Micro Focus Net Express®

データベースアクセス

第 4 版
2003 年 5 月

このマニュアルでは、Net Express を使用して、埋め込み SQL でリレーショナルデータベースにアクセスする COBOL アプリケーションを作成する方法を説明します。

第 1 章 : はじめに

1.1 概要

Net Express には、多くの SQL プリプロセッサ (OpenESQL、DB2 ECM、および COBSQL) が組み込まれています。これらのプリプロセッサでは、COBOL プログラム内に埋め込み SQL 文を記述してリレーショナルデータベースにアクセスできます。

- OpenESQL

  OpenESQL プリプロセッサを使用する場合は、COBOL プログラム内に記述された埋め込み SQL 文から ODBC ドライバを通じてリレーショナルデータベースにアクセスできます。

- DB2 ECM

  DB2 外部チェッカーモジュール (ECM) は、Net Express が提供する、Micro Focus COBOL コンパイラとの連携を重視した新しいタイプの統合プリプロセッサです。DB2 ECM は、埋め込み SQL 文を DB2 データベースサービスへの適切な呼び出しに変換します。DB2 ECM は次の各ツールとともに使用できます。

  - IBM Software Development Kit (SDK) for Windows 95/NT Version 2.1
  - IBM DB2 for Windows 95/NT Single User Version 2.1
  - IBM Distributed Database Connection Services (DDCS) for Windows NT Version 2.3
  - IBM DB2 Universal Database Version 5.0 以降
  - IBM DB2 Connect Version 5 以降

- DB2 データベースサービスにアクセスするプログラムをコンパイルおよび実行するには、次のツールをインストールする必要があります。
  - DB2 UDB Software Development Kit (Version 6.1 以前)
  - DB2 UDB Application Development Client (Version 7.1 以降)

- COBSQL

  COBSQL は、リレーショナルデータベースのベンダが提供している COBOL プリコンパイラ向けの統合プリプロセッサです。COBSQL は次の各ツールとともに使用できます。

  - Oracle Pro*COBOL Version 1.8 以降
  - Oracle Pro*COBOL Version 8 以降
  - Informix Embedded SQL/COBOL Version 7.3 以降
  - Sybase Open Client Embedded SQL/COBOL Version 11.1 以降

すでに上記のプリコンパイラを Micro Focus COBOL の旧製品とともに使用しており、作成済みのアプリケーションを Net Express に移行する場合、または、UNIX に展開して Oracle または Sybase のリレーショナルデータベースにアクセスするアプリケーションを開発している場合には、COBSQL を使用することをお奨めします。
それ以外の場合の埋め込み SQL アプリケーションの開発には、OpenESQL を使用することをお奨めします。

注:

- Oracle Version 1.8 プリコンパイラでは、入れ子プログラムはサポートされません。
- COBSQL はオブジェクト指向 COBOL 構文 (OO COBOL) をサポートしていません。OO COBOL を使用する場合には、OpenESQL を使用してください。
- SOURCEFORMAT"FREE" 指令でコンパイルされたプログラムでは COBSQL を使用できません。プログラムが自由形式の場合は、OpenESQL を使用してください。

1.2 埋め込み SQL

各プリプロセッサは、COBOL プログラム内に記述された埋め込み SQL 文を取り出し、対応するデータベースの関数呼び出しに変換します。
COBOL プログラム内に埋め込み SQL 文を記述する際には、その前に次のキーワードを記述する必要があります。
EXEC SQL
SQL 文の後には次のキーワードを記述します。
END-EXEC
次に例を示します。

```
EXEC SQL
   SELECT au_lname INTO :lastname FROM authors
      WHERE au_id = '124-59-3864'
END-EXEC
```

埋め込み SQL 文は、必要に応じて複数の行に続けて記述できます。その際の継続に対する規則は、通常の COBOL 規則に従います。ただし、EXEC SQL と END EXEC キーワードの間に記述できるのは埋め込み SQL 文のみで、通常の COBOL コードは記述できません。
多くのデータベースソフトウェア製品には、埋め込み SQL 文に関する情報を網羅した SQL リファレンスマニュアルが付属しています。ただし、次に挙げるような基本的なデータベース操作を実行できることが前提になります。

操作	SQL 文
テーブルへのデータの追加	INSERT
テーブル内のデータの変更	UPDATE
テーブルからの行データの取得	SELECT
名前付きカーソルの作成	DECLARE CURSOR
カーソルを使用した複数行のデータの取得	OPEN、FETCH、CLOSE

構文の説明は、その使用例とともに、各埋め込み SQL 文のヘルプに記載されています。

1.2.1 大文字と小文字の扱い

プログラム内に記述した埋め込み SQL キーワードでは、大文字と小文字は区別されません。次に例を示します。

```
EXEC SQL CONNECT
exec sql connect
Exec Sql Connect
```

これらの 3 つの行は、同じ文として認識されます。
特定のデータベースの設定によって、接続名、テーブル名やカラム名などの大文字と小文字の扱いが決定されます。

テーブルやカラム名などの SQL 識別名にはハイフン (-) を使用できません。

1.2.2 OpenESQL アシスタント

Net Express には、OpenESQL アシスタントが組み込まれています。この対話型ツールを使用して、SQL SELECT、INSERT、UPDATE、DELETE、およびストアドプロシージャ文のプロトタイプを作成し、それらの文をデータベースに照らし合わせてテストできます。
詳細は、『*OpenESQL*』の章を参照してください。

1.3 アプリケーションのビルド

埋め込み SQL を含む COBOL アプリケーションの記述を終えたら、適切なコンパイラ指令を指定してコンパイルし、プリプロセッサで埋め込み SQL 文をデータベースの関数呼び出しに変換する必要があります。

- OpenESQL プリプロセッサ

  SQL コンパイラ指令を指定します。詳細は、『*OpenESQL*』の章を参照してください。

- DB2 ECM プリプロセッサ

  DB2 コンパイラ指定を指定します。詳細は、『*DB2*』の章を参照してください。

- COBSQL プリプロセッサ

  PREPROCESS"COBSQL" コンパイラ指令を指定します。詳細は、『*COBSQL*』の章を参照してください。

1.3.1 インターネットアプリケーションウィザード

Net Express には、インターネットアプリケーションウィザードが組み込まれています。このウィザードを使用して、リレーショナルデータベースにアクセスする完全な Web アプリケーションを生成します。動作する SQL アプリケーションを簡単に作成できます。
詳細は、『***分散コンピューティング***』を参照してください。

1.4 複数のプログラムモジュール

複数の埋め込み SQL ソースファイルは、個別にコンパイルして同じ実行可能ファイルにリンクすれば、実行時に同じデータベース接続を共有できます。独立した動的リンクライブラリ (.dll ファイル) にコンパイルされるプログラムでも同様です。同じプロセス内の後続のプログラムモジュールが CONNECT 文を処理しない場合には、それらのモジュールは、CONNECT 文をインクルードしたモジュールと同じデータベース接続を共有します。
次の表に、各種 SQL プリプロセッサで複数のプログラムモジュールを使用する方法に関するガイドラインを示します。

SQL プリプロセッサ	ガイドライン

OpenESQL	個別にコンパイルされた複数のモジュールを含むプログラムでは、SQL コンパイラ指令の INIT オプションを使用して 1 つのモジュールのみをコンパイルしてください。プログラム内の他のすべてのモジュールは最初の自動接続を共有するか、または、CONNECT 文を使用して明示的に接続します。
OpenESQL、DB2	文の名前は、特定のプログラムモジュール (コンパイル単位) に固有です。つまり、1 つのモジュールで定義された文を別のモジュールで実行できません。
OpenESQL、DB2	カーソル名は、アプリケーション内で一意である必要があります。
COBSQL	INIT 指令を複数回指定すると、Net Express では 2 回目以降の指令が無視されます。

第 2 章：ホスト変数

ホスト変数とは、COBOL プログラム内で定義されるデータ項目です。ホスト変数は、データベースとの値の受け渡しに使用されます。ホスト変数は、COBOL プログラムのファイル節、作業場所節、局所記憶節、または連絡節で定義し、1 〜 48 の範囲の任意のレベル番号を割り当てることができます。レベル-49 は VARCHAR データ項目に予約されています。
埋め込み SQL 文内でホスト変数名を記述する場合は、データ項目名の先頭にコロン (:) を付けてください。コンパイラはこのコロンによって、ホスト変数と同じ名前のテーブルまたはカラムを区別できます。
ホスト変数は、使用方法によって次の 2 つの種類に分けられます。

- **入力ホスト変数**

  COBOL プログラムからデータベースに転送されるデータを指定します。

- **出力ホスト変数**

  データベースから COBOL プログラムに返されるデータを格納します。

たとえば、次の文では、`:book-id` は、検索する本の ID を表す入力ホスト変数で、`:book-title` は検索結果を返す出力ホスト変数です。

```
EXEC SQL
   SELECT title INTO :book-title FROM titles
      WHERE title_id=:book-id
END-EXEC
```

2.1 ホスト変数の宣言

埋め込み SQL 文でホスト変数を使用するには、その前にホスト変数を宣言する必要があります。ホスト変数は、埋め込み SQL 文の BEGIN DECLARE SECTION と END DECLARE SECTION で囲んで宣言します。次に例を示します。

```
EXEC SQL
   BEGIN DECLARE SECTION
END-EXEC
01 id             pic x(4).
01 name           pic x(30).
EXEC SQL
   END DECLARE SECTION
END-EXEC

display "あなたの ID を入力してください : "
accept id.

* 次の文は、ホスト変数「id」に格納された
* 内容と同じ社員番号をもつ社員の名前を
* 取得し、ホスト変数
* 「name」に返します。

EXEC SQL
   SELECT emp_name INTO :name FROM employees
      WHERE emp_id=:id
```

```
END-EXEC
display "ようこそ " name.
```

注:

- データ項目の集団を単一のホスト変数として使用できます。
- OpenESQL は、文字型のホスト変数から後部空白文字を取り除きます。ホスト変数が空白文字のみで構成される場合、長さ 0 の文字列を NULL として扱うサーバもあるため、OpenESQL は先頭の空白文字を残します。

2.1.1 OpenESQL と DB2 プリプロセッサ

BEGIN DECLARE SECTION および END DECLARE SECTION を使用して宣言されていなくても、データ項目をホスト変数として使用できます。
ホスト変数の宣言では、次の点に留意してください。

- ホスト変数名は COBOL におけるデータ項目の命名規則に準拠する必要があります。
- ホスト変数の宣言は、COBOL データ項目を宣言できる部分であればどこにでも記述できます。
- ホスト変数名にはアンダスコア (_) は使用できません。

2.2 ホスト配列

配列とは、1 つの変数名に関連付けられた複数のデータ項目の集まりです。複数のホスト変数を配列 (ホスト配列) として定義すると、これらの変数を単一の SQL 文で処理できます。
ホスト配列は、INSERT、UPDATE、および DELETE 文の入力変数として使用したり、SELECT 文や FETCH 文の INFO 句で出力変数として使用したりできます。配列を SELECT 文、FETCH 文、DELETE 文、INSERT 文、および UPDATE 文で使用して、大量のデータを扱うことができます。
ホスト配列は、単一のホスト変数と同様に BEGIN DECLARE SECTION と END DECLARE SECTION を使用して宣言します。ただし、配列の次元を OCCURS 句で指定する必要があります。次に例を示します。

```
EXEC SQL
   BEGIN DECLARE SECTION
END-EXEC
01 AUTH-REC-TABLES
   05 Auth-id       OCCURS 25 TIMES PIC X(12).
   05 Auth-Lname    OCCURS 25 TIMES PIC X(40).
EXEC SQL
   END DECLARE SECTION
END-EXEC.
.
.
.
EXEC SQL
   CONNECT USERID 'user' IDENTIFIED BY 'pwd' USING 'db_alias'
END-EXEC
EXEC SQL
   SELECT au-id, au-lname
      INTO :Auth-id, :Auth-Lname FROM authors
END-EXEC
display sqlerrd(3)
```

この例では、SELECT 文により配列のサイズとして 25 行分が戻されます。SELECT 文が 25 行を超える行を戻すことができる場合には、25 行が戻され、それ以上の行が取り出し可能であるが戻せなかったことを SQLCODE で示します。
SELECT 文は、選択する行の最大数が分かっている場合に使用してください。戻される行数が不明の場合は、FETCH 文を使用してください。配列の使用では、データをバッチで取り込むことができます。これは、情報のスクロールリストを作成する場合に便利です。
単一 SQL 文内で複数のホスト配列を使用する場合は、各配列の次元を同一にする必要があります。

注:
OpenESQL

- 単一 SQL 文内で、ホスト配列と通常のホスト変数は併用できません。ホスト配列を使用する場合には、同一 SQL 文内のホスト変数をすべてホスト配列にする必要があります。
- ホスト配列内のホスト変数のオカレンス数は、すべて同一数で定義する必要があります。1 つの変数が 25 オカレンスをもつ場合は、そのホスト配列内のすべての変数が 25 オカレンスをもつ必要があります。

COBSQL

- ホスト配列に関するこれらの情報は、Oracle データベースを使用している場合そのまま適用されます。ただし、Sybase データベースを使用している場合、ホスト変数は、SELECT 文または FETCH 文の出力変数としてのみ使用できます。
- Oracle と Sybase の両方とも、SELECT 文の WHERE 句で通常のホスト変数を使用できます。この場合のみに、ホスト配列と通常のホスト変数を併用できます。

2.2.1 FOR 句

デフォルトでは、配列全体を SQL 文で処理しますが、必要に応じて FOR 句を使用し、処理する配列の要素数を制限できます。これは、UPDATE、INSERT、および DELETE 文で配列の一部のみを処理する場合に特に便利です。
FOR 句では整数型のホスト変数を使用する必要があります。次に例を示します。

```
EXEC SQL
   BEGIN DECLARE SECTION
END-EXEC
01 AUTH-REC-TABLES
   05 Auth-id       OCCURS 25 TIMES PIC X(12).
   05 Auth-Lname    OCCURS 25 TIMES PIC X(40).
01 maxitems         PIC S9(4) COMP-5 VALUE 10.
EXEC SQL
   END DECLARE SECTION
END-EXEC.
.
.
.
EXEC SQL
   CONNECT USERID 'user' IDENTIFIED BY 'pwd' USING 'db_alias'
END-EXEC
EXEC SQL
   FOR :maxitems
      UPDATE   authors
```

```
        SET   au_lname = :Auth_Lname
        WHERE au_id  = :Auth_id
END-EXEC
display sqlerrd(3)
```

この例では、UPDATE 文により 10 行 (:maxitems の値) が変更されます。
処理される配列の要素数は、ホスト配列の次元数と FOR 句の変数を比較して決定されます。この場合は、最も小さい値が使用されます。
FOR 句の変数の値が 0 以下の場合は、行は処理されません。

注:
COBSQL プリプロセッサ
COBSQL を使用している場合には、FOR 句に関するこの情報は、Oracle データベースを使用している場合のみに適用されます。Sybase データベースや Informix データベースを使用している場合には適用されません。

2.2.2 処理済み行数の確認

SQLDA の SQLERRD の第 3 要素 (SQLERRD(3)) には、INSERT、UPDATE、DELETE、および SELECT INTO の各文で処理された行数が記録されます。
OpenESQL プリプロセッサ
FETCH 文の場合には、SQLERRD(3) は必ず直前の FETCH 文で取り込まれた行数が格納されます。
COBSQL および DB2 プリプロセッサ
FETCH 文の場合は、SQLERRD(3) に処理済み行の累計数が記録されます。

2.2.2.1 DB2 プリプロセッサ

SQLERRD(3) には、次の内容が格納されます。

- PREPARE が呼び出されて成功した場合は、戻される予想行数。
- INSERT、UPDATE、および DELETE の後には、実際に影響された行数。
- 複合 SQL が呼び出された場合は、成功した副文の数。
- CONNECT が呼び出されたときは、データベースが更新可能な場合は 1、データベースが読み取り専用の場合は 2。
- ホスト配列の処理時にエラーが発生した場合は、処理が成功した最後の行。

SQLERRD(4) には、次の内容が格納されます。

- PREPARE 呼び出されて成功した場合は、文の処理に必要なリソースの相対コスト評価。
- 複合 SQL が呼び出された場合は、成功した副文の数。
- CONNECT が呼び出されたときは、下位レベルのクライアントからの 1 フェーズコミットの場合は 0、1 フェーズコミットの場合は 1、1 フェーズ、読み取り専用コミットの場合は 2、2 フェーズコミットの場合は 3。

SQLERRD(5) には、次の内容が格納されます。

- 次の両方の結果により削除、挿入、または更新された総行数。
    - 削除操作が成功した後の制約の強制。
    - 有効化されたトリガからトリガされた SQL 文の処理。

- 複合 SQL が呼び出された場合は、副文すべての行数の累計値。場合によっては、エラーが発生すると、このフィールドに負数値が含まれます。これは内部エラーポインタです。
- CONNECT が呼び出された場合は、サーバ認証の場合は認証型値 0、クライアント認証の場合は 1、DB2 コネクト使用の認証の場合は 2、DCE セキュリティサービス認証の場合は 3、不特定の認証には 255。

2.3 インジケータ変数

埋め込み SQL では、インジケータ変数を使用してデータベースに NULL 値を保存したり、データベースから NULL 値を取り込むことができます。インジケータ変数は、常に次のように定義されます。
`pic S9(4) comp-5.`

2.3.1 NULL 値

COBOL とは異なり、SQL では NULL 値を格納できる変数をサポートしています。NULL 値はエントリが存在しないことを意味し、通常は値が不明または未定義です。NULL 値を使用する場合は、数字カラムのゼロや文字カラムの空白文字などの意図的なエントリを、不明なエントリや適用不能なエントリと区別できます。たとえば、価格カラムから取り込んだ値が NULL の場合でも、対応する品目は無料ではありません。価格は無効または未設定です。
また、ホスト変数と対応するインジケータ変数は、両方で 1 つの SQL 値を示します。これらの変数は、どちらも先頭にコロン (:) を付けて記述します。ホスト変数が NULL の場合は、対応するインジケータ変数の値は -1 になります。ホスト変数が NULL 以外の値であれば、インジケータ変数は -1 以外の値になります。
埋め込み SQL 文では、インジケータ変数は対応するホスト変数の直後に記述してください。たとえば、次の例では、埋め込み UPDATE 文内のホスト変数 (`saleprice`) の直後に、対応するインジケータ変数 (`saleprice-null:`) が記述されています。

```
EXEC SQL
   UPDATE closeoutsale
      SET temp_price = :saleprice:saleprice-null,
           listprice = :oldprice
END-EXEC
```

この例で、`saleprice-null` が -1 の値をもつと、UPDATE 文が実行されたときに、次の文として読み込まれます。

```
EXEC SQL
   UPDATE closeoutsale
      SET temp_price = null, listprice = :oldprice
END-EXEC
```

検索条件にはインジケータ変数は使用できません。NULL 値の検索をするには、`is null` 構成を使用します。たとえば、次のように使用します。

```
     if saleprice-null equal -1
         EXEC SQL
             DELETE FROM closeoutsale
                    WHERE temp_price is null
         END-EXEC
     else
         EXEC SQL
             DELETE FROM closeoutsale
                    WHERE temp_price = :saleprice
         END-EXEC
     end-if
```

2.3.2 データの切り捨て

データがデータベースからホスト変数に取り込まれたときに、切り捨てが実行される場合は、インジケータ変数を別の用途で使用できます。ホスト変数がデータベースから返されるデータを保持できるほど大きくない場合には、SQLCA データ構造体に警告フラグ (sqlwarn1) が設定され、インジケータ変数はデータベース内のデータのサイズに設定されます。

2.3.3 インジケータ配列

インジケータ変数と同じ方法で、インジケータ配列を使用します。つまり、次のようなことです。

- NULL 値の割り当て
- NULL 値の検出
- データの切り捨ての検出

次の例では、インジケータ配列を -1 に設定して、カラムに NULL 値を挿入します。

```
01 ix                        PIC 99 COMP-5.
.
.
.
EXEC SQL
   BEGIN DECLARE SECTION
END-EXEC
01 sales-id       OCCURS 25 TIMES PIC X(12).
01 sales-name     OCCURS 25 TIMES PIC X(40).
01 sales-comm     OCCURS 25 TIMES PIC S9(9) COMP-5.
01 ind-comm       OCCURS 25 TIMES PIC S9(4) COMP-5.
EXEC SQL
   END DECLARE SECTION
END-EXEC.
.
.
.
PERFORM VARYING iX FROM 1 BY 1 UNTIL ix > 25
  MOVE -1 TO ind-comm (ix)
END-PERFORM.
.
.
.
EXEC SQL
   INSERT INTO SALES (ID, NAME, COMM)
      VALUES (:sales_id, :sales_name, :sales_comm:ind-comm)
END-EXEC
```

COBSQL

COBSQL を使用している場合には、インジケータ配列に関するこの情報は、Oracle データベースを使用している場合のみに適用されます。Sybase データベースを使用している場合は、適用されません。

第 3 章 : データ型

SQL データ型は、COBOL で使用されるデータ形式と異なります。
SQL には一連の標準データ型がありますが、実際の実装状況はデーベースによって異なり、これらのデータ型をすべて実装するデータベースはほとんどありません。

3.1 データ型の変換

COBOL プログラム内では、ホスト変数は COBOL プログラム変数としてのみでなく、SQL データベース変数としても機能するため、プリプロセッサにより COBOL データ形式を適切な SQL データ型に変換したり、マップしたりする必要があります。つまり、プリプロセッサが COBOL データ形式を正しい SQL データ型にマップするように、ホスト変数を正しい COBOL PICTURE 句で宣言する必要があります。これを行うには、接続するデータソースで使用される SQL データ型を把握しておくことが必要です。以降では、さまざまな SQL データ型と、直接それらにマップするホスト変数の宣言方法について説明します。

3.1.1 COBSQL プリプロセッサ

COBSQL で Sybase、Informix、または Oracle を使用している場合は、データベースエンジンで変換の一種を行って、COBOL データ形式からデータベースのデータ型にデータを変換できます。通常、数字または整数データ形式のホスト変数は、次のように定義します。
`PIC S9(..)..COMP..`
文字またはテキストデータ形式は、次のように定義します。
`PIC X(...).`
Oracle と Sybase では、データベースデータ型を特定のホスト変数に定義できます。これは、より複雑なデータ型が使用される場合に便利です。

3.1.1.1 Oracle

Oracle では、次のように記述します。

```
EXEC SQL
BEGIN DECLARE SECTION
END-EXEC.
*
* データ項目を Oracle データ型の DISPLAY として定義します。
*
01 emp-comm pic s9(6)v99 DISPLAY SIGN LEADING SEPARATE
*
EXEC SQL
   VAR emp-comm IS DISPLAY(8,2)
END-EXEC.
EXEC SQL
   END DECLARE SECTION
END-EXEC.
```

3.1.1.2 Sybase

Sybase では、次のように記述します。

```
EXEC SQL
   BEGIN DECLARE SECTION
```

```
END-EXEC.
*
* データ項目を Sybase 固有のデータ型として定義します。
*
01 money-item CS-MONEY.
*
EXEC SQL
   END DECLARE SECTION
END-EXEC.
```

ホスト変数のデータベース型定義の詳細については、各データベースベンダが提供している COBOL プリコンパイラマニュアルを参照してください。

3.1.1.3 Informix

Informix では、さまざまなデータ型を操作するために呼び出すことができるシステムルーチンが多数提供されています。これらのルーチンの詳細については、『*Programming with INFORMIX-ESQL/COBOL*』マニュアルを参照してください。

3.2 整数データ型

3.2.1 TINYINT

TINYINT は、SQL の 1 バイトの整数データ型です。COBOL では次のように宣言されます。

```
PIC S9(4) COMP-5.
```

DB2

DB2 では TINYINT データ型をサポートしていません。

COBSQL

Sybase では、TINYINT ホスト変数の使用をサポートしています。Sybase では次のように定義されます。

```
03 tinyint1     PIC S9(2) COMP-5.
03 tinyint2     PIC S9(2) COMP.
03 tinyint3     PIC S9(2) BINARY.
```

これらは、Sybase データ型 TINYINT にマップされます。

3.2.2 SMALLINT

SMALLINT は、SQL の 2 バイトの整数データ型です。COBOL では、BINARY、COMP、COMP-X、COMP-5、または COMP-4 の用途で宣言されます。
たとえば、次の定義はすべて、ホスト変数が直接 SMALLINT データ型にマップされます。

```
03 shortint1      PIC S9(4)  COMP.
03 shortint2      PIC S9(4)  BINARY.
03 shortint3      PIC X(2)   COMP-5.
03 shortint4      PIC S9(4)  COMP-4.
03 shortint5      PIC 9(4)   USAGE DISPLAY.
03 shortint6      PIC S9(4)  USAGE DISPLAY.
```

OpenESQL

- OpenESQL では現在、符号付き SMALLINT をサポートしていますが、符号なし SMALLINT はサポートしていません。
- 最も効率よくアクセスするためには、SMALLINT を COMP-5 として宣言する必要があります。

COBSQL - Oracle
Oracle では、ホスト変数を shortint1、shortint2、または次のように定義するのが最良の方法です。
```
03 shortint7     PIC S9(4) COMP-5.
```
これらは、Oracle データ型 NUMBER(38) にマップされます。
COBSQL - Sybase
Sybase では、shortint3 以外はすべて受け入れられます。使用できる方法は、次のとおりです。
```
03 shortint7     PIC S9(4) COMP-5.
```
これらは、Sybase データ型 SMALLINT にマップされます。
COBSQL - Informix
Informix では、ホスト変数を shortint1、shortint2、または次のように定義するのが最良の方法です。
```
03 shortint7     PIC S9(4) COMP-5.
```
これらは、Informix データ型 SMALLINT にマップされます。

3.2.3 INT

INT は、SQL の 4 バイトの整数データ型です。COBOL では、BINARY、COMP、COMP-X、COMP-5、または COMP-4 の用途で宣言されます。
次の定義は、ホスト変数を直接 INT データ型にマップされます。
```
03 longint1      PIC S9(9)  COMP.
03 longint2      PIC S9(9)  COMP-5.
03 longint3      PIC X(4)   COMP-5.
03 longint4      PIC X(4)   COMP-X.
03 longint5      PIC 9(9)   USAGE DISPLAY.
03 longint6      PIC S9(9)  USAGE DISPLAY.
```
OpenESQL

- 現在 OpenESQL は、符号付き INT をサポートしていますが、符号なし INT はサポートしていません。
- 最も効率よくアクセスするためには、INT を COMP-5 として宣言する必要があります。

COBSQL - Oracle
Oracle では、ホスト変数を longint1、longint2、または次のように定義するのが最良の方法です。
```
03 longint7     PIC S9(9) COMP-5.
```
これらは、Oracle データ型 NUMBER(38) にマップされます。
COBSQL - Sybase
Sybase では、longint3 以外はすべて受け入れられます。使用できる方法は、次のとおりです。
```
03 longint7     PIC S9(9) COMP-5.
```
これらは、Sybase データ型 INT にマップされます。
COBSQL - Informix
Informix では、ホスト変数を longint1、longint2、または次のように定義するのが最良の方法です。
```
03 longint7     PIC S9(9) COMP-5.
```
これらは、Informix データ型 INT にマップされます。

3.2.4 BIGINT

BIGINT は、SQL の 8 バイトの整数データ型です。COBOL では次のように宣言されます。
```
PIC S9(18) COMP-3.
```
OpenESQL
OpenESQL では、ホスト変数として使用される COBOL データ項目に S9(18) という最大サイズをサ

ポートして、SQL データ型 BIGINT からマップされた値を保持します。ただし、BIGINT データ型は、PIC S9(18) データ項目の最大値以上の値を格納できます。そのため、データ切り捨てに対するコード検査が必要です。
DB2
BIGINT データ型は、DB2 UDB V6.1 以降でサポートされます。
COBSQL
Oracle、Informix、および Sybase では、BIGINT をサポートしていません。

3.3 文字データ型

3.3.1 CHAR

固定長文字列 (CHAR) は、ドライバによって最大長が定義された SQL データ型です。COBOL では PIC X(*n*) と宣言します。*n* は、1 から最大長までの整数です。
たとえば、次のように記述します。

```
03 char-field1      pic x(5).
03 char-field2      pic x(254).
```

COBSQL
これは、Oracle データ型 CHAR(*n*)、Sybase データ型 CHAR(*n*)、および Informix データ型 CHAR(*n*) にマップします。Oracle または Sybase の場合に、サポートされる固定長文字列の最大長は 255 バイトです。Informix の場合に、サポートされる固定長文字列の最大長は 32KB です。
DB2
これは、DB2 データ型 CHAR にマップされます。サポートされる固定長文字列の最大長は 254 バイトです。254 バイトを超える長さの文字列が必要な場合は、VARCHAR フィールドを使用してください。

3.3.2 VARCHAR

3.3.2.1 OpenESQL

OpenESQL を使用する場合は、長さフィールドを COMP-5 として宣言する必要があります。

3.3.2.2 OpenESQL および DB2

可変長文字列 (VARCHAR) は、SQL データ型です。COBOL では、次の 2 通りで宣言できます。

- 固定長文字列 (PIC X(*n*)).
- レベル-49 の基本項目 2 つのみを含む集団項目。最初の項目は、2 バイトのフィールドで有効な文字列の長さを示す COMP または COMP-5 の用途で宣言します。もう 1 つの項目は PIC X(*n*) データ形式で宣言し、実際のデータを格納します。*n* は、整数です。

  次に宣言例を示します。

```
03 varchar1.
   49 varchar1-len        pic 9(4) comp-5.
   49 varchar1-data       pic x(200).
03 Longvarchar1.
   49 Longvarchar1-len    pic 9(4) comp.
   49 Longvarchar1-data   pic x(30000).
```

SQL 文では、集団名を参照する必要があります。

SQL の CHAR、VARCHAR、または LONG BARCHAR データ型にコピーされたデータが、これらのデータ型に定義された長さを超える場合には、データが切り捨てられ、SQLCA データ構造体の SQLWARN1 フラグが設定されます。また、定義されたデータ長より短い文字列には、受け取った CHAR データ型に空白文字が付加されます。

3.3.2.3 COBSQL - Oracle

Oracle では、ホスト変数は Oracle キーワード VARYING を使用して定義されます。次に使用例を示します。

```
EXEC SQL
   BEGIN DECLARE SECTION
END-EXEC.
01 USERNAME     PIC X(20) VARYING.
EXEC SQL
   END DECLARE SECTION
END-EXEC.
```

Oracle は、データ項目 USERNAME を次のような集団項目に展開します。

```
01 USERNAME
   02 USERNAME-LEN     PIC S9(4) COMP-5.
   02 USERNAME-ARR     PIC X(20).
```

COBOL コード内では、USERNAME-LEN または USERNAME-ARR のどちらかを参照する必要がありますが、SQL 文内では集団名 USERNAME を使用する必要があります。次に例を示します。

```
move "SCOTT" to USERNAME-ARR.
move 5 to USERNAME-LEN.
exec sql
   connect :USERNAME identified by :pwd
   using :db-alias
end-exec.
```

これは Oracle データ型 VARCHAR(*n*) または VARCHAR2(*n*) にマップされます。非常に長い文字項目については、Oracle はデータ型 LONG を提供しています。

3.3.2.4 COBSQL - Sybase

Sybase では、ホスト変数を PIC X(*n*) PICTURE 句を使用して宣言する必要があります。これは、Sybase プリコンパイラが、VARCHAR SQL データ型を処理する集団項目の使用をサポートしないためです。
これらは、Sybase データ型 VARCHAR(*n*) にマップされます。

3.3.2.5 COBSQL - Informix

Informix では、ホスト変数を PIC X(*n*) PICTURE 句を使用して宣言する必要があります。これは、Informix プリコンパイラが、VARCHAR SQL データ型を処理する集団項目の使用をサポートしていないためです。
これらは、Informix データ型 VARCHAR(*n*) にマップされます。VARCHAR フィールドの最大長は、使用している Informix のバージョンによって異なります。VARCHAR データ項目の詳細については、『*Informix SQL ガイド*』のマニュアルを参照してください。

3.4 概数データ型

32 ビット SQL 浮動小数点データ型の REAL は、COBOL では COMP-1 として宣言します。
64 ビット SQL 浮動小数点データ型の FLOAT と DOUBLE は、COBOL では COMP-2 として宣言します。
次に宣言例を示します。

```
01 float1      usage comp-2.
```

OpenESQL
OpenESQL では埋め込み SQL の単精度浮動小数点数がサポートされないため、32 ビットおよび 64 ビットの浮動小数点データ型は COMP-2 COBOL データ項目にマップされます。
DB2

- DB2 ユニバーサルデータベースでは、単精度浮動小数点数 (REAL) を COMP-1 として、倍精度浮動小数点数 (FLOAT または DOUBLE) を COMP-2 としてサポートします。
- DB2 バージョン 2.1 では、倍精度浮動小数点数 (FLOAT または DOUBLE) のみ COMP-2 としてサポートします。

COBSQL - Oracle
Oracle は、COMP-1 データ項目および COMP-2 データ項目の使用をサポートしています。これらはどちらも、Oracle データ型 NUMBER にマップされます。
COBSQL - Sybase
Sybase は、COMP-1 データ項目および COMP-2 データ項目の使用をサポートしています。COMP-1 データ項目は、Sybase データ型 REAL にマップされます。COMP-2 データ項目は、Sybase データ型 FLOAT にマップされます。
COBSQL - Informix
Informix は COMP-1 データ項目と COMP-2 データ項目のどちらもサポートしていません。Informix は、COBOL の固定数字データ項目 `PIC S9(m)V9(n)` のみサポートしています。Informix では、FLOAT カラムと SMALLFLOAT SQL カラムがこの形式に変換されます。

3.5 真数データ型

真数データ型 DECIMAL および NUMERIC には、ドライバで指定された精度と位取りで値を格納できます。
COBOL では、これらは COMP-3、PACKED-DECIMAL、または NUMERIC USAGE DISPLAY として宣言されます。
次に宣言例を示します。

```
03 packed1      pic s9(8)v9(10) usage comp-3.
03 packed2      pic s9(8)v9(10) usage display.
```

COBSQL - Oracle
Oracle では、これらはデータ型 NUMBER(*p*,*s*) にマップされます。Sybase では、NUMBER(*p*,*s*) または DECIMAL(*p*,*s*) にマップされます。Informix では、DECIMAL(*p*,*s*) または MONEY(*p*,*s*) にマップされます。

- NUMERIC データ型と DECIMAL データ型の相違については、『*Sybase Transact-SQL ユーザーズ・ガイド*』の『*データ型の作成と使用方法*』の章を参照してください。
- DECIMAL データ型と MONEY データ型の相違については、『*Informix SQL ガイド*』の『*データ型*』の章を参照してください。

3.6 日時データ型

COBOL には、日付データや時刻データ専用のデータ形式はありません。そのため、SQL の日付カラムや時刻カラムは文字列に変換されます。
SQL タイムスタンプ値に対して COBOL で出力するホスト変数を PIC X(*n*) と定義した場合には、日付と時刻は *yyyy-mm-dd hh:mm:ss.ff* の形式で指定されます。この場合には、*n* は 19 以上の整数です。また、小数部の桁数はドライバで指定されます。
たとえば、次のようになります。
1994-05-24 12:34:00.000

3.6.1 OpenESQL

OpenESQL は、どのリレーションナルデータベースにもアクセスでき、各データベースには日付や時刻を指定するさまざまな方法があるため、通常は、入力ホスト変数で日付や時刻を指定します。この方法を使用する場合は、プログラムのコンパイル時に SQL 指令で DETECTDATE オプションを使用する必要があります。

- 日付を指定するには、日付を {**d***yyyy-mm-dd*} の形式でホスト変数に転記します。
- 時刻を指定するには、時刻を {**t***hh:mm;ss*} の形式でホスト変数に転記します。
- 日付と時刻を指定するには、日付と時刻を {**ts***yyyy-mm-dd hh:mm:ss*} の形式でホスト変数に転記します。

たとえば、次のように記述します。

```
$set sql(dbman=odbc, detectdate)

 01 Hire-Date pic x(26).

 move "{d'1965-11-02'} to Hire-Date
 exec sql
     insert into emp (HireDate) values (:Hire-Date)
 end-exec
```

3.6.1.1 DB2

DB2 では、TIMESTAMP データ型の最大長は 26 文字です。

3.6.2 COBSQL

3.6.2.1 Oracle

Oracle データ項目には一意なデータ定義があり、これらのデータ項目を COBOL プログラム内で使用したときに、日付、時刻および日時フィールドを変換する関数があります。これらの関数は、次のとおりです。

- TO_CHAR

  Oracle の日付形式を文字列に変換します。

- TO_DATE

  文字列を Oracle の日付に変換します。

どちらの関数も変換する項目を引数にとり、その後にデータ項目に適用される日付、時刻、または日時マスクが続きます。次に例を示します。

```
exec sql
   select ename, TO_CHAR(hiredate, 'DD-MM-YYYY')
      from emp
      into :ename, :hiredate
      where empno = :empno
end-exec.

exec sql
  insert into emp (ename, TO_DATE(hiredate, 'DD-MM-YYYY'))
          values   (:ename, :hiredate)
end-exec.
```

これは、Oracle データ型 DATE にマップされます。DATE データ型の詳細については、Oracle の『*SQL 言語リファレンスマニュアル*』を参照してください。このマニュアルでは、Oracle SQL 文内でのこれらの関数の使用方法について詳しく説明しています。

3.6.2.2 Sybase

Sybase には、データ型の形式を変換する、convert という名前の関数があります。前述の Oracle 例を使用する場合には、SQL 構文は次のようになります。

```
exec sql
   select ename, convert(varchar(12) hiredate, 105)
      from emp
      into :ename, :hiredate
      where empno = :empno
end-exec.

exec sql
   insert into emp (ename, hiredate)
          values   (:ename, convert(datetime :hiredate, 105)
end-exec.
```

