使用して、C 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

コンパイル・オプション:

$CC
gcc または xlc_r コンパイラー。

$EXTRA_C_FLAGS
次のうちの 1 つが含まれています。

- Linux for zSeries の場合のみ **-m31** を指定して、32 ビット・ライブラリーを作成します。
- Linux for x86, x86_64 and POWER の場合には **-m32** を指定して、32 ビット・ライブラリーを作成します。
- Linux for zSeries, POWER, x86_64 の場合には **-m64** を指定して、64 ビット・ライブラリーを作成します。
- Linux for IA64 の場合には値を指定しないで、64 ビット・ライブラリーを作成します。

-I$DB2PATH/include
DB2 組み込みファイルのロケーションを指定します。

-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
gcc または xlc_r コンパイラー。コンパイラーをリンカーのフロントエンドとして使用します。

$EXTRA_C_FLAGS
次のうちの 1 つが含まれています。

- Linux for zSeries の場合のみ -m31 を指定して、32 ビット・ライブラリーを作成します。
- Linux for x86, x86_64 and POWER の場合には -m32 を指定して、32 ビット・ライブラリーを作成します。
- Linux for zSeries, POWER, x86_64 の場合には -m64 を指定して、64 ビット・ライブラリーを作成します。
- Linux for IA64 の場合には値を指定しないで、64 ビット・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。

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・ファイルを組み込みます。

$EXTRA_LFLAG
32 ビットの場合は値 ″-Wl,-rpath,$DB2PATH/lib32″ が入り、 64 ビットの場合は値 ″-Wl,-rpath,$DB2PATH/lib64″ が入ります。

-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示します。例えば、32 ビットの場合は $HOME/sqllib/lib32、64 ビットの場合は $HOME/sqllib/lib64 です。

-ldb2
DB2 ライブラリーとリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

Linux C++ アプリケーションのコンパイルとリンクのオプション:

以下は、bldapp ビルド・スクリプトに示されているように、Linux C++ コンパイラーを使用して、C++ 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

コンパイル・オプション:

g++ GNU/Linux C++ コンパイラー。

$EXTRA_C_FLAGS
次のうちの 1 つが含まれています。

- Linux for zSeries の場合のみ **-m31** を指定して、32 ビット・ライブラリーを作成します。
- Linux for x86, x86_64 and POWER の場合には **-m32** を指定して、32 ビット・ライブラリーを作成します。
- Linux for zSeries, POWER, x86_64 の場合には **-m64** を指定して、64 ビット・ライブラリーを作成します。
- Linux for IA64 の場合には値を指定しないで、64 ビット・ライブラリーを作成します。

-I$DB2PATH/include
DB2 組み込みファイルのロケーションを指定します。

-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:

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。

$EXTRA_C_FLAGS
次のうちの 1 つが含まれています。

- Linux for zSeries の場合のみ **-m31** を指定して、32 ビット・ライブラリーを作成します。
- Linux for x86, x86_64 and POWER の場合には **-m32** を指定して、32 ビット・ライブラリーを作成します。
- Linux for zSeries, POWER, x86_64 の場合には **-m64** を指定して、64 ビット・ライブラリーを作成します。
- Linux for IA64 の場合には値を指定しないで、64 ビット・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・ファイルを組み込みます。

$EXTRA_LFLAG
32 ビットの場合は値 ″-Wl,-rpath,$DB2PATH/lib32″ が入り、 64 ビットの場合は値 ″-Wl,-rpath,$DB2PATH/lib64″ が入ります。

-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示します。例えば、32 ビットの場合は **$HOME/sqllib/lib32**、64 ビットの場合は **$HOME/sqllib/lib64** です。

-ldb2 DB2 ライブラリーとリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

Solaris C アプリケーションのコンパイルとリンクのオプション:

以下は、**bldapp** ビルド・スクリプトに示されているように、Forte C コンパイラーを使用して、 C 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

コンパイル・オプション:

cc C コンパイラー。

-xarch=$CFLAG_ARCH
このオプションを使用すると、 libdb2.so へのリンク時に必ず正しい実行可能ファイルがコンパイラーで生成されるようにすることができます。 $CFLAG_ARCH の値は、32 ビットの場合は v8plusa に、64 ビットの場合は v9 に設定されます。

-I$DB2PATH/include
DB2 組み込みファイルのロケーションを指定します。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。このスクリプトでは、コンパイルとリンクは別個のステップです。

リンク・オプション:

cc コンパイラーをリンカーのフロントエンドとして使用します。

-xarch=$CFLAG_ARCH
このオプションを使用すると、 libdb2.so へのリンク時に必ず正しい実行可能ファイルがコンパイラーで生成されるようにすることができます。 $CFLAG_ARCH の値は、32 ビットの場合は v8plusa に、64 ビットの場合は v9 に設定されます。

-mt マルチスレッド・サポートにリンクします。 libdb2 を使ったリンクに必要です。
注: POSIX スレッドを使用する場合、DB2 アプリケーションは、スレッド化されていてもいなくても -lpthread にリンクする必要もあります。

-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。

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示します。例えば、32 ビットの場合は $HOME/sqllib/lib32、64 ビットの場合は $HOME/sqllib/lib64 です。

$EXTRA_LFLAG
実行時の DB2 共用ライブラリーのロケーションを示します。 32 ビットの場合は値 ″-R$DB2PATH/lib32″ が入り、 64 ビットの場合は値 ″-R$DB2PATH/lib64″ が入ります。

-ldb2 DB2 ライブラリーとリンクします。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

Solaris C++ アプリケーションのコンパイルとリンクのオプション:

以下は、bldapp ビルド・スクリプトに示されているように、Forte C++ コンパイラーを使用して、C++ 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

コンパイル・オプション:

CC C++ コンパイラー。

-xarch=$CFLAG_ARCH
このオプションを使用すると、 libdb2.so へのリンク時に必ず正しい実行可能ファイルがコンパイラーで生成されるようにすることができます。 $CFLAG_ARCH の値は、32 ビットの場合は v8plusa に、64 ビットの場合は v9 に設定されます。

-I$DB2PATH/include
DB2 組み込みファイルのロケーションを指定します。たとえば、$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。このスクリプトでは、コンパイルとリンクは別個のステップです。

リンク・オプション:

CC コンパイラーをリンカーのフロントエンドとして使用します。

-xarch=$CFLAG_ARCH
このオプションを使用すると、 libdb2.so へのリンク時に必ず正しい実行可能ファイルがコンパイラーで生成されるようにすることができます。 $CFLAG_ARCH の値は、32 ビットの場合は v8plusa に、64 ビットの場合は v9 に設定されます。

-mt マルチスレッド・サポートにリンクします。 libdb2 を使ったリンクに必要です。
注: POSIX スレッドを使用する場合、DB2 アプリケーションは、スレッド化されていてもいなくても -lpthread にリンクする必要もあります。

-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。

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示します。例えば、32 ビットの場合は $HOME/sqllib/lib32、64 ビットの場合は $HOME/sqllib/lib64 です。

$EXTRA_LFLAG
実行時の DB2 共用ライブラリーのロケーションを示します。 32 ビットの場合は値 ″-R$DB2PATH/lib32″ が入り、 64 ビットの場合は値 ″-R$DB2PATH/lib64″ が入ります。

-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。

Windows C および C++ アプリケーションのコンパイルとリンクのオプション:

以下は、bldapp.bat バッチ・ファイルに示されているように、Windows 上で Microsoft Visual C++ コンパイラーを使用して、 C および C++ 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

コンパイル・オプション:
%BLDCOMP% コンパイラー用の変数です。デフォルトは、cl (Microsoft Visual C++ コンパイラー) です。またこれは、icl (32 ビットおよび 64 ビット・アプリケーション用の Intel® C++ コンパイラー)、または ecl (Itanium® 64 ビット・アプリケーション用の Intel C++ コンパイラー) に設定することもできます。
-Zi デバッグ情報を使用可能にします。
-Od 最適化なし。最適化をオフにしてデバッガーを使用する方が簡単です。
-c コンパイルのみを実行し、リンクは実行しません。バッチ・ファイルでは、コンパイルとリンクは別個のステップです。
-W2 警告、エラー、重大、およびリカバリー不能エラー・メッセージを出力します。
-DWIN32 Windows オペレーティング・システムに必要なコンパイラー・オプション。
リンク・オプション:
link リンクにリンカーを使用します。
-debug デバッグ情報を組み込みます。
-out:%1.exe 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。

サンプル・ビルド・スクリプトを使用した C または C++ でのアプリケーションの構築 (UNIX)

DB2 には、C あるいは C++ の組み込み SQL と DB2管理 API プログラムをコンパイルしてリンクするためのビルド・スクリプトが用意されています。 C のアプリケーション用には sqllib/samples/c ディレクトリーに、C++ のアプリケーション