これは、Sybase データ型 SMALLDATETIME または DATETIME にマップされます。
SMALLDATETIME データ型と DATETIME データ型の相違については、『*Sybase Transact-SQL ユーザーズ・ガイド*』の『*データ型の作成と使用方法*』の章を参照してください。
Sybase の convert 関数の詳細については、Sybase の『*SQL Server Reference Manual: Volume 1 Commands, Functions and Topics*』を参照してください。

3.6.2.3 Informix

Informix では、日付はユリウス形式または *mm/dd/yyyy* 形式のどちらかを要求します。

- ユリウス日付を使用する場合は、フィールドを `PIC S9(9) COMP` として定義してください。
- *mm/dd/yyyy* 形式で日付を表す場合は、次のようにします。
    - COBOL フィールドを `PIC X(10)` として定義します。
    - DATE_TYPE 関数を使用します。

Informix に日付を渡す方法については、『*INFORMIX-ESQL/COBOL Programmer's Manual*』を参照してください。

3.7 バイナリデータ型

3.7.1 OpenESQL

SQL の BINARY、VARBINARY、および IMAGE データは、COBOL では PIC X (*n*) フィールドとして表されます。データの変換は実行されません。データベースからデータを取り込むときに、データのサイズが格納先の COBOL フィールドよりも大きい場合は、フィールドに格納できない部分のデータは切り捨てられ、SQLCA データ構造体の SQLWARN1 フィールドに「W」が設定されます。また、データ長が COBOL フィールドよりも短い場合は、フィールドの空き部分に NULL 文字 (x"00") が付加されます。BINARY、VARBINARY、または LONG VARBINARY カラムにデータを挿入するには、動的 SQL 文を使用します。

3.7.1.1 DB2

DB2 では、BINARY を表すには CHAR FOR BIT DATA、VARBINARY を表すには VARCHAR(*n*) FOR BIT DATA、LONG VARBINARY を表すには LONG VARCHAR FOR BIT DATA を使用します。IBM ODBC ドライバを使用してる場合は、IBM 互換データ型のかわりに、BINARY、VARBINARY、および LONG VARBINARY が戻されます。IMAGE データ型は、BLOB で表されます。DB2 では、非常に大きいカラム (最大 2GB) を定義するために、LOB (文字型ラージオブジェクト、バイナリ型ラージオブジェクトまたはグラフィック型ラージオブジェクト) を使用します。これらのデータ型には静的 SQL を使用できます。

3.7.2 COBSQL

3.7.2.1 Oracle

Oracle では、バイナリデータをサポートしています。Oracle でのバイナリデータと文字データの相違は、文字データには文字符号系変換が行われますが、バイナリデータには何も行われないという点です。これらの Oracle データ型は RAW と LONG RAW の 2 つです。RAW および LONG RAW の使用には制約があります。詳細については、Oracle のマニュアルを参照してください。

3.7.2.2 Sybase

Sybase には、BINARY、VARBINARY、および IMAGE の 3 つのバイナリデータ型があります。IMAGE は、複雑なデータ型であるため、ホスト変数は CS-IMAGE として定義できます。たとえば、次のように記述します。

```
EXEC SQL
   BEGIN DECLARE SECTION
END-EXEC.
*
* データ項目を Sybase 固有のデータ型として定義します。
*
01 image-item     CS-IMAGE.
*
EXEC SQL
   END DECLARE SECTION
END-EXEC.
```

注：Sybase データ型 BINARY、VARBINARY、および IMAGE の使用については、『*Sybase Transact-SQL ユーザーズ・ガイド*』の『*データ型の作成と使用方法*』の章を参照してください。

3.7.2.3 Informix

Informix では、TEXT と BYTE の 2 種類のバイナリデータ項目がサポートされます。これらのデータ型には実際のデータは格納されません。これらは、ファイル名です。このため、COBOL の対応項目は PIC X(*n*) になります。
TEXT および BYTE データ項目の詳細については、『*Informix SQL ガイド*』を参照してください。

3.8 OpenESQL SQL TYPE

OpenESQL を使用しない場合や、他の ESQL プリプロセッサとの互換性を維持するには、ここを飛ばしてもかまいません。それ以外の場合は、ここで説明する SQL TYPE を使用することをお奨めします。日付 / 時刻のデータやバイナリデータに関連する SQL データを処理するときに、通常の COBOL ホスト変数を使用すると複雑になったり、これまでの方法で可変長文字列のデータを処理すると問題になったりする可能性があることが確認されています。このため、OpenESQL を拡張して、SQL TYPE 関数により、SQL テーブルに格納されるデータ型により密接に影響を与えるホスト変数をより簡単に宣言できるようにしました。これにより、動的 SQL 構文よりも静的 SQL 構文でより多くのアプリケーションを作成できます。
次のデータ型は、SQL TYPE 関数でホスト変数として使用できます。

- BINARY
- CHAR-VARYING
- DATE
- DATE-RECORD
- LONG-VARBINARY
- LONG-VARCHAR
- TIME
- TIME-RECORD
- TIMESTAMP
- TIMESTAMP-RECORD
- VARBINARY

次に示すどの例でも、トークン TYPE と IS はオプションです。

BINARY の例

```
01 hv-name SQL TYPE IS BINARY(n)
generates
01 hv-name pic x(n).
```

CHAR-VARYING の例

```
01 hv-name SQL TYPE IS CHAR-VARYING(n)
generates
01 hv-name pic x(n).
```

CHAR-VARYING データは、OpenESQL に SQL_VARCHAR として渡されます。データがデータソースに送信されると、値がすべて空白文字の場合は最初の空白文字以外の後続の空白文字は削除され、データソースから取り込んだ値には空白文字が付加されます。

DATE の例

```
01 hv-name SQL TYPE IS DATE
generates
01 hv-name pic x(10).
```

DATE データは、YYYY-MM-DD の形式である必要があります。

DATE-RECORD の例

```
01 hv-name SQL TYPE IS DATE-RECORD
generates
01 hv-name.
  03 hv-name-year   pic s9(4) comp-5.
  03 hv-name-month  pic 9(4) comp-5.
  03 hv-name-day    pic 9(4) comp-5.
```

TIMESTAMP の例

```
01 hv-name SQL TYPE IS TIMESTAMP
generates
01 hv-name pic x(29).
```

TIMESTAMP データは、YYYY-MM-DD HH:MM:SS の形式である必要があります。

TIMESTAMP-RECORD の例

```
01 hv-name SQL TYPE IS TIMESTAMP-RECORD
generates
01 hv-name.
  03 hv-name-year   pic s9(4) comp-5.
  03 hv-name-month  pic 9(4) comp-5.
  03 hv-name-day    pic 9(4) comp-5.
  03 hv-name-hour   pic 9(4) comp-5.
  03 hv-name-min    pic 9(4) comp-5.
  03 hv-name-sec    pic 9(4) comp-5.
  03 hv-name-frac   pic 9(9) comp-5.
```

第 4 章 : カーソル

SELECT 文によって返される結果集合に複数行のデータが含まれるコードを作成する場合は、カーソルを宣言して使用する必要があります。カーソルは、結果集合内の現在の位置を、画面上のカーソルが現在の位置を示すのと同じように示します。
カーソルを使用すると、次の処理が行えます。

- 1 回の操作での複数行データの取り出し。
- 結果集合内の指定位置での更新や削除の実行。

次に示すコード例は、次の一連のイベントを示しています。

1. DECLARE CURSOR 文で、SELECT 文をカーソル `Cursor1` に関連付けます。
2. OPEN 文で、カーソルをオープンします。カーソルをオープンする場合は、関連付けた SELECT 文が実行されます。
3. FETCH 文で、カラム `au_fname` と `au_lname` の現在行のデータを検索し、ホスト変数 `first_name` と `last_name` にそのデータを格納します。
4. FETCH 文がループされ、取り込むデータがなくなるまで繰り返し実行されます。
5. CLOSE 文でカーソルをクローズします。

```
EXEC SQL DECLARE Cursor1 CURSOR FOR
   SELECT au_fname, au_lname FROM authors
END-EXEC
...
EXEC SQL
   OPEN Cursor1
END-EXEC
...
perform until sqlcode not = zero
   EXEC SQL
      FETCH Cursor1 INTO :first_name,:last_name
   END-EXEC
      display first_name, last_name
end-perform
...
EXEC SQL
   CLOSE Cursor1
END-EXEC
```

4.1 カーソルの宣言

カーソルを使用する前に、カーソルを宣言する必要があります。カーソルを宣言するには、DECLARE CURSOR 文を使用し、この文で、カーソル名と、SELECT 文または PREPARE 文で定義した SQL 文の名前を指定します。
カーソル名は、接続するデータベースの識別子に対する規則に準拠する必要があります。たとえば、識別子にハイフンを使用できないデータベースでは、カーソル名にもハイフンは使用できません。

```
EXEC SQL
    DECLARE Cur1 CURSOR FOR
        SELECT first_name FROM employee
         WHERE last_name = :last-name
END-EXEC
```

この例では、入力ホスト変数 (:last-name) を使用して SELECT 文を指定しています。カーソルの OPEN 文が実行されると、入力ホスト変数の値が読み取られ、SELECT 文が実行されます。

```
EXEC SQL
    DECLARE Cur2 CURSOR FOR stmt1
END-EXEC
...
move "SELECT first_name FROM emp " &
     "WHERE last_name=?" to prep.
EXEC SQL
   PREPARE stmt1 FROM :prep
END-EXEC
 ...
EXEC SQL
   OPEN Cur2 USING :last-name
END-EXEC
```

この例では、DECLARE CURSOR 文が PREPARE 文で定義した文 (stmt1) を参照しています。PREPARE 文で定義した SELECT 文には、パラメータマーカーとして疑問符 (?) を使用できます。パラメータマーカーは、カーソルをオープンするとデータに置き換えられます。カーソルは、SELECT 文を PREPARE 文で定義する前に宣言してください。

COBSQL

カーソルは、プログラムのデータ部または手続き部のどちらかで宣言できます。DECLARE CURSOR 文はコードを生成しませんが、カーソルが手続き部で宣言された場合には、COBSQL は DECLARE CURSOR 文に対してアニメーションブレークポイントを生成します。

4.1.1 オブジェクト指向 COBOL 構文

オブジェクト指向 (OO) プログラム内では、データ項目を宣言するために有効な場所であればどこでもカーソルを宣言できます。カーソルは、カーソルが開かれたオブジェクト内でローカルです。つまり、それぞれ「同じ」カーソルを開いているオブジェクトの 2 つのインスタンスは、独自のカーソルインスタンスを取得します。

カーソルをあるメソッドでオープンし、第 2 のメソッドで取り込み、第 3 のメソッドでクローズできますが、その場合には、カーソルをオブジェクト記憶で宣言する必要があります。

注:

COBSQL

- オブジェクト指向 COBOL 構文は、COBSQL ではサポートされていません。
- Informix プリコンパイラの一部のバージョンでは、カーソルが作業場所節内で宣言されると、不正なコードが生成されることがあります。このため、INFORMIX を使用する場合には、カーソルは手続き部のみで宣言することをお奨めします。

4.2 カーソルのオープン

宣言したカーソルを使用する前に、カーソルをオープンする必要があります。カーソルをオープンするには、OPEN 文を使用します。次に例を示します。

```
EXEC SQL
   OPEN Cur1
END-EXEC
```

DECLARE CURSOR 文が PREPARE 文で定義した文を参照しており、この参照文にパラメータマーカーが含まれる場合には、パラメータマーカーの値を提供するホスト変数または SQLDA 構造体の名前を OPEN 文で指定する必要があります。次に例を示します。

```
EXEC SQL
   OPEN Cur2 USING :last-name
END-EXEC
```

SQLDA データ構造体を使用するには、OPEN 文の実行時にデータ型、データ長、およびアドレスの各フィールドに、有効値が設定済みであることが必要です。

COBSQL

カーソルがオープンされると、データが選択されているテーブルにロックは適用されません。

COBSQL

Oracle データベースでは、カーソルをクローズする前に再オープンし、SELECT 文を再評価することができます。プログラムが ANSI モードでコンパイルされている場合には、カーソルをクローズする前に再オープンするとエラーが発生します。MODE プリコンパイラ指令の詳細については、『***ORACLE プリコンパイラ・プログラマーズ・ガイド***』を参照してください。

4.3 カーソルでのデータ取り出し方法

カーソルをオープンすると、データベースからデータを取り出すことができます。データの取り込みには、FETCH 文を使用します。FETCH 文は、OPEN 文によって生成された結果集合から次の行を取り出し、指定されたホスト変数 (または SQLDA 構造体で指定されたアドレス) にそのデータを書き込みます。次に使用例を示します。

```
     perform until sqlcode not = 0
         EXEC SQL
            FETCH Cur1 INTO :first_name
         END-EXEC
         display '姓 : ' fname
         display '名 : ' lname
         display spaces
     end-perform
```

カーソルが結果集合の末尾に達する場合は、SQLCA データ構造体の SQLCODE の値として 100 が戻され、SQLSTATE の値が「02000」に設定されます。

データがカーソルから取り出されると、データが選択されているテーブルにロックを置くことができます。カーソルの異なるタイプ、ロックされたデータの読み込み、およびデータに置くロックの詳細については、『*カーソルオプション*』を参照してください。

COBSQL

ORACLE プリコンパイラ指令 MODE は、データが見つからない場合に SQLCODE に格納される値に影響を与えます。MODE プリコンパイラ指令の使用方法の詳細については、『***ORACLE プリコンパイラ・プログラマーズ・ガイド***』を参照してください。

4.4 カーソルのクローズ

カーソルの使用が終了したら、使用したカーソルを CLOSE 文でクローズします。次に使用例を示します。

```
EXEC SQL
   CLOSE Cur1
END-EXEC
```

通常は、カーソルがクローズされると、データおよびテーブルへのロックがすべて解放されます。ただし、トランザクション内でカーソルがクローズされると、ロックが解放されない場合があります。

COBSQL
ORACLE プリコンパイラ指令 MODE は、コミットまたはロールバックコマンドを使用したときのカーソルの動作へ影響を与えます。MODE プリコンパイラ指令の使用方法の詳細については、『***ORACLE プリコンパイラ・プログラマーズ・ガイド***』を参照してください。
COBSQL
カーソルをクローズすると、ORACLE クライアントはそのカーソルに関連付けられたメモリおよびリソースの割り当てを解除することがあります。プリコンパイラオプション HOLD_CURSOR、MAXOPENCURSORS、および RELEASE_CURSOR でカーソルの割り当て解除を制御します。プリコンパイラ指令の使用方法の詳細については、『***ORACLE プリコンパイラ・プログラマーズ・ガイド***』を参照してください。

4.5 カーソルオプション

OpenESQL
ここで述べるカーソルオプションの説明は、OpenESQL のみに適用されます。
カーソルの動作やパフォーマンスは、次の埋め込み SQL 文を使用して調整できます。

埋め込み SQL 文	説明
SET SCROLLOPTION	カーソルの結果集合に含まれる行の決定方法を選択します。
SET CONCURRENCY	並行制御を有効にします。並行アクセスでは、データの信頼性を維持するために何らかの制御を行う必要があります。この文は、カーソルをオープンする前に使用します。

注：SET SCROLLOPTION と SET CONCURRENCY は、どちらも拡張 SQL 文の一部であるため、使用する ODBC ドライバによってはサポートされないことがあります。

4.6 位置づけ UPDATE 文と DELETE 文

位置づけ UPDATE 文と DELTE 文はカーソルと併用して使用し、検索条件句のかわりに WHERE CURRENT OF 句を記述します。WHERE CURRENT OF 句によって、併用するカーソルを指定します。

```
EXEC SQL
   UPDATE emp SET last_name = :last-name
      WHERE CURRENT OF Cur1
END-EXEC
```

この例では、カーソル `Cur1` を使用して、データベースから前回取り込んだ行に含まれる `last_name` の値を更新します。

```
EXEC SQL
   DELETE emp WHERE CURRENT OF Cur1
END-EXEC
```

この例では、カーソル `Cur1` を使用して、データベースから前回取り込んだ行を削除します。
OpenESQL
ODBC ドライバによっては、位置づけ更新や位置づけ削除に使用するカーソルに FOR UPDATE 句を記述する必要があります。位置づけ UPDATE や位置づけ DELETE は拡張 ODBC 構文の一部です。そのため、使用する ODBC ドライバによってはサポートされない場合もあります。

OpenESQL
COBSQL では、位置づけ更新や位置づけ削除に使用するカーソルには FOR UPDATE 句を記述する必要があります。

4.7 カーソルの使用

カーソルは大量のデータを扱う場合に便利ですが、カーソルを使用する際には、データ並行性、データ整合性、およびデータ一貫性に留意する必要があります。
データの整合性を確保するために、データベースサーバはいくつかのロック方法を実装できます。ロックを取得しないデータアクセスのタイプ、共有ロックを取得するタイプ、および排他ロックがあります。共有ロックでは、他のプロセスがデータにアクセスできますが、更新はできません。排他ロックでは、他のプロセスはデータにアクセスできません。
カーソルを使用する際には、次のように 3 つの分離レベルがあり、カーソルが読み込んでロックできるデータを制御します。

- レベル 0

  レベル 0 は、読み取り専用カーソルのみで使用できます。レベル 0 では、カーソルは行をロックしませんが、コミットされていないデータを読み込む可能性があります。コミットされていないデータを読み込むのは危険です (ロールバック操作でデータが元の状態に戻るため)。これは通常「ダーティリード」と呼ばれます。データベースによっては、ダーティリードは行えません。

- レベル 1

  レベル 1 は、読み取り専用カーソルまたは更新可能カーソルに使用できます。レベル 1 では、FOR UPDATE 句を使用している場合を除き、共有ロックがデータに置かれます。FOR UPDATE 句を使用している場合には、排他ロックがデータに置かれます。カーソルをクローズする場合は、ロックが解放されます。通常、FOR UPDATE 句なしの標準カーソルは、分離レベル 1 にあり、共有ロックを使用します。

- レベル 3

  レベル 3 カーソルは、トランザクションに使用されます。カーソルをクローズしたときにロックが解放されるのではなく、トランザクションが終了したときにロックが解放されます。通常、レベル 3 では、データを排他ロックします。

2 つのプロセスが同じデータで競合する状態「デッドロック」または「膠着状態」の問題が発生することがあります。代表的な例を説明します。あるプロセスがデータ A をロックしてデータ B にロックを要求します。ただし、別のプロセスがデータ B をロックしており、データ A にロックを要求します。このように両方のプロセスが、相手プロセスが要求しているデータをロックしている場合です。このような場合には、データベースサーバが問題を発見し、どちらか片方、または両方のプロセスにエラーを送ります。
COBSQL
Oracle、Sybase、および Informix では、アプリケーションでカーソルの分離レベルを設定することができます。適用可能なロックタイプと動作については、マニュアルで説明しています。マニュアルでは、データがロックされる物理レベルについても説明しています。物理レベルには、単一行、行セット (ページレベル)、またはテーブル全体があります。複数のテーブルや、多数のプロセスに使用されているテーブルを走査するカーソルを使用する場合は、ロックされたデータのアクセス可能性が減少するので、注意が必要です。

注:
COBSQL
Oracle、Sybase、**および** Informix **では、**FOR READ ONLY **や** FOR UPDATE **などの多くの異なる句で定義されたカーソルを使用できます。これらの句は、カーソルの分離レベルに影響し、トランザクション処理に影響を及ぼします。これらの各句の影響の詳細については、使用しているデータベースに付属の** SQL **リファレンスマニュアルを参照してください。**

第 5 章 : データ構造体

Net Express SQL プリプロセッサでは、次の 2 つのデータ構造体を使用します。

データ構造体	説明	機能
SQLCA	SQL 通信領域	ステータスとエラーの情報を返します。
SQLDA	SQL 記述子領域	動的 SQL 文で使用される変数を記述します。

5.1 SQL 通信領域 (SQLCA)

埋め込み SQL 文を実行するたびに、エラーおよびステータスの情報が SQL 通信領域 (SQLCA) に返されます。
SQLCA データ構造体のレイアウトの詳細については、オンラインヘルプを参照してください。ヘルプファイルの索引で「SQLCA」を選択してください。
SQLCA には 2 つの変数 (`sqlcode` および `sqlstate`) の他、前回実行した SQL 文でエラーが発生したかどうかを示す一連の警告フラグが含まれています。
COBSQL
COBSQL では、SQLSTATE は独立したデータ項目です。現在サポートされているバージョンの Oracle と Sybase では、SQLSTATE 変数よりも SQLCA を優先して使用してください。将来的には、データベースとクライアントアプリケーション間でのデータの受け渡しの方法として、SQLCA ではなく SQLSTATE 変数が使用されるようになりますが、まだ実現されていません。

5.1.1 SQLCODE 変数

埋め込み SQL 文が正常に実行できたかどうかを確認する手段としては、*sqlcode* 変数の値をチェックするのが最も一般的です。`sqlcode` の値は、次のとおりです。

値	説明
0	この文は、正常に実行されました。
1	文は実行されましたが、警告が生成されました。エラーの種類については、`sqlwarn` フラグの値を確認してください。
100	クエリーに一致するデータが見つからないか、または、結果集合の末尾に達しました。処理された行はありません。
< 0 (負の値)	アプリケーション、データベース、システム、またはネットワークのエラーのため、文が実行されませんでした。

SQLCODE 値の詳細については、オンラインヘルプを参照してください。ヘルプファイルの索引で「SQLCA」を選択してください。
COBSQL および DB2
COBSQL と DB2 の場合には、上記以外にも正の値が返される可能性があります。これは、SQL 文は実行されたが、警告が生成されたということを意味します。
COBSQL

- SQLCODE に設定できる正の値がとる範囲の詳細については、Oracle、Sybase、または Informix のエラーメッセージマニュアルをそれぞれ参照してください。上記の SQLCODE は、OpenSQL で生成されます。Oracle、Sybase、および Informix で報告される SQLCODE の

値とエラーは、それぞれ異なります。返されるエラーの詳細については、『*Oracle エラー・メッセージ*』、『*Sybase Error Messages*』、または『*Informix Error Messages*』のマニュアルを参照してください。

- 値 +100 は、「データが見つからない」ことを表す ANSI 標準規格です。Oradle では「データが見つからない」場合には、別の値を返すことがあります。Oracle で「データが見つからない」場合に値 +100 が返されるようにするには、Oracle プリコンパイラ指令 MODE=ANSI または END_OF_FETCH=100 を設定する必要があります。この設定は、Oracle プリコンパイラが SQL 文を処理する方法にも影響を与えます。Oracle プリコンパイラ MODE または END_OF_FETCH=+100 指令の詳細については、『***ORACLE プリコンパイラ・プログラマーズ・ガイド***』を参照してください。
- SQLCODE が 0 の場合でも警告が生成されている可能性があります。`sqlwarn` フラグの値をチェックして、警告の種類を確認してください。Oracle、Sybase、および Informix では、データベースサーバがアプリケーションに警告を返すときには、常に `sqlwarn0` が設定されます。

5.1.2 SQLSTATE 変数

DB2
DB2 ユニバーサルデータベースでは、SQL-92 準拠の SQLSTATE 値を返します。DB2 バージョン 2.1 では異なります。
`sqlstate` 変数は SQL-92 標準に導入された、将来のアプリケーションの推奨機構です。この変数は、次の 2 つの構成要素からなります。

- 最初の 2 文字は `class` コードと呼びます。文字 A ～ H、または桁 0 ～ 4 で始まるクラスコードは、SQL 標準または他の標準による定義の `sqlstate` 値を示します。
- 後ろの 3 文字は `subclass` コードと呼びます。

値「00000」は、前の埋め込み SQL 文が正常に実行されたことを示します。
Oracle、Sybase、または Informix 使用時の SQLSTATE に格納される値の詳細については、関連の*データベースエラーメッセージマニュアル*を参照してください。SQLSTATE 値の詳細については、オンラインヘルプも参照してください。ヘルプファイルの索引で「**SQLSTATE**」を選択してください。

5.1.3 警告フラグ

文を実行すると警告が生成されることがあります。警告の種類を確認するには、アプリケーションで `sqlwarn` フラグの値を確認する必要があります。このフラグには、特定の警告が発生すると「W」、それ以外の場合には空白文字が設定されます。
各 `sqlwarn` フラグにはそれぞれ特定の意味があります。`sqlwarn` フラグの意味については、オンラインヘルプを参照してください。ヘルプファイルの索引で「SQLCA」を選択してください。

5.1.4 WHENEVER 文

埋め込み SQL 文を実行するたびに `sqlcode` または `sqlstate` の値を明確に確認するには、多くのコードを記述する必要があります。この問題を回避するには、アプリケーションで WHENEVER 文を使用して、SQL 文のステータスを確認します。
WHENEVER 文は実行文ではありません。埋め込み SQL 文が実行されるたびに、コンパイラ指令として、コンパイラにエラー処理コードを自動生成させます。
WHENEVER 文では、次の各条件に対して 3 つのデフォルト動作 (CONTINUE、GOTO、PERFORM) のどれかを登録できます。

条件	sqlcode の値
NOT FOUND	100
SQLWARNING	+1
SQLERROR	< 0 (負の値)

COBSQL
Oracle、Sybase、および Informix では、sqlwarn0 が「W」に設定されると、「SQLWARNING」句がトリガされます。
Oracle
Oracle プリコンパイラ指令 MODE の設定に関わらず、SELECT 文または FETCH 文からデータが戻されなかった場合には、常に NOT FOUND 条件がトリガされます。
Informix
Informix では、WHENEVER 文内から STOP または CALL を実行できます。これは ANSI 標準に追加され、『**INFORMIX-ESQL/COBOL Programmer's Manual**』で説明されています。
ある特定の条件に対する WHENEVER 文は、その条件に対する以前の WHENEVER 文をすべて置き換えます。
WHENEVER 文の範囲は、実行シーケンス内の論理位置ではなく、原始プログラム内の物理位置に関連します。たとえば、次のコードで、最初の SELECT 文が何も戻さない場合には、段落 C ではなく段落 A が処理されます。

```
     EXEC SQL
         WHENEVER NOT FOUND PERFORM A
     END-EXEC.
     perform B.
     EXEC SQL
        SELECT col1 into :host-var1 FROM table1
         WHERE col2 = :host-var2
     END-EXEC.
 A.
     display "最初の項目が見つかりません".
 B.
     EXEC SQL
         WHENEVER NOT FOUND PERFORM C.
     END-EXEC.
 C.
     display "2 番目の項目が見つかりません".
```

5.1.5 SQLERRM

SQLERRM データ領域は、データベースサーバからアプリケーションにエラーメッセージを渡すために使用されます。SQLERRM データ領域は、SQLERRML と SQLERRMC の 2 つの部分で構成されています。SQLERRML には、エラーメッセージの長さが保持され、SQLERRMC にはエラーテキストが保持されます。エラールーチン内では、次のコードを使用して、SQL エラーメッセージを表示できます。

```
     if (SQLERRML > ZERO) and (SQLERRML < 80)
         display 'エラーメッセージ : ', SQLERRMC(1:SQLERRML)
     else
         display 'エラーメッセージ : ', SQLERRMC
     end-if.
```

5.1.6 SQLERRD

SQLERRD データ領域は、6 つの整数の状態値をもつ配列です。

COBSQL
Oracle、Sybase、および Informix では、SQLERRD 配列内に 6 つの値のうちの 1 つ (またはそれ以上) が設定される場合があります。これらは、直前に実行した SQL 文により影響を受ける行数を示します。たとえば、SQLERRD(3) には、SELECT 文または一連の FETCH 文で返された行の合計数が格納されています。

5.2 SQL 記述子領域 (SQLDA)

次の状況では、ホスト変数ではなく、SQL 記述子領域 (SQLDA) を使用します。

- 動的 SQL 文

  多数のパラメータを渡す場合や、コンパイル時にデータ型が不明な場合

- 静的 SQL 文

  カーソル FETCH 文を使用する場合

SQLDA には、入力パラメータや出力カラムの詳細情報が格納されます。

- カラム名
- データ型
- 長さ
- 各入力パラメータや出力パラメータで使用するデータバッファへのポインタ

SQLDA は、プリコンパイラごとに一意であるため、データは必ず適切な形式に変換されます。
通常、SQLDA はパラメータマーカーと併用して、PREPARE 文で定義した SQL 文への入力値を指定します。ただし、PREPARE 文で定義した SELECT 文からデータを取得するには、DESCRIBE 文、または PREPARE 文の INTO オプションと SQLDA を併用することもできます。
Oracle SQLDA は、Sybase、OpenESQL、または DB2 で使用される SQLDA と互換性がありません。
同様に、Sybase、OpenESQL、または DB2 SQLDA は、Oracle SQLDA と互換性がありません。
COBSQL
Oracle と Sybase のどちらの場合でも、プログラムが動的 SQL を使用する場合に限り SQLDA が必要です。
COBSQL
Oracle、Sybase、および Informix では、SQLDA を標準の COBOL コピーファイルとして定義する必要があります。たとえば、次のように定義します。
```
COPY "SQLDA".
```
Oracle、Sybase、および Informix では、EXEC SQL INCLUDE 構文を使用してプログラムに SQLDA をインクルードできません。
COBSQL
Oracle には、動的 SQL と使用する特別なコピーファイル ORACA が用意されています。これは、次の構文を使用してプログラムにインクルードできます。
```
EXEC SQL
   INCLUDE ORACA
END-EXEC
```
ORACA を使用するには、Oracle プリコンパイラオプション ORACA=YES を設定する必要があります。
Oracle プリコンパイラオプションの設定の詳細については、『***ORACLE プリコンパイラ・プログラマーズ・ガイド***』を参照してください。

COBSQL
Oracle では SQLDA が提供されませんが、『***ORACLE プリコンパイラ・プログラマーズ・ガイド***』には、レイアウトの定義が記述されています。
COBSQL
Sybase では SQLDA コピーファイルは提供されません。Sybase プリコンパイラのマニュアルでは、SQLDA のレイアウトと、そのレイアウト内の各種項目に値を割り当てる方法が説明されています。また、このマニュアルでは、Sybase を使用して COBOL データ形式と Sybase データ型の変換を行う方法も説明されています。
OpenESQL
SQLDA 構造体は、Net Express の基本インストールディレクトリにある **source** ディレクトリの **sqlda.cpy** ファイルで提供されています。次の文をデータ部に追加して、COBOL プログラムに SQLDA 構造体をインクルードできます。

```
EXEC SQL
   INCLUDE SQLDA
END-EXEC
```

OpenESQL SQLDA の詳細については、オンラインヘルプで参照してください。ヘルプファイルの索引で「SQLCA」を選択してください。

5.2.1 SQLDA の使用方法

SQLDA 構造体を使用する前に、アプリケーションで次のフィールドを初期化する必要があります。

SQLN	このフィールドは、SQLDA 構造体で保持できる SQLVAR 項目の最大数に設定します。
SQLDABC	SQLDA の最大サイズ。SQLN × 44 + 16 という式で計算します。

5.2.1.1 PREPARE 文と DESCRIBE 文

DESCRIBE 文 (または INFO オプションを指定した PREPARE 文) を使用して、SQLDA 構造体の適切なフィールドにカラム名やデータ型などの情報を入力できます。
これらの文を実行するには、前述したように SQLN フィールドと SQLDABC フィールドを初期化する必要があります。
文を実行する場合は、PREPARE で定義された文のパラメータ数が SQLD フィールドに含まれます。パラメータごとに SQLTYPE と SQLLEN フィールドが設定され、SQLVAR レコードが設定されます。SQLN にどの位の大きさの値を設定すればよいかわからない場合には、SQLN に 1、SQLD に 0 を設定して DESCRIBE 文を実行できます。この場合は、詳細なカラム情報が SQLDA 構造体に転記されませんが、結果集合に含まれるカラム数が SQLD に挿入されます。

5.2.1.2 FETCH 文

SQLDA 構造体を使用して FETCH 文を実行するには、前述したようにアプリケーションで SQLN と SQLDABC を初期化しておく必要があります。また、対応するカラムデータから受け取る各プログラム変数のアドレスを SQLDATA フィールドに挿入する必要があります (SQLDATA フィールドは SQLVAR の一部です)。インジケータ変数を使用する場合には、SQLIND にも対応するインジケータ変数のアドレスを設定する必要があります。
データ型フィールド (SQLTYPE) とデータ長 (SQLLEN) は、PREPARE INTO または DESCRIBE 文で取り込まれた情報が格納されます。これらの値は、FETCH 文の実行前にアプリケーションで上書きできます。

5.2.1.3 OPEN 文または EXECUTE 文

SQLDA 構造体を使用して、OPEN 文または EXECUTE 文への入力データを指定するには、アプリケーションで SQLDA 構造体の全フィールドのデータを設定する必要があります。この場合は、各変数の SQLN、SQLD、SQLDABC、SQLTYPE、SQLLEN、および SQLDATA フィールドも含めて設定します。SQLTYPE フィールドの値が奇数の場合は、SQLIND フィールドにもインジケータ変数のアドレスを設定する必要があります。

5.2.2 DESCRIBE 文

PREPARE 文の後に DESCRIBE 文を実行する場合は、指定の PREPARE 文で定義した文によって戻される各カラムのデータ型、データ長、および名前を取り出すことができます。この情報は SQL 記述子領域 (SQLDA) に戻されます。

```
EXEC SQL
   DESCRIBE stmt1 INTO :sqlda
END-EXEC
```

PREPARE 文の実行後すぐに DESCRIBE 文を実行する場合には、次のように PREPARE 文と INFO オプションを使用する場合に、両方の操作を同時に実行できます。

```
EXEC SQL
   PREPARE stmt1 INTO :sqlda FROM :stmtbuf
END-EXEC
```

第 6 章 : 動的 SQL

アプリケーションのコンパイル時にソースコードに完全に記述されている SQL 文を、静的 SQL と呼びます。
ただし、アプリケーションの作成時に SQL 文の内容が完全に特定できないこともあります。たとえば、アプリケーションの実行時にエンドユーザが任意の SQL 文を入力する場合などです。この場合は、SQL 文を実行時に作成する必要があります。このような文を、動的 SQL 文を呼びます。

6.1 動的 SQL 文のタイプ

動的 SQL 文には、次のように 4 つのタイプがあります。

動的 SQL 文のタイプ	クエリーを実行するかどうか	データを返すかどうか
文を 1 回実行する	実行しない	成功または失敗のみ返す
同じ文を複数回実行する	実行しない	成功または失敗のみ返す
特定の選択基準を使用して、特定のデータリストを選択する	実行する	返す
何らかの選択基準で任意の量のデータを選択する	実行する	返す

これらの型の動的 SQL 文については、以降に詳しく説明します。

6.1.1 文を 1 回実行する

このタイプの動的 SQL 文では、SQL 文が直ちに実行されます。文が実行されるたびに、構文解析が行われます。

6.1.2 同じ文を複数回実行する

このタイプの動的 SQL 文は、複数回実行する可能性のある文か、または、ホスト変数を必要とする文です。この場合は、PREPARE 文で定義した文を実行する必要があります。

6.1.3 特定のデータリストを選択する

このタイプの動的 SQL 文は、ホスト変数の数と型が判明している SELECT 文です。この SQL 文の通常のシーケンスは次のとおりです。

1. 文を準備します。
2. 結果を保持するカーソルを宣言します。
3. カーソルをオープンします。
4. 変数を取り出します。
5. カーソルをクローズします。

6.1.4 任意の量のデータを選択する

この動的 SQL 文は、コーディングするのが最も難しいタイプです。変数の型 / 個数は実行時のみに解決されます。この SQL 文の通常のシーケンスは次のとおりです。

1. 文を準備します。
2. 文に対してカーソルを宣言します。
3. 使用する変数を記述します。
4. 記述した変数を使用してカーソルをオープンします。
5. 取り出す変数を記述します。
6. その記述内容を使用して変数を取りします。
7. カーソルをクローズします。

入力ホスト変数、または出力ホスト変数のどちらかがコンパイル時に判明している場合には、OPEN または FETCH がホスト変数を命名できるので、ホスト変数を記述する必要はありません。

6.2 動的 SQL 文の準備

PREPARE 文では、動的 SQL 文を含む文字列を受け取り、名前と文を関連付けます。次に例を示します。

```
     move "INSERT INTO publishers " &
               "VALUES (?,?,?,?)" to stmtbuf
     EXEC SQL
         PREPARE stmt1 FROM :stmtbuf
     END-EXEC
```

動的 SQL 文では、値に対するプレースホルダとしてパラメータマーカーである、疑問符 (?) を使用できます。上記の例では、4 つの疑問符 (?) に対応する値を、文の実行時に指定する必要があります。
PREPARE 文で定義した SQL 文は、次のどちらかの方法で実行します。

- 定義した文を実行する。
- 定義した文を参照するカーソルをオープンする。

COBSQL
Oracle では、プレースホルダとして疑問符を使用しません。ホスト変数表記を使用します。便宜上、プレースホルダは V*n* と命名され、*n* にはプレースホルダを文内で一意にする数字を指定します。読みやすくするために、同じプレースホルダを複数回使用できますが、文の実行時には (またはカーソルを使用している場合はオープン時)、各プレースホルダに対してホスト変数が 1 つ必要です。次に例を示します。

```
     string "update ordtab " delimited by size
            "set order_no = :v1, "
            "line_no = :v2, "
            "cust_code = :v3, "
            "part_no = :v4, "
            "part_name = :v5, "
            "order_val = :v6, "
            "pay_value = :v7 "
            "where order_no = :v1 and "
            "line_no = :v2 and "
            "cust_code = :v3 " delimited by size
      into Updt-Ord-Stmt-Arr
     end-string
```

```
move 190 to Updt-Ord-Stmt-Len

EXEC SQL PREPARE updt_ord FROM :Updt-Ord-Stmt END-EXEC

EXEC SQL EXECUTE updt_ord USING
    :dcl-order-no, :dcl-line-no, :dcl-cust-code,
    :dcl-part-no,  :dcl-part-name:ind-part-name,
    :dcl-order-val,:dcl-pay-value,
    :dcl-order-no, :dcl-line-no, :dcl-cust-code
END-EXEC
```