用には sqllib/samples/cpp ディレクトリーに置かれており、それと一緒に、それらのファイルを使用してビルドできるサンプル・プログラムも置かれています。

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 管理 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 プログラムを構築するにはデータベースへの接続が必要なため、 3 つのオプション・パラメーターも用意されています。 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、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ていない場合は、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置かれているインスタンスが異なる場合にのみ必要になります。

以下の例は、DB2 管理 API と組み込み SQL アプリケーションをビルドして実行する方法を示しています。

DB2 管理 API アプリケーションの構築および実行

C 用の cli_info.c および C++ 用の cli_info.C ソース・ファイルから DB2 管理 API サンプル・プログラム、cli_info をビルドするには、以下のように入力します。

```
bldapp cli_info
```

結果として、実行可能ファイル cli_info 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。

```
cli_info
```

組み込み SQL アプリケーションの構築と実行

- C 用の tbmod.sqc、および C++ 用の tbmod.sqC ソース・ファイルから組み込み SQL アプリケーション tbmod をビルドする場合、次の 3 つの方法があります。
  1. 同じインスタンス上のサンプル・データベースに接続している場合には、次のように入力します。

     ```
     bldapp tbmod
     ```
  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 tbmod 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 tbmod database userid password
     ```

結果として、実行可能ファイル tbmod 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。

     ```
     tbmod
     ```

  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。

     ```
     tbmod 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。

     ```
     tbmod database userid password
     ```

Windows での C/C++ アプリケーションの構築

DB2 には、 DB2 API と組み込み SQL C/C++ プログラムをコンパイルおよびリンクするためのビルド・スクリプトが用意されています。このファイルを使用して構築できるサンプル・プログラムと一緒に sqllib¥samples¥c および sqllib¥samples¥cpp ディレクトリーに置かれています。

バッチ・ファイル bldapp.bat には、 DB2 API と組み込み SQL プログラムを構築するためのコマンドが入っています。これは最大 4 個のパラメーターを取り、それらはバッチ・ファイル内で変数 %1、%2、%3、および %4 で表されます。

最初のパラメーター %1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を使用し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ためにはデータベースへの接続が必要なため、さらに別の 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 %2 は、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 %3 は、データベースのユーザー ID を指定します。そして %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、bldapp は、プリコンパイルおよびバインドのファイル embprep.bat にパラメーターを渡します。 データベース名が指定されていない場合は、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置かれているインスタンスが異なる場合にのみ必要になります。

- **組み込み SQL アプリケーションの構築と実行**

  sqllib¥samples¥c の C ソース・ファイル tbmod.sqc、または sqllib¥samples¥cpp の C++ ソース・ファイル tbmod.sqx から組み込み SQL アプリケーション tbmod を構築する方法は 3 つあります。

  – 同じインスタンス上のサンプル・データベースに接続している場合には、次のように入力します。

    ```
    bldapp tbmod
    ```

  – 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 tbmod database
    ```

– 他のインスタンスにあるデータベースに接続している場合は、さらにそのデータベース・インスタンスのユーザー ID とパスワードも入力します。

```
bldapp tbmod database userid password
```

結果として、実行可能ファイル tbmod.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。

– 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。

```
tbmod
```

– 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。

```
tbmod database
```

– 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

```
tbmod database userid password
```

• **マルチスレッド・アプリケーションの構築と実行**

Windows 用の C/C++ マルチスレッド・アプリケーションは、 -MT または -MD オプションを指定してコンパイルする必要があります。 -MT オプションを指定すると静的ライブラリー LIBCMT.LIB を使用してリンクされ、 -MD を指定すると動的ライブラリー MSVCRT.LIB を使用してリンクされます。 -MD を指定してリンクされたバイナリーはサイズは小さいですが MSVCRT.DLL がないと動作せず、 -MT を指定してリンクされたバイナリーはサイズは大きくても実行時に必要なものを内蔵しています。

バッチ・ファイル bldmt.bat は、-MT オプションを使用してマルチスレッド・プログラムを構築します。その他のコンパイルとリンクのオプションは、バッチ・ファイル bldapp.bat が通常のスタンドアロン・アプリケーションを構築するときに使用するものと同じです。

ソース・ファイル samples¥c¥dbthrds.sqc または samples¥cpp¥dbthrds.sqx からマルチスレッド・サンプル・プログラム dbthrds を構築するには、次のように入力します。

```
bldmt dbthrds
```

結果として、実行可能ファイル dbthrds.exe が作成されます。

このマルチスレッド・アプリケーションを実行する方法には次の 3 つがあります。

– 同じインスタンスにある sample データベースにアクセスする場合は、ただ実行可能ファイルの名前 (拡張子なし) を入力します。

```
dbthrds
```

– 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。

```
dbthrds database
```

– 他のインスタンスにあるデータベースにアクセスする場合は、実行可能ファイル名、データベース名、およびそのデータベース・インスタンスのユーザー ID とパスワードを入力します。

```
dbthrds database userid password
```

以下の例は、DB2 API と組み込み SQL のアプリケーションを構築して実行する方法を示しています。

sqllib¥samples¥c のソース・ファイル cli_info.c 、または sqllib¥samples¥cpp のソース・ファイル cli_info.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 cli_info を構築するには、次のように入力します。

```
bldapp cli_info
```

結果として、実行可能ファイル cli_info.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。

```
cli_info
```

VisualAge C++ で作成された組み込み SQL アプリケーションの構成ファイルによる構築

VisualAge C++ には、増分コンパイラーとバッチ・モード・コンパイラーの両方が装備されています。バッチ・モード・コンパイラーは Make ファイルとビルド・ファイルを使用するのに対して、増分コンパイラーは構成ファイルを使用します。これに関して詳しく知りたい場合は、VisualAge C++ バージョン 5.0 に添付されている資料を参照してください。

DB2 は、VisualAge C++ コンパイラーでビルド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。以下のトピックに、DB2 に用意されている構成ファイルを使用して各種のプログラムをコンパイルする方法が説明されています。

- 構成ファイルを使用した C または C++ の組み込み SQL および DB2 管理 API アプリケーションの構築 (AIX)
- 「*SQL および外部ルーチンの開発* 」にある『構成ファイルを使用した C または C++ の組み込み SQL ストアード・プロシージャーの構築』
- 「*SQL および外部ルーチンの開発* 」にある『構成ファイルを使用した C または C++ のユーザー定義関数の構築 (AIX)』

構成ファイルを使用した C または C++ の組み込み SQL および DB2 API アプリケーションの構築 (AIX)

構成ファイルは、基本的には、VisualAge で作成されたアプリケーションのビルドに使用されるビルド・スクリプトです。このスクリプトは、エラー・チェック、デー

タベースへの接続、コードのプリコンパイル、 DB2 にアクセスするためのパスの設定、および最終的な実行可能ファイルの生成など、さまざまなことを行います。

以下の例は、構成ファイルを使用して、DB2 管理 API と組み込み SQL アプリケーションをビルドして実行する方法を示しています。

- **構成ファイルによる C および C++ 組み込み SQL アプリケーションのビルド**

  sqllib/samples/c と sqllib/samples/cpp の中の構成ファイル emb.icc を使用すれば、 AIX 上で C および C++ の DB2 組み込み SQL アプリケーションを構築することができます。構成ファイルを使用してソース・ファイル tbmod.sqc から組み込み SQL アプリケーション tbmod 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。
     - bash または Korn シェルの場合

       ```
       export EMB=tbmod
       ```

     - C シェルの場合

       ```
       setenv EMB tbmod
       ```

  2. emb.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 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++ で提供されます。

     結果として、実行可能ファイル tbmod が作成されます。このプログラムを実行するには、次の実行可能ファイル名を入力します。

     ```
     tbmod
     ```

- **構成ファイルによる C および C++ DB2 API アプリケーションのビルド**

  sqllib/samples/c および sqllib/samples/cpp にある構成ファイル、api.icc を使用すれば、Cまたは C++ の DB2 管理 API プログラムをAIX 上でビルドできます。

  構成ファイルを使用して、DB2 管理 API サンプル・プログラム cli_info をソース・ファイル cli_info.c から構築するには、以下のようにします。

  1. 次のように入力して、API 環境変数をプログラム名に設定します。
     - bash または Korn シェルの場合

       ```
       export API=cli_info
       ```

     - C シェルの場合

       ```
       setenv API cli_info
       ```

2. api.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 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++ で提供されます。

   結果として、実行可能ファイル cli_info が作成されます。このプログラムを実行するには、次の実行可能ファイル名を入力します。

   ```
   cli_info
   ```

Windows での C/C++ 複数接続アプリケーションの構築

DB2 には、 C および C++ 組み込み SQL と DB2 API プログラムをコンパイルしてリンクするためのビルド・スクリプトが用意されていて、このファイルを使用して構築できるサンプル・プログラムと一緒に sqllib¥samples¥c および sqllib¥samples¥cpp ディレクトリーに置かれています。

バッチ・ファイル bldmc.bat には、 2 つのデータベースを必要とする DB2 複数接続アプリケーション・プログラムを作成するコマンドが入っています。コンパイルとリンクのオプションは、bldapp.bat ファイルが使用するものと同じです。

最初のパラメーター %1 には、ソース・ファイルの名前を指定します。第 2 パラメーター %2 には、接続先の 1 つ目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、接続先の 2 つ目のデータベースの名前を指定します。これらはすべて必要パラメーターです。

注: ビルド・スクリプトには、データベース名のデフォルト値 ″sample″ と ″sample2″ (それぞれ %2 と %3) がハードコーディングされているため、このビルド・スクリプトを使用してこれらのデフォルトを受け入れる場合は、指定する必要があるのはプログラム名 (%1 パラメーター) だけです。 bldmc.bat スクリプトを使用する場合は、3 つのパラメーターすべてを指定します。

オプション・パラメーターは、ローカル接続の場合は必要ありませんが、リモート・クライアントからサーバーに接続する場合は必要です。オプション・パラメーターの %4 と %5 は、それぞれ最初のデータベースのユーザー ID とパスワードを指定し、%6 と %7 は、それぞれ 2 番目のデータベースのユーザー ID とパスワードを指定します。

複数接続サンプル・プログラム dbmcon.exe には、2 つのデータベースが必要です。 sample データベースをまだ作成していなければ、 DB2 コマンド・ウィンドウのコマンド行で db2sampl と入力して作成できます。 2 番目のデータベース (ここでは sample2 という名前) は、以下のいずれかのコマンドを使用して作成できます。

データベースをローカルで作成する場合:

```
db2 create db sample2
```

データベースをリモートから作成する場合:

```
db2 attach to <node_name>
db2 create db sample2
db2 detach
db2 catalog db sample2 as sample2 at node <node_name>
```

<node_name> は、データベースが常駐するノードです。

複数接続では、TCP/IP Listener も実行されている必要があります。これを確実に実行するには、次のようにします。

1. 環境変数 DB2COMM を TCP/IP に設定します。

   ```
   db2set DB2COMM=TCPIP
   ```

2. サービス・ファイルで指定されるように、データベース・マネージャーの構成ファイルを TCP/IP サービス名で更新します。

   ```
   db2 update dbm cfg using SVCENAME <TCP/IP service name>
   ```

   各インスタンスは TCP/IP サービス名を持っており、この名前はサービス・ファイルにリストされています。このファイルが見つからない場合、またはサービス・ファイルを変更するファイル許可がない場合は、システム管理者に連絡してください。

3. データベース・マネージャーをいったん停止してから再始動して、これらの変更を有効にします。

   ```
   db2stop
   db2start
   ```

dbmcon.exe プログラムは、 samples¥c または samples¥cpp ディレクトリーにある次の 5 つのファイルから作成されます。

dbmcon.sqc または dbmcon.sqx
: 両方のデータベースに接続するためのメイン・ソース・ファイル。

dbmcon1.sqc または dbmcon1.sqx
: 最初のデータベースにバインドされるパッケージを作成するためのソース・ファイル。

dbmcon1.h
: 主ソース・ファイル dbmcon.sqc または dbmcon.sqx に組み込まれた dbmcon1.sqc または dbmcon1.sqx のヘッダー・ファイルで、最初のデータベースにバインドされた表を作成およびドロップするための SQL ステートメントにアクセスするためのものです。

dbmcon2.sqc または dbmcon2.sqx
: 2 番目のデータベースにバインドされるパッケージを作成するためのソース・ファイル。

dbmcon2.h
: 主ソース・ファイル dbmcon2.sqc または dbmcon2.sqx に組み込まれた dbmcon.sqc または dbmcon.sqx のヘッダー・ファイルで、 2 番目のデータ

ベースにバインドされた表を作成およびドロップするための SQL ステートメントにアクセスするためのものです。

複数接続サンプル・プログラム dbmcon.exe を構築するには、次のようにします。

```
bldmc dbmcon sample sample2
```

結果として、実行可能ファイル dbmcon.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を拡張子なしで実行します。

```
dbmcon
```

2 つのデータベースへの 1 フェーズ・コミットが、プログラムによって例示されます。

COBOL で作成されたアプリケーションおよびルーチンのビルド

COBOL の組み込み SQL アプリケーションのビルドを可能にする、さまざまなオペレーティング・システム・プラットフォーム向けのビルド・スクリプトが製品とともに提供されています。アプリケーションのビルドに使用されるビルド・スクリプトに加え、ルーチン (ストアード・プロシージャーおよびユーザー定義関数) をビルドするために特に使用される bldrtn スクリプトも提供されています。 Linux 上で、Micro Focus COBOL 言語で書かれたアプリケーションを扱う場合は、コンパイラーを、特定の COBOL 共用ライブラリーにアクセスできるよう必ず構成してください。 IBM COBOL のサンプルが提供されています。それらは、UNIX 用には sqllib/samples/cobol ディレクトリーに、Windows 用には sqllib¥samples¥cobol ディレクトリーにあります。Micro Focus COBOL サンプルの場合、ディレクトリーのパスの末尾の 'cobol' を 'cobol_mf' に置き換えてください。

COBOL のコンパイルとリンクのオプション

AIX IBM COBOL アプリケーションのコンパイルとリンクのオプション:

以下は、bldapp ビルド・スクリプトに示されているように、IBM COBOL for AIX コンパイラーを使用して、COBOL 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

コンパイル・オプション:
cob2 IBM COBOL for AIX コンパイラー。
-qpgmname¥(mixed¥) コンパイラーに、大/小文字混合の名前を持つライブラリーのエントリー・ポイントの 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$DB2PATH/include/cobol_a DB2 組み込みファイルのロケーションを指定します。たとえば、$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。

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・チェック用のユーティリティー・オブジェクト・ファイルを組み込みます。

-L$DB2PATH/$LIB
DB2 ランタイム共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/lib32 のように指定します。

-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。

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。

AIX Micro Focus COBOL アプリケーションのコンパイルとリンクのオプション:

以下は、bldapp ビルド・スクリプトに示されているように、AIX 上で Micro Focus COBOL コンパイラーを使用して、COBOL 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。 DB2 MicroFocus COBOL 組み込みファイルは、COBCPY 環境変数をセットアップすることによって検出されます。したがって、コンパイルのステップで -I フラグは必要ありません。この例は、bldapp スクリプトを参照してください。

bldapp のコンパイルとリンクのオプション

コンパイル・オプション:

cob MicroFocus 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。

$EXTRA_COBOL_FLAG="-C MFSYNC"
64 ビット・サポートを使用可能にします。

-x -c と一緒に使用すると、オブジェクト・ファイルが作成されます。

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。

-L$DB2PATH/$LIB
DB2 ランタイム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/lib32 のように指定します。

-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。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

HP-UX Micro Focus COBOL アプリケーションのコンパイルとリンクのオプション:

以下は、bldapp ビルド・スクリプトに示されているように、HP-UX 上で Micro Focus COBOL コンパイラーを使用して、COBOL 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

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。 **$EXTRA_COBOL_FLAG** HP-UX プラットフォームが IA64 で、64 ビット・サポートが有効な場合は、″-C MFSYNC″ が入ります。
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。 **$1.o** プログラム・オブジェクト・ファイルを組み込みます。 **checkerr.o** エラー・チェック用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/$LIB** DB2 ランタイム共用ライブラリーのロケーションを指定します。 **-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。 他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

Solaris Micro Focus COBOL アプリケーションのコンパイルとリンクのオプション:

以下は、bldapp ビルド・スクリプトに示されているように、Solaris 上で Micro Focus COBOL コンパイラーを使用して、COBOL 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

コンパイル・オプション: **cob** Micro Focus COBOL コンパイラー。 **$EXTRA_COBOL_FLAG** 64 ビット・サポート用には、値 ″-C MFSYNC″ を入れます。その他の場合には、値を入れません。 **-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。 **$1.o** プログラム・オブジェクト・ファイルを組み込みます。 **checkerr.o** エラー・チェック用のユーティリティー・オブジェクト・ファイルを組み込みます。 **-L$DB2PATH/$LIB** リンク時の DB2 静的ライブラリーおよび共用ライブラリーのロケーションを示します。たとえば、$HOME/sqllib/lib64。 **-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。

Linux Micro Focus COBOL アプリケーションのコンパイルとリンクのオプション:

以下は、bldapp ビルド・スクリプトに示されているように、Linux 上で Micro Focus COBOL コンパイラーを使用して、COBOL 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

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。 **$EXTRA_COBOL_FLAG** 64 ビット・サポート用には、値 ″-C MFSYNC″ を入れます。その他の場合には、値を入れません。

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。

-o $1 実行可能ファイルを組み込みます。

$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・チェック用のユーティリティー・オブジェクト・ファイルを組み込みます。