ここでは Updt-Ord-Stmt は、VARYING 型のホスト変数として定義されています。

COBSQL

Oracle プリコンパイラを使用する場合は、PREPARE 文の物理的な位置が重要になります。PREPARE 文は、EXECUTE 文または DECLARE 文の前に記述する必要があります。

6.3 動的 SQL 文の実行

PREPARE 文で定義した SQL 文は、EXECUTE 文で個別に実行されます。

注:この方法で実行できるのは、結果を返さない SQL 文のみです。

PREPARE 文で定義した文にパラメータマーカーが含まれている場合は、EXECUTE 文で using :hvar オプションを使用し、ホスト変数名を記述してパラメータを指定するか、または、using descriptor :sqlda_struct オプションを使用し、アプリケーションによって値がすでに格納されている SQLDA データ構造体を識別する必要があります。PREPARE 文で定義した文に含まれるパラメータマーカー数は、SQLDATA エントリのメンバ数 (using descriptor :sqlda) またはホスト変数 (using :hvar) の数と一致する必要があります。

```
move "INSERT INTO publishers " &
          "VALUES (?,?,?,?)" to stmtbuf
EXEC SQL
    PREPARE stmt1 FROM :stmtbuf
END-EXEC
 ...
EXEC SQL
    EXECUTE stmt1 USING :pubid,:pubname,:city,:state
END-EXEC.
```

この例では、4 つのパラメータマーカーを EXECUTE 文の USING 句で指定した 4 つのホスト変数の値によって置き換えます。

6.3.1 EXECUTE IMMEDIATE

パラメータマーカーを含まない動的 SQL 文は、PREPARE と EXECUTE を順次実行するかわりに、EXECUTE IMMEDIATE を使用して、直ちに実行できます。次に例を示します。

```
move "DELETE FROM emp " &
          "WHERE last_name = 'Smith'" to stmtbuf
EXEC SQL
    EXECUTE IMMEDIATE :stmtbuf
END-EXEC
```

COBSQL

EXECUTE IMMEDIATE を使用する場合は、文が実行されるたびに構文解析が再度行われます。文を何回も使用するような場合には、文を PREPARE として実行し、必要に応じて EXECUTE を実行するほうが効率的です。

6.3.2 FREE 文 (COBSQL Informix)

Informix プリコンパイラには、PREPARE 文で定義された文またはカーソルに割り当てられたリソースを解放する FREE 文があります。
PREPARE 文で定義された文が終了した後に、FREE 文を使用します。たとえば、次のように記述します。

```
     move "INSERT INTO publishers " " &
               "VALUES (?,?,?,?)" to stmtbuf
     EXEC SQL
         PREPARE stmt1 FROM :stmtbuf
     END-EXEC
      ...
     EXEC SQL
         EXECUTE stmt1 USING :pubid,:pubname,:city,:state
     END-EXEC.
      ...
     EXEC SQL
         FREE stmt1
     END-EXEC
```

6.4 動的 SQL 文とカーソル

結果を返す動的 SQL 文では、EXECUTE 文を使用できません。この場合は、カーソルを宣言して使用する必要があります。
まず、DECLARE CURSOR 文で次のようにカーソルを宣言します。

```
EXEC SQL
   DECLARE C1 CURSOR FOR dynamic_sql
END-EXEC
```

この例では、`dynamic_sql` が動的 SQL 文の名前です。この動的 SQL 文は、宣言したカーソルをオープンする前に PREPARE 文で定義する必要があります。

```
     move "SELECT char_col FROM mfesqltest " &
          "WHERE int_col = ?" to sql-text
     EXEC SQL
        PREPARE dynamic_sql FROM :sql-text
     END-EXEC
```

そして、OPEN 文を使用してカーソルをオープンする場合は、PREPARE 文で定義した文が実行されます。

```
EXEC SQL
   OPEN C1 USING :int-col
END-EXEC
```

PREPARE で定義した文でパラメータマーカーを使用している場合は、ホスト変数または SQLDA 構造体を指定してこれらのパラメータに OPEN 文で値を指定する必要があります。
カーソルをオープンした後に、FETCH 文を使用してデータを取り出すことができます。次に例を示します。

```
EXEC SQL
   FETCH C1 INTO :char-col
END-EXEC
```

FETCH 文の詳細については、『*カーソル*』の章を参照してください。
最後に、CLOSE 文を使用してカーソルをクローズします。

```
EXEC SQL
   CLOSE C1
END-EXEC
```

CLOSE 文の詳細については、『*カーソル*』の章を参照してください。

6.5 CALL 文

CALL 文は、動的 SQL として準備および実行できます。これは、OpenESQL プリコンパイラのみでサポートされます。

- 静的 SQL でホスト変数を使用する場合は、常にパラメータマーカー (?) を動的 SQL で使用できます。
- パラメータマーカーの後に IN、INPUT、OUT、OUTPUT、INOUT、および CURSOR キーワードを使用することは、静的 SQL でホスト変数パラメータの後にこれらのキーワードを使用することと同じです。
- CALL 文全体は、ODBC の標準的なストアドプロシージャ構文に準拠するために波かっこで囲む必要があります (Open ESQL プリコンパイラは、静的 SQL でこれを行います)。次に例を示します。

```
move '{call myproc(?, ? out)}' to sql-text
exec sql
    prepare mycall from :sql-text
end-exec
exec sql
    execute mycall using :parm1, :param2
end-exec
```

- パラメータ配列を使用する場合は、EXECUTE 文の FOR 句で使用される要素の数を制限できます。次に例を示します。

```
move 5 to param-count
exec sql
    for :param-count
     execute mycall using :parm1, :param2
end-exec
```

例

次に、データソース「SQLServer 2000」を使用してストアドプロシージャ「mfexecsptest」を作成し、動的 SQL でカーソル「c1」を使用して「publishers」テーブルからデータを取り出すプログラム例を示します。

```
$SET SQL
 WORKING-STORAGE SECTION.

 EXEC SQL INCLUDE SQLCA  END-EXEC

 *> SQL エラーが発生する場合は、ここに詳細なメッセージテキストが示されます。
 01 MFSQLMESSAGETEXT  PIC X(250).
 01 IDX               PIC X(04)  COMP-5.

 EXEC SQL BEGIN DECLARE SECTION  END-EXEC
 *> 他の COBOL コンパイラに移植する必要がある場合は、
 *> ここにホスト変数を記述します。

 01  stateParam          pic xx.
 01  pubid               pic x(4).
 01  pubname             pic x(40).
 01  pubcity             pic x(20).
```

```
01  sql-stat            pic x(256).

EXEC SQL END DECLARE SECTION END-EXEC

PROCEDURE DIVISION.

EXEC SQL
    WHENEVER SQLERROR perform OpenESQL-Error
END-EXEC

EXEC SQL
  CONNECT TO 'SQLServer 2000' USER 'SA'
END-EXEC

*> プログラムロジックと SQL 文をここに記述します。

    EXEC SQL
            create procedure mfexecsptest
                (@stateParam char(2) = 'NY' ) as

            select pub_id, pub_name, city from publishers
                where state = @stateParam
    END-EXEC

    exec sql
        declare c1 scroll cursor for dsql2 for read only
    end-exec

    move "{call mfexecsptest(?)}" to sql-stat
    exec sql prepare dsql2 from :sql-stat end-exec

    move "CA" to stateParam
    exec sql
        open c1 using :stateParam
    end-exec

    display "ストアドプロシージャをもつカーソルをテストします"
    perform until exit
        exec sql
            fetch c1 into :pubid, :pubname, :pubcity
        end-exec

        if sqlcode = 100
            exec sql close c1 end-exec
            exit perform
        else
            display pubid " " pubname " " pubcity
        end-if
    end-perform

EXEC SQL close c1  END-EXEC

EXEC SQL DISCONNECT CURRENT END-EXEC
EXIT PROGRAM.
STOP RUN.
```

```
*> デフォルトの SQL エラールーチン。必要に応じて修正してプログラムを停止します。
OpenESQL-Error Section.

    display "SQL エラー = " sqlstate " " sqlcode
    display MFSQLMESSAGETEXT
*> 実行を停止します。
    exit.
```

第 7 章 : OpenESQL

OpenESQL プリプロセッサを使用すると、COBOL プログラム内に記述された埋め込み SQL 文から ODBC ドライバを通じてリレーショナルデータベースにアクセスできます。
個別のプリプロセッサとは異なり、OpenESQL は、アプリケーションのコンパイル時に SQL 指令を指定して制御します。
Net Express 3.0 以前を使用してアプリケーションをビルドした場合は、再コンパイルする必要があります。

7.1 ODBC ドライバとデータソース名

ODBC を使用するには、次の操作を行う必要があります。

1. ODBC ドライバのインストール
2. ODBC データソース名 (DSN) の設定

7.1.1 XDB ドライバのインストール

Net Express のインストール時に、「**SQL オプション**」を選択すると、XDB ドライバが自動的にインストールされます。

7.1.2 データソース名の設定

Net Express の ODBC サポートを使用するには、ODBC マネージャでデータソース名 (DSN) を設定する必要があります。ODBC マネージャは、Windows 98、Windows 2000、Windows Me、Windows XP Professional、または Windows NT デスクトップの [**コントロール パネル**] の [**管理ツール**] を使用して開くことができます。適切な [**32 ビット ODBC**] アイコンをクリックします。[**システム DSN**] タブをクリックします。
「ODBC データソースアドミニストレータ」ダイアログボックスが開きます。Net Express のドライバ名は **Micro Focus XDB** です。
DSN を割り当てる方法の詳細については、「ODBC データソースアドミニストレータ」ダイアログボックスの [**ヘルプ**] ボタンをクリックしてください。

7.2 ORACLE OCI サポート

OpenESQL では、開発者は ORACLE OCI インターフェイス形式で ORACLE データソースを使用することもできます。ORACLE OCI を使用すると、OpenESQL が通常使用する一般的な ODBC インターフェイスよりも SQL 文の処理を高速で行うことができます。このインターフェイスを使用する場合は、次の指令でアプリケーションをコンパイルする必要があります。

```
sql(targetdb=ORACLEOCI)
```

Oracle サーバに接続するときには、CONNECT 文で ODBC データソース名のかわりに Oracle Net8 サービス名を使用します。ORACLE Net8 サービスの設定方法については、Oracle のマニュアルを参照してください。
Oracle OCI を使用する場合は、次の OpenESQL 機能がサポートされないことに注意してください。

- BEGIN TRANSACTION 文
- CALL 文

- EXECSP 文
- QUERY ODBC 文
- SET AUTOCOMMIT 文
- SET TRANSACTION ISOLATION 文
- カーソルのスクロール (SET SCROLLOPTION 文と SET CONCURRENCY 文を含む)
- CONNECT 文の WITH PROMPT パラメータ
- SQL (CONNECTIONPOOL) 指令

7.3 SQL コンパイラ指令

プログラムのコンパイル時には、SQL コンパイラ指令と適切なオプションを指定する必要があります。これらを指定すれば、埋め込み SQL 文がプリプロセッサによって対応するデータソースの関数呼び出しに変換されます。プログラムから呼び出す ODBC ドライバは、アクセスする特定のデータソースによって決定されます。
SQL コンパイラ指令のオプションは、次の 2 通りの方法で指定できます。

- プログラム内に $SET 文で記述します。

  たとえば、次のように記述します。

  ```
  $set sql(dbman=odbc, datecheck, autocommit)
  ```

- Net Express の「**高度な指令**」画面で指定します。この画面を表示するには、[**プロジェクト**] メニューから [**ビルド設定**] を選択し、プログラムを選択します。そして、[**コンパイル**] タブをクリックして、[**SQL 指令**] ボタンを押します。

注：これらの 2 つの方法は併用できません。どちらかの方法を使用してください。

次の表に、DB2 コンパイラ指令のオプションを示します。

オプション	説明
ALLOWNULLCHAR	プログラムで PIC X(n) ホスト変数を使用したり、CHAR カラムで 16 進数文字を選択、挿入、更新したりできます。この場合は、ソースを変更して SQL TYPE BINARY ホスト変数を使用する必要はありません。
[NO]ANSI92ENTRY	OpenESQL を SQL ANSI 92 エントリレベル規格に準拠させます。
[NO]AUTOCOMMIT	各 SQL 文を個別のトランザクションとして処理し、実行後直ちにコミットします。このオプションを設定しないで、トランザクションに対応した ODBC ドライバを使用する場合には、文をトランザクションの一部として明示的にコミット (またはロールバック) する必要があります。
[NO]AUTOFETCH	SELECT 文に AUTOFETCH 属性を設定して、Microsoft SQL Server データソースで実行します。この指令を使用してコンパイルする場合は、アプリケーションのパフォーマンスが向上します。この指令は、プログラムを SQL(TARGETDB=MSSQLSERVER) 指令でコンパイルする

	場合のみに機能します。
[NO]CHECK	各 SQL 文のコンパイル時にデータベースに送信します。コンパイル時の文チェックを指定する場合は、DB と PASS も設定する必要があります。
CHECKDUPCURSOR	カーソルがすでにオープンし、プログラムが NOANSI92ENTRY 指令でコンパイルされている場合に、OpenESQL から SQLCODE -19516 が返されます。プログラムを NOANSI92ENTRY でコンパイルした場合は、通常、自動的にカーソルがクローズして再オープンします。
CHECKSINGLETON	実行時にシングルトンの SELECT が複数行を返すかどうか OpenESQL でチェックします。複数行が返される場合には、OpenESQL は SQLCODE を -811 に設定します。
CONCAT=ASCII 文字コード	CONCAT 記号 (|) に使用する ASCII 文字コードを指定します。この指令は、デフォルトの 33 を変更する場合のみに使用してください。
CONNECTIONPOOL=[DRIVER | ENVIRONMENT | **NONE**]	ODBC 3.0 の接続プールの使用を有効化します。接続プールを有効化する場合は、アプリケーションで閉じた接続がドライバマネージャでは一定時間にわたって維持されるため、アプリケーションで同じ接続を再使用するときに接続を再確立する手間が省けます。ODBC では、接続プールを、特定の ODBC 環境を対象にして設定するか、または、各ドライバ単位で設定できます。詳細については、ODBC のマニュアルを参照してください。このオプションは、接続のオープンとクローズが頻繁なアプリケーションのみに効果を発揮します。Microsoft Transaction Server (MTS) など、接続プール機能を独自に実装している環境もあります。このオプションは、MTS を実行していない環境で ISAPI アプリケーションのパフォーマンスを向上させる場合などに効果的です。デフォルトは NONE です。
[NO]CTRACE	デバッグ情報を sqltrace.txt ファイルに記述します。デフォルトは NOCTRACE です。
[NO]CURSORCASE	ESQLVERSION の値が 2.0 であれば CURSORCASE が暗黙で指定されます。デフォルトは NOCURSORCASE で、カーソル名の大文字と小文字が区別されません。CURSORCASE の場合は、大文字と小文字が区別されます。以前のバージョンの OpenESQL では、カーソル名の大文字と小文字は区別されていました。
[NO]DB	接続するデータソースの名前。このオプションは、INIT や CHECK オプション (またはその両方) とともに使用します。
DBMAN=*preprocessor*	使用するプリプロセッサを指定します。この値は常に odbc に設定されるため、dbman=odbc となります。この指令は、OpenESQL でプログラムをコンパイルするときは不要です。
[NO]DECDEL	OpenESQL で、10 進数データを格納する変数に対してルーチン GetLocaleInfo を呼び出す手間を省きます。このルーチンは、変数にアクセスするたびに呼び出され、使用される 10 進数の区切り文字を返します。多くの場合には、区切り文字はピリオドまたはコンマです。この指令のオプションは次のとおりです。

	DECDEL=PERIOD　10 進数の区切り文字として常にピリオドを使用します。 DECDEL=COMMA　10 進数の区切り文字として常にコンマを使用します。 DECDEL=LOCAL　GetLocaleInfo を 1 回呼び出して 10 進数の区切り文字を取得します。 デフォルトは NODECDEL で、10 進数変数が参照されるたびに OpenESQL で GetLocaleInfo が呼び出されます。
[NO]DETECTDATE	DETECTDATE を設定する場合には、OpenESQL は ODBC エスケープシーケンスで文字ホスト変数を検査します。 {d<データ>} - 日付 {t<データ>} - 時刻 {ts<データ>} - タイムスタンプ 検出されたパラメータは適切にバインドされ、文字カラムとしては使用されません。この処理は、サーバがネイティブな日付表記文字列をサポートしていない場合 (Microsoft Access など) に必要です。また、汎用的なアプリケーションでも有用です。ただし、「{d」、「{t」、または「{ts」で開始するデータを正当に含む他の文字カラムが存在する場合には問題が発生する可能性があります。デフォルトは NODETECTDATE です。
[NO]ESQLVERSION	OpenESQL の構文レベルを設定します。
[NO]FIPSFLAG	FIPS フラグを有効にします (NIST 証明が必要な場合のみ)。デフォルトは NOFIPSFLAG です。
GEN-CC2	OpenESQL データベースインターフェイスの呼び出しを、呼び出し規約 74 ではなく呼び出し規約 2 を使用して生成します。Net Express 3.0 と同じ方法でアプリケーションを生成する場合にこの指令を使用します。**.exe** を作成する場合は、**odbcrw32.lib** をリンク文に追加する必要があります。
[NO]IGNORE-NESTED=program-id	入れ子のプログラムで、データベースインターフェイスコードの生成を開始する program-id を指定します。プログラムファイル名が program-id と一致する場合は、IGNORE-NESTED を指定するのみです。デフォルトは NOIGNORE-NESTED です。
[NO]INIT	プリプロセッサはデータベース接続用のコードを自動生成します。INIT を指定する場合には、**DB** と **PASS** も指定する必要があります。この指令のかわりにアプリケーションで EXEC SQL CONNECT 文を使用することをお奨めします。
[NO]NIST	OpenESQL の動作を NIST による SQL ANSI 92 エントリレベル規格に準拠させます。
NOT=ASCII 文字コード	NOT 文字 (¬) に使用する ASCII 文字コードを指定します。この指令は、デフォルトの 170 を変更する場合のみに使用してください。
ODBCTRACE= [ALWAYS | NEVER | **USER**]	ODBCTRACE=USER を指定する場合は、ODBC トレース処理を OBDC コントロールパネルで制御できます。このコントロールパネルでトレース対象のファイルを指定できます。ALWAYS は OBDC トレース処理を指令で制御する設定であり、IDE での

	アプリケーション開発に適しています。ODBC コントロールパネルの設定に関わらず、この設定では常に現在のディレクトリにトレースファイル MFSQLTRACE.LOG が生成されます。通常の開発では、プロジェクトのビルド設定に応じて、現在のプロジェクトの Debug ディレクトリまたは Release ディレクトリが使用されます。NEVER を設定する場合は、アプリケーションのトレースは実行されず、コントロールパネルの設定も無視されます。ODBC トレースファイルには重要な情報が含まれる場合があるため、運用時に情報の安全性を重視する場合には NEVER を使用します。デフォルトは USER です。
[NO]PARAMARRAY	すべての入力パラメータに ODBC 配列バインドを使用します (ODBC ドライバが同機能をサポートしている場合のみ)。デフォルトは PARAMARRAY です。
[NO]PASS	データソースへの接続に使用するログイン名。このオプションは、**INIT** や **CHECK** オプション (またはその両方) とともに使用します。
[NO]PRE	デフォルトは PRE です。この場合は、実行時にプリプロセッサで OpenESQL ランタイムモジュール (**odbcrw32.dll**) を動的にロードするコードが生成されます。この設定は、LITLINK コンパイラ指令と競合します。このため、LITLINK を使用する場合には、NOPRE を指定して動的ロードのコード生成を停止します。この場合は、ビルド設定で、リンクする LIB のリストに **odbcrw32.lib** を追加する必要があります。これにより、リンカがコードを実行可能ファイルとして生成する場合には、その実行可能ファイルのロード時にオペレーティングシステムは **odbcrw32.dll** を暗黙でロードします。
[NO]QUALFIX	ホスト変数を ODBC に宣言するときに、プリプロセッサがこれらのホスト変数の名前に 3 文字を追加します。デフォルトは QUALFIX です。
[NO]RESULTARRAY	すべての出力パラメータに ODBC 配列バインドを使用します (ODBC ドライバが同機能をサポートしている場合のみ)。デフォルトは RESULTARRAY です。
STMTCACHE	OpenESQL で使用される PREPARE 文で定義した文のキャッシュサイズを設定します。デフォルトは 20 です。この値にデフォルト値よりも大きい値を設定する場合は、使用しているアプリケーションとデータソースによって、パフォーマンスが向上する場合と、データエラーが発生する場合があります。
[NO]TARGETDB=[MSQLSERVER | ORACLEOCI | ORACLE | INFORMIX | SYBASE | DB2 | ORACLE7]	特定のデータソースに対してパフォーマンスを最適化する場合や、アプリケーションで ODBC 呼び出しではなく ORACLE OCI を使用してデータベース呼び出しを生成する場合は、この指令を設定してください。
THREAD=[**SHARE** | ISOLATE]	ISOLATE を設定する場合は、すべての接続やカーソルなどがそれらを作成したスレッドごとに生成されます。この設定は、マルチスレッドのアプリケーションサーバ環境 (IIS/ISAPI など) で必要です。SHARE に指定した場合は、CONNET 文とスレッド 1 をハードコードして実行した後に、スレッド 2 で同じ文を実行する場合に、最初のスレッドですでに接続がオープンしているため、

	スレッド 2 ではエラーが発生します。ISOLATE に指定した場合は、スレッドごとに独自の接続が確立されます。デフォルトは SHARE です。
[NO]USECURLIB[NO | YES | **IFNEEDED**]	ODBC のカーソルライブラリの使用を制御します。カーソルライブラリは、基礎となるドライバがスクロールカーソルの機能をサポートしない場合にスクロールカーソルのサポートを提供します。また、位置指定更新を「シミュレート」します。USECURLIB=YES を設定する場合は、常にカーソルライブラリが使用されます。USERCURLIB=NO を設定する場合には、カーソルライブラリは使用されません。デフォルトの USERCURLIB=IFNEEDED を設定する場合は、ドライバマネージャがドライバがサポートしていないと判断した機能をアプリケーションが実行しようとする場合に、カーソルライブラリが使用されます。スクロールカーソル機能をカーソルライブラリで使用する場合は、STATIC カーソルを使用する必要があります。カーソルライブラリを使用して位置指定更新を実行する場合は、OPTCCVAL 並行性を使用する必要があります。「シミュレートされた」位置指定更新は複数の行を選択する可能性があることに注意してください。このため、SELECT に主キーを含めることをお奨めします。

7.4 デバッグファイルの作成方法

プログラムのコンパイル時にサポート窓口に問い合わせする必要のあるエラーが発生した場合には、サポート窓口の担当者は、問題の原因を特定するために追加のデバッグファイルの提供を求める場合があります。サポート窓口の担当者は、問題を再現するために、ソースファイルとデータファイルの他に、3 つのデバッグファイルのうち 1 つ以上の提供を求める場合があります。これらの指令のいくつかを指定すると、独自でデバッグする場合に役に立ちます。次の指定があります。

指令	作成ファイル	ファイル内の情報
CHKECM(CTRACE)	ecmtrace.txt	このファイルには、EXEC SQL 文を置き換えるために生成されるコードを示す擬似 COBOL コードが含まれます。このファイルは、OpenESQL ODBC プリコンパイラで生成される出力ファイルと等価です。
CHKECM(TRACE)	ecmtrace.txt	このファイルには、ODBC ECM とコンパイラ間で受け渡しされる情報に関する詳細情報が含まれます。無効な構文を生成するエラーが発生した場合に、このファイルを使用して、問題が発生した箇所を分離できます。
SQL(CTRACE)	sqltrace.txt	このファイルには、OpenESQL プリコンパイラサービスに渡される情報の詳細なリストと、その結果が含まれます。このファイルは、エラーが OpenESQL ECM のみでなく OpenESQL ランタイムのバグに関連する場合にも役に立ちます。
ECMLIST	*program-name*.lst	プログラム全体に対して生成された COBOL コードを COBOL リストファイルに生成します。CHKECM(CTRACE) 指令と LIST 指令を使用してプ

ログラムをコンパイルする必要があります。

7.5 データベース接続

プログラムからデータベース内のデータにアクセスするには、その前にデータベースへの接続を確立する必要があります。
プログラムは、次の 2 通りの方法でデータベースに接続できます。

- 明示的接続 (推奨方法)

  通常、プログラムがコンパイル時にデータベース名を特定できない異なるデータソースにアクセスする場合や、複数のデータベースにアクセスする場合に CONNECT 文を使用します。

- 暗黙的接続

  通常、プログラムがコンパイル時に特定の 1 つのデータベースに接続する場合のみに使用します。SQL コンパイラ指令の INIT オプションを指定すると、SQL コンパイラ指令の DB オプションでデータソースに PASS オプションで指定されたログイン情報を使用して自動接続する呼び出しが、コンパイラによってプログラムの開始時に挿入されます。

アプリケーションによるデータベース操作が完了するときには、データベースとの接続を解除する必要があります。この処理には、DISCONNECT 文を使用します。
暗黙的接続を使用している場合には、プログラムの終了時に OpenESQL によってデータソースとの接続が自動的に解除されます。
プログラムの異常終了時に、OpenESQL に接続の解除とロールバックを暗黙的に実行させるには、SQL コンパイラ指令の INIT=PROT オプションを指定します。

7.6 キーワード

OpenESQL では数多くのキーワードが予約されています。これらのキーワードはプログラム内で他の用途に使用することはできません。予約されているキーワードの完全なリストについては、オンラインヘルプに記載されています。ヘルプファイルの索引で「OpenESQL」を選択してください。

7.7 アプリケーションのビルド

OpenESQL アプリケーションをビルドするには、次の操作を行う必要があります。

1. SQL 文を EXEC SQL と END-EXEC のキーワードで囲んで、アプリケーションを記述します。Micro Focus Server Express も使用している場合は、PC で OpenESQL アシスタントを使用してアプリケーションを開発し、UNIX システムへパブリッシュする方法が簡単です。Net Express OpenESQL アシスタントでは、SQL 文の開発にグラフィカルなドラッグアンドドロップの方法を提供しています。
2. SQL コンパイラ指令を使用してアプリケーションをコンパイルします。
3. [コントロール パネル] の [ODBC データソース] でデータソースを設定します (『*データソース名の設定*』を参照)。

Net Express の基本インストールディレクトリの source ディレクトリには、sqlca.cpy と sqlda.cpy のコピーファイルがインストールされており、通常の方法でプログラムにインクルードできます。
Net Express を使用して .exe を作成する場合は、SQL(GEN-CC2) 指令でプログラムをコンパイルしていない限り、必要なオブジェクトファイルがすべて自動的にリンクされます。

注：作成したアプリケーションを他のシステムに移動する場合には、移動先のシステムに odbcrw32.dll ファイルが存在することを確認してください。

7.8 デモンストレーションアプリケーション

Net Express の基本インストールディレクトリの demo ディレクトリにある odbcesql ディレクトリには、さまざまなデモンストレーションアプリケーションが提供されています。
デモンストレーションアプリケーションを使用するには、1 つ以上の ODBC ドライバと、デモンストレーションに使用するために作成した DNS をインストールする必要があります。
デモンストレーションアプリケーションの中には、接続しているデータベースに EMP というテーブルが存在することを要求するものもあります。
すべての OpenESQL デモンストレーションアプリケーションではコンソールログが作成され、処理の進行状況を表示し、クエリー結果を表示する場合もあります。処理中にエラーが発生する場合は、エラーメッセージを表示して終了します。

- **connect.app**

  データベースソース名、ユーザ名、およびパスワードの入力を求めます。作成したデータベース名とユーザ名「admin」を入力し、パスワードは空欄のまま Enter キーを押します。異なる構文オプションを使用した 4 種類の接続および接続解除テストが実行されます。5 番目のテストでは、「SQL データソース」ダイアログが表示されます。「マシンデータソース」リストから適切な名前を選択し、[OK] をクリックします。「ログイン」ダイアログが表示されます。ログイン名として「admin」を入力し、パスワードは空欄のまま [OK] をクリックします。5 番目のテストが実行され、プログラムが終了します。

- **select.app**

  サンプルデータベースに接続し、顧客コードの入力を求めます。メッセージに表示された BLUEL を入力します。該当の顧客コードの 2 つのフィールドが表示され、顧客コードの入力が再要求されます。ここで Enter キーを押します。地域の入力が求められます。メッセージに表示された CA を入力します。その地域の顧客が一覧表示され、さらに地域の入力が求められます。ここで、Enter キーを押すと、プログラムが終了します。

- **static.app および dynamic.app**

  この 2 つのアプリケーションは、複数の同じテストを異なる SQL 構文オプションで実行します。どちらを起動しても、データソースとユーザ名の入力を求められます。テストのシーケンスは次のとおりです。

  接続
  テストテーブルの削除
  テストテーブルの作成
  行の挿入

コミット
行の更新
読み取りと検証
ロールバック
読み取りと検証
テストテーブルの削除
接続の解除
テストテーブルの作成

2 番目のテストではエラーメッセージが表示されることがありますが、これは意図的に生成されたものであり、プログラムは続行されます。最後のテストでもエラーメッセージが表示されますが、これについても同様です。逆に、この段階で ODBC エラーが表示されない場合には、テストは失敗です。

- **whenever.app**

  接続を試行して、エラーメッセージを表示します。「SQL データソース」ダイアログが表示されます。DSN 名を選択し、[**OK**] をクリックします。「ログイン」ダイアログが表示されます。ログイン名として「admin」を入力し、パスワードは空欄のまま [**OK**] をクリックします。意図的にエラーが生成され、2 つのエラーメッセージが表示されます。

- **catalog.app**

  「SQL データソース」ダイアログが表示されます。名前を選択するか、または入力し、[**OK**] をクリックします。「ログイン」ダイアログが表示されます。ログイン名として「admin」を入力し、パスワードは空欄のまま [**OK**] をクリックします。3 種類のデータディクショナリクエリーが実行され、結果が出力されます。

7.9 トランザクションの管理

OpenESQL では、COMMIT 文と ROLLBACK 文を使用して、ODBC のトランザクション管理機能を制御できます。ODBC では各文の実行後にトランザクションが標準で自動コミットされるように指定しても、OpenESQL では他の SQL システムとの互換性を考慮して、この機能は無効化されています。この機能が必要な場合は、SQL コンパイラ指令の AUTOCOMMIT オプションを指定してください。

注：トランザクション処理機能を実装していない ODBC ドライバもあります。そのようなドライバでは、データベースの更新が直ちに確定されない可能性があります。詳細については、使用しているデータベースドライバのマニュアルを参照してください。

7.10 データ型

オンラインヘルプには、SQL のデータ型と COBOL データ形式を変換する際に OpenESQL で使用する対応関係を示す表が記載されています。オンラインヘルプの目次で、『*リファレンス*』、『*データベースアクセス*』、『*OpenESQL*』、『*データ型*』を選択してください。
ODBC では、日付は *yyyy-mm-dd*、時刻は *hh:mm:ss* 形式で表記されます。これらの形式は、使用するデータソースのネイティブな日付や時刻の表記形式と一致しない可能性があります。入力文字ホスト

変数には、データソースのネイティブな日付日時形式を使用できます。ほとんどのデータソースでは、PICTURE 句 PIC X(29) を使用することをお奨めします。次に例を示します。

```
 01  mydate      PIC x(29).
 ...
     EXEC SQL
         INSERT INTO TABLE1 VALUES (1,'1997-01-24 12:24')
     END-EXEC
 ...
     EXEC SQL
         SELECT DT INTO :mydate FROM TABLE1 WHERE X = 1
     END-EXEC
     display mydate
```

また、ODBC エスケープシーケンスを使用することもできます。ODBC では、日付、時刻、およびタイムスタンプの各定数用のエスケープシーケンスが定義されています。これらのエスケープシーケンスは、ODBC ドライバによって認識され、データソース側の構文に変換されます。

エスケープシーケンスによる、日付、時刻、およびタイムスタンプの各定数の表記方法は、次のとおりです。

{d 'yyyy-mm-dd'} - 日付

{t 'hh:mm:ss'} - 時刻

{ts yyyy-mm-dd hh:mm:ss[.f...] - タイムスタンプ

日付、時刻、およびタイムスタンプのエスケープシーケンスを使用したコード例を、次に示します。

```
 working-storage section.
 EXEC SQL INCLUDE SQLCA END-EXEC

 01  date-field1      pic x(29).
 01  date-field2      pic x(29).
 01  date-field3      pic x(29).

 procedure division.
     EXEC SQL
        CONNECT TO 'Net Express 4.0 Sample 1' USER 'admin'
     END-EXEC
* テーブルが存在する場合は削除します。
     EXEC SQL
        DROP TABLE DT
     END-EXEC

* DATE、TIME、および DATE/TIME のカラムをもつテーブルを作成します。
* 注：Access では、これら 3 つのカラムのかわりに DATETIME を使用します。
*     専用のカラム型を使用するデータベースもあります。
* 別のデータソースに DATE/TIME カラムを作成する場合には、
* 該当データベースのマニュアルでカラムの定義方法を
* 確認してください。

     EXEC SQL
        CREATE TABLE DT ( id  INT,
                          myDate DATE NULL,
                          myTime TIME NULL,
                          myTimestamp TIMESTAMP NULL)
     END-EXEC

* ODBC エスケープシーケンスを使用してテーブルに挿入します。

     EXEC SQL
        INSERT into DT values (1 ,
```

```
            {d '1961-10-08'},  *> 日付を設定します。
            {t '12:21:54'  },  *> 時刻を設定します。
            {ts '1966-01-24 08:21:56' } *> 日付と時刻を設定します。
                                )
     END-EXEC
```

* 挿入した値を取り込みます。

```
     EXEC SQL
        SELECT myDate
              ,myTime
              ,myTimestamp
         INTO  :date-field1
              ,:date-field2
              ,:date-field3
         FROM DT
             where id = 1
     END-EXEC
```

* 取り込み結果を表示します。

```
     display 'ここに日付が設定されています :'
             date-field1
     display 'ここに時刻が設定されています :'
             date-field2
     display '注：ほとんどのデータソースは、日付に対して'
             'デフォルトが設定されます'
     display 'ここに日付と時刻が設定されています :'
             date-field3
```

* テーブルを削除します。

```
     EXEC SQL
        DROP TABLE DT
     END-EXEC
```

* データソースから接続を解除します。

```
     EXEC SQL
        DISCONNECT CURRENT
     END-EXEC

     stop run.
```

この例では、日付 / 時刻変数に SQL TYPE で定義されたホスト変数を使用できます。次のホスト変数を定義できます。

```
01  my-id          pic s9(08) COMP-5.
01  my-date        sql type is date.
01  my-time        sql type is time.
01  my-timestamp   sql type is timestamp.
```

INSERT 文を次のコードに書き換えます。

```
*> SQL TYPE ホスト変数を使用したテーブルへの挿入
    MOVE 1                          TO  MY-ID
    MOVE "1961-10-08"               TO  MY-DATE
    MOVE "12:21:54"                 TO  MY-TIME
    MOVE "1966-01-24 08:21:56"      TO  MY-TIMESTAMP
```

```
EXEC SQL
   INSERT into DT value (
     :MY-ID
    ,:MY-DATE
    ,:MY-TIME
    ,:MY-TIMESTAMP  )
END-EXEC
```

7.11 SQLCA の使用方法

SQLCA データ構造体は、Net Express 基本インストールディレクトリの **source** ディレクトリ内にある **sqlca.cpy** ファイルで定義されています。SQLCA をプログラムで使用するには、データ部に次の文を記述します。

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

この文を記述しない場合でも、COBOL コンパイラによって自動的に領域が割り当てられますが、プログラムではその領域にはアクセスできません。ただし、SQLCODE または SQLSTATE のデータ項目を宣言する場合は、各 EXEC SQL 文の後に SQLCA の対応フィールドをユーザ定義フィールドにコピーするコードが COBOL コンパイラによって生成されます。
MFSQLMESSAGETEXT データ項目を宣言した場合には、SQLCODE にゼロ以外の値が検出されるたびに、例外条件の説明によって同データ項目が更新されます。MFSQLMESSAGETEXT は、文字列データ項目 PIC x(*n*) として宣言します。*n* には、有効値を入力します。ODBC エラーメッセージは、70 バイトの SQLCA メッセージフィールドを超えることがあるので、MFSQLMESSAGETEXT データ項目は特に有用です。

注：SQLCA、SQLCODE、SQLSTATE、または MFSQLMESSAGETEXT はホスト変数として宣言する必要はありません。

7.12 動的 SQL

動的 SQL のデモンストレーションアプリケーション **dynamic.app** は、Net Express の基本インストールディレクトリの **demo** ディレクトリ内の **odbcesql** ディレクトリに格納されています。このプロジェクトを開き、必要に応じてプロジェクトをリビルドした後に、[**アニメート**] メニューから [**ステップ実行**] を選択すると、アプリケーションがステップ実行され、COBOL プログラムでの動的 SQL の使用方法が具体例を通じて示されます。

7.13 位置指定更新

ODBC は、位置指定更新をサポートします。位置指定更新では、カーソルを使用して最後に取り込まれた行が更新されます。ただし、位置指定更新をサポートしないドライバもあります。

注：位置指定更新ではホスト配列を使用できません。

ODBC ドライバによっては、位置指定更新を有効にするために、カーソルで使用される SELECT 文に FOR UPDATE 句を含める必要があります。多くのデータソースでは、SET 文または DECLARE CURSOR 文のどちらかで SCROLLOPTION と CONCURRENCY の特定の組み合わせを指定することが必要です。これが機能しない場合は、ODBC カーソルライブラリにより、位置指定更新が制限されて実装されます。この位置指定更新は、指令 SQL(USECURLIB=YES) および SCROLLOPTION

STATIC と CONCURRENCY OPTCCVAL (または OPTIMISTIC) を使用してコンパイルすると有効化できます。ODBC カーソルライブラリを使用しているときに複数の行が更新されないようにするには、カーソルクエリーで、更新するテーブルに主キーカラムを含めます。

例

```
$SET SQL(usecurlib=yes)
  WORKING-STORAGE SECTION.

  EXEC SQL INCLUDE SQLCA  END-EXEC

  *> SQL エラーが発生する場合は、ここに詳細なメッセージテキストが示されます。
  01 MFSQLMESSAGETEXT  PIC X(250).
  01 IDX               PIC X(04)  COMP-5.

  EXEC SQL BEGIN DECLARE SECTION  END-EXEC
  *> 他の COBOL コンパイラに移植する必要がある場合は、
  *> ここにホスト変数を記述します。
  EXEC SQL INCLUDE Products END-EXEC

  EXEC SQL END DECLARE SECTION END-EXEC

  PROCEDURE DIVISION.

  EXEC SQL
      WHENEVER SQLERROR perform OpenESQL-Error
  END-EXEC

  *> ACCESS データソースを使用して位置指定更新を行うデモです。
  EXEC SQL
    CONNECT TO 'Inventory' USER 'admin'
  END-EXEC

  *> プログラムロジックと SQL 文をここに記述します。

      EXEC SQL
       DECLARE CSR679 CURSOR
         FOR SELECT
              A.ProductID
             ,A.ProductName
             ,A.UnitPrice
         FROM Products A
        WHERE ( A.ProductID <  3 )
      END-EXEC

     EXEC SQL SET SCROLLOPTION static  END-EXEC
     EXEC SQL SET CONCURRENCY optccval END-EXEC

      EXEC SQL OPEN CSR679 END-EXEC
      PERFORM UNTIL SQLSTATE >= "02000"
        EXEC SQL
        FETCH CSR679 INTO
            :ProductID
           ,:ProductName:ProductName-NULL
           ,:UnitPrice:UnitPrice-NULL
        END-EXEC
```

```
        *> FETCH 処理のデータを処理します。
        IF SQLSTATE = "00000"
           *> 価格を 10% 上げます。
            compute unitprice = unitprice * 1.10
           EXEC SQL
               UPDATE Products
                 SET UnitPrice = :UnitPrice:UnitPrice-NULL
               WHERE CURRENT OF CSR679
           END-EXEC
        END-IF
      END-PERFORM
      EXEC SQL CLOSE CSR679 END-EXEC

    EXEC SQL COMMIT END-EXEC

  EXEC SQL DISCONNECT CURRENT END-EXEC
  EXIT PROGRAM.
  STOP RUN.
  *> デフォルトの SQL エラールーチン。必要に応じて修正してプログラムを停止します。
  OpenESQL-Error Section.

      display "SQL エラー = " sqlstate " " sqlcode
      display MFSQLMESSAGETEXT
  *> 実行を停止します。
      exit.
```

7.14 Web サーバおよびアプリケーションサーバでの OpenESQL の使用方法

ここでは、IIS、MTS、COM+、CICS、Tuxedo などの Web サーバおよびアプリケーションサーバで制御される環境で OpenESQL を使用するときに実行する作業について説明します。
サーバの種類は異なりますが、ここでは、すべてのサーバに共通の内容を説明します。

7.14.1 スレッドセーフティ

OpenESQL はスレッドセーフです。通常、アプリケーション内のすべてのスレッドは、接続やカーソルなどの SQL リソースを共有します。ただし、アプリケーションサーバで実行している場合には、スレッドは、異なるユーザからの要求を処理できるようにスケジュールされます。このため、次の指令を使用します。
SQL(THREAD=ISOLATE)
これにより各スレッドのリソースをそれぞれ分離する必要があります。
注：OpenESQL では、スレッドごとに 72KB のオーバーヘッドをもちます。このメモリは、アプリケーションが終了するまで解放されません。

7.14.2 接続管理

多くの環境では、アプリケーションサーバが接続プールを管理します。つまり、データベースに対して実際の接続と接続解除の要求を行うことはほとんどありません。アプリケーションが実行されると、既存の接続が再使用されます。多くの場合には、接続プーリングは ODBC ドライバマネージャで管理され、アプリケーションに対して透過的です。アプリケーションサーバ自体が接続プールを管理する場合は、アプリケーションは他の処理を実行する前に「SET CONNECTION」文を使用する必要があります。

アプリケーションで OpenESQL の CONNECT 文と DISCONNECT 文を使用し、アプリケーションサーバ自体が ODBC 接続プーリングを有効化しているか不明な場合は、SQL(CONNECTIONPOOL=...) 指令を使用することもできます。ただし、この方法を使用することはほとんどありません。

7.14.3 トランザクション

多くの場合には、アプリケーションサーバはトランザクション管理を提供します。これは、構成要素をアプリケーションサーバの制御下に置くときに決定されます。アプリケーションサーバがトランザクション管理を提供していない場合は、OpenESQL の COMMIT 文と ROLLBACK 文を使用してトランザクションを管理する必要があります。ただし、アプリケーションサーバでトランザクション管理を提供する場合は、次の作業を実行する必要があります。

- 接続管理がアプリケーションサーバで提供されない場合 (つまり、CONNECT 文と DISCONNECT 文を使用する場合) は、SQL(AUTOCOMMIT) 指令を使用する必要があります。これは、SQL 文が自動的にコミットされることを意味するのではなく、アプリケーションが COMMIT 文と ROLLBACK 文でトランザクションを処理しないことを表します。
- アプリケーションでは、OpenESQL の COMMIT 文や ROLLBACK 文は使用できません。そのかわりに、アプリケーションサーバが API 呼び出しを提供するか、または、処理の成功や失敗を表すコードを返します。

MTS または COM+ を使用している場合は、アプリケーションサーバがトランザクションを管理しているときに、デフォルトのトランザクション分離レベルをシリアル化できます。これによって、過剰なロックが行われたり、並行性が低下したりすることがあります。アプリケーションサーバでデッドロックを解決しようとすると異常終了するトランザクションを処理できるようにアプリケーションを準備する必要があります。これらの問題を解決するには、次の文を使用します。

```
exec sql set transaction isolation read committed end-exec
```

これにより厳格性の低い分離レベルを設定します。この場合は、この文を、CONNEDT 文の直後に実行する文にする必要があります。SQL(AUTOCOMMIT) が使用されていない場合は、SET TRANSACTION ISOLATION 文を実行する直前にコミットまたはロールバックを実行する必要があります。

7.14.4 ユーザアカウント、スキーマ、および認証

アプリケーションサーバ環境で実行する場合は、アプリケーションを実行するユーザアカウントが、開発用のアカウントと異なることがあります。このため、ユーザデータソースとして設定された ODBC データソースが使用できない場合があります。実装システムでデータソースをシステムデータソースとして設定すると便利な場合があります。ファイルベースのデータソースを使用する場合は、アプリケーションサーバで使用されるアカウントでデータベースファイルにアクセスできるようにする必要があります。データベースへのアクセス時、特に統合セキュリティが使用されている場合 (DBMS がオペレーティングシステムと同じアカウント番号を使用している場合) には、デフォルトのスキーマが、開発時に使用されていたスキーマと異なる場合があります。開発および実装にそれぞれ別のスキーマ名が使用されている場合には、これは意図的なものである可能性があります。また、テーブルがアプリケーションに表示されていないことを意味する場合もあります。明示的なオーナー修飾語を使用するか、または、データベース固有の文を実行する場合は、正しいスキーマがデフォルトとして選択されます。

7.14.5 トランザクションラッパーのサンプル

次に、OCX ウィザードで生成されたトランザクションラッパーの例を示します。このトランザクションラッパーは、SQL Server データソースを使用して次のシナリオを処理する OpenESQL ロジックを含めるために変更されています。

- MTS/COM+ が関連しないスタンドアローンのトランザクション
- MTS/COM+ で管理されているが、MTS/COM+ でトランザクションが処理されない構成要素
- MTS/COM+ で処理される構成要素とトランザクション

```
 $set ooctrl(+p) sql(thread=isolate autocommit)
      *>-------------------------------------------------------------
      *> クラスの記述
      *>-------------------------------------------------------------
       class-id. cblsqlwrapper
                 inherits from olebase.
       object section.
       class-control.
           cblsqlwrapper is class "cblsqlwrapper"
      *> OCWIZARD - クラスの一覧表示の開始
       objectcontext is class "objectcontext"
           olebase is class "olebase"
           oleSafeArray is class "olesafea"
           oleVariant is class "olevar"
      *> OCWIZARD - クラスの一覧表示の終了
      *>---USER-CODE。次のクラス名を追加します。
           *>-------------------------------------------------------------
--
       working-storage section. *> グローバルデータの定義
      *>-------------------------------------------------------------

      *>-------------------------------------------------------------
       class-object.   *> クラスデータとメソッドの定義
      *>-------------------------------------------------------------
       object-storage section.

      *> OCWIZARD - 標準クラスメソッドの開始
      *>-------------------------------------------------------------
      *> クラスに関する詳細情報を返します。
      *> タイプライブラリがある場合は、この theClassId と theInterfaceId
      *> が一致する必要があります。
      *> theProgId はこのクラスのレジストリエントリと一致する必要があります。
      *>   (ゼロ長文字列はクラスファイル名を暗黙で使用します。)
      *> theClassId はレジストリに保存された CLSID と一致する必要があります。
      *> theVersion は現在無視されています (デフォルトの 1 を使用)。
      *>-------------------------------------------------------------
       method-id. queryClassInfo.
       linkage section.
       01 theProgId             pic x(256).
       01 theClassId            pic x(39).
       01 theInterfceId         pic x(39).
       01 theVersion            pic x(4) comp-5.
       01 theDescription        pic x(256).
       01 theThreadModel        pic x(20).
       procedure division using by reference theProgId
                                by reference theClassId
```

```
                                by reference theInterfceId
                                by reference theVersion
                                by reference theDescription
                                by reference theThreadModel.
         move z"{3EADD92C-06C5-46F2-A2E0-7EB0794C14DF}" to theClassId
         move z"{5BF3F966-9932-4835-BFF6-2582CA2592AD}" to
theInterfceId
         move z"Description for class cblsqlwrapper"
              to theDescription
         move z"Apartment" to theThreadModel
         exit method.
       end method queryClassInfo.
      .
      *>-------------------------------------------------------------
      *> タイプライブラリの詳細を返します。使用しない場合は、削除してください。
      *> theLocale は現在無視されています (デフォルトの 0 を使用).
      *> theLibraryName は自動登録に使用する NULL 終了文字列で、
      *> 次の値をサポートします。
      *>    <no string> - このバイナリにはライブラリが埋められます。
      *>    <number>    - 上記と同様に、このリソース番号が使用されます。
      *>    <Path>      - ライブラリはこの位置にあります (完全なパス)。
      *>-------------------------------------------------------------
       method-id. queryLibraryInfo.
       linkage section.
       01 theLibraryName        pic x(512).
       01 theMajorVersion       pic x(4) comp-5.
       01 theMinorVersion       pic x(4) comp-5.
       01 theLibraryId          pic x(39).
       01 theLocale             pic x(4) comp-5.
       procedure division using by reference theLibraryName
                                by reference theMajorVersion
                                by reference theMinorVersion
                                by reference theLibraryId
                                by reference theLocale.
         move 1 to theMajorVersion
         move 0 to theMinorVersion
         move z"{24207F46-7136-4285-A660-4594F5EE7B87}" to
theLibraryId
         exit method.
       end method queryLibraryInfo.

      *>-------------------------------------------------------------

      *> OCWIZARD - 標準クラスメソッドの終了

       end class-object.

      *>-------------------------------------------------------------
       object.         *> インスタンスデータとメソッドの定義
      *>-------------------------------------------------------------
       object-storage section.

      *> OCWIZARD - 標準インスタンスメソッドの開始
      *> OCWIZARD - 標準インスタンスメソッドの終了
```

```
      *>-----------------------------------------------------------------
-
       method-id. "RetrieveString".
       working-storage section.

       01 mfsqlmessagetext pic x(400).
       01 ESQLAction       pic x(100).

       COPY DFHEIBLK.

       COPY SQLCA.
      *> トランザクションプログラム名
       01 transactionPgm           PIC X(7) VALUE 'mytran'.


       local-storage section.
       01 theContext              object reference.
       01 transactionStatusFlag   pic 9.
         88 transactionPassed     value 1.
         88 transactionFailed     value 0.
      *>---USER-CODE。次の必要な局所場所項目を追加します。

       01 ReturnValue             pic x(4) comp-5.
         88 IsNotInTransaction    value 0.

       01 transactionControlFlag  pic 9.
         88 TxnControlledByMTS    value 0.
         88 TxnNotControlledByMTS value 1.

       linkage section.

      *> トランザクションに渡される情報
       01 transaction-Info.
          05 transaction-Info-RC   pic 9.
          05 transaction-Info-data pic x(100).

      *> トランザクションから返される情報
       01 transaction-Info-Returned pic x(100).


       procedure division using by reference transaction-Info
                                returning  transaction-Info-Returned.

      *> 初期化コード
           perform A-Initialise
           perform B-ConnectToDB
           if TxnNotControlledByMTS
              perform C-SetAutoCommitOff
           end-if

      *> 分離レベルを設定して、SQLServer のデフォルト、serialize を上書きします。
           perform D-ResetDefaultIsolationLevel

      *> カーソルタイプを設定し、OpenESQL のデフォルト dynamic+lock を上書きし
ます。
           perform E-ResetDefaultCursorType
```

```
      *> トランザクションを呼び出します。
           perform F-CallTransaction

      *> コードの終了 - MTS/COM+ で制御されていない場合は、Commit/Rollback を
      *> 実行します。
           if TxnNotControlledByMTS
              if transactionPassed
                 perform X-Commit
              else
                 perform X-Rollback
           end-if

           perform Y-Disconnect

      *> トランザクションサーバ - メソッドに失敗した場合は、setAbort を使用しま
す。
           if theContext not = null
              if transactionPassed
                 invoke theContext "setComplete"
              else
                 invoke theContext "setAbort"
              end-if
              invoke theContext "finalize" returning theContext
           end-if

           exit method
           .

       A-Initialise.

      *> トランザクションサーバ - 実行中のコンテキストを取得します。
           invoke objectcontext "GetObjectContext" returning
theContext

      *> この構成要素が MTS トランザクションに含まれるかどうか確認します。
           if theContext = null
              set TxnNotControlledByMTS to true
           else
              invoke theContext "IsInTransaction" returning
ReturnValue
              if IsNotInTransaction
                 set TxnNotControlledByMTS to true
              else
                 set TxnControlledByMTS    to true
              end-if
           end-if

      *> プログラム変数を初期化します。
           set transactionPassed to true

           INITIALIZE DFHEIBLK
           .

       B-ConnectToDB.
```

```
*> データソースに接続します。

     EXEC SQL
         CONNECT TO 'SQLServer 2000' USER 'SA'
     END-EXEC

     if sqlcode  zero
        move z"connection failed " to ESQLAction
        perform Z-ReportSQLErrorAndExit
     end-if
     .

 C-SetAutoCommitOff.

     EXEC SQL
         SET AUTOCOMMIT OFF
     END-EXEC
     if sqlcode  zero
         move z"Set Autocommit Off failed " to ESQLAction
         perform Z-ReportSQLErrorAndExit
     end-if

     perform X-Commit
    .

 D-ResetDefaultIsolationLevel.
*> SQLServer のデフォルトの分離レベルは「Serialized」であるため、
*> ここでは、このデフォルト値をより適切な値にリセットします。

     EXEC SQL
         SET TRANSACTION ISOLATION READ COMMITTED
     END-EXEC
     if sqlcode  zero
         move z"set transaction isoation failed " to ESQLAction
         perform Z-ReportSQLErrorAndExit
     end-if
     .

 E-ResetDefaultCursorType.
*> OpenESQL のデフォルトのカーソルタイプは、dynamic + lock です。
*> 最も効率のよいのは「client」または「firehose」カーソルです。
*> これは、forward + read only として宣言されたカーソルです - ここでこれを実行する場合は、
*> この時点からこの値がデフォルトとして設定されます。Forward で問題が発生した場合は、
*> 並行性を fast forward に変更します (ただし、これは
*> クライアントカーソルではなくなります)。

     EXEC SQL
         SET CONCURRENCY READ ONLY
     END-EXEC
     if sqlcode  zero
         move z"Set Concurrency Read Only" to ESQLAction
         perform Z-ReportSQLErrorAndExit
     end-if
```

```
    EXEC SQL
        SET SCROLLOPTION FORWARD
    END-EXEC
    if sqlcode  zero
        move z"Set Concurrancy Read Only" to ESQLAction
        perform Z-ReportSQLErrorAndExit
    end-if
    .

F-CallTransaction.

*> プログラムを呼び出してトランザクションを処理します。
    move 0               to  transaction-Info-RC
    call tranactionPgm using dfheiblk transaction-Info

*> 処理が適切かどうか確認します。
    if transaction-Info-RC = 0
       set transactionPassed to true
    else
       set transactionFailed to true
    end-if
    .

X-Commit.

    EXEC SQL
        COMMIT
    END-EXEC
    if sqlcode  zero
        move z"Commit failed " to ESQLAction
        perform Z-ReportSQLErrorAndExit
    end-if
    .

X-Rollback.

    EXEC SQL
        ROLLBACK
    END-EXEC
    if sqlcode  zero
        move z"Rollback failed " to ESQLAction
        perform Z-ReportSQLErrorAndExit
    end-if
    .

Y-Disconnect.

    EXEC SQL
        DISCONNECT CURRENT
    END-EXEC
    if sqlcode  zero
        move z"Disconnect failed " to ESQLAction
        perform Z-ReportSQLErrorAndExit
    end-if
    .

Z-ReportSQLErrorAndExit.
```

```
            move spaces to transaction-Info-Returned
            string ESQLAction delimited by x"00"
                   "SQLSTATE = "
                   SQLSTATE
                   "  "
                   mfsqlmessagetext
                   into transaction-Info-Returned
            end-string

            exit method
            .

        exit method.
        end method "RetrieveString".
       *>-------------------------------------------------------------
-

        end object.
        end class cblsqlwrapper.
```

MTS/COM+ または WebSphere トランザクションの設定方法の詳細については、『*分散コンピューティング*』のマニュアルを参照してください。

7.15 XML サポート

XML ODBC ドライバがある場合は、ODBC データソースと同様に XML ファイルにアクセスするためにそのドライバを設定できます。そして、XML データソースで OpenESQL アシスタントを使用して SQL 文をビルドできます。

7.15.1 PERSIST 文

XML への情報の変換をするために、OpenESQL に PERSIST 文が追加されました。PERSIST 文を使用する場合は、カーソルの SELECT 文で定義された情報を XML ファイルとして保存できます。構文は次のとおりです。

```
PERSIST cursor_name TO xml_destination
```

xml_destination には識別子、ホスト変数、または定数を単一引用符または二重引用符で囲んで指定できます。カーソルでは SCROLLOPTION を static に設定する必要もあります。 次に、例を示します。

```
    01  hv  pic x(50).
    procedure-division.

    *> SQL エラーを処理するために whenever 句を設定します。
    exec sql whenever sqlerror goto sql-error end-exec
    exec sql whenever sqlwarning perform sql-warning end-exec

    *> データソースに接続します。
    exec sql connect to "data source"  end-exec

    *> xml ファイルに保存するカラム情報を使用して静的カーソルを宣言します。
    exec sql
      declare c static cursor for
        select * from emp
    end-exec
```

```
*> カーソルをオープンします。
exec sql open c end-exec

*> 二重引用符で囲まれた定数を使用してデータを xml ファイルに保存します。
exec sql
  persist c to "c:¥XML Files¥xmltest1.xml"
end-exec

*> 単一引用符で囲まれた定数を使用してデータを xml ファイルに保存します。
exec sql
  persist c to 'c:¥XML Files¥xmltest2.xml'
end-exec

*> ホスト変数を使用してデータを xml ファイルに保存します。
move "c:¥XML Files¥xmltest3.xml" to hv
exec sql
 persist c to :hv
end-exec

*> カーソルをクローズします。
exec sql close c end-exec

*> データソースとの接続を解除します。
exec sql disconnect current end-exec

goback.
```

注: Data Direct Connect ODBC ドライバを使用している場合は、バージョン 3.70 以降を使用する必要があります。

7.16 OpenESQL の Unicode サポート

一部のアプリケーションでは、Unicode データを ANSI に変換しないで Microsoft SQL Server データソースに取り込んだり保存したりすることがあります。以前のバージョンの OpenESQL では、データを自動変換しないとデータにアクセスできませんでした。現在のバージョンの OpenESQL と Net Express では新しいタイプのホスト変数を使用することで、Unicode データを ANSI に変換しなくても直接操作できるようになりました。Microsoft SQL Server では、次の 3 つの Unicode カラム型がサポートされます。

- NCHAR
- NVARCHAR
- NTEXT

データを自動的変換しないで、これらのカラムにアクセスするには、次の定義を使用してホスト変数を定義します。
`PIC N(xx) USAGE NATIONAL`
xx には、カラムのサイズを指定します。この形式は、現在固定長データと可変長データの両方についてサポートされています。可変長データを NULL で終了して、データの挿入や更新時にカラムのデータの末尾を示すことができます。データソースからデータが取り込まれると、ホスト変数の末尾に空白文字が付加されます。

たとえば、次のプログラムでは Microsoft SQL Server 2000 製品に付属している Northwind サンプルデータベースから従業員情報を取り込みます。

```
        $SET UNICODE(NATIVE)
        $SET SQL
        WORKING-STORAGE SECTION.

        EXEC SQL INCLUDE SQLCA  END-EXEC

        *> SQL エラーが発生する場合は、ここに詳細なメッセージテキストが示されます。
        01 MFSQLMESSAGETEXT  PIC X(250).
        01 IDX               PIC X(04)  COMP-5.

        EXEC SQL BEGIN DECLARE SECTION  END-EXEC
        *> 他の COBOL コンパイラに移植する必要がある場合は、
        *> ここにホスト変数を記述します。
        EXEC SQL I       NCLUDE Employees END-EXEC

        EXEC SQL END DECLARE SECTION END-EXEC

        PROCEDURE DIVISION.

        EXEC SQL
            WHENEVER SQLERROR perform OpenESQL-Error
        END-EXEC

        EXEC SQL
          CONNECT TO 'LocalServer'
        END-EXEC

        *> プログラムロジックと SQL 文をここに記述します。
             EXEC SQL
              DECLARE CSR135 CURSOR FOR SELECT
                     A.FirstName
                    ,A.LastName
                    ,A.EmployeeID
                    ,A.HireDate
                FROM Employees A
             END-EXEC
             EXEC SQL OPEN CSR135 END-EXEC
             PERFORM UNTIL SQLSTATE >= "02000"
               EXEC SQL
               FETCH CSR135 INTO
                   :Employees-FirstName
                  ,:Employees-LastName
                  ,:Employees-EmployeeID
                  ,:Employees-HireDate:Employees-HireDate-NULL
               END-EXEC

               *> FETCH 処理のデータを処理します。
               IF SQLSTATE < "02000"

               *> 配列の取り込みの場合は、フィールド sqlerrd(3) に返される行数が
               *> 格納されます。
               *>  PERFORM VARYING IDX FROM 1 BY 1 UNTIL IDX >
SQLERRD(3)
```

```
            *>   ここに、配列を処理するコードを追加する必要があります。

            *>  END-PERFORM
            END-IF
          END-PERFORM
          EXEC SQL CLOSE CSR135 END-EXEC

     EXEC SQL DISCONNECT CURRENT END-EXEC
     EXIT PROGRAM.
     STOP RUN.
     *> デフォルトの SQL エラールーチン。必要に応じて修正してプログラムを停止します。
     OpenESQL-Error Section.

         display "SQL エラー = " sqlstate " " sqlcode
         display MFSQLMESSAGETEXT
     *> 実行を停止します。
         exit.
```

次の例は、ANSI でデータを取り込んだ場合と同じコードですが、次のように INCLUDE Employees コピーブックの定義が異なります。

```
     *>  ------------------------------------------------------------
--
     *> Employee テーブルに対する COBOL 宣言
     *>  ------------------------------------------------------------
--
     01  DCLEmployees.
         03 Employees-EmployeeID             PIC S9(09)  COMP-5.
         03 Employees-LastName               PIC N(20)
                                  USAGE NATIONAL.
         03 Employees-FirstName              PIC N(10)
                                  USAGE NATIONAL.
         03 Employees-Title                  PIC N(30)
                                  USAGE NATIONAL.
         03 Employees-TitleOfCourtesy        PIC N(25)
                                  USAGE NATIONAL.
         03 Employees-BirthDate              PIC X(23).
         03 Employees-HireDate               PIC X(23).
         03 Employees-Address                PIC N(60)
                                  USAGE NATIONAL.
         03 Employees-City                   PIC N(15)
                                  USAGE NATIONAL.
         03 Employees-Region                 PIC N(15)
                                  USAGE NATIONAL.
         03 Employees-PostalCode             PIC N(10)
                                  USAGE NATIONAL.
         03 Employees-Country                PIC N(15)
                                  USAGE NATIONAL.
         03 Employees-HomePhone              PIC N(24)
                                  USAGE NATIONAL.
         03 Employees-Extension              PIC N(4)
                                  USAGE NATIONAL.
         03 Employees-Photo                  PIC X(64000).
         03 Employees-Notes                  PIC N(32000)
                                  USAGE NATIONAL.
         03 Employees-ReportsTo              PIC S9(09)  COMP-5.
         03 Employees-PhotoPath              PIC N(255)
```

USAGE NATIONAL.

OpenESQL アシスタントも Unicode データをサポートするように拡張されています。詳細は、『*OpenESQL アシスタント*』の章を参照してください。

第 8 章 : OpenESQL アシスタント

OpenESQL アシスタントは、次の作業を簡単に行うための対話型ツールです。

- SQL の SELECT 文のプロトタイプ定義と、データベースに対するこの SELECT 文のテスト
- SQL の INSERT 文、UPDATE 文、および DELETE 文の作成

作成した SQL クエリーは、OpenESQL アシスタントを使用して Net Express の COBOL コードに挿入できます。適切なプロジェクトとプログラムを開くのみで、現在の挿入箇所に OpenESQL アシスタントにより SQL クエリーを挿入できます。また、SQL クエリーを挿入するときに必要となる補助コードを、OpenESQL アシスタントを使用して作成し、挿入する方法も選択できます。
ここでは、チュートリアル形式で次の項目を説明します。

- OpenESQL アシスタントオプションの設定方法
- OpenESQL アシスタントの起動方法
- データソースへの接続方法
- テーブルの選択方法
- カラムの選択方法
- カラムの選択解除方法
- テーブルに含まれる全カラムの選択方法
- テーブルの選択解除方法
- カラムの詳細の表示方法
- 新規クエリーの作成方法

- 別のテーブルの選択方法
- クエリーの型の変更方法
- 別のデータソースへの接続方法

- SELECT 文によるクエリーの実行方法
- 検索条件の指定方法
- 取り出すデータのソート方法
- データソースからの接続解除方法
- 結合テーブルの作成方法
- プログラムへの埋め込み SQL 文の追加方法
- プログラムへの補助コードの追加方法
- SELECT (カーソル) クエリーの変更による配列取り込みの実行方法
- ストアドプロシージャとしてのクエリーの生成方法
- OpenESQL アシスタントの終了方法

8.1 OpenESQL アシスタントオプションの設定方法

「OpenESQL アシスタント構成オプション」ダイアログボックスで、特定のデフォルト設定を変更して、OpenESQL アシスタントでクエリーを生成する方法を制御できます。これを実行するには、[**オプション**] メニューの [**埋め込み SQL**] をクリックします。次のダイアログボックスが表示されます。

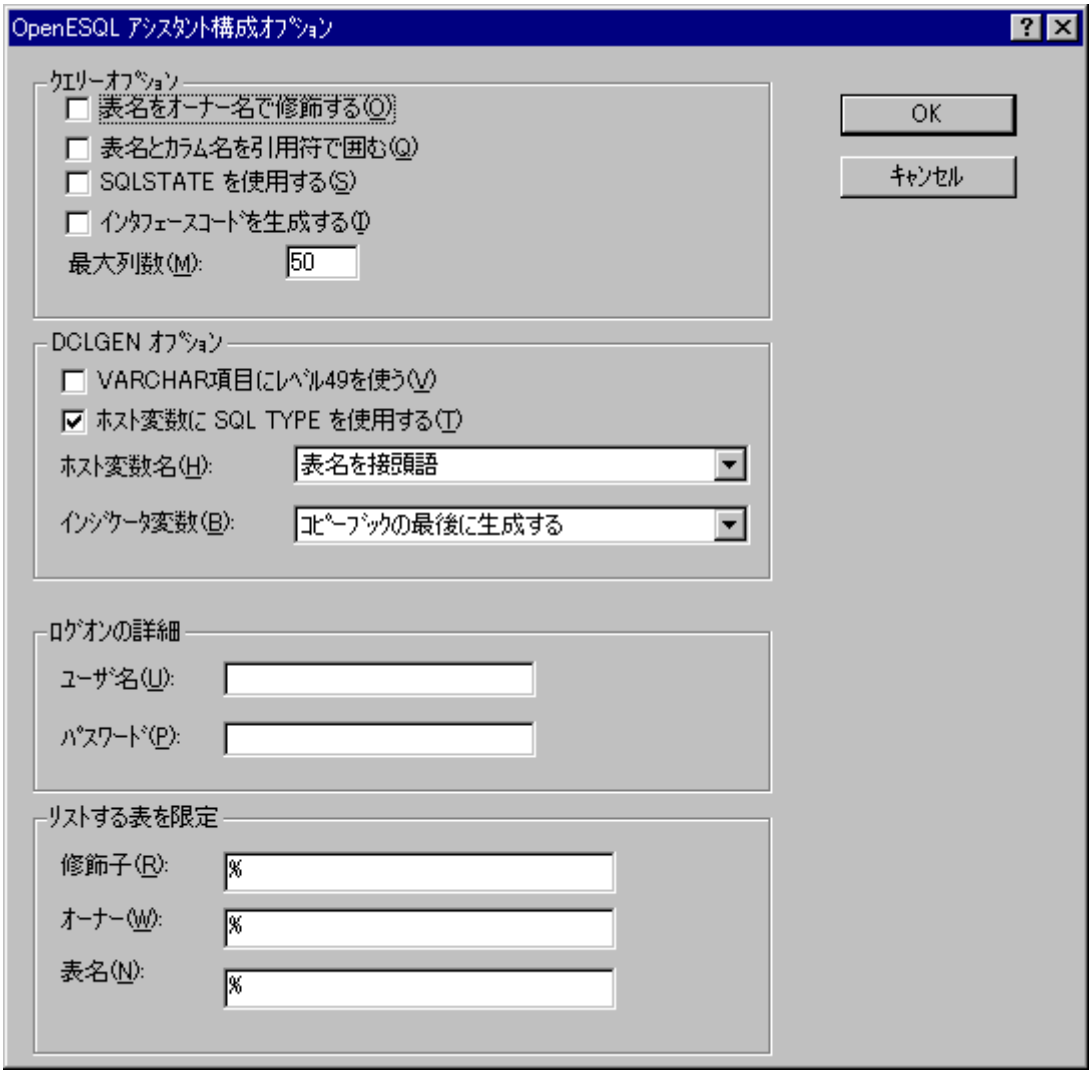


図 8-1：OpenESQL アシスタント構成オプション
次のオプションを設定できます。

- **「表名をオーナー名で修飾する」**

  デフォルトでは、すべてのクエリーは修飾しないでビルドされます。ただし、データソースに同じ名前のテーブルが複数ある場合は、特定のオーナーを指定してクエリーをビルドできます。このオプションをチェックすると、ツリービュー内のテーブルリストで、テーブル名の後ろにかっこで囲まれたオーナー名が表示されます。このリストで、クエリーに使用するオーナー名をもつテーブル名を選択すると、そのクエリーのすべてのテーブル名がオーナー名で修飾されます。

- **「表名とカラム名を引用符で囲む」**

  デフォルトでは、カラム名またはテーブル名に空白文字、特殊文字 (「$」など)、DBCS 文字が含まれていない場合は、カラム名やテーブル名はデータソースに関連付けられている引用符識別子で囲まれません。このオプションをチェックすると、すべてのテーブル名とカラム名が引用符識別子で囲まれます。

- 「SQLSTATE を使用する」

  デフォルトでは、SQLCODE の確認を使用して SQL 文が生成されます。SQLSTATE の確認を使用して SQL 文を生成する場合は、このオプションをチェックします。

- 「インターフェイスコードを生成する」

  デフォルトでは、アプリケーションで使用する SQL 文が生成されます。Web サービスとして使用できる SQL 文を作成する場合は、このオプションをチェックします。

- 「最大列数」

  デフォルトでは、クエリーの実行時に最初の 50 行のみ返されます。このツールの目的は対話形式でクエリーをビルドおよびテストすることなので、必要なすべての基準を指定していない場合があります。そのような場合にネットワークサーバを使用してクエリーをテストする場合は、クエリーから多量の行が返されてマシンがクラッシュしたりネットワークがオーバーロードしたりすることがあります。それを避けるために、このデフォルト値が設定されています。この値を大きくすることは可能ですが、返されるデータ量またはテストに必要なデータ量を考慮して適切な値を選択してください。

- 「VARCHAR 項目にレベル 49 を使う」

  デフォルトでは、VARCHAR カラムの PIC X(n) フィールドとしてホスト変数が生成されます。データをホスト変数にマップする場合には、データは NULL で終了します。ただし、返されるカラムデータの長さを取得する場合は、このオプションをチェックして、2 つのレベル-49 変数 (1 つはマップされたデータの長さで、もう 1 つは実際のテキストデータの長さ) を使用してホスト変数を生成します。

- 「ホスト変数に SQL TYPE を使用する」

  デフォルトでは、適用可能な場合は SQL TYPE 定義を使用して COBOL ホスト変数が生成されるようになりました。これにより、SQL プリコンパイラは、ホスト変数の用途のタイプをより的確に判断できます。このオプションのチェックを外すと、以前の方法でホスト変数を生成できます。また、配列取り込みでコピーブックを使用している場合も、このオプションのチェックを外します。

- 「ホスト変数名」

  このドロップダウンリスト内で選択されたカラム名と接頭語の組み合わせを使用してホスト変数が作成されます。この組み合わせの文字列が長すぎたり、無効な文字が含まれたりする場合は、COBOL 名が無効になり、カラム番号が使用されます。有効な COBOL 名を作成するために、アンダスコアはすべてハイフンに変換されます。次のオプションから選択することができます。

  - 「接頭語なし」

    カラム名のみを使用して変数名を生成します。カラム名が 32 文字以上の場合や、名前に無効な COBOL 文字が含まれる場合は、FLD が前に付いたカラム番号が生成されます。

- 「表名を接頭語」

  テーブル名をカラム名の接頭語として使用します。この組み合わせの文字列の長さは 31 文字以内にする必要があります。32 文字以上の場合は、カラム名ではなくカラム番号が使用されます。これがデフォルトの接頭語設定です。

- 「アルファベット文字の接頭語」

  最初に選択されたテーブルの接頭語を「A」、次に選択されたテーブルの接頭語を「B」といったようにアルファベットを順番に割り当てて変数名を生成します。

- 「インジケータ変数」

  インジケータ変数を生成するタイミングを選択できます。

  - 「コピーブックの最後に生成する」

    これがデフォルトの接頭語設定です。

  - 「各ホスト変数の最後に生成する」

- 「ログオンの詳細」

  常に同じデータソースに接続する場合や、データソースへ接続する際に、常に同じログオン情報を使用する場合は、次のフィールドを使用してその情報を保存できます。

  - 「ユーザ名」

    データソースへの接続に必要なユーザ ID またはログオン ID。

  - 「パスワード」

    以前に指定したユーザ名に対応する、データソースへの接続に必要なパスワード。このフィールドに入力したパスワードはエントリ時にはそのまま表示されず、アスタリスクで表示されます。

- 「リストする表を限定」

  デフォルトでは、ツリービューに、データソース内で検出されたすべてのテーブル、ビュー、別名、および同義語が表示されます。特定の種類のデータソースでは、非常に大きなリストが作成される可能性があります。このリストの作成応答時間を短くするためには、返されるテーブルを限定してこのリストの内容を制限できます。たとえば、「XYZ%」で始まるオーナーから開始されるテーブルのみを返すように要求できます。通常、これは次の点で LIKE 関数と似ています。

  - アンダスコア文字 (_) は単一文字を表します。
  - パーセント記号 (%) は 0 以上の文字で構成される文字列を表します。
  - 他の文字はその文字自体を表します。

  リストを制限するために次の 3 つのフィールドを使用できます。

- 「修飾子」
- 「オーナー」
- 「表名」

データソースによっては、これらの 3 つのフィールドがすべてサポートされない場合があるため、データソースのプロパティを参照して、接続先のデータソースでサポートされるフィールドを確認してください。