-L$DB2PATH/$LIB
DB2 ランタイム共用ライブラリーのロケーションを指定します。

-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。

他のコンパイラー・オプションについては、コンパイラーの資料をご覧ください。

Windows IBM COBOL アプリケーションのコンパイルとリンクのオプション:

以下は、`bldapp.bat` バッチ・ファイルに示されているように、 Windows 上で IBM VisualAge COBOL コンパイラーを使用して、 COBOL 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

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/小文字混合の名前を持つライブラリーのエントリー・ポイントの 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。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 組み込みファイルのロケーションを指定します。例えば、`-I"%DB2PATH %¥include¥cobol_a"`。

%EXTRA_COMPFLAG%
"set IBMCOB_PRECOMP=true" がコメント化されていない場合、IBM COBOL プリコンパイラーを使用して組み込み SQL がプリコンパイルされます。入力パラメーターに応じて、以下のいずれかの公式で呼び出されます。

-q"SQL('database sample CALL_RESOLUTION DEFERRED')"
デフォルトのサンプル・データベースを使用してプリコンパイルし、呼び出し解決を据え置きます。

-q"SQL('database %2 CALL_RESOLUTION DEFERRED')"
ユーザー指定のデータベースを使用してプリコンパイルし、呼び出し解決を据え置きます。

-q"SQL('database %2 user %3 using %4 CALL_RESOLUTION DEFERRED')"
ユーザー指定のデータベース、ユーザー ID、およびパスワードを使用してプリコンパイルし、呼び出し解決を据え置きます。これは、リモート・クライアント・アクセス用の書式です。

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。

%1.obj プログラム・オブジェクト・ファイルを組み込みます。

checkerr.obj
エラー・チェック・ユーティリティー・オブジェクト・ファイルを組み込みます。

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。

Windows Micro Focus COBOL アプリケーションのコンパイルとリンクのオプション:

以下は、`bldapp.bat` バッチ・ファイルに示されているように、Windows 上で Micro Focus COBOL コンパイラーを使用して、 COBOL 組み込み SQL および DB2 API アプリケーションを構築するのにお勧めするコンパイルとリンクのオプションです。

bldapp のコンパイルとリンクのオプション

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** リンカーを使用してリンク・エディットします。 **-l**　lcobol ライブラリーとリンクします。 **checkerr.obj** エラー・チェック・ユーティリティー・オブジェクト・ファイルとリンクします。 **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。

COBOL コンパイラーの構成

AIX での IBM COBOL コンパイラーの構成:

組み込み SQL および DB2 API 呼び出しの入ったアプリケーションを開発する場合に、IBM COBOL Set for AIX コンパイラーを使用していれば、以下のステップを行う必要があります。

- コマンド行プロセッサーのコマンド db2 prep を使用し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。
- 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・タイプ・サポート」機能を使用する場合、 アプリケーション用の DB2 組み込みファイルは以下のディレクトリーにあります。

  ```
  $HOME/sqllib/include/cobol_i
  ```

  提供されたスクリプト・ファイルを使用して DB2 サンプル・プログラムを構築している場合、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・タイプ・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
  ```

次のように、COPY ファイル名を .cbl 拡張子を含めて指定します。

```
COPY "sql.cbl".
```

Windows での IBM COBOL コンパイラーの構成:

組み込み SQL および DB2 API 呼び出しの入ったアプリケーションを開発する場合に、 IBM VisualAge COBOL コンパイラーを使用するときは、気をつける点がいくつかあります。

- DB2 プリコンパイラーでアプリケーションをプリコンパイルし、コマンド行プロセッサーのコマンド db2 prep を使用する場合は、 target ibmcob オプション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・ファイル内で PROCESS および CBL キーワードを使用して、コンパイル・オプションを設定します。キーワードは 8 桁目から 72 桁目まで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・タイプ・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・ファイルを使用して DB2 サンプル・プログラムを構築している場合、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・タイプ・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
  ```

  cobol_a ディレクトリーはデフォルトです。
- 次のように、COPY ファイル名を .cbl 拡張子を含めて指定します。

  ```
  COPY "sql.cbl".
  ```

Windows での Micro Focus COBOL コンパイラーの構成:

組み込み SQL および DB2 API 呼び出しの入ったアプリケーションを開発する場合に、 Micro Focus コンパイラーを使用するときは、気をつける点がいくつかあります。

- コマンド行プロセッサーのコマンド db2 prep を使用してアプリケーションをプリコンパイルする場合は、 target mfcob オプションを使用してください。
- 以下のコマンドを使用して、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%"
```

- 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%"
  ```

  上に挙げた環境変数は、必ず、System 設定で永続的に設定されているようにしてください。これは、次のようなステップで確認できます。

  1. 「**コントロール パネル**」を開きます。
  2. 「**システム**」を選択します。
  3. 「**詳細設定**」タブを選択します。
  4. 「**環境変数**」をクリックします。
  5. 「**システム環境変数**」のリストに、必要な環境変数があるかどうかを確認してください。必要な環境変数がリストにない場合は、「**システム環境変数**」のリストに追加してください。

  コマンド・プロンプトまたはスクリプトのいずれで行う場合でも、User 設定での設定では不十分です。

DB2 アプリケーション・プログラミング・インターフェースの呼び出しはすべて、呼び出し規則 74 を使用して行わ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シンボル 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、手動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 シンボルを手動で追加する必要があります。

Linux での Micro Focus COBOL コンパイラーの構成:

Micro Focus COBOL ルーチンを実行するには、 Linux ランタイム・リンカーが特定の COBOL 共用ライブラリーにアクセスできる必要があり、しかも DB2 がそのライブラリーをロードできる必要があります。そのロードを行うプログラムは、setuid 特権のもとで実行されるので、 /usr/lib 内の従属ライブラリーだけが探索されることになります。

COBOL 共用ライブラリー用に /usr/lib へのシンボリック・リンクを作成します。これは root として実行する必要があります。これを行う最も簡単な方法は、次のようにして、すべての COBOL ライブラリー・ファイルを $COBDIR/lib から /usr/lib にリンクするという方法です。

```
ln -s $COBDIR/lib/libcob* /usr/lib
```

$COBDIR は Micro Focus COBOL のインストール先で、通常は /opt/lib/mfcobol です。

以下は、各個別ファイルをリンクするためのコマンドです (Micro Focus COBOL は /opt/lib/mfcobol にインストールされていると想定しています)。

```
ln -s /opt/lib/mfcobol/lib/libcobrts.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobrts_t.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobrts.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobrts_t.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobcrtn.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobcrtn.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobmisc.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobmisc_t.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobmisc.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobmisc_t.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobscreen.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobscreen.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobtrace.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobtrace_t.so /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobtrace.so.2 /usr/lib
ln -s /opt/lib/mfcobol/lib/libcobtrace_t.so.2 /usr/lib
```

各 DB2 インスタンスで、次のようにする必要があります。

- コマンド行プロセッサーのコマンド db2 prep を使用してアプリケーションをプリコンパイルする場合は、 target mfcob オプションを使用してください。
- 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含めるには、次のように入力します。

  – bash または korn シェルの場合

    ```
    export COBCPY=$HOME/sqllib/include/cobol_mf:$COBDIR/cpylib
    ```

  – C シェルの場合

    ```
    setenv COBCPY $HOME/sqllib/include/cobol_mf:$COBDIR/cpylib
    ```

- 環境変数を次のように更新します。

  – bash または korn シェルの場合

    ```
    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/sqllib/lib:$COBDIR/lib
    ```

  – C シェルの場合

    ```
    setenv LD_LIBRARY_PATH $LD_LIBRARY_PATH:$HOME/sqllib/lib:$COBDIR/lib
    ```

- DB2 環境リストを設定します。

  ```
  db2set DB2ENVLIST="COBDIR LD_LIBRARY_PATH"
  ```

注: COBCPY、COBDIR、および LD_LIBRARY_PATH を、 .bashrc、.kshrc (使用中のシェルによって異なる)、 .bash_profile、.profile (使用中のシェルによって異なる)、または .login に設定することもできます。

AIX での Micro Focus COBOL コンパイラーの構成:

Micro Focus COBOL コンパイラーを使用して、組み込み SQL および DB2 API 呼び出しの入ったアプリケーションを開発する場合は、次のようにします。

- コマンド行プロセッサーのコマンド db2 prep を使用してアプリケーションをプリコンパイルする場合は、 target mfcob オプションを使用してください。
- 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含めるには、次のように入力します。

  – bash または korn シェルの場合

    ```
    export COBCPY=$COBCPY:$HOME/sqllib/include/cobol_mf
    ```

  – C シェルの場合

    ```
    setenv COBCPY $COBCPY:$HOME/sqllib/include/cobol_mf
    ```

  注: COBCPY を .profile または .login ファイル中に設定することもできます。

HP-UX での Micro Focus COBOL コンパイラーの構成:

組み込み SQL および DB2 API 呼び出しの入ったアプリケーションを開発する場合に、Micro Focus COBOL コンパイラーを使用するときは、気をつける点がいくつかあります。

- コマンド行プロセッサーのコマンド db2 prep を使用してアプリケーションをプリコンパイルする場合は、 target mfcob オプションを使用してください。
- 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

  – bash または korn シェルでは以下を入力します。

    ```
    export COBCPY=$COBCPY:$HOME/sqllib/include/cobol_mf
    ```

  – C シェルでは以下を入力します。

    ```
    setenv COBCPY ${COBCPY}:${HOME}/sqllib/include/cobol_mf
    ```

  注: COBCPY を .profile または .login ファイル中に設定することもできます。

Solaris での Micro Focus COBOL コンパイラーの構成:

組み込み SQL および DB2 API 呼び出しの入ったアプリケーションを開発する場合に、Micro Focus COBOL コンパイラーを使用するには、気を付けなければならない点があります。

- コマンド行プロセッサーのコマンド db2 prep を使用してアプリケーションをプリコンパイルする場合は、 target mfcob オプションを使用してください。
- 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。

  – bash または korn シェルの場合

  ```
  export COBCPY=$COBCPY:$HOME/sqllib/include/cobol_mf
  ```

  – C シェルの場合

  ```
  setenv 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。

AIX での IBM COBOL アプリケーションの構築

DB2 には、 IBM COBOL 組み込み SQL と DB2 管理 API プログラムをコンパイルしてリンクするためのビルド・スクリプトが用意されていて、このファイルを使用して構築できるサンプル・プログラムと一緒に sqllib/samples/cobol ディレクトリーに置かれています。

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 を使用し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ため、 3 つのオプション・パラメーターも用意されています。 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、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ていない場合は、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置かれているインスタンスが異なる場合にのみ必要になります。

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 を構築するには、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
     ```

UNIX Micro Focus COBOL アプリケーションの構築

DB2 には、 Micro Focus COBOL 組み込み SQL と DB2 管理 API プログラムをコンパイルしてリンクするためのビルド・スクリプトが用意されていて、このファイルを使用して構築できるサンプル・プログラムと一緒に sqllib/samples/cobol_mf ディレクトリーに置かれています。

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 を使用し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ため、 3 つのオプション・パラメーターも用意されています。 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、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ていない場合は、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置かれているインスタンスが異なる場合にのみ必要になります。

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 を構築するには、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
     ```

Windows での IBM COBOL アプリケーションの構築

DB2 には、 DB2 API と組み込み SQL プログラムのコンパイルとリンクのためのビルド・スクリプトが用意されています。このファイルを使用して構築できるサンプル・プログラムと一緒に sqllib¥samples¥cobol ディレクトリーに置かれています。

DB2 は、Windows での IBM COBOL アプリケーションの構築用に、 DB2 プリコンパイラーと IBM COBOL プリコンパイラーの 2 種類のプリコンパイラーをサポートします。デフォルトは DB2 プリコンパイラーです。使用するバッチ・ファイルの該当する行のコメントを外すことにより、 IBM COBOL プリコンパイラーを選択できます。 IBM COBOL でのプリコンパイルは、特定のプリコンパイル・オプションを使用して、コンパイラー単体で実行できます。

バッチ・ファイル bldapp.bat には、 DB2 アプリケーション・プログラムを構築するコマンドが入っています。これは最大 4 個のパラメーターを取り、それらはバッチ・ファイル内で変数 %1、%2、%3、および %4 で表されます。

最初のパラメーター %1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を使用し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ためにはデータベースへの接続が必要なため、3 つのオプション・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、パスワードを指定します。

デフォルトの DB2 プリコンパイラーを使用する組み込み SQL プログラムの場合、bldapp.bat は、プリコンパイルおよびバインドのファイル embprep.bat にパラメーターを渡します。

IBM COBOL プリコンパイラーを使用する組み込み SQL プログラムの場合、bldapp.bat は .sqb ソース・ファイルを .cbl ソース・ファイルにコピーします。コンパイラーは .cbl ソース・ファイルに対して特定のプリコンパイル・オプションを使用して、プリコンパイルを実行します。

いずれのプリコンパイラーの場合も、データベース名が指定されていない場合は、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置かれているインスタンスが異なる場合にのみ必要になります。

以下の例は、DB2 API と組み込み SQL のアプリケーションを構築して実行する方法を示しています。

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 を構築するには、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
     ```

Windows での Micro Focus COBOL アプリケーションの構築

DB2 には、 DB2 API と組み込み SQL プログラムのコンパイルとリンクのためのビルド・スクリプトが用意されています。このファイルを使用して構築できるサンプル・プログラムと一緒に sqllib¥samples¥cobol_mf ディレクトリーに置かれています。

バッチ・ファイル bldapp.bat には、 DB2 アプリケーション・プログラムを構築するコマンドが入っています。これは最大 4 個のパラメーターを取り、それらはバッチ・ファイル内で変数 %1、%2、%3、および %4 で表されます。

最初のパラメーター %1 には、ソース・ファイルの名前を指定します。これは組み込み SQL を使用し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ためにはデータベースへの接続が必要なため、3 つのオプション・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、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.bat にパラメーターを渡します。 データベース名が指定されていない場合は、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置かれているインスタンスが異なる場合にのみ必要になります。

以下の例は、DB2 API と組み込み SQL のアプリケーションを構築して実行する方法を示しています。

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 を構築するには、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.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
     ```

REXX で作成された組み込み SQL アプリケーションの構築および実行

REXX アプリケーションではプリコンパイル、およびリンクは行われません。以下は、Windows オペレーティング・システムおよび AIX オペレーティング・システムで REXX アプリケーションをビルドして実行する方法について説明しています。

Windows ベースのプラットフォームの場合、アプリケーション・ファイルの拡張子は .CMD になっている必要があります。この拡張子を付けると、アプリケーション

をオペレーティング・システムのコマンド・プロンプトから直接実行できます。AIX 上で、アプリケーション・ファイルには、任意の拡張子を付けることができます。

REXX アプリケーションを構築して実行するには、以下のようにします。

- Windows オペレーティング・システムの場合、アプリケーション・ファイルには、任意の名前を付けることができます。ファイルを作成後、次のように REXX インタープリターを起動して、アプリケーションをオペレーティング・システムのコマンド・プロンプトから実行できます。

  ```
  REXX file_name
  ```

- AIX では、アプリケーションは、次の 2 つの方法のうちいずれかを使用して実行することができます。
  - シェル・コマンド・プロンプトで、rexx name と入力する。 name は REXX プログラムの名前です。
  - 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 インタープリター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 インタープリター・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。

注: AIX では、LIBPATH 環境変数をセットして、 REXX SQL ライブラリー db2rexx があるディレクトリーを含めます。以下に例を示します。

```
export LIBPATH=/lib:/usr/lib:/$DB2PATH/lib
```

REXX のバインド・ファイル

REXX アプリケーションをサポートするために、5 つのバインド・ファイルが提供されています。それらのファイルの名前は、DB2UBIND.LST ファイルの中に組み込まれています。各バインド・ファイルは、それぞれ別々の分離レベルを使用してプリコンパイルされます。したがって、5 つの異なるパッケージがデータベース内に保管されます。

以下に、5 つのバインド・ファイルを示します。

DB2ARXCS.BND
カーソル固定の分離レベルをサポートします。

DB2ARXRR.BND
反復可能読み取り分離レベルをサポートします。

DB2ARXUR.BND
非コミット読み取りの分離レベルをサポートします。

DB2ARXRS.BND
読み取り固定の分離レベルをサポートします。
DB2ARXNC.BND
コミットなしの分離レベルをサポートします。この分離レベルは、一部のホストまたは System i データベース・サーバーを作動させる際に使用されます。他のデータベースでは、非コミット読み取りの分離レベルと同様に動作します。

注: これらのファイルを、データベースに明示的にバインドしなければならない場合もあります。

SQLEXEC ルーチンを使用する場合は、カーソル固定により作成されたパッケージがデフォルトのパッケージとして使用されます。他の分離レベルを使用する必要がある場合は、データベースに接続する前に SQLDBS CHANGE SQL ISOLATION LEVEL API を使用して分離レベルを変更できます。分離レベルを変更すると、その後に続く SQLEXEC ルーチンに対する呼び出しが、新たに指定した分離レベルと関連付けられます。

Windows ベースの REXX アプリケーションは、セッション内の他の REXX プログラムの設定が変更されていないことを確認しない限り、デフォルトの分離レベルが有効であると見なしません。 REXX アプリケーションは、データベースに接続する前に分離レベルを明示的に設定しなければなりません。

Windows でのオブジェクト REXX アプリケーションの構築

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。オブジェクト指向の拡張子が従来の 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 インタープリターは、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
- メッセージング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 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、移植性の理由から、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 `sqllib¥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。