この例では、「オーナー」フィールドに「XYZ%」と入力して、[OK] をクリックすると、設定が保存されてデータソースに接続されます。指定した制限によりテーブルが見つからない場合は、テーブル名として「**<テーブルが選択されていません>**」が返されます。

8.2 OpenESQL アシスタントの起動方法

OpenESQL アシスタントを起動するには、Net Express の [**ビュー**] メニューから [**ドッキングできるウィンドウ**] を選択します。[**OpenESQL アシスタント**] チェックボックスをチェックして、「**ドッキングできるウィンドウ**」ダイアログボックスを閉じます。

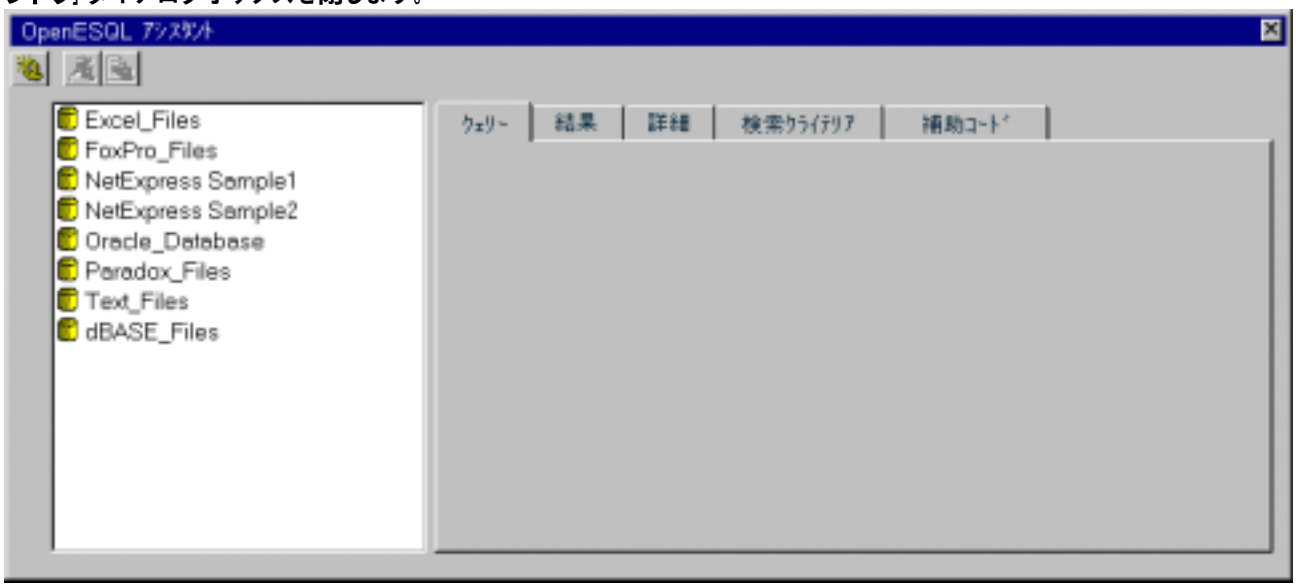


図 8-2：OpenESQL アシスタント
「OpenESQL アシスタント」ウィンドウは、ドッキング可能なウィンドウです。この機能を使用する場合は、タイトルバーのすぐ下にある灰色の領域を右クリックし、「**ドッキングを許可する**」をチェックします。また、この機能を使用しない場合は、「**ドッキングを許可する**」のチェックを外します。また、ウィンドウを非表示にするには、右クリックし、「**非表示**」をオンにします。ドッキングやウィンドウの非表示に慣れていない場合は、オンラインヘルプファイルの『**ドッキング**』を参照し、『**ビューとドッキング可能なウィンドウの再整列**』を選択してください。

8.3 データソースへの接続方法

OpenESQL アシスタントを起動すると、設定したすべての ODBC データソースのリストが表示されます。たとえば、*図 8-2* では、OpenESQL アシスタントにより **Excel_Files** と **Oracle_Database** の 2 つの既存データソースが表示されます。

データソースに接続するには、次のようなデータソース名または該当するデータソースのアイコンをダブルクリックします。

NetExpress Sample2

図 8-3：データソースアイコンとデータソース名

注:一度に接続できるデータソースは 1 つのみです。接続するデータソースの変更方法については、『*別のデータベースへの接続方法*』を参照してください。

データソースの設定方法によっては、次のどれか (複数の場合もあります) を入力する必要があることがあります。

- ユーザ ID
- パスワード
- データベース名

起動する前に、独自のデータソースを設定する必要があります。この方法については、『*ヘルプ*』を参照してください。2 つのデータソース UserSample1 と UserSample2 を作成することをお奨めします。データソースに接続した後に、データソースを右クリックし、ポップアップメニューの [**データソースのプロパティ**] をクリックすると、データソースのプロパティを参照できます。

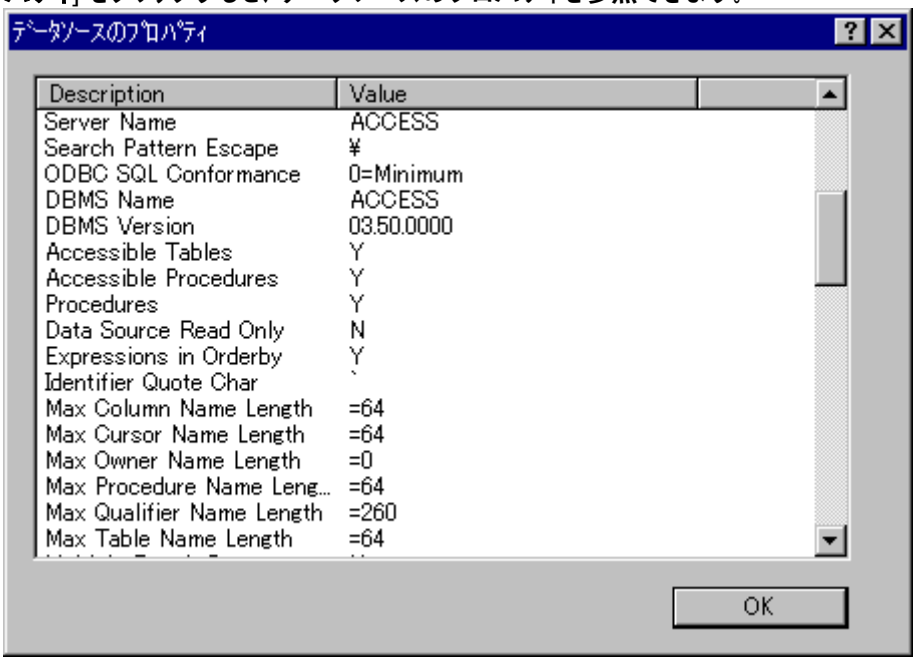


図 8-4：データソースのプロパティ

データソースに関する情報には、使用される識別子を囲む引用符と、カラム名とテーブル名の最大サイズがあります。この例では、「Max Owner Name Length」が 0 に設定されており、このデータソースが「オーナー」フィールドをサポートしていないことがわかります。このため、前述した設定機能でこのフィールドを使用してテーブルのリストを制限できません。

以後のチュートリアルでは、sample.mdb データベースを使用することを仮定とし、すべての例でこのデータベースを使用します。

8.4 テーブルの選択方法

データソースに接続すると、データソース名の下部に、このデータソースに含まれるすべてのテーブル名が表示されます。

図 8-5：テーブルの選択

テーブルを選択するには、テーブル名をダブルクリックします。その結果、必要なクエリーの型を選択するダイアログボックスが表示されます。この段階では選択するクエリーの型は関係ないため、デフォルト (「SELECT(singleton)」) を選択して、[**OK**] ボタンをクリックします。

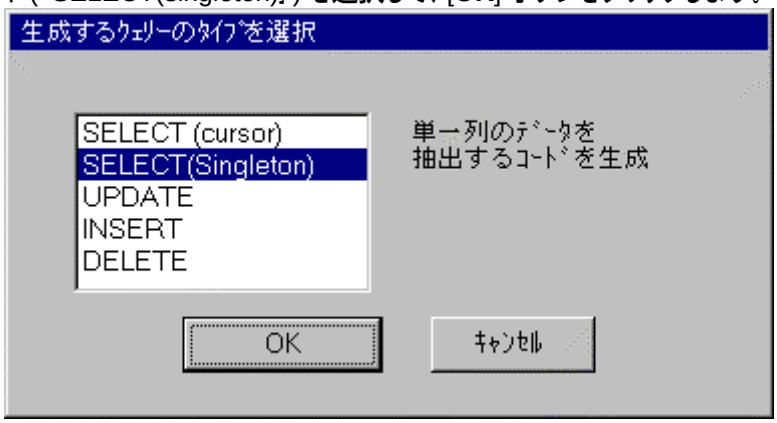


図 8-6：クエリーの選択

クエリーを選択する場合は、選択したクエリーに対する COBOL コードが自動生成され、「**クエリー**」タブの下部に表示されます。同時に、テーブル名の下部にテーブルの全カラムが一覧表示されます。

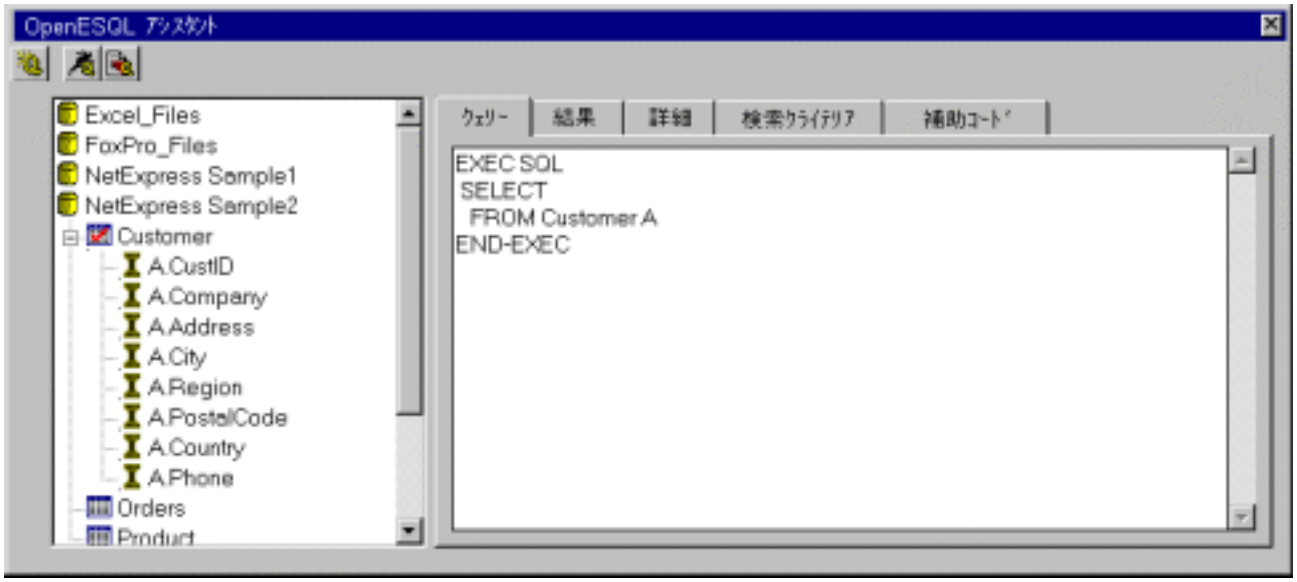


図 8-7：カラムの表示

OpenESQL アシスタントによりテーブルの別名が自動生成されていることに注意してください。

```
SELECT FROM Customer A
```

これは、最初に選択したテーブルなので、別名には、「A」の文字が使用されています。2 番目に選択するテーブルについては、OpenESQL アシスタント は「B」の文字を使用して別名を生成します。その後は同様に、テーブルの選択順に対応したアルファベットを使用して別名を生成します。

また、各カラム名の前にも、別名 (A.CustID、A.Company など) が付くことに注意してください。この別名により、テーブル内のカラムを他のテーブル内のカラムと区別できます。

注：

一部のデータベースでは、システムテーブルの名前に特殊文字が使用されます。たとえば、Oracle では、システムテーブル名にドル記号 ($) を使用されることがあります。このため、OpenESQL アシスタントは、引用符で囲まれていないと規約違反となるカラム名やテーブル名を検出するたびに自動的にカラム名やテーブル名が引用符で囲まれるオプションを設定します。

カラム名から生成された名前 (接頭語と接尾語を含む) の文字数が 31 文字を超えたり、名前に規約違反な文字が含まれたりするために COBOL で規約違反になった場合は、カラム名ではなく、カラム番号 (COL005 など) を使用してホスト変数が生成されます。

8.4.1 カラムの選択方法

カラムを選択するには、カラム名をダブルクリックします。この場合は、COBOL コードが自動更新されることに注意してください。

8.4.2 カラムの選択解除方法

選択したカラムを選択解除するには、そのカラム名をダブルクリックします。カラムの選択と選択解除を行うたびに、COBOL コードが自動更新されます。

8.4.3 テーブルに含まれる全カラムの選択方法

テーブルの全カラムを選択するには、次の操作を行います。

- テーブル名を右クリックします。
- [**すべてのカラムを選択**] をクリックします。

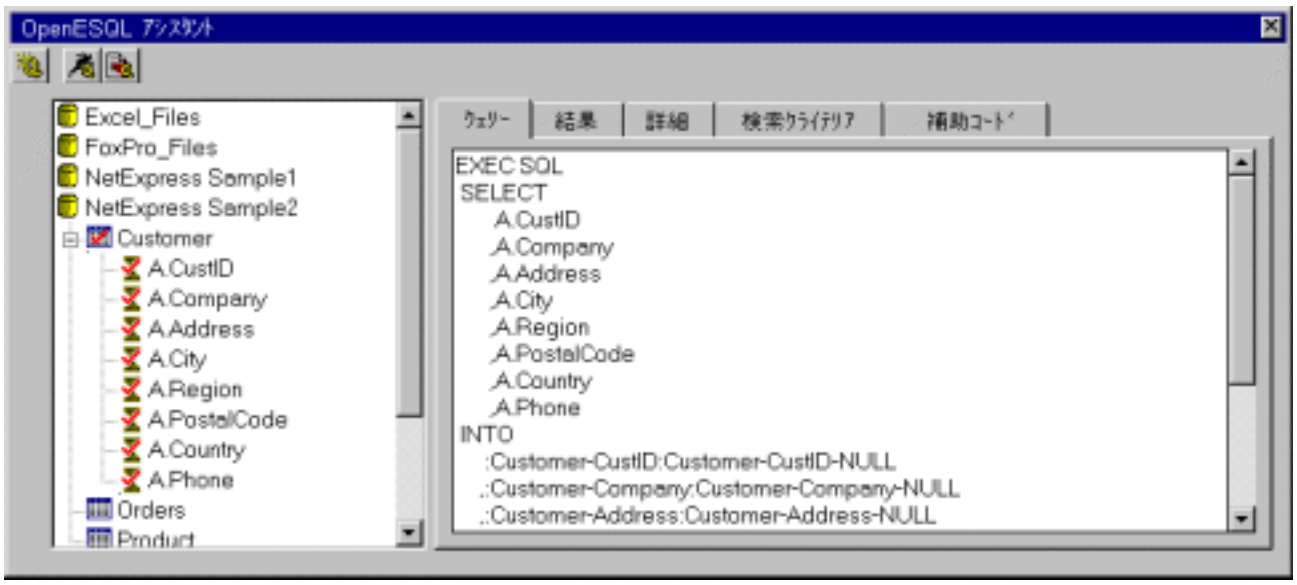


図 8-8：カラムの選択

カラムが現在選択されている場合は、カラムアイコンがチェックされるので、カラムが現在選択されているかどうかを常に把握できます。

図 8-9：カラムアイコン

8.5 テーブルの選択解除方法

選択したテーブルをダブルクリックしてテーブルを選択解除できます。このテーブルからカラムを選択している場合は、選択解除を確認するダイアログが表示されます。[**はい**] ボタンをクリックします。テーブルの選択は解除され、生成されたコードが更新されます。

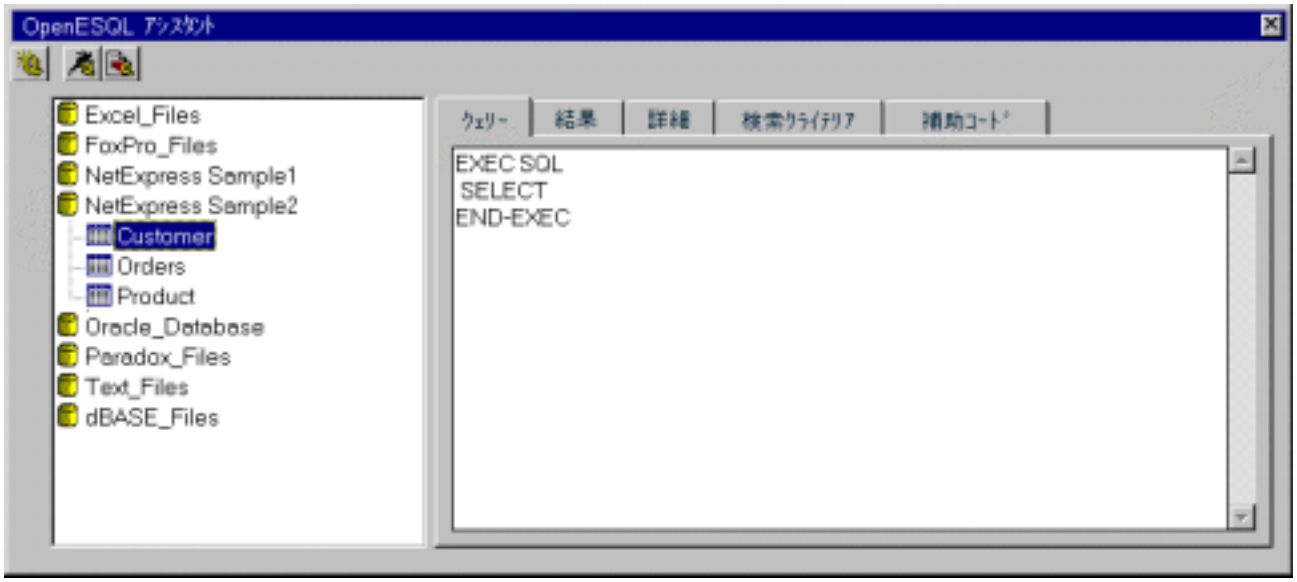


図 8-10：テーブルの選択解除

テーブルを選択解除するには、テーブル名またはテーブルアイコンをダブルクリックしてください。マイナス記号 (-) をクリックしても、テーブルのカラムの表示 / 非表示が切り替わるのみです (プラス記号 (+) をクリックするとすべてのカラムが表示され、マイナス記号 (-) をクリックするとすべてのカラムが非表示になります)。

たとえば、Customer テーブルをダブルクリックして、再度選択します。テーブル名を右クリックし、[**すべてのカラムを選択**] をクリックして、すべてのカラムを選択します。さらに、カラムを非表示するためにマイナス記号 (-) をクリックします。

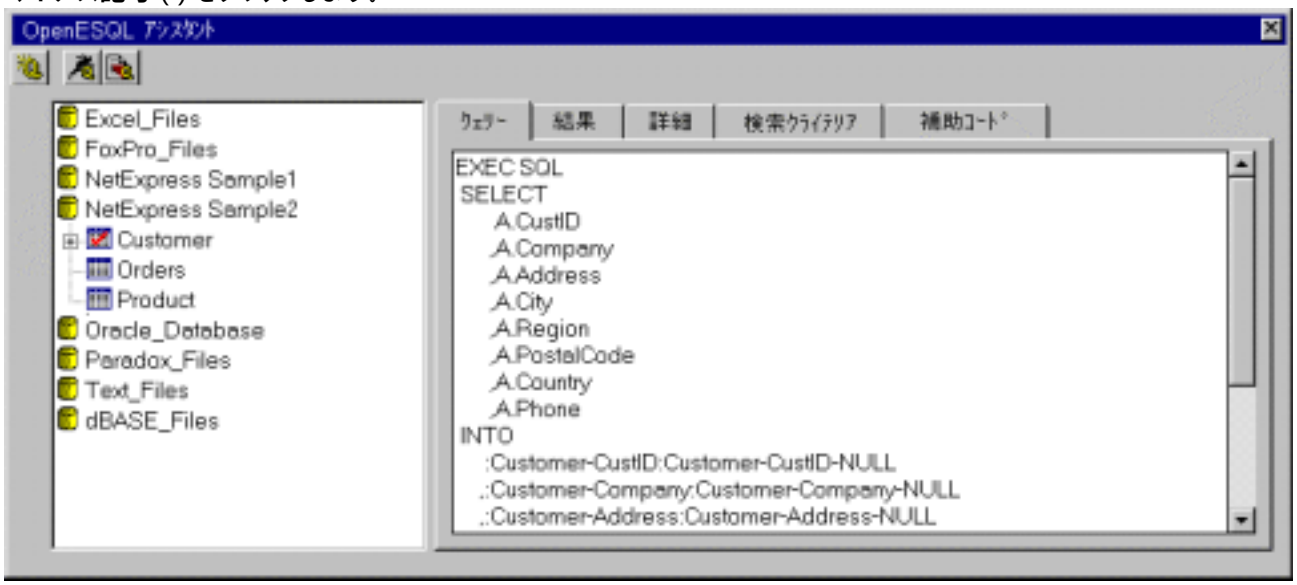


図 8-11：選択したテーブルに含まれるカラムの非表示

8.6 カラムの詳細の表示方法

テーブルに含まれるカラムの詳細情報を表示するには、[**詳細**] タブをクリックします。

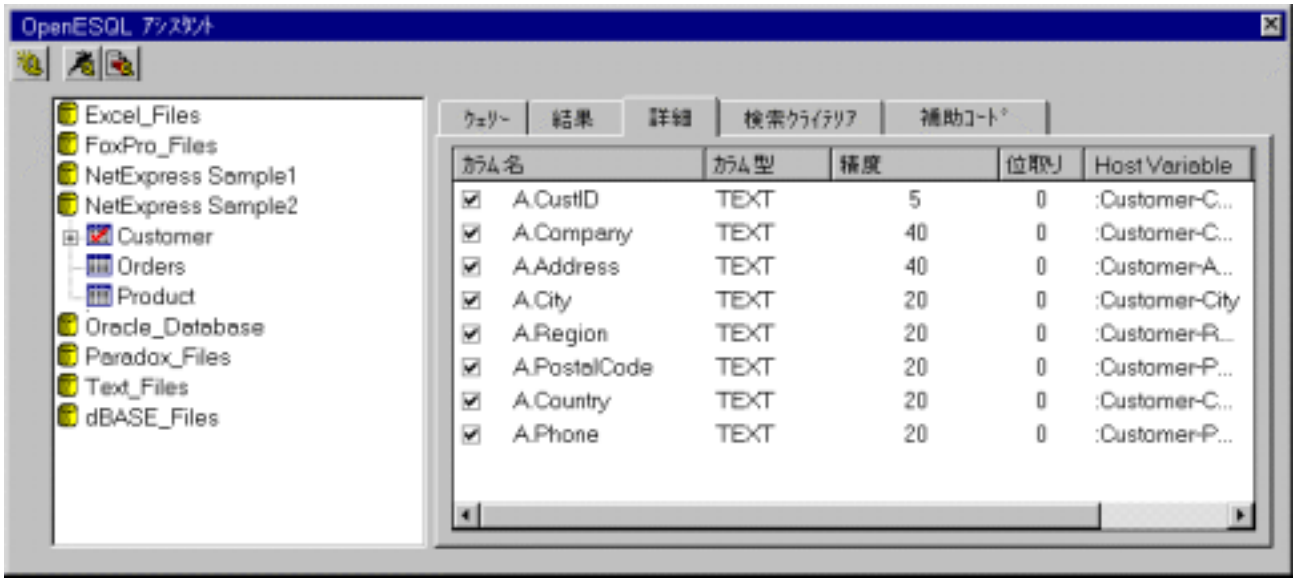


図 8-12：カラムの詳細表示

表示される情報は、次のとおりです。

- 「カラム名」

  テーブルの各カラム名が表示されます。各カラム名の前には、テーブルの別名が付加されることに注意してください。たとえば、CustID は、A.CustID のように表示されます。カラム名の左横にあるチェックボックスがチェックされている場合は、現在そのカラムが選択されていることを示します。

- 「型」

  カラムのデータ型が表示されます。このデータ型は、接続するデータソースで使用します。カラムのデータ型は、そのカラムの値をデータソースと受け渡しするホスト変数の COBOL PICTURE 句で指定するデータ形式と一致する必要があります。

  OpenESQL アシスタントを使用して、現在のディレクトリに *tablename*.cpy というコピーファイルを生成できます。このコピーファイルでは、正しい COBOL PICTURE 句 (つまり、テーブルのカラムのデータ型と一致する) をもつ必要なホスト変数がすべて宣言されます。『*プログラムへの補助コードの追加方法*』を参照してください。

- 「精度」

  カラムの総桁数が表示されます。精度は、関連するカラムのみについて表示されます。カラムがテキストカラムの場合には、精度はカラム長を示します。

- 「位取り」

  カラムの数値を丸める桁数が表示されます。位取りは、関連するカラムのみについて表示されます。

- 「ホスト変数」

  各カラムについてホスト変数が自動生成されます。ホスト変数名は、次の形式です。

  `:<table-name>-<column-name>`

  たとえば、CustID カラムに対して生成されるホスト変数名は `:Customer-CustID` になります。また、City カラムに対して生成されるホスト変数名は `:Customer-City`、Phone カラムに対して生成されるホスト変数名は `:Customer-Phone` になります。

- 「インジケータ変数」

  ホスト変数と同様に、各カラムに対してインジケータ変数を生成します。インジケータ変数名は、次の形式です。

  `:<table-name>-<column-name>-NULL`

  たとえば、CustID カラムに対して生成されるインジケータ変数名は `:Customer-CustID-NULL` になります。また、City カラムに対して生成されるインジケータ変数名は `:Customer-City-NULL`、Phone カラムに対して生成されるインジケータ変数名は `:Customer-Phone-NULL` になります。

8.7 新規クエリーの作成方法

次では、既存のクエリーの変更方法や新規クエリーの作成方法について説明します。

8.7.1 別のテーブルの選択方法

別のテーブルを選択するには、まず、現在選択しているテーブルをダブルクリックして、選択を解除します (このテーブルのカラムが現在選択されている場合は、テーブルの選択解除を確認するダイアログが表示されます)。そして、別のテーブルを選択してダブルクリックする場合は、クエリーの型を選択するダイアログが表示されます。

8.7.2 クエリーの型の変更方法

クエリーの型を変更するには、まず、現在選択されているテーブルを選択解除し、再度同じテーブルを選択するか (同じテーブルで別の型のクエリーを作成する場合)、または、新しいテーブルを選択する必要があります。テーブルを選択すると、クエリーの型を選択するダイアログが表示されます。必要な型のクエリーを選択し、[OK] ボタンをクリックします。

8.7.3 別のデータソースへの接続方法

別のデータソースへ接続するには、[**クエリーの新規作成**] ボタン をクリックします。このボタンをクリックすると、現在のデータソースとの接続が解除されます。そのため、別のデータソース名を新たにダブルクリックして選択できます。現在のデータソースから接続を解除しないで別のデータソースへ接続しようとすると、この別のデータソースに接続できないことを示すエラーメッセージが表示されます。

8.8 SELECT 文によるクエリーの実行方法

UserSample2 データソースに接続し、**Customer** テーブルを選択します。次に、クエリーの型を選択するダイアログが表示されます。「**SELECT (カーソル)**」を選択し、「**カーソル名**」エントリフィールドに「CSR506」と入力し、[**OK**] ボタンをクリックします。

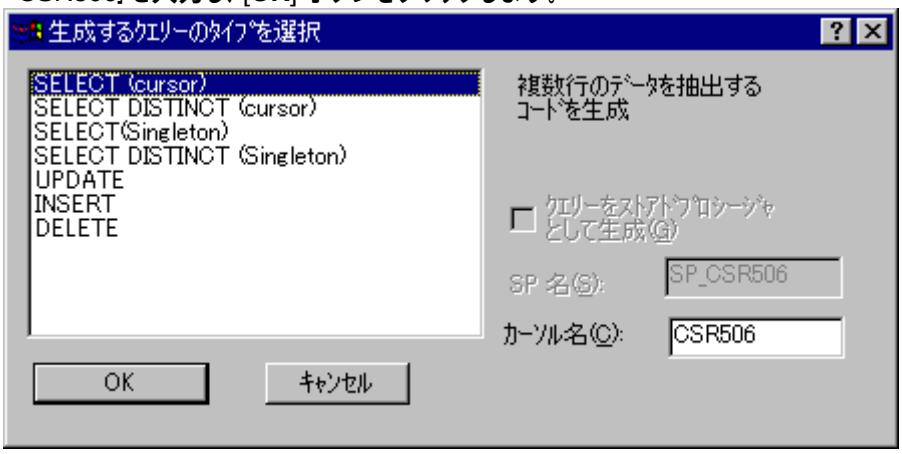


図 8-13：クエリーの型「SELECT (カーソル)」の選択

SELECT 文を使用した COBOL コードが自動生成され、表示されます。

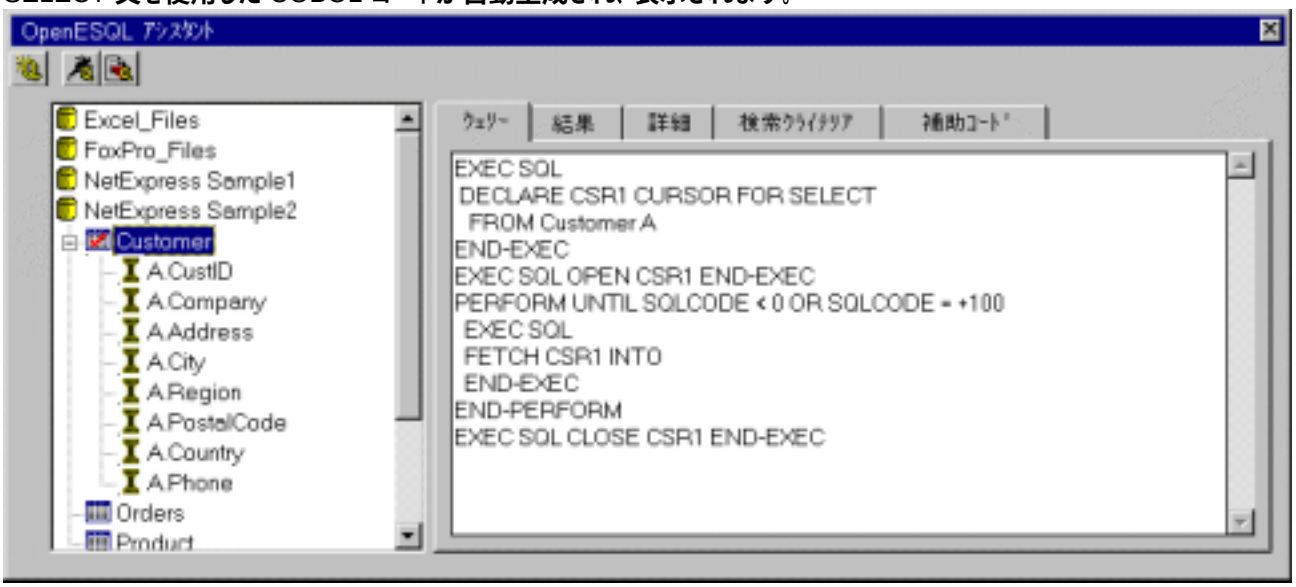


図 8-14：「SELECT (カーソル)」によるクエリーコード

このコードには、次の文が含まれています。

- SELECT 文で使用するカーソルを作成する DECLARE CURSOR 文

```
DECLARE CSR506 CURSOR FOR SELECT FROM Customer A
```

- 対応する DECLARE CURSOR 文で指定した SELECT 文を実行する OPEN 文

```
OPEN CSR506
```

- カーソルの結果集合から次の行を検索する FETCH 文

```
FETCH CSR506 INTO
```

- カーソルをクローズする CLOSE 文

```
CLOSE CSR506
```

このクエリーのままでは、選択するカラムと選択する内容が指定されていないため、構文エラーになります。[**クエリーの実行**] ボタン をクリックすると、エラーが表示されます。
カラムを選択するには、カラムをダブルクリックします。ここでは、A.CustID を選択します。選択した各カラムについて、次のように自動生成コードが更新されます。

- カラムが DECLARE CURSOR 文に追加されます。次に例を示します。
- DECLARE CSR506 CURSOR FOR SELECT
- A.CustID
FROM Customer A

- カラムを読み込むホスト変数が FETCH 文の INTO 句に追加されます。次に例を示します。
- FETCH CSR506 INTO
:Customer-CustID:Customer-CustID-NULL

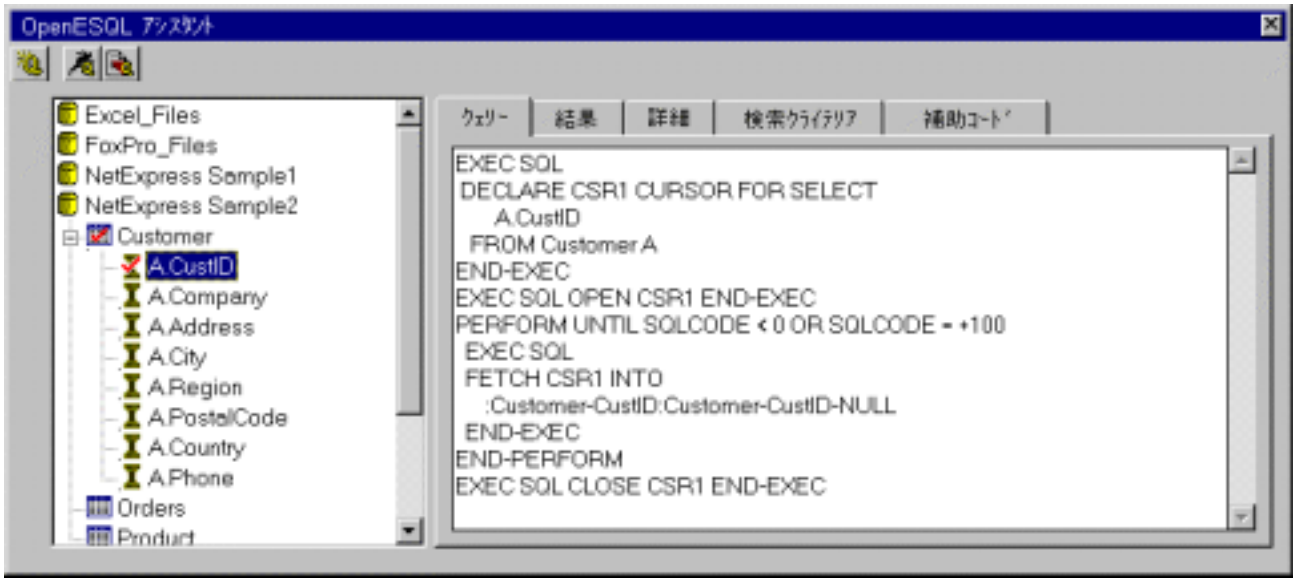


図 8-14：SELECT 文へのカラムの追加
ここでは、A.Company と A.Phone を選択します。必要なカラムをすべて選択した後で、[**クエリーの実行**] ボタン をクリックすると、クエリーが実行されます。クエリー結果は、OpenESQL アシスタントにより自動的に表示されます。

図 8-15：クエリー結果

8.9 検索条件の指定方法

検索条件 (WHERE 句) を指定すると、SELECT 文で返される行を制限できます。OpenESQL アシスタントには、WHERE 句で使用する検索条件をすばやく簡単に指定できる特別なダイアログがあります。SELECT 文で返される行数を制限するには、[**検索条件**] タブをクリックします。

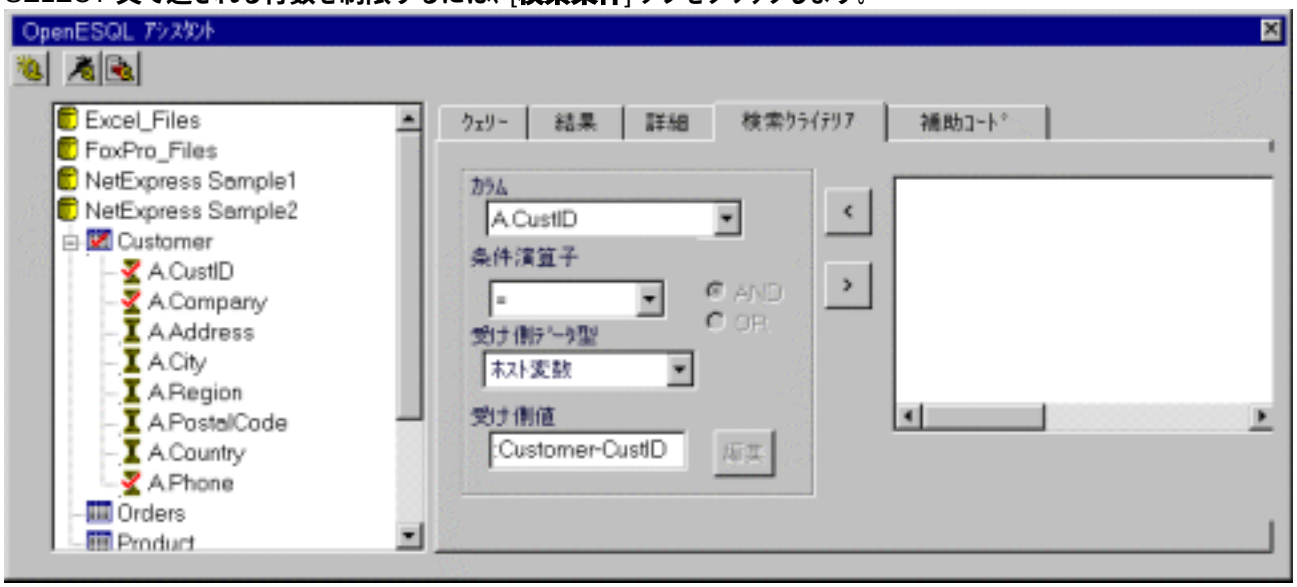


図 8-16：[検索条件] タブ

検索条件を指定するには、次の手順を実行します。

1. 「**カラム**」リストボックスからカラム名を選択します。このリストボックスには、現在選択されているテーブルのすべてのカラム名が表示されます。

2. 「条件演算子」を選択します。下向きの矢印をクリックし、有効な条件演算子が見つかるまでスクロールします。
3. 「**受け側データ型**」を選択します。「受け側データ型」は、ホスト変数、定数、特殊レジスタ、またはカラム名です。
4. 「**受け側値**」を選択するか、または、入力します。「受け側データ型」にカラム名を選択した場合は、OpenESQL アシスタントにより生成された有効なカラム名がすべて「受け側値」リストボックスに一覧表示されます。「受け側データ型」にホスト変数を選択した場合は、OpenESQL アシスタントにより生成された有効なホスト変数がすべて「受け側値」リストボックスに一覧表示されます。また、「受け側データ型」に定数または特殊レジスタを選択した場合は、「受け側値」リストボックスの右側にある [**編集**] ボタンが使用可能になります。この場合は、「受け側値」リストボックスに、値を直接入力するか、または、[**編集**] ボタンをクリックして、**定数値エディタ**を表示することも可能です。

選択内容に問題がない場合は、右向きの矢印 (>) をクリックします。その結果、検索条件が右側のペインに移動され、同時にクエリーウィンドウ内の COBOL コードが更新されます。
たとえば、上記のとおり作成した SELECT 文によって返される顧客 ID を制限する場合は、次のように指定します。

1. 「**カラム**」リストボックスで **A.City** を選択します。
2. 「条件演算子」で = を選択します。
3. 「受け側データ型」で**定数**を選択します。
4. この段階で「受け側値」リストボックスの右側にある [**編集**] ボタンが使用可能になります。このボタンをクリックし、「**定数値エディタ**」ダイアログボックスを表示します。

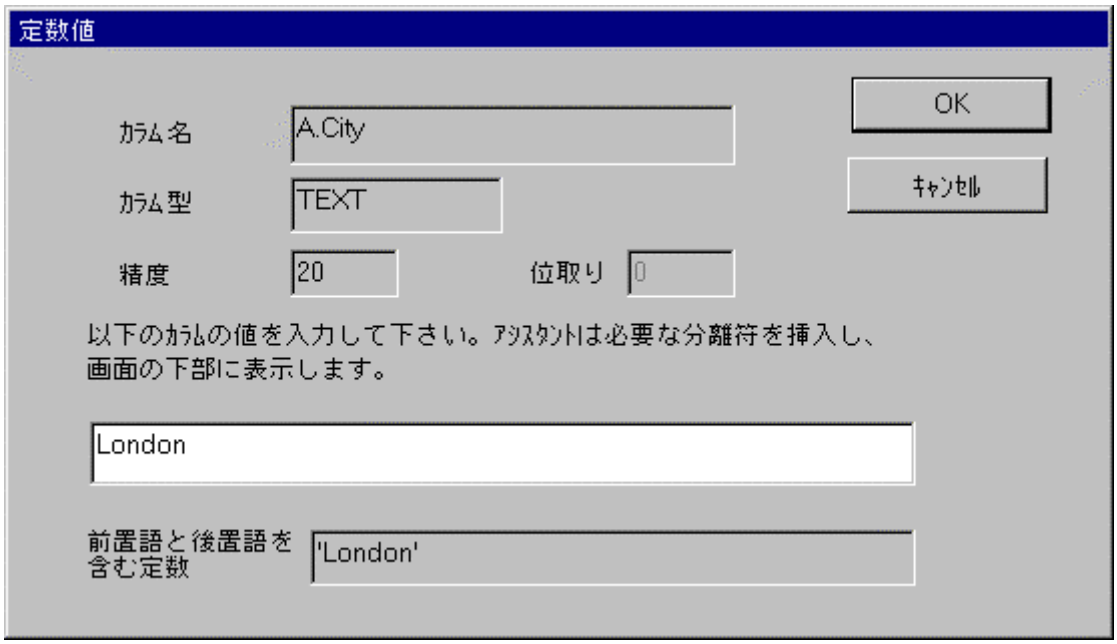


図 8-17：定数値エディタ

5. 「受け側データ型」として **London** を選択します。OpenESQL アシスタント により正しい区切り文字が自動的に追加されていることに注意してください (この例では、単一引用符 (') が使用されています)。[**OK**] ボタンをクリックします。
6. そして、右向きの矢印をクリックし、検索条件を右側のペインに移動させます。

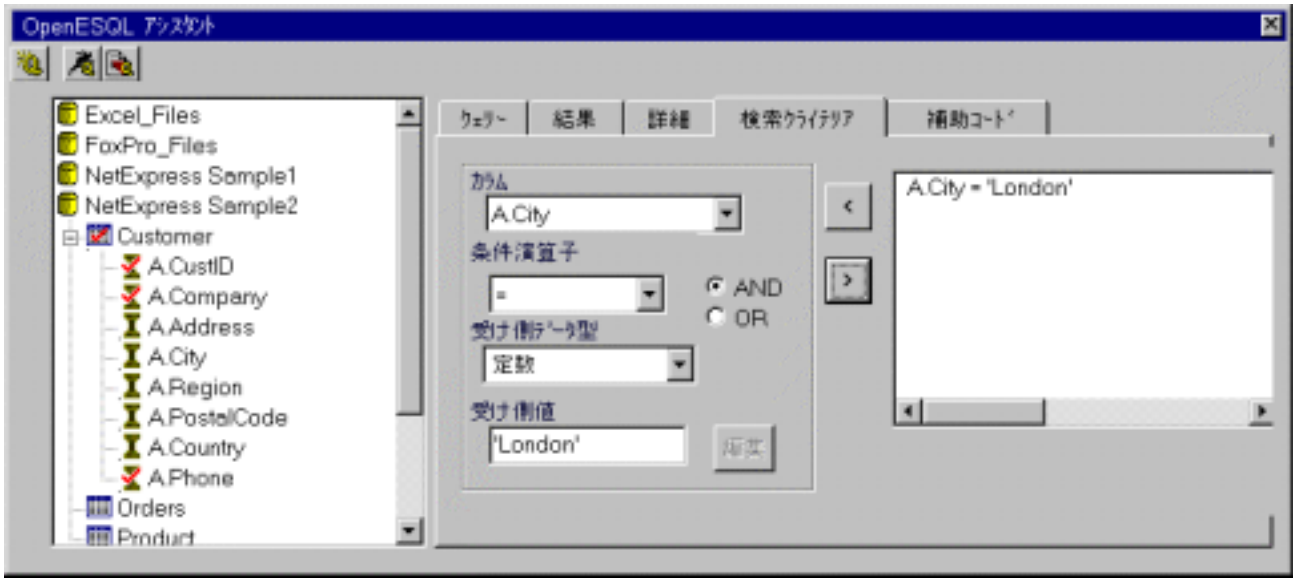


図 8-18：検索条件の指定

このクエリーを再実行 ([**クエリーを実行**] ボタンをクリック) すると、City カラムのエントリが London である顧客の顧客 ID のみが表示されます。

図 8-19：検索条件を指定して制限した顧客 ID

最初の検索条件を指定して、右側のペインに移動させると、それ以降の条件は、適切なオプションボタンをクリックして選択する論理積 (AND) または論理和 (OR) に従います。

右側のペインに表示された検索条件を選択し、左向きの矢印 (<) をクリックすると、この選択条件を右側のペインから削除し、編集できるように左側のペインの該当するボックスに戻すことができます。検索条件を右側のペインから左側のペインへ移動させると、自動生成されたクエリーコードから関連する COBOL コードが削除されます。

8.10 取り出すデータのソート方法

ソートするカラムを指定して (ORDER BY 句)、SELECT 文から返される行の順序を決定できます。OpenESQL アシスタントには、ORDER BY 句で順序を決定するカラムをすばやく簡単に選択できるダイアログボックスがあります。
SELECT 文から返されるデータの順序を指定するには、[ソート] タブをクリックします。

図 8-20：[ソート] タブ

ORDER BY 句に含めるカラムを指定するには、次の操作を実行します。

1. 「SELECT のカラム」リストボックスからカラム名を選択します。このリストボックスには、クエリーに対して現在選択されているすべてのカラムのカラム名が表示されます。
2. 「昇順」または「降順」オプションボタンの演算子をクリックして選択されたカラムのソート順序を選択します。
3. カラム名に数字を使用してクエリーを生成する場合は、「**カラム名に整数を使う**」をチェックします。この場合には、カラム名は「**ORDER BY のカラム**」フィールドに表示されていても、クエリーに対してカラムの整数値が生成されます。

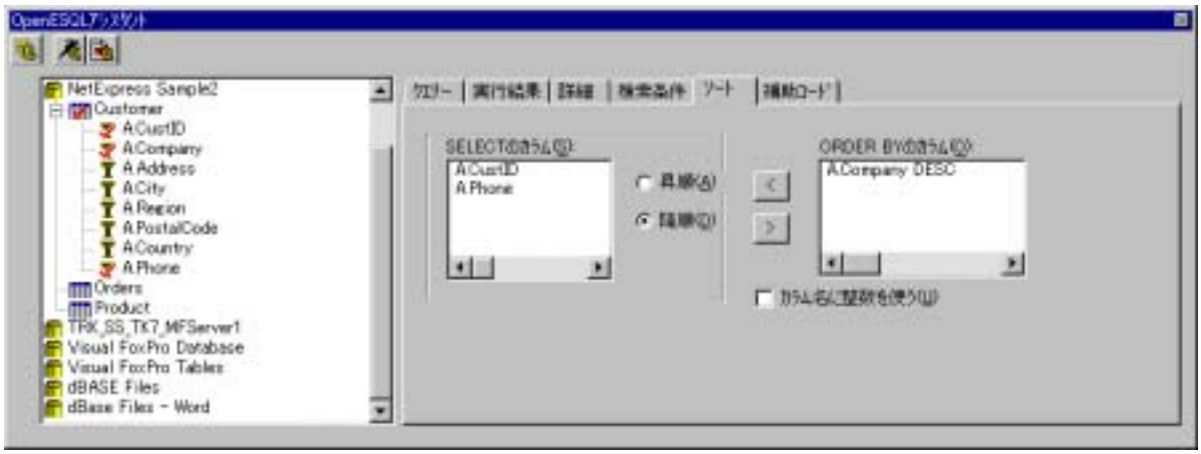


図 8-21：ORDER BY シーケンスの指定

4. 選択内容に問題がない場合は、右向きの矢印 (>) をクリックします。これにより、ORDER BY 選択項目が「**ORDER BY のカラム**」フィールドに移動し、同時にクエリーウィンドウ内の COBOL コードが更新されます。

図 8-22：ORDER BY 句をもつ SELECT 文

このクエリーを再実行 ([**クエリーを実行**] ボタン をクリック) すると、選択された行が会社名で降順にソートされます。

図 8-23：ORDER BY 句をもつ文の実行
削除するカラムを選択してから左向き矢印 (<) をクリックすると、ORDER BY 句からカラムを削除できます。

図 8-24：ORDER BY 句からカラムを削除
これにより、「**ORDER BY のカラム**」フィールドから選択項目が削除され、「**SELECT のカラム**」フィールドにカラムが戻されます。同時に、クエリーウィンドウ内の COBOL コードが更新されます。

8.11 データソースからの接続解除方法

データソースから接続解除するには、[**クエリーの新規作成**] ボタン をクリックします。

8.12 結合テーブルの作成方法

複数のテーブル間に共通するカラムが 1 つでもあれば、これらのテーブルを結合させ、複数のテーブルから同時に情報を検索できます。
たとえば、**UserSample2** から、ある製品を注文した全顧客の企業名と電話番号を検索すると仮定します。一方、製品の詳細情報は **Product** テーブルに保存されているとします。この場合は、次のように検索します。

1. OpenESQL アシスタントを起動し、**UserSample2** データソースへ接続します。
2. **Product** テーブルをダブルクリックし、クエリーの型として **SELECT (カーソル)** を選択して、[**OK**] ボタンをクリックします。
3. **Product** テーブルを右クリックし、[**すべてのカラムを選択**] をクリックします。

[**クエリーの実行**] ボタン をクリックしてクエリーを実行します。ここでは、16 番の製品 ID をもつ製品は「パブロワ」です。検索する情報は、パブロワを注文した全顧客の企業名と電話番号です。
Customer テーブルには、全顧客の企業名と電話番号が含まれていますが、注文された製品の製品 ID のような注文に関する詳細情報は、すべて **Orders** テーブルに保存されているとします。この場合は、必要な情報を表示するには、Customer テーブルから企業名と電話番号を表示し、この情報を Orders テーブルの製品 ID カラムを使用して選別する必要があります。そのためには、Customer テーブルと Orders テーブルをリンクさせる結合テーブルを作成する必要があります。OpenESQL アシスタントを使用すると、この作業を簡単に行えます。OpenESQL アシスタントで、SQL クエリーを作成し、2 番目以降のテーブルを選択すると、自動的にテーブルが結合されます。

1. **Product テーブルをダブルクリックして選択解除します。選択解除を確認するダイアログが表示されたら、[はい] ボタンをクリックします。**
2. **Customer テーブルをダブルクリックし、クエリーの型として「SELECT (カーソル)」を選択して、[OK] ボタンをクリックします。**
3. **A.CustID、A.Company、および A.Phone カラムをダブルクリックします。**

このクエリーを生成する COBOL コードは、次のように表示されます。

図 8-25：SELECT 文

この段階で、クエリーを実行すると、顧客ごとの企業名と電話番号が表示されます。
次に、Orders テーブルをダブルクリックして、結合テーブルを作成します。

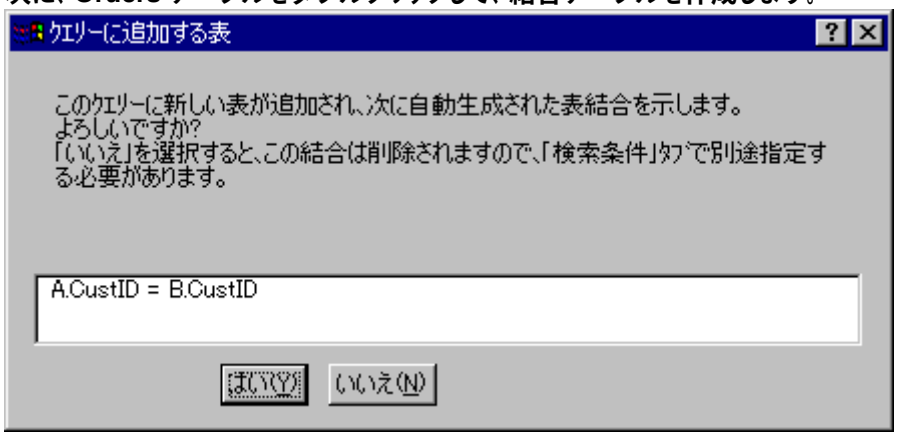


図 8-26：クエリーへのテーブルの追加

OpenESQL **アシスタントは、最初に一致するカラム** (**ここでは** CustID) **を使用して結合テーブルを作成します** (**この結合テーブル以外を使用するには、**[**いいえ**] **ボタンをクリックして、**[**検索条件**] **タブで別の結合テーブルを指定します**)。OpenESQL **アシスタントにより提示された結合テーブルを使用する場合は、**[**はい**] **ボタンをクリックします。その結果、**COBOL **コードに** WHERE **句が自動的に追加されます。**

図 8-27：COBOL コードへの結合テーブルの追加
次の操作を行って、クエリーを完成させます。

1. [**検索条件**] タブをクリックします。
2. 「**カラム**」リストボックスで、**B.ProdID** を選択します。
3. 「**条件演算子**」リストボックスで、= を選択します。
4. 「**受け側データ型**」で、**定数**を選択します。
5. 「**受け側値**」で、16 を入力します。
6. 右向きの矢印をクリックします。

論理積 (AND) の選択と挿入は、自動的に行われます。

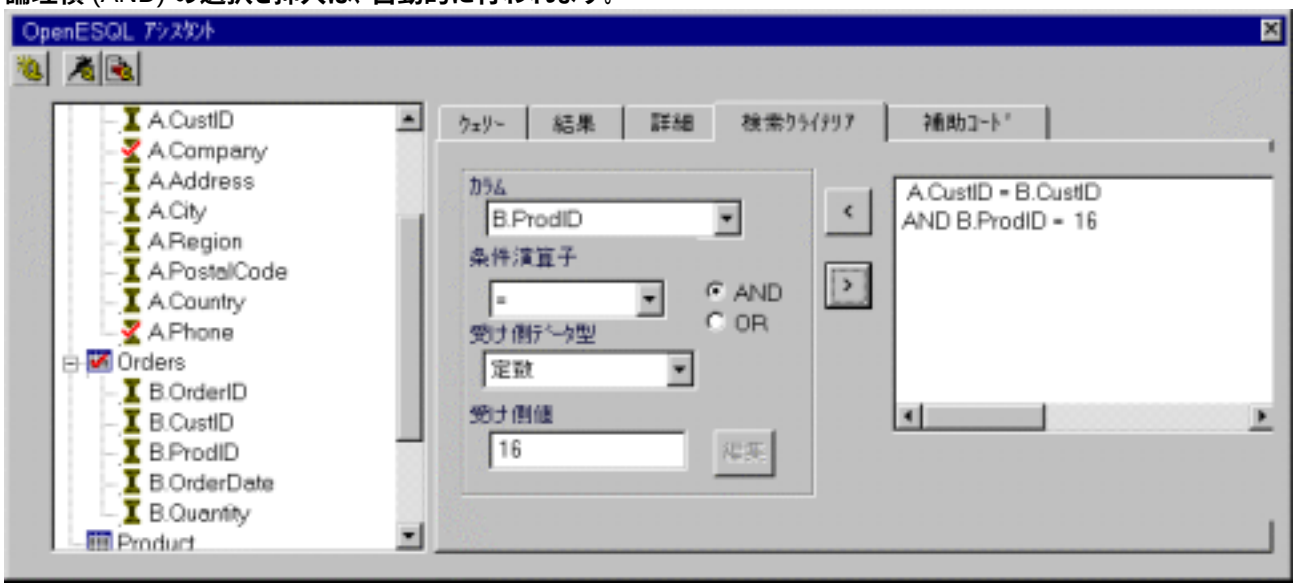


図 8-28：検索条件の指定
[**クエリー**] タブをクリックすると、WHERE 句が拡張され、新しい検索条件が追加されているのがわかります。

図 8-29：コードに追加された検索条件

[**クエリーの実行**] をクリックしてクエリーを再実行すると、製品 ID が 16 である製品 (パブロワ) を注文した顧客の企業名と会社番号のみが表示されます。

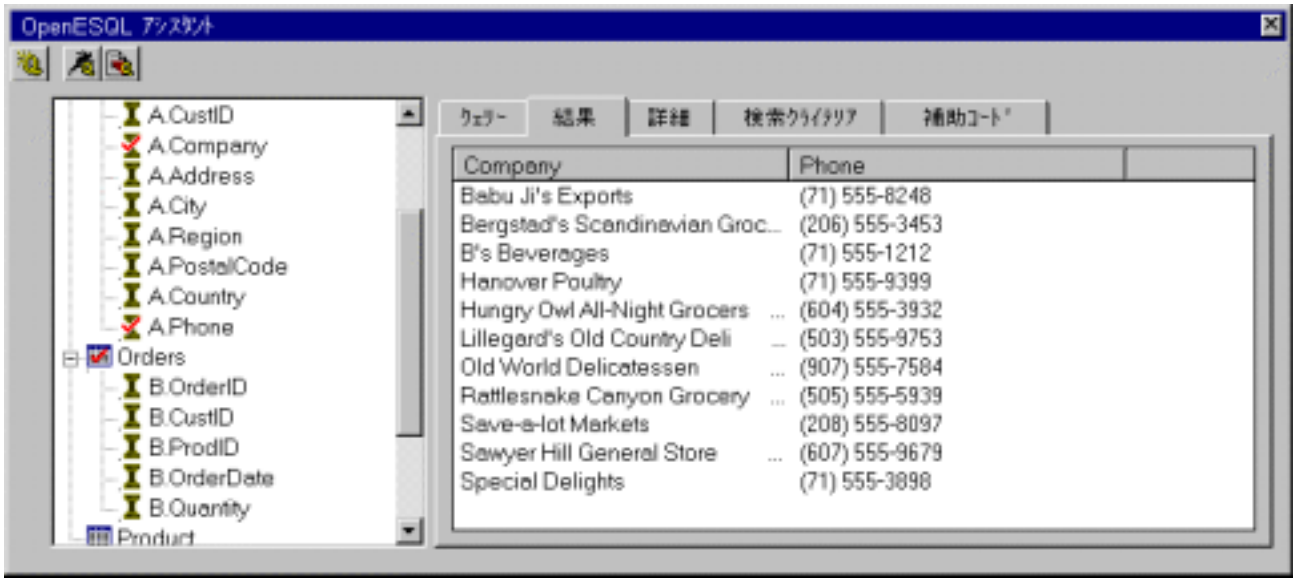


図 8-30：コードの実行

8.13 プログラムへの埋め込み SQL の追加方法

作成した SQL クエリーに問題がない場合には、[**現在のプログラムにクエリーを挿入する**] ボタン をクリックして、このクエリーをプログラムに追加します。このときは、Net Express で適切なプロジェクトが開いていること、また、コードを追加するプログラムも開いていて編集できる状態であることを確認してください。このチュートリアルでは、次の操作を行います。

1. Net Express の **Demo** ディレクトリに **OpenESQL** ディレクトリを作成します。

2. **OpenESQL** ディレクトリに **OpenESQL** という名前の新規プロジェクトを作成します。このプロジェクトは開いたままの状態にします。
3. 新規プログラムを作成します。このチュートリアルでは、プログラムが空白でもかまいませんが、この状態ではプログラムを保存できません。この場合は、**Enter** キーを押して、「reports.cbl」として保存してください。[**プロジェクト**] メニューの [**プロジェクトへのファイルの追加**] をクリックして、新規プログラムを **OpenESQL** プロジェクトに追加します。
4. カーソルをプログラムの最上部に移動し、[**現在のプログラムにクエリーを挿入する**] ボタン をクリックします。

これで、プログラムの現在の挿入箇所に SQL クエリーが追加されます。

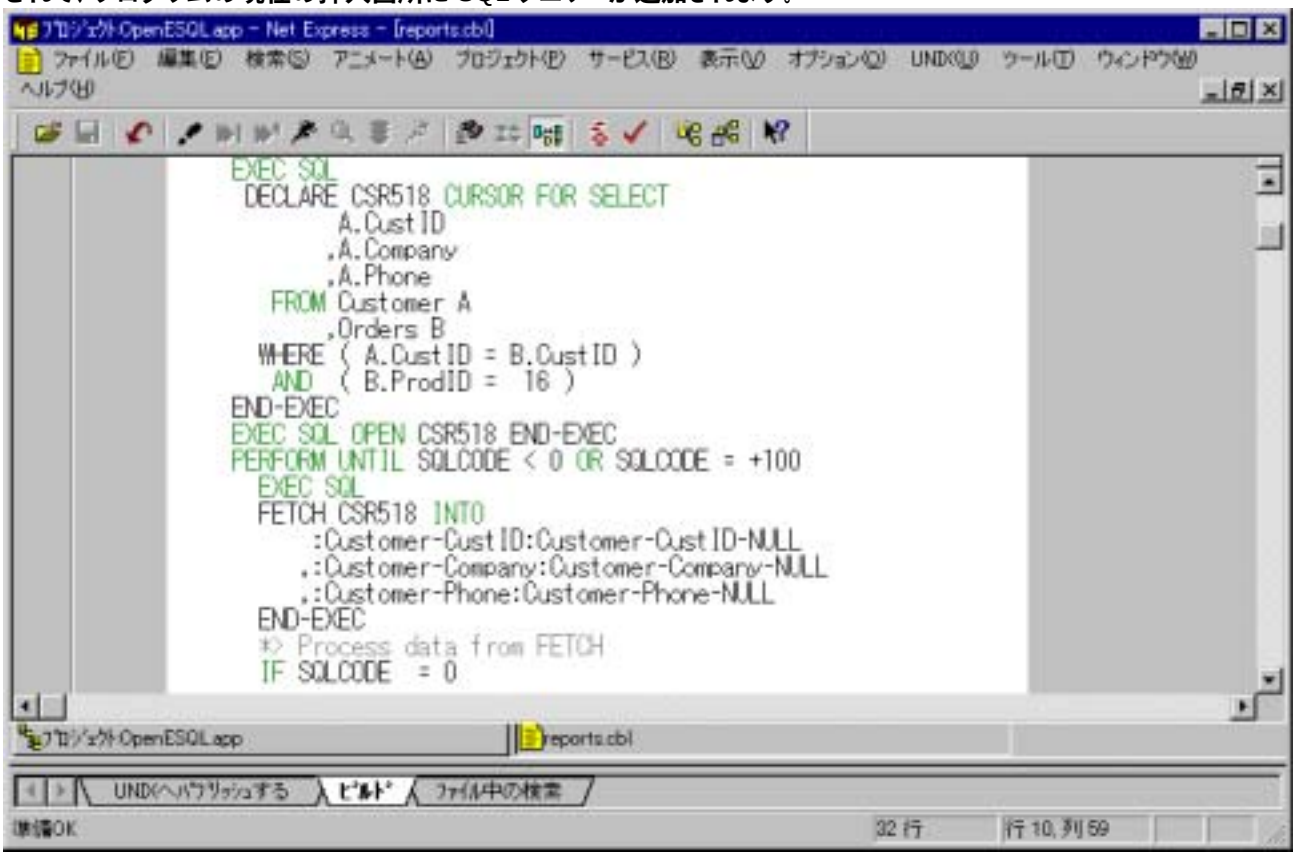


図 8-31：プログラムへの埋め込み SQL の追加

8.14 プログラムへの補助コードの追加方法

プログラムに埋め込み SQL を追加した後で、正常にプログラムを実行するために必要なコード修正を行います。たとえば、データベースへアクセスする SQL クエリーを正常に実行するには、データソースへ接続する CONNECT 文を挿入する必要があります。
OpenESQL アシスタントでは、必要な補助コードを自動生成できます。この機能を使用するには、[**補助コード**] タブをクリックします。「CONNECT 文」オプションボタンをクリックします。その結果、次のような画面が表示されます。

図 8-32：[補助コード] タブ

OpenESQL アシスタントを使用する場合は、次の文や宣言を挿入できます。

- 「CONNECT 文」

  現在のデータソース接続に関する情報を基に CONNECT 文を自動生成できます。

- 「SQLCA 宣言」

  INCLUDE 文を自動生成し、プログラムに SQL 通信領域 (SQLCA) を追加できます。

- 「ホスト変数の宣言」

  選択した各テーブルに対して、必要なホスト変数宣言をすべて含むコピーファイルが生成されます。また、OpenESQL アシスタントでは、INCLUDE 文を自動生成し、プログラムに各コピーファイルを追加することができます。

- 「DISCONNECT 文」

  DISCONNECT 文を自動生成し、現在のデータソースとの接続を解除できます。

- 「COMMIT 文」

  COMMIT 文を自動生成し、直前の SQL 文で実行されたデータソースへの変更をコミットできます。

- 「ROLLBACK 文」

  ROLLBACK 文を自動生成し、直前の SQL 文で実行されたデータソースへの変更をロールバックできます。

- 「汎用 SQL プログラム」

INCLUDE SQLCA、CONNECT、および DISCONNECT SQL 文を含むスケルトンの SQL プログラムと、デフォルトの SQL エラールーチンを自動生成し、簡単に起動できるようにします。

図 8-33：汎用 SQL プログラム

- 「入力インターフェイス LINKAGE」
- 「出力インターフェイス LINKAGE」

OpenESQL アシスタントでは、自動生成した補助コードをプログラム内の現在の挿入箇所に追加します。そのため、補助コードを挿入する前に、プログラム内のカーソル位置が正しいことを確認する必要があります。たとえば、CONNECT 文は、データソースへアクセスするコードの前に挿入する必要があります。また、SQLCA またはホスト変数を含むコピーファイルを追加する INCLUDE 文は、プログラムのデータ部に挿入する必要があります。
OpenESQL アシスタントを使用して挿入するコードと、手動で挿入するコードを選択できます。
OpenESQL アシスタントを使用して、補助コードを生成し挿入するには、次の操作を行います。

1. 「**汎用 SQL プログラム**」などの、生成するコードのタイプの次にあるオプションボタンをチェックします。
2. プログラム内のカーソル位置が正しいことを確認します。
3. [**現在のプログラムにクエリーを挿入する**] ボタン をクリックします。

これで、プログラム内の現在の挿入箇所に補助コードが挿入されます。

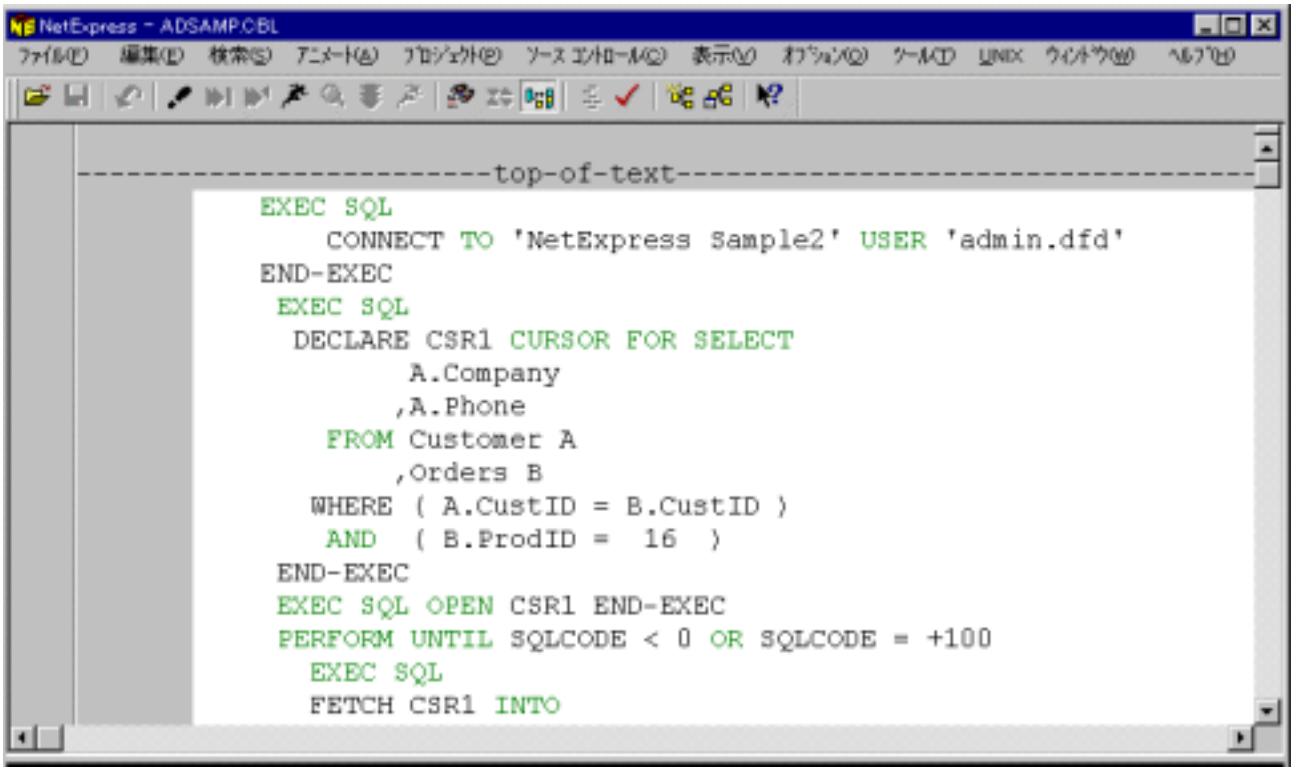


図 8-34：補助コードの挿入

8.15 SELECT (カーソル) クエリーの変更による配列取り込みの実行方法

OpenESQL アシスタントでは、配列取り込みのためのコードは直接生成されません。ただし、OpenESQL アシスタントで生成された SELECT (カーソル) コードを変更し、配列取り込み処理するのは非常に簡単です。これを行うには、次の操作を実行します。

- 埋め込み SQL オプションで「**ホスト変数に SQL TYPE を使用する**」が設定されていないことを確認します。このコンパイラでは、SQL TYPE ホスト変数で定義された OCCURS 句は現在サポートされていません。
- ホスト変数のコピーブックを生成する場合は、「**ホスト変数配列サイズ**」フィールドの値を、各配列取り込みで返される最大行数に変更します。次に示す例では、この値が「20」に設定されています。

図 8-35：「ホスト変数配列サイズ」の設定

- SELECT (カーソル) 文に対して生成されたコードの FETCH 文の後の PERFORM/END-PERFORM 文ブロックから (次に表示されている) 注釈を削除します。

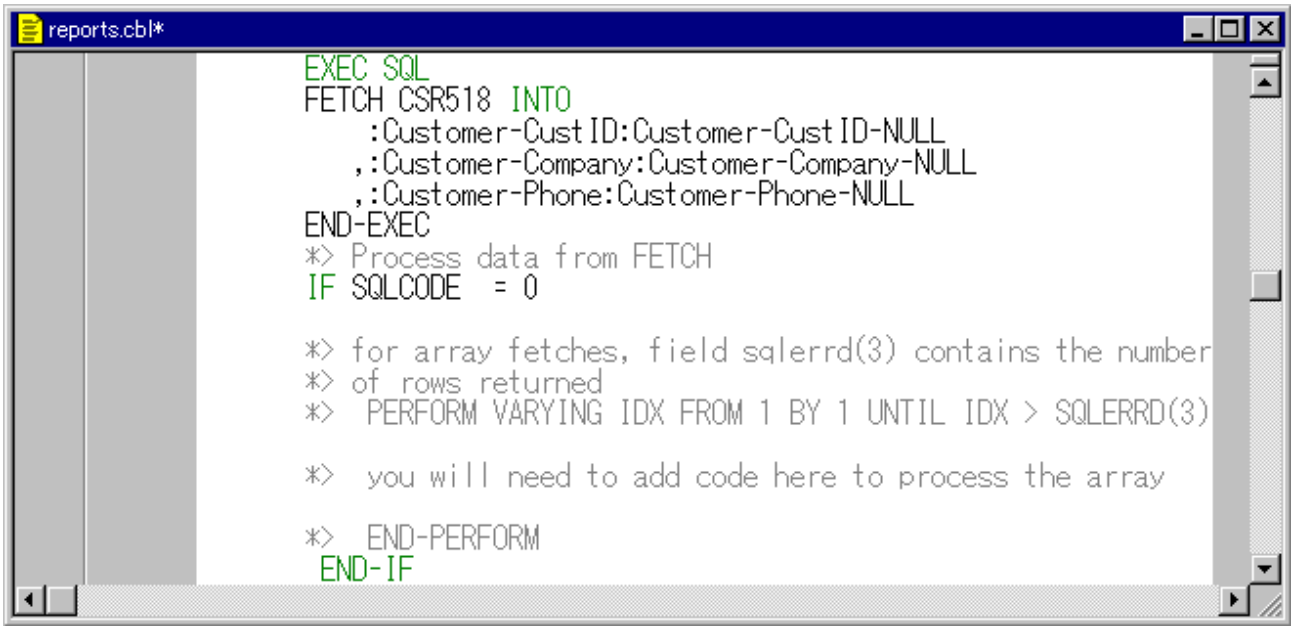


図 8-36：配列取り込み処理の変更コーディング
GENERIC スタブを使用しない場合は、指標変数 IDX を定義するか、または、配列ループを処理するために定義されている変数を使用するコードを変更する必要もあります。

8.16 ストアドプロシージャとしてのクエリーの生成方法

Microsoft は、ストアドプロシージャを作成、編集、およびテストするツールを提供しています。プログラマは、COBOL クライアントプログラムからこれらのストアドプロシージャを呼び出すコードを生成する必

要があります。OpenESQL アシスタントは、このプロセスをより簡単に行えるように機能拡張されました。
Microsoft SQL Server データソースを使用している場合は、OpenESQL アシスタントで、クエリーをストアドプロシージャとして生成するオプションを使用できます。このオプションを使用する場合は、ストアドプロシージャと、そのプロシージャを呼び出すクライアントコードを作成する 2 つの SQL 文が生成されます。

8.16.1 ストアドプロシージャ用のクエリーの型の選択方法

SQL Server データソースに接続した後に、クエリーの作成に使用するテーブルを選択します。「**生成するクエリーのタイプを選択**」ウィンドウが表示されます。

1. 「**クエリーをストアドプロシージャとして生成**」をチェックします。
2. 作成するクエリーの型を選択します。
3. 「**SP 名**」エントリフィールドでストアドプロシージャの名前を指定します。