```
rexx updat.cmd
```

コマンド行からの組み込み SQL アプリケーションの構築

コマンド行から組み込み SQL アプリケーションをビルドするには、以下のステップに従ってください。

1. PRECOMPILE コマンドを発行して、アプリケーションをプリコンパイルします。
2. バインド・ファイルを作成した場合は、BIND コマンドを発行することによって、このファイルをデータベースにバインドし、アプリケーション・パッケージを作成します。
3. 修正されたアプリケーション・ソースと、組み込み SQL を含まないソース・ファイルをコンパイルして、アプリケーション・オブジェクト・ファイル (.obj ファイル) を作成します。
4. link コマンドを使用して、アプリケーション・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します。

C または C++ で作成された組み込み SQL アプリケーションの構築 (Windows)

ソース・ファイルを作成した後は、その組み込み SQL アプリケーションをビルドする必要があります。ビルド・プロセスのいくつかのステップは、使用するコンパイラーによって異なります。手順の各ステップで紹介している例は、C コンパイラーである Microsoft Visual Studio 6.0 コンパイラーを使用して、myapp というアプリケーションをビルドする方法を示すためのものです。手順の各ステップは、個々に行うことも、DB2 コマンド・ウィンドウのプロンプトからバッチ・ファイルを使用して一度にまとめて実行することもできます。 %DB2PATH %¥SQLLIB¥samples¥c¥ ディレクトリーの組み込み SQL サンプル・アプリケーションの構築に使用できるバッチ・ファイルの例については、%DB2PATH %¥SQLLIB¥samples¥c¥bldapp.bat ファイルを参照してください。このバッチ・ファイルは、もう 1 つのバッチ・ファイル、%DB2PATH %¥SQLLIB¥samples¥c¥embprep.bat を呼び出して、アプリケーションのプリコンパイル、およびデータベースへのアプリケーションのバインドを行います。

- アクティブなデータベース接続
- アプリケーションのソース・コード・ファイル。拡張子は C の場合 .sqc、C++ の場合 .sqx であり、組み込み SQLを含む。
- サポートされている C または C++ コンパイラー
- PRECOMPILE コマンドおよび BIND コマンドの実行に必要な権限または特権

1. PRECOMPILE コマンドを発行して、アプリケーションをプリコンパイルします。以下に例を示します。

```
C application: db2 PRECOMPILE myapp.sqc BINDFILE
C++ application: db2 PRECOMPILE myapp.sqx BINDFILE
```

   PRECOMPILE コマンドによって、.sqc または .sqC ファイルのソース・コードを修正したものを収めた .c または .C ファイル、およびアプリケーション・パッケージが生成されます。 BINDFILE オプションを使用した場合、PRECOMPILE コマンドはバインド・ファイルを生成します。上記の例では、バインド・ファイルの名前は myapp.bnd となります。

2. バインド・ファイルを作成した場合は、BIND コマンドを発行することによって、このファイルをデータベースにバインドし、アプリケーション・パッケージを作成します。以下に例を示します。

```
db2 bind myapp.bnd
```

   BIND コマンドを実行すると、アプリケーション・パッケージがデータベースに関連付けられ、パッケージがデータベース内に保管されます。

3. 修正されたアプリケーション・ソースと、組み込み SQL を含まないソース・ファイルをコンパイルして、アプリケーション・オブジェクト・ファイル (.obj ファイル) を作成します。以下に例を示します。

```
C application: cl -Zi -Od -c -W2 -DWIN32 myapp.c
C++ application: cl -Zi -Od -c -W2 -DWIN32 myapp.cxx
```

4. link コマンドを使用して、アプリケーション・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します。以下に例を示します。

```
link -debug -out:myapp.exe myapp.obj
```

次の作業

次に、以下の作業をいずれも行うことができます。

- 組み込み SQL アプリケーションのコンパイル時エラーのデバッグ
- 組み込み SQL アプリケーションのパフォーマンスのチューニング
- 実行可能アプリケーション、myapp.exe のデプロイ
- 組み込み SQL アプリケーションの実行

第 7 章 組み込み SQL アプリケーションのデプロイおよび実行

組み込み SQL アプリケーションは移植可能であり、リモート・マシンに配置できます。 1 つの場所でアプリケーションをコンパイルしてから、パッケージを別のマシン上で実行して、新しい方のマシン上にあるデータベースを使用することができます。

libdb2.so へのリンクに関する制約事項

いくつかの Linux ディストリビューションで、 libc 開発 rpm には、