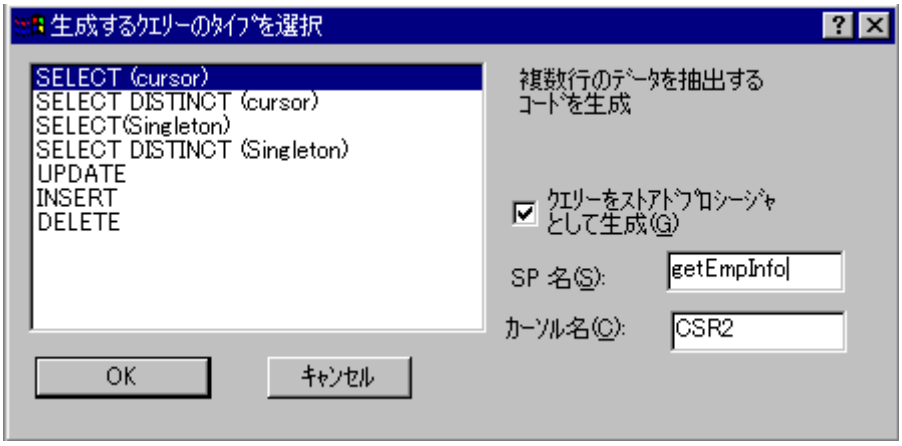


図 8-37：クエリーをストアドプロシージャとして生成
この画面では、ストアドプロシージャの名前は「getEMpInfo」です。OpenESQL アシスタントでは、ストアドプロシージャを削除し、作成する 2 つの SQL 文が生成されます。これらの SQL 文は 1 回のみ実行する必要があるため、通常はサーバプログラムに記述されます。

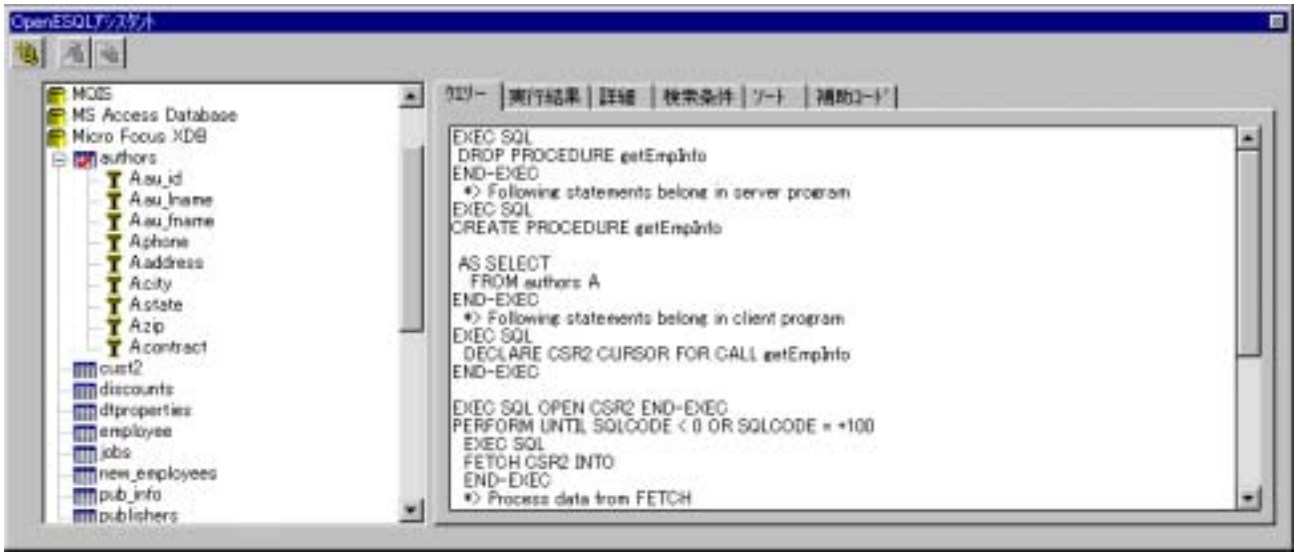


図 8-38：ストアドプロシージャ用に生成された初期文
OpenESQL アシスタントでは、ストアドプロシージャを呼び出すためにクライアントプログラムに記述するコードも生成します。この例では、ストアドプロシージャが結果集合を生成しているため、OpenESQL アシスタントでは、ストアドプロシージャを呼び出す DECLARE CURSOR 文が生成されます。他のクエリーの型では、SQL CALL 文のみが生成されます。

8.16.2 Create Procedure 文の作成方法

Create Procedure 文を作成するには、次の手順を実行します。

1. クエリーに含めるカラムを選択します。
2. ソートするカラムを選択します。
3. 結果を制限するために必要な検索基準を指定します。

選択によってさまざまなクエリーが作成されます。作成されたクエリーを次に示します。

図 8-39：getEmpInfo クエリー

8.16.3 ストアドプロシージャのテスト方法

ストアドプロシージャ文を作成した後に、[**クエリーの実行**] ボタン をクリックして、その文をテストできます。クエリーを最初に実行するときに、DROP PROCEDURE 文が実行されます。次のダイアログが表示されます。

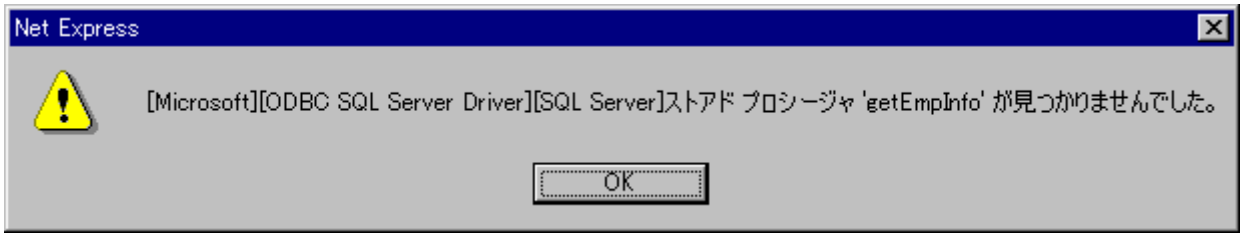


図 8-40：Drop Procedure の実行

SQL クエリーは必要に応じて何箇所でも変更できます。DROP PROCEDURE を複数回実行する場合は、それぞれ異なるメッセージが表示されますが、これは通常の動作です。
ストアドプロシージャのテストに使用するプロシージャは、SQL クエリーをテストするプロシージャに似ています。ただし、OpenESQL アシスタントでは、ストアドプロシージャ用に複数の SQL 文が作成されるので、[**クエリーの実行**] ボタンを複数回クリックする必要があります。[**クエリーの実行**] ボタンをクリックするたびに、次の SQL 文が実行されます。
次にクエリーを実行すると、CREATE PROCEDURE 文が実行されます。次のダイアログが表示されます。

図 8-41：Create Procedure の実行

ストアドプロシージャの CALL 文を実行するには、[**クエリーの実行**] ボタンをもう一度クリックします。検索条件が指定されているため、次のウィンドウが表示されます。作成される結果集合を制限するために国の値を入力するよう求められます。

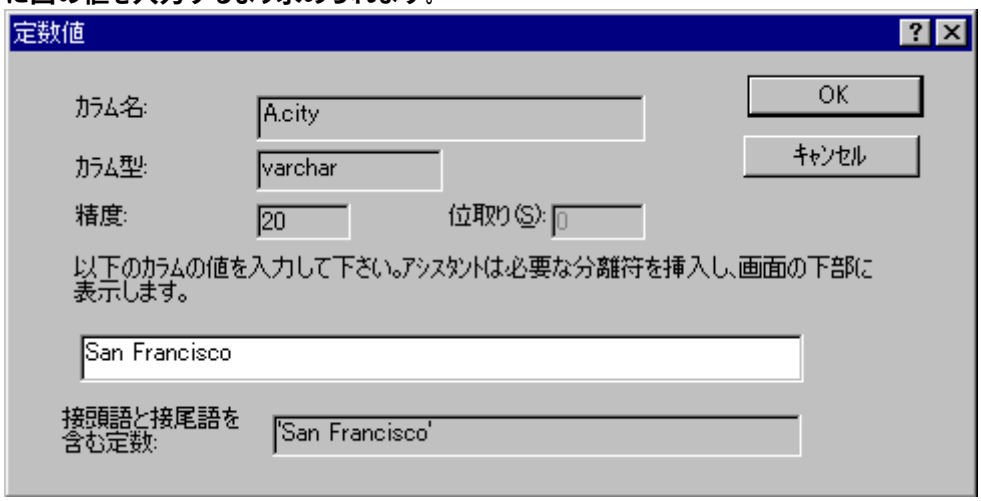


図 8-42：クエリー用の値の入力

値を入力して [OK] ボタンをクリックします。クエリーが実行され、結果が表示されます。

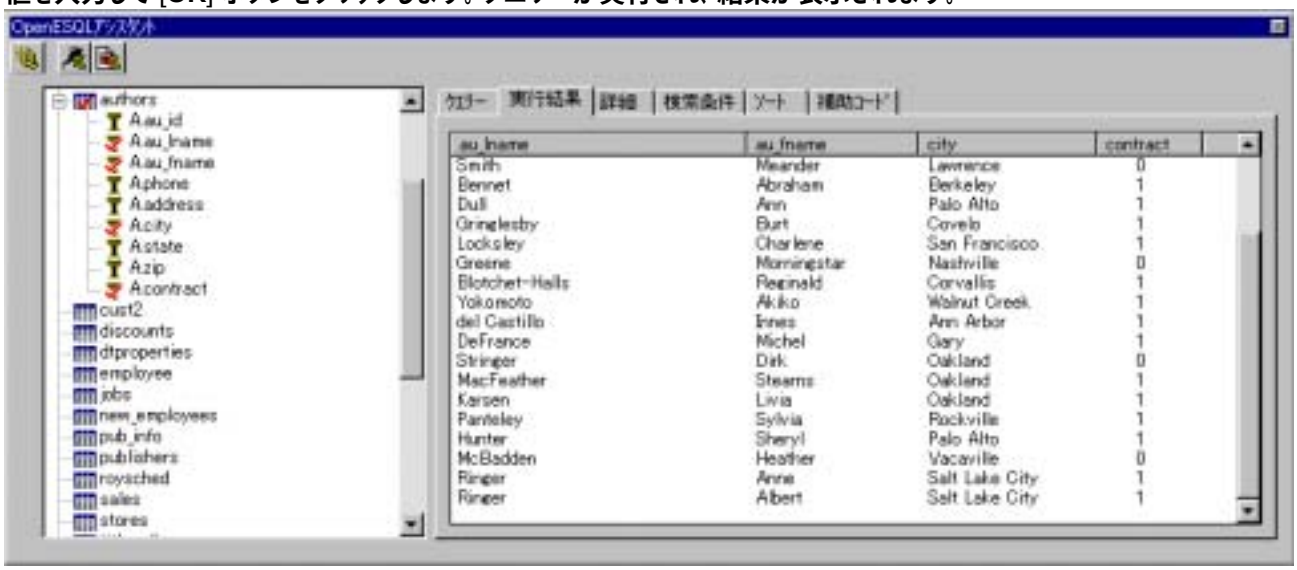


図 8-43：ストアドプロシージャ呼び出し結果
正しい結果になるまで、この処理を繰り返します。

8.16.4 ストアドプロシージャへのコードの追加方法

コードをプログラムに追加するには、プログラム内でコードを追加する箇所にカーソルを置き、[**現在のプログラムにクエリーを挿入する**] ボタン をクリックします。

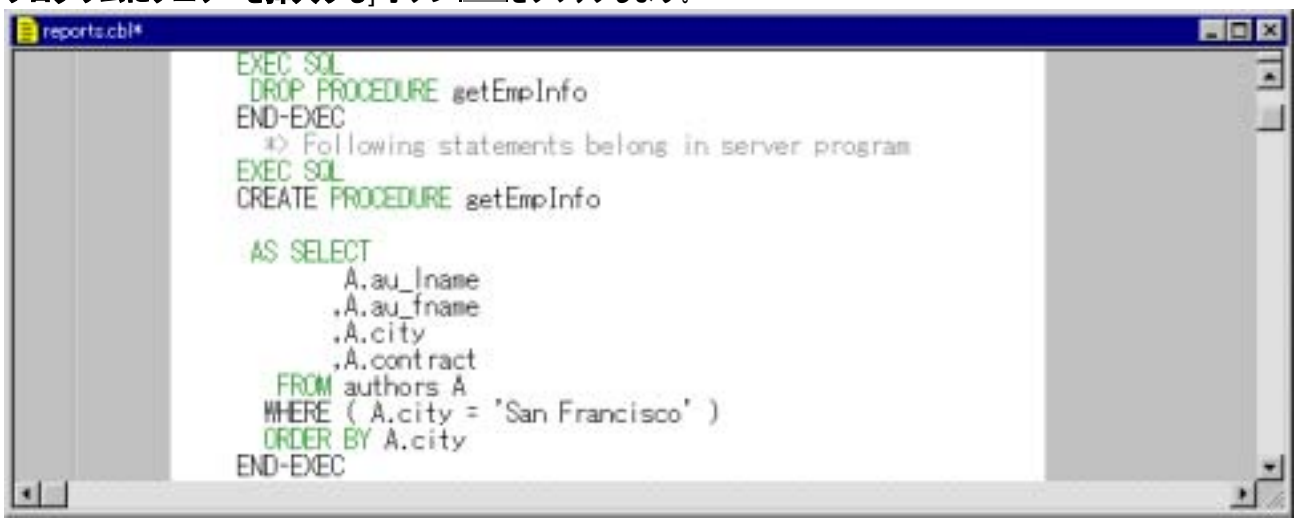


図 8-44：クライアントプログラムへのストアドプロシージャ文の追加
OpenESQL アシスタントでは、クライアントプログラムとサーバプログラムの両方に文が生成されます。不要な文は削除できます。この例でのプログラムは、クライアントプログラムであるため、DROP PROCEDURE 文と CREATE PROCEDURE 文を含める必要はありません。

注：前述の方法でクエリーをテストする場合は、ストアドプロシージャがすでに作成されているため、DROP PROCEDURE 文または CREATE PROCEDURE 文を実行する必要はありません。ストアド

プロシージャがまだ作成されていない場合は、SQL COMMIT 文または SQL(AUTOCOMMIT) 指令を、CREATE PROCEDURE を実行するプログラムに追加します。この方法を実行しないと、ストアドプロシージャはデータソースに保存されません。

8.16.5 追加機能

ストアドプロシージャ内には複数のタイプの SQL 文を記述できます。OpenESQL アシスタントでは、複数のタイプの SQL 文をもつストアドプロシージャを生成できません。そのため、これを行うには、Microsoft が提供しているツールなどを使用します。
OpenESQL アシスタントは、簡単なストアドプロシージャと、これらのストアドプロシージャを呼び出すために必要な COBOL コードの生成方法をすばやく理解するために提供されています。

8.17 OpenESQL アシスタントの終了方法

OpenESQL アシスタントを終了するには、画面右上隅の [**閉じる**] ボタンをクリックします。

8.18 OpenESQL アシスタントの Unicode サポート

OpenESQL アシスタントは、Unicode データをサポートするようになりました。OpenESQL アシスタントでは、ホスト変数の生成方法によって Unicode データの表示が異なります。ホスト変数のオプションを設定するには、Net Express IDE の [**オプション**] で、[**埋め込み SQL**] を選択します。次のようなダイアログが表示されます。

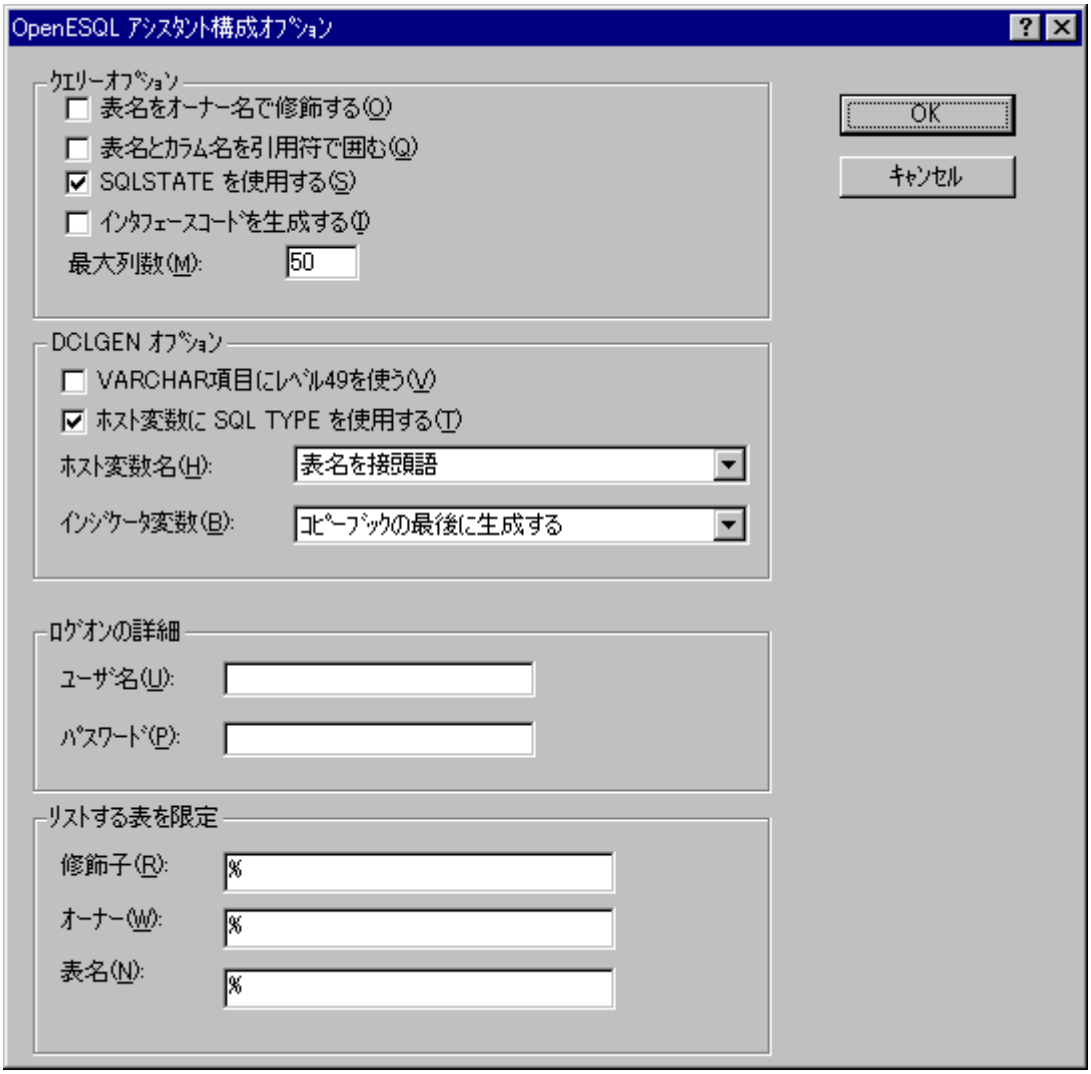


図 8-45：「OpenESQL アシスタント構成オプション」ダイアログ

「DCLGEN オプション」グループボックスで、「**VARCHAR 項目にレベル 49 を使う**」または「**ホスト変数に SQL TYPE を使用する**」チェックボックスをどちらもチェックを外すと、NCHAR、NVARCHAR、および NTEXT カラム型では、ホスト変数が PIC N(xx) USAGE NATIONAL 定義を使用して生成されます。これらの型のカラムを選択してクエリーを実行すると、[クエリー] タブに Unicode 文字列のデータが表示されます。

どちらかのボックスをチェックすると、同じクエリーは、以前のバージョンの Net Express と同じ形式でその結果が表示されます。

第 9 章 : DB2

ここでは、Net Express を使用してコンパイルおよびリンクされた、埋め込み SQL 文をもつ COBOL プログラムから DB2 データベースへアクセスする方法を説明します。
DB2 外部コンパイラモジュール (ECM) は、Net Express が提供する統合プリプロセッサであり、Micro Focus COBOL コンパイラとより密接に動作するように設計されています。DB2 ECM により、埋め込み SQL 文は DB2 データベースサービスへの適切な呼び出しに変換されます。

9.1 文字データ

文字データに対して適切なデータ型を使用するように注意が必要です。254 文字以下の固定長の文字列の場合は、CHAR データを使用します。254 文字を超える固定長文字列、および可変長文字列の場合は、VARCHAR データを使用します。詳細は、『*データ型*』の章を参照してください。

9.1.1 追加データ型

『*データ型*』の章で説明しているデータ型以外にも、DB2 は DECIMAL データ型をサポートします。DECIMAL データ型は、パック 10 進数項目を記述します。小数点がある場合とない場合があります。COBOL ではこのような項目を COMP-3 または PACKED-DECIMAL として宣言できます。
追加データ型は、次の SQL 構文を使用して宣言してください。

```
>>--level_number--name--+----------------+--SQL--+---------------+-->
                        |                |       |               |
                        +--USAGE--+------+       +--TYPE--+------+
                                  |      |                 |      |
                                  +--IS--+                 +--IS--+

 >--sql_type--+--------------+--><
              |              |
              +--(--size--)--+
```

構文の詳細は、次のとおりです。

level_number 1 ～ 48 の範囲内。

sql_type 次の新 SQL データ型の 1 つ。BLOB、CLOB、DBCLOB、BLOB-FILE、CLOB-FILE、DBCLOB-FILE、BLOB-LOCATOR、CLOB-LOCATOR、DBCLOB-LOCATOR、または TIMESTAMP。TIMESTAMP 型 は DB2 V5.2 では新型でないので、DB2 ECM でも提供されています。

size BLOB 型、CLOB 型、および DBCLOB 型の場合には必ず指定します。K (KB)、M (MB)、または G (GB) で修飾できます。

VALUE 句は、新 SQL データ型では許可されていません。
指定する *sql_type* によって、実際に生成されるデータは基本項目または集団項目になります。集団項目の要素名は自動的に生成されます。
次の表は、SQL 構文を使用して生成したデータ項目の構造を、等価のネイティブな COBOL 定義で示したものです。どちらの場合も同じデータが生成されますが、DB2 ECM で使用可能なホスト変数として認識されるには、項目は SQL 構文を使用して宣言する必要があります (これは、COBOL の定義があいまいなためです。多くの新しい SQL 型と個々のホスト変数に展開される集団項目は、COBOL では区別できません)。既存のデータ型はすべて引き続き、通常の COBOL 構文を使用して宣言してください。例外は TIMESTAMP のみです。TIMESTAMP はどちらの形式を使用しても宣言できます。

SQL 構文	等価の COBOL 構文

01 MY-BLOB SQL BLOB(125M).	01 MY-BLOB. 49 MY-BLOB-LENGTH PIC S9(9) COMP-5. 49 MY-BLOB-DATA PIC X(131072000).
03 A SQL CLOB(3K).	03 A. 49 A-LENGTH PIC S9(9) COMP-5. 49 A-DATA PIC X(3072).
03 HV SQL DBCLOB(125).	03 HV. 49 HV-LENGTH PIC S9(9) COMP-5. 49 HV-DATA PIC G(125).
01 B SQL BLOB-LOCATOR.	01 B PIC S9(9) COMP-5.
01 C SQL CLOB-FILE.	01 C. 49 C-NAME-LENGTH PIC S9(9) COMP-5. 49 C-DATA-LENGTH PIC S9(9) COMP-5. 49 C-FILE-OPTIONS PIC S9(9) COMP-5. 49 C-NAME PIC X(255).
01 TS SQL TIMESTAMP.	01 TS PIC X(29).

9.2 複合 SQL

現在 DB2 V2 で使用可能な拡張形式も含め、複合 SQL をサポートしています。不完全な複合 SQL 文は DB2 ECM で検知され、エラーが発生します。ただし、DB2 はこの状態から常に回復するとは限りません。プログラムソース後半の有効な SQL 文がさらにエラーを起こす場合がありますので、注意してください。

9.3 ユーザ定義関数

ユーザ定義関数 (UDF) への参照を含むプログラムは、他のモジュールを呼び出します。これは、そのモジュールに、適切な値を返す、ユーザ提供のコードが含まれているからです。UDF コード自体には SQL が含まれません。
埋め込み SQL 文を含むプログラムを実行すると、DB2 が呼び出されます。さらに DB2 が UDF モジュールを呼び出すこともあります。UDF を宣言する際には、このモジュールが記述された言語を指定する必要があります。一部のプラットフォームでは COBOL でモジュールを記述できますが、現在 DB2 で指定できる言語は C のみです。次の例は、これらがどのように行われるかを示しています。ユーザ定義関数とパラメータ記述の詳細については、DB2 のマニュアルに記載されています。
COBOL で記述されたユーザ定義関数は、現在 UNIX ではサポートされていません。

注:クライアント / サーバの構成では、UDF モジュールはサーバ上で呼び出され、これらの制限はサーバのみに適用されます。サーバに問題がない場合は、どのクライアントも UDF にアクセスでききます。

UDF のエントリポイントは、C 呼び出し規約を使用して定義してください。次のサンプルコードでは、指数を計算する単純な UDF を使用および定義しています。
次のプログラム 1 は DB2 への関数を宣言します。このプログラムをコンパイルおよび実行してから、プログラム 2 をコンパイルする必要があります。

```
exec sql
   create function mfexp(integer, integer)
      returns integer
      fenced
      external name 'db2v2fun!mfexp'
      not variant
```

```
      no sql
      parameter style db2sql
      language cobol
      no external action
end-exec
```

LANGUAGE COBOL 句に注目してください。これは、DB2 構文への拡張として Micro Focus COBOL が提供するものです。この句は LANGUAGE C と等価で、どちらを使用しても、呼び出されるモジュールは C 呼び出し規約に準拠しています。ここでは、EXTERNAL NAME 句で、呼び出されるモジュールの名前を db2v2fun (プラットフォームにより .dll または .dlw の拡張子)、この中のエントリポイントの名前を mfexp と指定します。

次のプログラム 2 は、この UDF を使用します。

```
     move 2 to hv-integer
     move 3 to hv-integer-2
     exec sql
         values (mfexp(:hv-integer, :hv-integer-2))
          into :hv-integer-3
     end-exec
```

次のプログラム 3 は、この UDF そのものを含む、純粋な COBOL プログラムです。

```
$set case
 special-names.
     call-convention 0 is cc.
 linkage section.
 01  a pic s9(9) comp-5.
 01  b pic s9(9) comp-5.
 01  c pic s9(9) comp-5.
 01  an pic s9(4) comp-5.
 01  bn pic s9(4) comp-5.
 01  cn pic s9(4) comp-5.
 01  udf-sqlstate pic x(6).
 01  udf-fname pic x(28).
 01  udf-fspecname pic x(19).
 01  udf-msgtext pic x(71).
 procedure division cc.
    goback
    .
 entry "mfexp" cc
     using a b c an bn cn
           udf-sqlstate
           udf-fname
           udf-fspecname
           udf-msgtext.
     if an not = 0 or bn not = 0
         move -1 to cn
     else
         compute c = a ** b
         move 0 to cn
     end-if
     goback
     .
```

このモジュールは、動的にロード可能な実行可能プログラム (dll) を作成するためにコンパイルして、オペレーティングシステムが (PATH 上で) 検索できる場所に置く必要があります。

注: エントリポイント名は、すべてのシステムで大文字と小文字が区別されます。大文字と小文字の名前をマッチングさせる場合には、注意が必要です。また、上記のプログラム例での $SET 文 のように、CASE コンパイラ指令を指定する必要があります。

9.4 埋め込み SQL サポートの拡張

ここでは、埋め込み SQL サポートの Micro Focus 拡張を説明します。

9.4.1 INCLUDE 文

次の形式の文があります。

```
exec sql
   include filename
end-exec
```

この文は、次の文とまったく同じように許可および処理されます。

```
copy filename
```

インクルードされたファイルには、コピーファイルが含むことのできる COBOL 文をすべて含むことができます。さらに EXEC SQL 文を含むこともできます。

UNIX

どのように指定しても、AIX では sqlca または sqlda の特殊な場合としてファイル名は小文字に変換されます。

9.4.2 DECLARE TABLE 文

次の形式の文があります。

```
exec sql
   DECLARE table-name TABLE
   ...
end-exec
```

この文は許可され、注釈として扱われます。

9.4.3 整数ホスト変数

埋め込み SQL サポートでは、整数の形式を USAGE COMP-5 にする必要があります。ユーザが使用しやすいように、DB2 ECM ではホスト変数に USAGE COMP、COMP-4、および BINARY を使用することもでき、これらの形式を変換する追加コードを生成します。最も効率的なコードが生成されるのは、COMP-5 を使用した場合です。

9.4.4 修飾付きホスト変数

ホスト変数は、DB2 for MVS 互換構文を使用して修飾できます。

たとえば、次のようなホスト変数を定義します。

```
01 block-1.
   03 hostvar pic s9(4) comp-5.
01 block-2.
   03 hostvar pic s9(4) comp-5.
```

次のように修飾して、hostvar のどちらのインスタンスを構文で使用するかを指定できます。

```
exec sql
   fetch s2 into :block-1.hostvar
end-exec
```

9.4.5 ホスト変数集団とインジケータ配列

集団項目内でホスト変数が宣言されると、これらの変数をそれぞれ参照する必要のある SQL 文が、かわりに集団名を参照することによって、参照を短縮できます。インジケータ変数をこれらのホスト変数と関連付ける必要がある場合には、ホスト変数と同じ数のインスタンスをもつインジケータ変数のテーブルを定義して、このテーブルを参照します (OCCURS 句をもつ項目です。その項目を含む集団項目ではありません)。
たとえば、次のようなホスト変数を定義します。

```
01  host-structure.
   03 sumh           pic s9(9) comp-5.
   03 avgh           pic s9(9) comp-5.
   03 minh           pic s9(9) comp-5.
   03 maxh           pic s9(9) comp-5.
   03 varchar.
      49 varchar-l   pic s9(4) comp.
      49 varchar-d   pic x(1000).
01  indicator-table.
   03 indic          pic s9(4) comp-5 occurs 4.
01  redefines indicator-table.
   03 indic1         pic s9(4) comp-5.
   03 indic2         pic s9(4) comp-5.
   03 indic3         pic s9(4) comp-5.
   03 indic4         pic s9(4) comp-5.
```

次の手続き文があります。

```
exec sql fetch s3 into
  :host-structure:indic
end-exec
```

この手続き文は、次の文と等価です。

```
exec sql fetch s3 into
  :sumh:indic1, :avgh:indic2, :minh:indic3,
  :maxh:indic4, :varchar
end-exec
```

宣言された 4 つのインジケータ変数が、はじめの 4 つのホスト変数に割り当てられます。もし 5 つ以上が宣言された場合には、5 つのホスト変数すべてが、1 つのインジケータ変数に関連付けられます。インジケータ変数のテーブルは、等価の SQL 文を示すためのみに再定義されます (添字指定は SQL 文では許可されていません)。再定義を省略し、COBOL プログラムで添字指定を使用してインジケータ変数を参照することもできます。

9.4.6 NOT 演算子

DB2 では、演算子 ¬=、¬>、および ¬< を使用できます。これらは、<>、<=、および >= にマップされます。NOT 演算子の文字表現はシステムによって異なるので、DB2 コンパイラ指令の NOT オプションを使用して NOT 演算子を定義できます。

9.4.7 連結演算子 (|)

国によっては、連結演算子に使用される記号は ASCII 文字の (|) ではありません。DB2 ECM では、DB2 コンパイラ指令の CONCAT オプションを使用して、他の ASCII 文字を連結演算子に指定できます。

9.4.8 SQL 通信領域

SQL 文を実行すると、プログラムの SQL 通信領域 (SQLCA) と呼ばれる領域に重要な情報が返されます。通常、SQL 通信領域は、次のような文を使用してユーザのプログラム内に含まれています。

```
exec sql include sqlca end-exec
```
この文により、Windows では sqlca.cpy、UNIX では sqlca.cbl というソースファイルがユーザのソースコードにインクルードされます。このソースファイルは、DB2 ECM によって提供され、SQLCA の COBOL 定義を含んでいます。
この文を含めない場合には、DB2 ECM は自動的に領域を割り当てますが、プログラム内でこの領域のアドレスを指定できません。ただし、SQLCODE および SQLSTATE のどちらか一方、または両方を宣言する場合は、DB2 ECM はコードを生成して、各 EXEC SQL 文の後に、SQLCA 領域の対応フィールドをユーザが定義したフィールドにコピーします。SQLCA 全体を定義することをお奨めします (この機能は ANSI 互換のために用意されています)。
MFSQLMESSAGETEXT が定義されている場合には、SQLCODE の非ゼロ条件の後ろで、DB2 ECM は 例外条件の記述をもつ MFSQLMESSAGETEXT データ項目の内容を更新します。この場合は、文字データ項目 PIC X(n) として宣言してください (n には有効な値を指定できます。このメッセージがデータ項目に入りきらない場合には、切り捨てられます)。
SQLCA、SQLCODE、SQLSTATE、および MFSQLMESSAGETEXT は、ホスト変数として宣言する必要はありません。

9.4.9 オブジェクト指向 COBOL 構文のサポート

DB2 ECM はオブジェクト指向 COBOL の構文 (OO プログラム) と動作するように拡張されました。ただし、いくつか制限がありますので、注意してください。

- DB2 コンパイラ指令の INIT オプションは、OO プログラム内で使用されると無効になります。つまり、DB2(INIT=PROT) コンパイラ指令の機能を使用する場合には、非オブジェクト指向モジュールを使用し、そのモジュールを DB2 コンパイラ指令でコンパイルする必要があります。
- METHOD 内で EXEC SQL WHENEVER 文を使用する場合には、SQL 文をもつ同一 CLASS 内でコード化された追加 METHOD は、先行する定義済み WHENEVER 文内で参照される節をもつ必要があります。これを行わない場合は、節が定義されていないことを示すコンパイルエラーが発生します。この制限は、他に EXEC SQL WHENEVER 文を定義することで回避できます。

9.4.10 入れ子の COBOL プログラムのサポート

DB2 ECM では、入れ子の COBOL プログラムを使用できます。
デフォルトでは、入れ子の COBOL プログラムのすべてに DB2 インターフェイスコードが生成されます。入れ子のプログラムごとに DB2 インターフェイスコードが生成されないようにするには、DB 指令の IGNORE-NESTED を使用してください。IGNORE-NESTED 指令を適切に使用するには、次の制限に注意してください。

- DB2 インターフェイスコードを生成するプログラム内で PROGRAM-ID 文を指定する必要があります。この文を指定しない場合は、コンパイルエラーが発生します。

9.5 DB2 INIT 指令

DB2 の初期のバージョンでは、実行時に SQL 構文でデータベースへ接続する方法を提供していませんでした。そのため、呼び出しを適切な SQL API ルーチンにコード化する必要がありました。Micro Focus 製品の以前のバージョンは、CONNECT 機能を行うために SQLINT モジュールまたは SQLINI2 モジュールを提供していました。これらのルーチンは現在は提供されていません。そのかわ

りに、DB2 ECM が DB2 コンパイラ指令の INIT オプションの設定に応じて、適切な CONNECT 文を生成します。
INIT 指令には、アプリケーションが異常終了した場合でも、データベース接続を正しく終了させる追加オプションがあります。これにより、データベースの破損を防ぎます。アプリケーションが異常終了した場合には、最後に行った COMMIT 以降の変更はすべてロールバックされます。このデータベース保護は、DB2 コンパイラ指令で INIT=PROT オプションを指定することで選択できます。
INIT=PROT オプションは、1 つのアプリケーションに対して一度しか設定できません。他の SQL プログラムから呼び出された SQL プログラムには INIT=PROT オプションを設定できません。そのかわりに、実行単位で最初に実行される SQL プログラムに INIT=PROT オプションを指定することができます。INIT=PROT オプションが設定されているアプリケーションで複数のモジュールをコンパイルする場合は、プログラムが異常終了する可能性があります。
FixPack 8 以降の DB2 UDB 7.2 を使用している場合に、INIT 指令オプションで PASS 指令オプションを指定する場合は、INIT ではデータベースの接続に使用するユーザ ID とパスワードに対して空白のホスト変数は生成されません。このバージョンの DB2 IDB では、これらの変数を空白文字または low-values で CONNET に渡すと、SQL エラーが生成されます。これは、INIT オプション動作の変更点の 1 つです。これらの値でコンパイラされたプログラムは機能しません。プログラムではこれらの指令を使用しないで、SQL CONNECT 文を使用することをお奨めします。

9.6 UDB-VERSION DB2 指令オプション

Net Express ECM は、使用している DB2 ユニバーサルデータベースのバージョンに応じて、2 セットある API 呼び出しのどちらかを使用できます。この呼び出しセットは、IBM 社がバージョン 7 から導入した新機能を使用できるようにするため、DB2 UBD バージョン 7.1 以降で使用されます。このバージョンで、IBM 社はプログラム識別子文字列 (PID) の長さを 40 バイトから 162 バイトに変更しました。この PID は、コンパイル時に各アプリケーションプログラムに格納されます。8 文字を超える COLLECTION-ID を使用する場合は、新しい PID 構造体を使用する必要があります。
DB2 ECM は、バージョン 7.1 以降の DB2 UDB サーバに接続してプログラムをコンパイルしたときに、自動的により新しい呼び出しセットの使用を試みます。UDB-VERSION の詳細については、『*DB2 コンパイラ指令*』を参照してください。ただし、たとえばバージョン 7.1 のクライアントソフトウェアを使用していない場合や DB2 コネクトを通じて別の DB2 サーバに接続している場合などのように、他の環境でも新しい呼び出しセットを使用したい場合があります。このような場合は、DB2 コンパイラ指令 UDB-VERSION を使用してください。この UDB-VERSION 指令では、使用する DB2 UDB のバージョンを指定できます。指定できる値は、V2、V5、V6、V7、および V8 です。V7 または V8 を指定する場合には、ECM は新しい呼び出しを実行します。デフォルトは DB2 V6 です。次に使用例を示します。

```
DB2(UDB-VERSION=V7)
```