```
/usr/lib/libdb2.so
```

または

```
/usr/lib64/libdb2.so
```

ライブラリーが備わっています。このライブラリーは、 Sleepycat Software 社の Berkeley DB のインプリメンテーション用に使用されるものであり、 IBM DB2 データベース・システムとは関係ありません。

Berkeley DB を使用する予定がない場合、これらのライブラリー・ファイルを名前変更するか、またはシステムから永久に削除することができます。

Berkeley DB を使用する場合、これらのライブラリー・ファイルを含むフォルダーを名前変更して、その新しいフォルダーを指し示すように環境変数を変更できます。

第 2 部 付録

付録 A. DB2 技術情報の概説

DB2 技術情報は、以下のツールと方法を介して利用できます。

- DB2 インフォメーション・センター
  - トピック (タスク、概念、およびリファレンス・トピック)
  - DB2 ツールのヘルプ
  - サンプル・プログラム
  - チュートリアル
- DB2 資料
  - PDF ファイル (ダウンロード可能)
  - PDF ファイル (DB2 PDF DVD に含まれる)
  - 印刷資料
- コマンド行ヘルプ
  - コマンド・ヘルプ
  - メッセージ・ヘルプ

注: DB2 インフォメーション・センターのトピックは、PDF やハードコピー資料よりも頻繁に更新されます。最新の情報を入手するには、資料の更新が発行されたときにそれをインストールするか、ibm.com® にある DB2 インフォメーション・センターを参照してください。

技術資料、ホワイト・ペーパー、IBM Redbooks® 資料などのその他の DB2 技術情報には、オンライン (ibm.com) でアクセスできます。 DB2 Information Management ソフトウェア・ライブラリー・サイト (http://www.ibm.com/software/data/sw-library/) にアクセスしてください。

資料についてのフィードバック

DB2 の資料についてのお客様からの貴重なご意見をお待ちしています。DB2 の資料を改善するための提案については、db2docs@ca.ibm.com まで E メールを送信してください。DB2 の資料チームは、お客様からのフィードバックすべてに目を通しますが、直接お客様に返答することはありません。お客様が関心をお持ちの内容について、可能な限り具体的な例を提供してください。特定のトピックまたはヘルプ・ファイルについてのフィードバックを提供する場合は、そのトピック・タイトルおよび URL を含めてください。

DB2 お客様サポートに連絡する場合には、この E メール・アドレスを使用しないでください。資料を参照しても、DB2 の技術的な問題が解決しない場合は、お近くの IBM サービス・センターにお問い合わせください。

DB2 テクニカル・ライブラリー (ハードコピーまたは PDF 形式)

以下の表は、DB2 ライブラリーについて説明しています。DB2 ライブラリーに関する詳細な説明については、www.ibm.com/shop/publications/order にある IBM Publications Center にアクセスしてください。英語の DB2 バージョン 9.5 のマニュアル (PDF 形式) とその翻訳版は、 www.ibm.com/support/docview.wss?rs=71&uid=swg2700947 からダウンロードできます。

この表には印刷資料が入手可能かどうかが示されていますが、国または地域によっては入手できない場合があります。

資料番号は、資料が更新される度に大きくなります。資料を参照する際は、以下にリストされている最新版であることを確認してください。

注: DB2 インフォメーション・センターは、PDF やハードコピー資料よりも頻繁に更新されます。

表 23. DB2 の技術情報

資料名	資料番号	印刷資料が入手可能かどうか
管理 API リファレンス	SC88-4431-01	入手可能
管理ルーチンおよびビュー	SC88-4435-01	入手不可
コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻	SC88-4433-01	入手可能
コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻	SC88-4434-01	入手可能
コマンド・リファレンス	SC88-4432-01	入手可能
データ移動ユーティリティー ガイドおよびリファレンス	SC88-4421-01	入手可能
データ・リカバリーと高可用性 ガイドおよびリファレンス	SC88-4423-01	入手可能
データ・サーバー、データベース、およびデータベース・オブジェクトのガイド	SC88-4259-01	入手可能
データベース・セキュリティー・ガイド	SC88-4418-01	入手可能
ADO.NET および OLE DB アプリケーションの開発	SC88-4425-01	入手可能
組み込み SQL アプリケーションの開発	SC88-4426-01	入手可能
Java アプリケーションの開発	SC88-4427-01	入手可能
Perl および PHP アプリケーションの開発	SC88-4428-01	入手不可
SQL および外部ルーチンの開発	SC88-4429-01	入手可能
データベース・アプリケーション開発の基礎	GC88-4430-01	入手可能

表 23. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか
DB2 インストールおよび管理概説 (Linux および Windows 版)	GC88-4439-01	入手可能
国際化対応ガイド	SC88-4420-01	入手可能
メッセージ・リファレンス 第 1 巻	GI88-4109-00	入手不可
メッセージ・リファレンス 第 2 巻	GI88-4110-00	入手不可
マイグレーション・ガイド	GC88-4438-01	入手可能
Net Search Extender 管理およびユーザーズ・ガイド	SC88-4630-01	入手可能
パーティションおよびクラスタリングのガイド	SC88-4419-01	入手可能
Query Patroller 管理およびユーザーズ・ガイド	SC88-4611-00	入手可能
IBM データ・サーバー・クライアント機能 概説およびインストール	GC88-4441-01	入手不可
DB2 サーバー機能 概説およびインストール	GC88-4440-01	入手可能
Spatial Extender and Geodetic Data Management Feature ユーザーズ・ガイドおよびリファレンス	SC88-4629-01	入手可能
SQL リファレンス 第 1 巻	SC88-4436-01	入手可能
SQL リファレンス 第 2 巻	SC88-4437-01	入手可能
システム・モニター ガイドおよびリファレンス	SC88-4422-01	入手可能
問題判別ガイド	GI88-4108-01	入手不可
データベース・パフォーマンスのチューニング	SC88-4417-01	入手可能
Visual Explain チュートリアル	SC88-4449-00	入手不可
新機能	SC88-4445-01	入手可能
ワークロード・マネージャー ガイドおよびリファレンス	SC88-4446-01	入手可能
pureXML ガイド	SC88-4447-01	入手可能
XQuery リファレンス	SC88-4448-01	入手不可

表 24. DB2 Connect 固有の技術情報

資料名	資料番号	印刷資料が入手可能かどうか
DB2 Connect Personal Edition 概説およびインストール	GC88-4443-01	入手可能

表 24. DB2 Connect 固有の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか
DB2 Connect サーバー機能 概説およびインストール	GC88-4444-01	入手可能
DB2 Connect ユーザーズ・ガイド	SC88-4442-01	入手可能

表 25. Information Integration の技術情報

資料名	資料番号	印刷資料が入手可能かどうか
Information Integration: フェデレーテッド・システム 管理ガイド	SC88-4166-01	入手可能
Information Integration: レプリケーションおよびイベント・パブリッシングのための ASNCLP プログラム・リファレンス	SC88-4167-02	入手可能
Information Integration: フェデレーテッド・データ・ソース構成ガイド	SC88-4185-01	入手不可
Information Integration: SQL レプリケーション ガイドおよびリファレンス	SC88-4168-01	入手可能
Information Integration: レプリケーションとイベント・パブリッシング 概説	GC88-4187-01	入手可能

DB2 の印刷資料の注文方法

DB2 の印刷資料が必要な場合、オンラインで購入することができますが、すべての国および地域で購入できるわけではありません。 DB2 の印刷資料については、IBM 営業担当員にお問い合わせください。 *DB2 PDF* ドキュメンテーション DVD の一部のソフトコピー・ブックは、印刷資料では入手できないことに留意してください。例えば、「*DB2* メッセージ・リファレンス」はどちらの巻も印刷資料としては入手できません。

DB2 PDF ドキュメンテーション DVD で利用できる DB2 の印刷資料の大半は、IBM に有償で注文することができます。国または地域によっては、資料を IBM Publications Center からオンラインで注文することもできます。お客様の国または地域でオンライン注文が利用できない場合、DB2 の印刷資料については、IBM 営業担当員にお問い合わせください。 DB2 PDF ドキュメンテーション DVD に収録されている資料の中には、印刷資料として提供されていないものもあります。

注: 最新で完全な DB2 資料は、DB2 インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2luw/v9r5) で参照することができます。

DB2 の印刷資料は以下の方法で注文することができます。

- 日本 IBM 発行のマニュアルはインターネット経由でご購入いただけます。詳しくは http://www.ibm.com/shop/publications/order の「ご注文について」をご覧ください。資料の注文情報にアクセスするには、お客様の国、地域、または言語を選択してください。その後、各ロケーションにおける注文についての指示に従ってください。
- DB2 の印刷資料を IBM 営業担当員に注文するには、以下のようにします。
  1. 以下の Web サイトのいずれかから、営業担当員の連絡先情報を見つけてください。
     - IBM Directory of world wide contacts (www.ibm.com/planetwide)
     - IBM Publications Web サイト (http://www.ibm.com/shop/publications/order) 国、地域、または言語を選択し、お客様の所在地に該当する Publications ホーム・ページにアクセスしてください。このページから、「このサイトについて」のリンクにアクセスしてください。
  2. 電話をご利用の場合は、DB2 資料の注文であることをご指定ください。
  3. 担当者に、注文する資料のタイトルと資料番号をお伝えください。タイトルと資料番号は、226 ページの『DB2 テクニカル・ライブラリー (ハードコピーまたは PDF 形式)』でご確認いただけます。

コマンド行プロセッサーから SQL 状態ヘルプを表示する

DB2 は、SQL ステートメントの結果の原因になったと考えられる条件の SQLSTATE 値を戻します。 SQLSTATE ヘルプは、SQL 状態および SQL 状態クラス・コードの意味を説明します。

SQL 状態ヘルプを呼び出すには、コマンド行プロセッサーを開いて以下のように入力します。

```
? sqlstate 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。

例えば、`? 08003` を指定すると SQL 状態 08003 のヘルプが表示され、 `? 08` を指定するとクラス・コード 08 のヘルプが表示されます。

異なるバージョンの DB2 インフォメーション・センターへのアクセス

DB2 バージョン 9.5 のトピックを扱っている DB2 インフォメーション・センターの URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/です。

DB2 バージョン 9 のトピックを扱っている DB2 インフォメーション・センターの URL は http://publib.boulder.ibm.com/infocenter/db2luw/v9/です。

DB2 バージョン 8 のトピックについては、バージョン 8 のインフォメーション・センターの URL http://publib.boulder.ibm.com/infocenter/db2luw/v8/にアクセスしてください。

DB2 インフォメーション・センターでの希望する言語でのトピックの表示

DB2 インフォメーション・センターでは、ブラウザーの設定で指定した言語でのトピックの表示が試みられます。トピックがその指定言語に翻訳されていない場合は、 DB2 インフォメーション・センターでは英語でトピックが表示されます。

- Internet Explorer Web ブラウザーで、指定どおりの言語でトピックを表示するには、以下のようにします。
  1. Internet Explorer の「**ツール**」 -> 「**インターネット オプション**」 -> 「**言語...**」ボタンをクリックします。「言語の優先順位」ウィンドウがオープンします。
  2. 該当する言語が、言語リストの先頭の項目に指定されていることを確認します。
     - リストに新しい言語を追加するには、「**追加...**」ボタンをクリックします。

       注: 言語を追加しても、特定の言語でトピックを表示するのに必要なフォントがコンピューターに備えられているとはかぎりません。
     - リストの先頭に新しい言語を移動するには、その言語を選択してから、その言語が言語リストに先頭に行くまで「**上に移動**」ボタンをクリックします。
  3. ブラウザー・キャッシュを消去してから、ページを最新表示します。希望する言語で DB2 インフォメーション・センターが表示されます。
- Firefox または Mozilla Web ブラウザーの場合に、希望する言語でトピックを表示するには、以下のようにします。
  1. 「**ツール**」 -> 「**オプション**」 -> 「**詳細**」 ダイアログの「**言語**」セクションにあるボタンを選択します。「設定」ウィンドウに「言語」パネルが表示されます。
  2. 該当する言語が、言語リストの先頭の項目に指定されていることを確認します。
     - リストに新しい言語を追加するには、「**追加...**」ボタンをクリックしてから、「言語を追加」ウィンドウで言語を選択します。
     - リストの先頭に新しい言語を移動するには、その言語を選択してから、その言語が言語リストに先頭に行くまで「**上に移動**」ボタンをクリックします。
  3. ブラウザー・キャッシュを消去してから、ページを最新表示します。希望する言語で DB2 インフォメーション・センターが表示されます。

ブラウザーとオペレーティング・システムの組み合わせによっては、オペレーティング・システムの地域の設定も希望のロケールと言語に変更しなければならない場合があります。

コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの更新

DB2 インフォメーション・センターをローカルにインストールしている場合は、IBM から資料の更新を入手してインストールすることができます。

ローカルにインストールされた DB2 インフォメーション・センターを更新するには、以下のことを行う必要があります。

1. コンピューター上の DB2 インフォメーション・センターを停止し、インフォメーション・センターをスタンドアロン・モードで再始動します。インフォメーション・センターをスタンドアロン・モードで実行すると、ネットワーク上の他のユーザーがそのインフォメーション・センターにアクセスできなくなります。これで、更新を適用できるようになります。非管理者および非 root の DB2 インフォメーション・センターは常にスタンドアロン・モードで実行されます。を参照してください。
2. 「更新」機能を使用することにより、どんな更新が利用できるかを確認します。インストールする更新がある場合は、「更新」機能を使用してそれを入手およびインストールできます。

   注: ご使用の環境において、インターネットに接続されていないマシンに DB2 インフォメーション・センターの更新をインストールする必要がある場合は、インターネットに接続されていて DB2 インフォメーション・センターがインストールされているマシンを使用して、更新サイトをローカル・ファイル・システムにミラーリングする必要があります。ネットワーク上の多数のユーザーが資料の更新をインストールする場合にも、更新サイトをローカルにミラーリングして、更新サイト用のプロキシーを作成することにより、個々のユーザーが更新を実行するのに要する時間を短縮できます。
   更新パッケージが入手可能な場合、「更新」機能を使用してパッケージを入手します。ただし、「更新」機能は、スタンドアロン・モードでのみ使用できます。
3. スタンドアロンのインフォメーション・センターを停止し、コンピューター上の DB2 インフォメーション・センターを再開します。

注: Windows Vista の場合、下記のコマンドは管理者として実行する必要があります。完全な管理者特権でコマンド・プロンプトまたはグラフィカル・ツールを起動するには、ショートカットを右クリックしてから、 **「管理者として実行」**を選択します。

コンピューターまたはイントラネット・サーバーにインストール済みの DB2 インフォメーション・センターを更新するには、以下のようにします。

1. DB2 インフォメーション・センターを停止します。
   - Windows では、**「スタート」** → **「コントロール パネル」** → **「管理ツール」** → **「サービス」**をクリックします。次に、**「DB2 インフォメーション・センター」**サービスを右クリックして**「停止」**を選択します。
   - Linux では、以下のコマンドを入力します。

     ```
     /etc/init.d/db2icdv95 stop
     ```
2. インフォメーション・センターをスタンドアロン・モードで開始します。
   - Windows の場合:
     a. コマンド・ウィンドウを開きます。
     b. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは <Program Files>¥IBM¥DB2 Information Center¥Version 9.5 ディレクトリーにインストールされています (<Program Files> は「Program Files」ディレクトリーのロケーション)。

c. インストール・ディレクトリーから doc¥bin ディレクトリーにナビゲートします。

d. 次のように help_start.bat ファイルを実行します。

```
help_start.bat
```

- Linux の場合:

a. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは /opt/ibm/db2ic/V9.5 ディレクトリーにインストールされています。

b. インストール・ディレクトリーから doc/bin ディレクトリーにナビゲートします。

c. 次のように help_start スクリプトを実行します。

```
help_start
```

システムのデフォルト Web ブラウザーが起動し、スタンドアロンのインフォメーション・センターが表示されます。

3. 「**更新**」ボタン () をクリックします。インフォメーション・センターの右側のパネルで、「**更新の検索 (Find Updates)**」をクリックします。 既存の文書に対する更新のリストが表示されます。
4. インストール・プロセスを開始するには、インストールする更新をチェックして選択し、「**更新のインストール**」をクリックします。
5. インストール・プロセスが完了したら、「**完了**」をクリックします。
6. 次のようにして、スタンドアロンのインフォメーション・センターを停止します。
   - Windows の場合は、インストール・ディレクトリーの doc¥bin ディレクトリーにナビゲートしてから、次のように help_end.bat ファイルを実行します。

     ```
     help_end.bat
     ```

     注: help_end バッチ・ファイルには、help_start バッチ・ファイルを使用して開始したプロセスを安全に終了するのに必要なコマンドが含まれています。help_start.bat は、Ctrl-C や他の方法を使用して終了しないでください。
   - Linux の場合は、インストール・ディレクトリーの doc/bin ディレクトリーにナビゲートしてから、次のように help_end スクリプトを実行します。

     ```
     help_end
     ```

     注: help_end スクリプトには、help_start スクリプトを使用して開始したプロセスを安全に終了するのに必要なコマンドが含まれています。他の方法を使用して、help_start スクリプトを終了しないでください。
7. DB2 インフォメーション・センターを再開します。
   - Windows では、「**スタート**」 → 「**コントロール パネル**」 → 「**管理ツール**」 → 「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**開始**」を選択します。
   - Linux では、以下のコマンドを入力します。

     ```
     /etc/init.d/db2icdv95 start
     ```

更新された DB2 インフォメーション・センターに、更新された新しいトピックが表示されます。

DB2 チュートリアル

DB2 チュートリアルは、DB2 製品のさまざまな機能について学習するのを支援します。この演習をとおして段階的に学習することができます。

はじめに

インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2help/) から、このチュートリアルの XHTML 版を表示できます。

演習の中で、サンプル・データまたはサンプル・コードを使用する場合があります。個々のタスクの前提条件については、チュートリアルを参照してください。

DB2 チュートリアル

チュートリアルを表示するには、タイトルをクリックします。

「*pureXML ガイド* 」の『**pureXML™**』
: XML データを保管し、ネイティブ XML データ・ストアに対して基本的な操作を実行できるように、DB2 データベースをセットアップします。

「*Visual Explain チュートリアル*」の『**Visual Explain**』
: Visual Explain を使用して、パフォーマンスを向上させるために SQL ステートメントを分析し、最適化し、調整します。

DB2 トラブルシューティング情報

DB2 製品を使用する際に役立つ、トラブルシューティングおよび問題判別に関する広範囲な情報を利用できます。

DB2 ドキュメンテーション
: トラブルシューティング情報は、DB2 問題判別ガイド、または DB2 インフォメーション・センターの「サポートおよびトラブルシューティング」セクションにあります。ここには、DB2 診断ツールおよびユーティリティーを使用して、問題を切り分けて識別する方法、最も頻繁に起こる幾つかの問題に対するソリューションについての情報、および DB2 製品を使用する際に発生する可能性のある問題の解決方法についての他のアドバイスがあります。

DB2 Technical Support の Web サイト
: 現在問題が発生していて、考えられる原因とソリューションを検索したい場合は、DB2 Technical Support の Web サイトを参照してください。
Technical Support サイトには、最新の DB2 資料、TechNotes、プログラム診断依頼書 (APAR またはバグ修正)、フィックスパック、およびその他のリソースへのリンクが用意されています。この知識ベースを活用して、問題に対する有効なソリューションを探し出すことができます。

  DB2 Technical Support の Web サイト (http://www.ibm.com/software/data/db2/udb/support.html) にアクセスしてください。

ご利用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。 ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

付録 B. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-8711
東京都港区六本木 3-2-12
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書は、IBM 以外の Web サイトおよびリソースへのリンクまたは参照を含む場合があります。 IBM は、本書より参照もしくはアクセスできる、または本書からリンクされた IBM 以外の Web サイトもしくは第三者のリソースに対して一切の責任を負いません。 IBM 以外の Web サイトにリンクが張られていることにより IBM が当該 Web サイトを推奨するものではなく、またその内容、使用もしくはサイトの所有者について IBM が責任を負うことを意味するものではありません。 また、IBM は、お客様が IBM Web サイトから第三者の存在を知ることになった場合にも (もしくは、IBM Web サイトから第三者へのリンクを使用した場合にも)、お客様と第三者との間のいかなる取引に対しても一切責任を負いません。 従って、お客様は、IBM が上記の外部サイトまたはリソースの利用について責任を負うものではなく、また、外部サイトまたはリソースからアクセス可能なコンテンツ、サービス、

製品、またはその他の資料一切に対して IBM が責任を負うものではないことを承諾し、同意するものとします。 第三者により提供されるソフトウェアには、そのソフトウェアと共に提供される固有の使用条件が適用されます。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

© (*お客様の会社名*) (*西暦年*). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _*年を入れる*_. All rights reserved.

商標

以下は、International Business Machines Corporation の米国およびその他の国における商標です。

pureXML	OS/390
VisualAge	DB2 Connect
DB2 Universal Database	z/OS
Redbooks	AS/400
IBM	DB2
zSeries	AIX
System/390	ibm.com

以下は、それぞれ各社の商標または登録商標です。

- Linux は、Linus Torvalds の米国およびその他の国における商標です。
- Java およびすべての Java 関連の商標およびロゴは Sun Microsystems, Inc.の米国およびその他の国における商標です。
- UNIX は The Open Group の米国およびその他の国における登録商標です。
- Intel、Itanium は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。
- Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

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です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

［サ行］

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[ワ行]

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

[特殊文字]

Printed in Japan

SC88-4426-01

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12

Spine information:

DB2 Version 9.5 for Linux, UNIX, and Windows

組み込み SQL アプリケーションの開発