DB2 ECM が新しい API を検出できない場合は、次のメッセージが表示されます。

```
* クリティカルエラー - DB2Initialize API は検出されませんでした。
* 指令 DB2(UDB-VERSION=V6) を使用して
* プログラムをコンパイルしてください。
```

続いて次のメッセージが表示されます。

```
** DB00010 DB2 は未指定のオプションを拒否しました。このエラーは SQL
** 処理の続行を防止します。 - 以後の EXEC SQL 文は
** 無視されます。
```

9.7 コンパイル方法

COBOL コンパイラで SQL プログラムをコンパイルするのは次の 2 つの手順と論理的に同等です - プリコンパイルによって SQL 行をホスト言語文に変更し、次に結果ソースをコンパイルする。これらの

手順は実際には単一のプロセスで行われており、COBOL コンパイラが DB2 ECM とともに実行します。
SQL プログラムをコンパイルするためには、事前に認証が付与されていることが前提になります。通常、認証の付与は DB2 データベース管理者が行います。次の権限のどれか 1 つが必要です。

- sysadm または dbadm 権限
- パッケージが存在しない場合は BINDADD 権限と次のどれか 1 つ
  - パッケージのスキーマ名が存在しない場合は、データベースの IMPLICIT_SCHEMA 権限
  - パッケージのスキーマ名が存在する場合は、スキーマの CREATIN 権限
  - パッケージが存在する場合は、スキーマの ALTERIN 権限
  - パッケージが存在する場合は、そのパッケージの BIND 権限

ユーザには、アプリケーションで静的 SQL 文をコンパイルするために必要なすべてのテーブル権限も必要です。グループに付与された権限は、静的 SQL 文の権限検査には使用されません。SQL オブジェクトに対する権限がないためにプログラムがコンパイルに失敗した場合は、社内の DB2 データベース管理者に確認してください。
ユーザが SQL を使用しているということや、どのデータベースを使用しているか、という情報を DB2 ECM に与えるには、DB2 コンパイラ指令を使用します。『*DB2 コンパイラ指令*』を参照してください。
DB2 コンパイラ指令が必要となる場合を除いて、通常、埋め込み SQL を含むプログラムは SQL 以外のプログラムと同じ方法でコンパイルされます。追加モジュールをリンクする必要があるときに実行可能 (バイナリ) ファイルを作成する場合のみに、特別な動作が必要です。SQL コードを含むプログラムは、他のプログラムと同じようにアニメートできます。SQL 文内のホスト変数は、通常の COBOL データ項目と同じように調べることができます。

9.7.1 リモート DB2 サーバを使用するプログラムのコンパイル方法

リモート DB2 サーバを使用するプログラムをコンパイルするには、最初にリモートサーバに接続する必要があります。DB2 ECM は最初に、ログイン時に使用したクライアントワークステーションのデフォルト値を使用してデータベースへの接続を試行します。ログインに失敗した場合には、DB2 ECM は「Micro Focus SQL ログイン」ダイアログを呼び出します。このダイアログには、プログラムをコンパイルするときに使用するデータベースのログイン ID とパスワードを入力できます。次のようなダイアログボックスが表示されます。

図 9-1：「Micro Focus SQL ログイン」ダイアログボックス

ログオン ID とパスワードをメモリに保存して、次回同じデータベースを使用してプログラムをコンパイルするときにそれらを再入力する手間を省くこともできます。この情報は、クライアントマシンを次に再起動するか、または Net Express のコマンドプロンプトで次のコマンドを入力すると無効になります。
MFDAEMON CLOSE

9.7.1.1 自動コンパイル

コマンドファイルなどのバックグランドプロセスからコンパイルを自動化する場合には、グラフィカルなログオンダイアログを表示できないことがあります。環境変数を設定し、ログオン ID とパスワードを含むテキストファイルで変数を示して、ログオン情報を指定できます。これを行うには、環境変数 **SQLPASS.TXT** に、ログオン ID とパスワードを含むテキストファイルの名前を設定します。次に使用例を示します。
SET SQLPASS.TXT=D:¥BATCH.TXT
batch.txt ファイルで、ログオン ID とパスワードを **id.password** の形式で指定します。次に使用例を示します。
MyId.Mypassword
ログオン ID とパスワードの妥当性検査に使用されているセキュリティシステムが大文字と小文字を区別する場合は、テキストファイルで **id.password** を正しい大文字小文字で指定する必要があります。

注:テキストファイルでログオン ID とパスワードを指定することはセキュリティ上問題があるので、この機能を実装する場合は注意してください。

9.7.2 DB2 コンパイラ指令

DB2 コンパイラ指令のオプションを指定するには、プログラムで $SET 文を使用します。
たとえば、次のように指定します。
$SET DB2(INIT=PROT BIND COLLECTION=MYSCHEMA)
コンパイラ指令にはデフォルト値があり、他に値が指定されない場合に使用されます。これは既存の DB2 指令オプションのすべてについても同様です。これらのオプションの多くがコンパイル時に DB2 に直接渡され、値が指定されない場合には、コンパイラのデフォルトが使用されます。ただし、このような場合には、これらのオプションが適切かどうか、およびそのデフォルト値は DB2 の設定に左右されます。特に、DDCS を通じて DRDA サーバに接続されているかどうかに依存します。このため、これらのオプションのデフォルトのコンパイラ設定は「not set」です。この場合は、DB2 には値が渡されず、デフォルト値は (適用可能な場合) DB2 によって判断されます。これらの値については、IBM DB2 の解説書を参照してください。
次の表に、DB2 コンパイラ指令のオプションを示します。デフォルト値は、太字でハイライトされています。
この表は、DB2 UDB V7.x を使用していることを仮定しています。一部の指令は、DB2 コネクトと DB2 ユニバーサルデータベースを併用してる場合のみに有効です。別のバージョンの DB2 UDB を使用している場合は、IBM DB2 の解説書の PRECOMPILE または BIND の説明の箇所を参照して、該当の DB2 指令オプションがローカル (DB2) でサポートされるか、または、DB2 コネクト (DRDA) のみでサポートされるか確認してください。サポートされるかどうかわからない場合は、指令を指定して、プログラムのコンパイルを試行します。サポートされない場合は、エラーメッセージが表示されます。

オプション	説明
ACCESS=*package name*,**ACCESS**, NOACCESS	パッケージの名前を作成しデータベースに格納することを指定します。パラメータなしで ACCESS を指定する場合には、パッケージ名のデフォルトはプログラム名になります (.CBL 拡張子は付きません)。

	同義語は **PACKAGE** です。
ACTION={ADD | REPLACE }, **NOACTION**	ACTION は、パッケージが追加できる、または、置き換えられるかを示します。 **ADD** 名前付きパッケージが存在しないで、新規パッケージが作成されることを示します。パッケージがすでに存在する場合には、実行は停止し、診断エラーメッセージが返されます。 **REPLACE** 古いパッケージが、同じ位置、集まり、およびパッケージ名をもつ新規パッケージに置き換えられることを示します。REPLACE を指定する場合に、REPLVER や RETAIN またはその両方を指定する場合は、これらを ACTION=REPLACE の直後に指定する必要があります。REPLVER と RETAIN の順序はどちらが先でもかまいません。

BIND=*bindfile*, BIND, **NOBIND**	作成するバインドファイル名を指定します。パラメータなしで BIND を指定する場合には、バインドファイルのデフォルトは、ファイル名の拡張子が .BND に置き換えられたプログラム名になります。また、バインドファイルは、現在のソースディレクトリに作成されます。
	バインドファイルは、現在のソースディレクトリ以外の特定のディレクトリ、または BIND 指令で指定したディレクトリに書き込むことができます。これを行うには、HCOBND 環境変数を設定します。次に使用例を示します。
	`HCOBND=d:¥mydir¥binds`
	`export HCOBND`
	HCOBND 環境変数で指定したディレクトリは、環境設定の設定を解除したり、DB2 バインド指令を使用してプログラムに特定のビルド設定を指定したりするまで、Net Express で作成されたすべてのバインドファイルに適用されます。次に使用例を示します。
	`db2(bind=d:¥mybinds¥test.bnd)`
	HCOBND 環境変数の設定は無視されます。
	同義語は **BINDFILE** です。

BLOCK={**UNAMBIG** | ALL | NO}	パッケージ作成の際に、レコードブロック化モードを使用することを指定します。行ブロック化についての情報は、『*IBM DB2 管理の手引き*』、または『*アプリケーション・プログラミングの手引き*』を参照してください。 **ALL** 読み取り専用カーソル、または FOR UPDATE OF として指定されていない

	定カーソルは読み取り専用として扱います。 **NO** カーソルをブロック化しないことを指定します。不定カーソルは更新可能として扱います。 **UNAMBIG** 読み取り専用カーソル、または FOR UPDATE OF として指定されていないカーソルのブロック化を指定します。不定カーソルは更新可能として扱います。 同義語は **BLOCKING** です。
CALL_RESOLUTION={IMMEDIATE | DEFERRED}, **IMMEDIATE**	CALL 文の解決方法を指定します。 **IMMEDIATE** CALL 文を通常の SQL 文として実行することを指定します。呼び出された名前が解決できない場合には、プリコンパイラはエラー SQL0204 を報告します。 **DEFERRED** CALL 文を、非推奨の sqleproc() API の呼び出しとして実行することを指定します。 この指令を使用するには、DB2 UDB V8.1 以降を使用する必要があります。 ホスト変数にストアドプロシージャの名前を格納する場合や、プリコンパイラが CALL 文でのプロシージャの解決に失敗した場合 (この場合には、プリコンパイラはエラー SQL0204 を報告します) には、CALL_RESOLUTION=DEFERRED を設定する必要があります。
CCSIDG=double-ccsid , **NOCCSIDG**	符号化文字集合識別番号 (CCSID) を、CREATE 文 および ALTER TABLE SQL 文の (特定の CCSID 句なしの) 文字カラム定義の 2 バイト文字に使用できることを指定する整数。この DRDA プリコンパイル / バインドオプションは DB2 ではサポートされていません。このオプションが指定されない場合には、DRDA サーバはシステム定義のデフォルト値を使用します。
CCSIDM=mixed-ccsid , **NOCCSIDM**	符号化文字集合識別番号 (CCSID) を、CREATE 文 および ALTER TABLE SQL 文の (特定の CCSID 句なしの) 文字カラム定義の混合バイト文字に使用できることを指定する整数。この DRDA プリコンパイル / バインドオプションは DB2 ではサポートされていません。このオプションが指定されない場合には、DRDA サーバはシステム定義のデフォルト値を使用します。
CCSIDS=sbcs-ccsid, **NOCCSIDS**	符号化文字集合識別番号 (CCSID) を、CREATE 文および ALTER TABLE SQL 文の (特定の

	CCSID 句なしの) 文字カラム定義の 1 バイト文字に使用できることを指定する整数。この DRDA プリコンパイル / バインドオプションは DB2 ではサポートされていません。このオプションが指定されない場合には、DRDA サーバはシステム定義のデフォルト値を使用します。
CHARSUB={DEFAULT | BIT | SBCS | MIXED}, **NOCHARSUB**	CREATE 文と ALTER TABLE SQL 文のカラム定義に、デフォルトの文字サブタイプを使用することを指定します。この DRDA プリコンパイル / バインドオプションは DB2 ではサポートされていません。 **BIT** 明示的にサブタイプが指定されていない新規文字カラムのすべてに FOR BIT DATA SQL 文字サブタイプを使用します。 **DEFAULT** 明示的にサブタイプが指定されていない新規文字カラムのすべてに、対象システムが定義したデフォルトを使用します。 **MIXED** 明示的にサブタイプが指定されていない新規文字カラムのすべてに FOR MIXED DATA SQL 文字サブタイプを使用します。 **SBCS** 明示的にサブタイプが指定されていない新規文字カラムのすべてに FOR SBCS DATA SQL 文字サブタイプを使用します。
COLLECTION=schema name, **NOCOLLECTION**	パッケージに 30 文字の COLLECTION (集合) 識別子を指定します。指定しない場合は、パッケージを処理しているユーザの権限識別子 (許可 ID) が使用されます。
COMMIT={1 | **2** | 3 | 4}	暗黙的な COMMIT 文を生成するかどうかを指定します。 **1** COMMIT 文は暗黙に生成されません。 **2** STOP RUN 文とプログラムの末尾に COMMIT 文が暗黙に生成されます。 **3** STOP RUN 文、EXIT PROGRAM 文、およびプログラムの末尾に COMMIT 文が暗黙に生成されます。 **4** すべての SQL 文の後に COMMIT が暗黙に生成されます。
CONCAT=(ASCII 文字コード | **33** }	CONCAT 記号 (|) に使用する ASCII 文字コードを指定します。
CONNECT={1 | 2}, **NOCONNECT**	CONNECT 文をタイプ 1 CONNECT またはタイプ 2 CONNECT として処理することを指定します。
CTRACE, **NOCTRACE**	サポート窓口へトレースファイルの提出が必要な場

	合に、トレースファイルを作成します。作成されるファイルのファイル名は sqltrace.txt です。
DB=*database name*, **DB**	プログラムがアクセスするデータベース名を指定します。パラメータなしで DB を指定する場合は、環境変数 DB2DBDFT で指定されたデータベースが使用されます。
DEC={31 | 15} , **NODEC**	10 進数算術演算に使用する最大の精度を指定します。この DRDA プリコンパイル / バインドオプションは DB2 ではサポートされていません。このオプションが指定されない場合には、DRDA サーバはシステム定義のデフォルト値を使用します。 **15** を指定する場合は、10 進数算術演算に 15 桁の精度を使用します。 **31** を指定する場合は、10 進数算術演算に 31 桁の精度を使用します。
DECDEL={PERIOD | COMMA}, **NODECDEL**	小数点定数および浮動小数点定数の小数点インジケータとして、ピリオド (.) またはコンマ (,) のどちらを使用するかを指定します。この DRDA プリコンパイル / バインドオプションは DB2 ではサポートされていません。このオプションが指定されない場合には、DRDA サーバはシステム定義のデフォルト値を使用します。 **COMMA** コンマ (,) を小数点インジケータとして使用します。 **PERIOD** ピリオド (.) を小数点インジケータとして使用します。
DEFERRED_PREPARE={NO | YES | ALL}, **NODEFERRED_PREPARE**	DB2 共通サーバデータベースまたは DRDA データベースにアクセスするときのパフォーマンスを拡張します。このオプションは、SQL PREPARE 文のフローと、それに関連する OPEN 文、DESCRIBE 文、または EXECUTE 文のフローを結び付け、プロセス間またはネットワークのフローを最小化します。 **NO** PREPARE 文は、発行と同時に実行されます。 **YES** PREPARE 文の実行は、対応する OPEN 文、DESCRIBE 文、または EXECUTE 文が発行されるまで遅延されます。PREPARE 文に INTO 句を使用した場合には、遅延されません。INTO 句は SQLDA がすぐに返される必要があるからです。ただし、パラメータマーカーを使用しないカーソルに対して PREPARE INTO 句が発行される場合には、PREPARE の **ALL** パラメータマーカーを含む PREPARE INTO 句が遅延されることを除いて、YES と同様です。PREPARE INTO 文がパラメータマーカーを含まない場合でも、カーソ

	INTO 句が遅延されることを除いて、YES と同様です。PREPARE INTO 文がパラメータマーカーを含まない場合でも、カーソルの事前 OPEN 処理が行われます。PREPARE 文が SQLDA を返す INTO 句を使用している場合には、OPEN 文、DESCRIBE 文、または EXECUTE 文が発行されて戻されるまで、アプリケーションはこの SQLDA の内容を参照できません。
DEGREE={1 | degree-of-parallelism | ANY}, **NODEGREE**	入出力並行処理を使用してクエリーを実行するかどうかを指定します。 **1** 並列入出力操作を禁止します。 **degree-of-I/O-parallelism** 並行入出力操作の次数を指定します。値は 2 〜 32767 (整数) です。 **ANY** 並列入出力操作を許可します。
DISCONNECT={EXPLICIT | CONDITIONAL | AUTOMATIC}, **NODISCONNECT**	**AUTOMATIC** コミット時にすべてのデータベース接続を切断することを指定します。 **CONDITIONAL** データベース接続が RELEASE とマークされている場合、またはオープンしている WITH HOLD カーソルをもたない場合には、コミット時に切断することを指定します。 **EXPLICIT** RELEASE 文によって明示的にリリースとマークされているデータベース接続のみ、コミット時に切断することを指定します。
DYNAMICRULES={BIND | RUN | DEFINE | DEFINEBIND | DEFINERUN | INVOKE | INVOKEBIND | INVOKERUN}, **NODYNAMICRULES**	パッケージ名の動的 SQL が実行されたときに、どの権限識別子 (許可 ID) を使用するかを指定します。 **BIND** 動的 SQL の実行に使用される権限識別子 (許可 ID) は、パッケージのオーナーであることを示します。 **RUN** 動的 SQL の実行に使用される権限識別子 (許可 ID) は、パッケージ実行者の authid であることを示します。

		内で使用される場合には、ルーチン定義側の権限識別子は、権限検査と、ルーチン内の動的 SQL 文内にある修飾されていないオブジェクト参照の暗黙的な修飾に使用されます。 パッケージがスタンドアローンのアプリケーションとして使用されている場合には、動的 SQL 文はパッケージが DYNAMICRULES BIND でバインドされているときの動作で処理されます。
	DEFINERUN	パッケージがルーチンコンテキスト内で使用される場合には、ルーチン定義側の権限識別子は、権限検査と、ルーチン内の動的 SQL 文内にある修飾されていないオブジェクト参照の暗黙的な修飾に使用されます。 パッケージがスタンドアローンのアプリケーションとして使用されている場合には、動的 SQL 文はパッケージが DYNAMICRULES RUN にバインドされているときの動作で処理されます。
	INVOKE	動的 SQL の実行に使用される権限識別子が、UDF またはストアドプロシージャの起動側であることを指定します。このオプションは DB2 ではサポートされていません。
	INVOKEBIND	パッケージがルーチンコンテキスト内で使用される場合には、ルーチンの呼び出し時に有効な現在の文の権限識別子が、動的 SQL 文の権限検査と、ルーチン内の 動的 SQL 文内にある修飾されていないオブジェクト参照の暗黙の修飾のために使用されます。 パッケージがスタンドアローンのアプリケーションとして使用されている場合には、動的 SQL 文はパッケージが DYNAMICRULES BIND でバインドされているときの動作で処理されます。
	INVOKERUN	パッケージがルーチンコンテキスト内で使用される場合には、ルーチンの呼び出し時に有効な現在の文の権限識別子が、動的 SQL 文の

	権限検査と、ルーチン内の 動的 SQL 文にある修飾されていないオブジェクト参照の暗黙の修飾のために使用されます。 パッケージがスタンドアローンのアプリケーションとして使用されている場合には、動的 SQL 文はパッケージが DYNAMICRULES RUN でバインドされているときの動作で処理されます。
EXPLAIN={NO | YES | ALL}, **NOEXPLAIN**	パッケージの各 SQL 文に対して選択されたアクセスプランに関する情報を Explain テーブルに格納します。DRDA はこのオプションの ALL 値をサポートしていません。 **NO** Explain 情報は取り込まれません。 **YES** Explain テーブルを生成して、選択されたアクセスプランの情報を格納します。 **ALL** 適格な静的 SQL 文それぞれに対する Explain 情報が Explain テーブルに格納されます。さらに、CURRENT EXPLAIN SNAPSHOT レジスタが NO に設定されている場合でも、適格な動的 SQL 文の Explain 情報が実行時に収集されます。特殊レジスタの詳細については、『***IBM DB2 SQL 解説書***』を参照してください。
EXPLSNAP={NO | YES | ALL}, **NOEXPLSNAP**	Explain Snapshot 情報を Explain テーブルに格納します。この DB2 プリコンパイル / バインドオプションは DRDA ではサポートされていません。 **NO** Explain Snapshot は取り込まれません。 **YES** 適格な静的 SQL 文に対する Explain Snapshot が Explain テーブルに格納されます。 **ALL** 適格な静的 SQL 文に対する Explain Snapshot が Explain テーブルに格納されます。さらに、CURRENT EXPLAIN SNAPSHOT レジスタが NO に設定されている場合でも、適格な動的 SQL 文の Explain Snapshot 情報が実行時に収集されます。特殊レジスタの詳細については、『***IBM DB2 SQL 解説書***』を参照してください。
FEDERATED={**NO**|YES}	静的 SQL 文がニックネームと連合ビューのどちらを参照するかを指定します。パッケージが連合ビューまたはニックネームを参照しないときにこのオプションを指定した場合、またはパッケージが連合ビューやニックネームを参照しているときにこのオプション

	を指定しないと、SQL エラーが返されます。 **NO** プログラムは DB2 ユニバーサルデータベースに接続されます。これは、デフォルト値です。 **YES** プログラムは、DB2 連合システムにアクセスします。
FORMAT={DEF | USA | EUR | ISO | JIS | **LOC**}	日付 / 時刻フィールドがホスト変数の文字列表現に割り当てられたときの日付と時刻の形式を指定します。DEF はデータベースの国番号に関連付けられている日付時刻形式です。 **EUR** 欧州日付時刻形式の IBM 標準。 **ISO** 国際標準化機構の日付時刻形式。 **JIS** 日本工業規格の日付時刻形式。 **LOC** データベースの国番号に関連付けられているローカルの日付時刻形式。 **USA** 米国日付時刻形式の IBM 標準。 同義語は **DATETIME** です。
FUNCPATH=schema-name , **NOFUNCPATH**	静的 SQL 内でユーザ定義の互いに異なる型と関数を解決するために使用される関数パスを指定します。このオプションを指定しない場合には、デフォルトの関数パスは次のとおりです。 "SYSIBM","SYS FUN",USER USER は USER 特殊レジスタの値です。この DB2 プリコンパイル / バインドオプションは DRDA ではサポートされていません。 **schema-name** 短い SQL 識別子 (SQL 短識別子)。通常識別子または区切り識別子であり、アプリケーションサーバに存在するスキーマを識別します。プリコンパイル時やバインド時に、スキーマが存在するかどうかの妥当性検査は行いません。同じスキーマが関数パスに複数回現れることはできません。指定できるスキーマの数は結果関数パスの長さによって制限され、最大 254 バイトです。SYSIBM スキーマを明示的に指定する必要はありません。SYSIBM スキーマは、関数パスに含まれていなくても、最初のスキーマであることが暗黙の仮定とされます。詳細については、『*IBM DB2 SQL 解説書*』を参照してください。
GENERIC=**文字列**	新しいバインドオプションを、ターゲットの DRDA データベースに渡すことができます。各オプションは、1 つ以上の空白文字で区切り、二重引用符で囲む必要があります。例：DB2(GENERIC="keepdynamic yes")

GEN-INIT-FLAG, NOGEN-INIT-FLAG	DBS for OS/390 互換のために SQL-INIT-FLAG 変数を生成します。このフラグは、SQL 環境が初期化されると更新されます。この変数をアプリケーションで定義する場合は、この指令を NOGEN-INIT-FLAG に設定してアプリケーションをコンパイルします。変数はプリコンパイラでは生成されません。
IGNORE-NESTED=*program-id*, IGNORE-NESTED, **NOIGNORE-NESTED**	入れ子のプログラムで、DB2 インターフェイスコードの生成を開始する箇所の program-id を指定します。program-id よりも前に現れる入れ子のプログラムは無視され、DB2 インターフェイスコードは生成されません。COBOL ソースコード内の program-id を指定してください。それ以外の場合は、コンパイルエラーになります。パラメータなしで IGNORE-NESTED を指定する場合には、program-id のデフォルトは、ファイル名の拡張子が .CBL に置き換わったプログラム名になります。
INIT={PROT }, **NOINIT**	プログラムで SQL を初期化します。このオプションは、OO プログラム内で使用された場合には、無効になります。 **PROT** STOP RUN 時にデータベースを保護する必要があるが、初期化する必要がない場合に、SQL プログラムに使用します。 『*DB2 INIT 指令*』も参照してください。
INSERT={DEF | BUF}, **NOINSERT**	プログラムをプリコンパイルまたはバインドして、DB2 V2.1 クライアントから DATASBASE 2 Parallel Edition サーバへ、パフォーマンス向上のためにデータ挿入のバッファを要求することを許可します。 **BUF** アプリケーションからの挿入をバッファします。 **DEF** アプリケーションからの挿入をバッファしません。
ISOLATION={**CS** | RR | UR | RS | NC}	このパッケージにバインドされたプログラムが、実行中の他のプログラムの影響から分離される程度を判断します。分離レベルの詳細については、『*IBM DB2 SQL 解説書*』を参照してください。 **CS** カーソル固定を分離レベルとして指定します。 **NC** (No Commit) コミットメントコントロールを使用しないことを指定します。この分離レベルは DB2 ではサポートされていません。 **RR** 繰り返し可能読み取り (反復可能読み取り) を分離レベルとして指定します。 **RS** 読み取り固定を分離レベルとして指定します。読み取り固定を使用する場合には、パッケージ内の SQL 文は、他のアプリケーションが読み込んで変更した行に対するプロセスか

	ら分離されて実行されます。 **UR** 非コミット読み取りを分離レベルとして指定します。
LANGLEVEL={**SAA1** | NONE | MIA|SQL92E}	このオプションの詳細については、『*IBM DB2 アプリケーション・プログラミングの手引き*』を参照してください。 **MIA** FOR UPDATE 句は位置指定更新のオプションです。C の NULL 終了文字列は、空白文字で埋められ、常に NULL 終了文字を含みます。このオプションは DB2 コネクトではサポートされていません。 **SAA1** 位置指定更新で更新されたすべてのカラムに FOR UPDATE 句が必要です。C の NULL 終了文字列は空白文字で埋められ、切り捨てが行われた場合には NULL 終了文字を含みません。 **NONE** SAA1 と同義語です。 SQL92E MIA と似ています。MIA との相違についてはマニュアルを参照してください。 同義語は **STDLVL** です。
LEVEL=consistency-token, **NOLEVEL**	一貫性トークンを使用してモジュールのレベルを定義します。一貫性トークンは、8 文字以下の英数字の値です。RDB パッケージの一貫性トークンは、リクエスタのアプリケーションとリレーショナルデータベースパッケージが同期していることを妥当性検査します。 注：通常の場合は、このオプションを使用しないことをお奨めします。
MSGAREA={data-item-name | **MFSQLMESSAGETEXT** },NOMSGAREA)	英数字データ項目の名前を指定します。この項目がプログラムソースに存在する場合は、自動的に DB2 エラー条件がこの項目に記述されます (SQLCODE がゼロではないとき)。
NOT={ASCII 文字コード | **170** }	NOT 文字 (¬) に使用する ASCII 文字コードを指定します。
OWNER=authorization-id, **NOOWNER**	パッケージのオーナーの権限識別子を 30 文字で指定します。オーナーは、パッケージで SQL 文を実行するために必要な権限をもつ必要があります。このオプションが明示的に指定されていない場合には、デフォルトはプリコンパイル / バインドプロセスの一次権限識別子 (1 次許可 ID) です。 同義語は **SCHEMA** です。
PRE, NOPRE	DB2 API ルーチンのエントリポイントを事前にロードすることによって、DB2 プログラムをアニメートしたり実行したりできます。実行可能ファイルまたは共有オ

	ブジェクトファイルを作成する場合は NOPRE を指定します。それ以外の場合は、リンクの問題が発生する場合があります。
QUALFIX, NOQUALFIX	ホスト変数を DB2 に宣言するときに、DB2 ECM がこれらのホスト変数の名前に 3 文字を追加します。これにより、修飾によって起こる問題 (修飾されないと 2 つ以上のホスト変数が同じ名前になる場合) を避けることができますが、次の副作用があります。 (1) DB2 ユニバーサルデータベース V5.0 以降を使用していない場合は、ホスト変数名の最大長は 27 文字になります。 (2) DB2 エラーメッセージで、ホスト変数の名前に 3 つの追加文字が付加されて表示されることがあります。 DB2 ユニバーサルデータベース V7.1 以降を使用している場合には、デフォルトは NOQUALFIX です。それ以外の場合は、QUALFIX です。
QUALIFIER=qualifier-name, **NOQUALIFIER**	パッケージに含まれる修飾なしのテーブル名、ビュー、指標、および別名に 30 文字の暗黙の修飾子を提供します。デフォルトは、オーナーの権限 ID です。DB2 ユニバーサルデータベース 5.2 以降を使用する場合には、QUALIFIER 指令はローカルの DB2 データベースで有効です。
QUERYOPT= optimization-level, **NOQUERYOPT**	パッケージに含まれるすべての静的 SQL 文に対して望ましい最適化レベルを示します。デフォルト値は 5 です。使用可能な最適化レベルの詳細は、『***SQL 解説書***』の SET CURRENT QUERY OPTIMIZATION 文の説明を参照してください。この DB2 プリコンパイル / バインドオプションは DRDA ではサポートされていません。
RELEASE={COMMIT | DEALLOCATE}, **NORELEASE**	各 COMMIT 箇所またはアプリケーションの終了時にリソースが解放されるかどうかを指定します。この DRDA プリコンパイル / バインドオプションは DB2 ではサポートされていません。 **COMMIT** 各 COMMIT 箇所でリソースを解放します。動的 SQL 文に使用されます。 **DEALLOCATE** アプリケーションの終了時のみにリソースを解放します。
REPLVER=version-id, **NOREPLVER**	パッケージの特定のバージョンを置き換えます。バージョン識別子は、どのバージョンのパッケージを置き換えるかを指定します。最大長は 254 文字です。REPLVER は、ACTION=REPLACE または RETAIN (指定している場合) の直後で指定する必要があります。

RETAIN={YES | NO} , **NORETAIN**	RETAIN はパッケージが置き換えられるときに EXECUTE 権限が保持されるかどうかを指定します。パッケージのオーナーが変更した場合は、新しいオーナーが以前のオーナーに BIND 権限と EXECUTE 権限を付与します。 **NO** パッケージを置き換えるときに EXECUTE 権限を保持しません。 **YES** パッケージを置き換えるときに EXECUTE 権限を保持します。
SQLERROR={NOPACKAGE | CHECK | CONTINUE}, **NOSQLERROR**	エラーが発生した場合に、パッケージを作成するか、または、ファイルをバインドするかを指定します。 **CHECK** 対象システムが、バインドされている SQL 文の構文および意味検査をすべて行うことを指定します。パッケージはこのプロセスの一部としては作成されません。パッケージの作成中に、同じ名前とバージョンをもつ既存のパッケージが検出された場合でも、既存のパッケージは削除されません。また、ACTION REPLACE が指定されていても置き換えられません。 **CONTINUE** SQL エラーが発生した場合でも、パッケージやバインドファイルは作成されます。このオプションは DB2 ではサポートされていません。 **NOPACKAGE** エラーが発生した場合には、パッケージやバインドファイルは作成されません。 構文が package オプションとともに使用されると、package は無視されます。 同義語は **ERROR** です。
SQLFLAG={MVSDB2V23 | MVSDB2V31 | MVSDB2V41|SQL92E}-SYNTAX, **NOSQLFLAG**	指定した SQL 言語構文から偏差を識別し報告します。 SQLFLAG オプションに加えて、BINDFILE または PACKAGE オプションが指定された場合のみに、バインドファイルまたはパッケージが作成されます。 BINDFILE、PACKAGE、SQLERROR CHECK、および SYNTAX オプションのどれか 1 つが指定された場合のみにローカル構文検査が行われます。 SQLFLAG を指定しない場合には、フラッガ機能は呼び出されず、バインドファイルまたはパッケージは影響されません。 **MVSDB2V23-SYNTAX** MVS DB2 バージョン 2.3 SQL 言語構文に対して SQL 文が検査されます。構文からの偏差は

	すべてプリコンパイラリストに報告されます。 **MVSDB2V31-SYNTAX** MVS DB2 バージョン 3.1 SQL 言語構文に対して SQL 文が検査されます。構文からの偏差はすべてプリコンパイラリストに報告されます。 **MVSDB2V41-SYNTAX** MVS DB2 バージョン 4.1 SQL 言語構文に対して SQL 文が検査されます。構文からの偏差はすべてプリコンパイラリストに報告されます。 **SQL92E-SYNTAX** ANSI または ISO SQL92 SQL 言語構文に対して SQL 文が検査されます。構文からの偏差はすべてプリコンパイラリストに報告されます。 同義語は **FLAG** です。
SQLRULES={DB2 | STD}, **NOSQLRULES**	タイプ 2 CONNECT を DB2 の規則に準じて処理するか、または、ISO/ANS SQL92 に基づく標準 (STD) 規則に準じて処理するかを指定します。 **DB2** 現在の接続から他の確立済み (休止状態) 接続へ切り替える SQL CONNECT 文の使用を許可します。 **STD** 新規接続を確立するのみの SQL CONNECT 文の使用を許可します。休止状態の接続へ切り替えるには、SQL SET CONNECTION 文を使用してください。 同義語は **RULES** です。
SQLWARN={YES | NO}, **NOSQLWARN**	動的 SQL 文のコンパイルから (PREPARE または EXECUTABLE IMMEDIATE 経由) 警告が返されるか、または、記述処理から (PREPARE...INTO または DESCRIBE 経由) 警告が返されるかを示します。この DB2 プリコンパイル / バインドオプションは DRDA ではサポートされていません。 **NO** SQL コンパイラから警告は返されません。 **YES** SQL コンパイラから警告が返されます。 注：SQLCODE +238 は例外です。この警告は、SQLWARN オプションの値に関わらず返されます。 同義語は **WARN** です。
STRDEL={APOSTROPHE | QUOTE}, **NOSTRDEL**	SQL 文内の文字区切りとして単一引用符 (') または二重引用符 (") のどちらを使用するか指定します。この DRDA プリコンパイル / バインドオプションは DB2 ではサポートされていません。このオプションが指定されない場合には、DRDA サーバはシステム定義のデフォルト値を使用します。

	単一引用符 (') を文字区切りとして使用する場合は、APOSTROPHE を指定します。 QUOTE を指定する場合は、二重引用符 (") を文字区切りとして使用します。
SYNCPOINT={ONEPHASE | TWOPHASE | NONE}, **NOSYNCPOINT**	複数のデータベース接続間でどのようにコミットまたはロールバックを調整するかを指定します。 **NONE** トランザクションマネージャ (TM) を使用して 2 フェーズコミットを行わないことを指定します。単一の更新操作、複数の読み取り操作を強制しません。COMMIT は各関係データベースに送られます。コミットがどれか失敗した場合には、アプリケーションに回復責任があります。 **ONEPHASE** TM を使用して 2 フェーズコミットを行わないことを指定します。複数データベーストランザクション内の各データベースによって行われた作業をコミットするには、1 フェーズコミットが使用されます。 **TWOPHASE** TM を使用して、このプロトコルをサポートするデータベース間の 2 フェーズコミットを調節することを指定します。
SYNTAX	SQLERROR=CHECK 指令の同義語です。
TEXT=label, **NOTEXT**	パッケージの説明。最大長は 255 文字です。デフォルト値は空白です。この DRDA プリコンパイル / バインドオプションは DB2 ではサポートされていません。
TRANSFORM-GROUP=identifier. **NOTRANSFORM-GROUP**	静的 SQL に使用する変換グループ名を指定します。これは、18 文以内の SQL 識別子です。
UDB-VERSION = {V2 | V5 | **V6** | V7 | V8 }	DB2 ECM でコードを生成する DB2 UDB のバージョンを指定します。詳細については、『*UDB-VERSION DB2 指令オプション*』を参照してください。
VALIDATE={RUN | BIND}, **NOVALIDATE**	データベース管理者が権限エラーとオブジェクトが存在しないエラーを検査するタイミングを決定します。妥当性検査にはパッケージオーナーの権限 ID が使用されます。 **BIND** 妥当性検査はプリコンパイル / バインド時に行われます。オブジェクトがすべて存在しない場合や、権限がすべて保持

	RUN またはバインドファイルが生成されます。ただし、エラー中の文は実行可能ではありません。 妥当性検査はバインド時に行われます。オブジェクトがすべて存在し、権限がすべて保持されている場合には、実行時に再検査は行われません。 プリコンパイル / バインド時にオブジェクトがすべて存在しない場合、または権限がすべて保持されていない場合には、警告メッセージが生成され、SQLERROR CONTINUE オプションの設定に関わらず、パッケージは問題なくバインドされます。ただし、プリコンパイル / バインド時に失敗した SQL 文の権限検査と存在検査は、実行時に再度行われる可能性があります。
VERSION=version-id, **NOVERSION**	パッケージのバージョン識別子を定義します。バージョン識別子は、英数字、$、#、@、_、-、または . で、長さは 254 文字以内です。

9.8 エラーコード

コンパイル時に、数字と説明でエラー条件が返されます。これらのメッセージの詳細については、使用しているデータベースシステムのマニュアルで説明されています。UB2 UDB の古いバージョンを使用している場合は、ホスト変数を参照してるメッセージがわずかに異なります。ハイフンがアンダスコア (_) になり、3 文字以下の文字が名前の末尾に追加されますが、これは無視してください。DB2 ECM から SQL コードに変更するときの副作用として、これらの変更が起こります。 詳細については、*QUALFIX* 指令を参照してください。
実行時のエラー条件は、SQLCODE の非ゼロ値によって示されます。MFSQLMESSAGETEXT の定義を行うと、MFSQLMESSAGETEXT データ項目に説明テキストが保存されます。このデータ項目の詳細については、『*SQL 通信領域*』を参照してください。
例

```
801-S
**      外部コンパイラモジュールメッセージ
**      SQ0100 SQL1032N   データベース管理者コマンドが起動できません。
**      SQLSTATE=57019
```

9.9 デバッグファイルの作成方法

プログラムのコンパイル時にサポート窓口に問い合わせする必要のあるエラーが発生した場合には、サポート窓口の担当者は、問題の原因を特定するために追加のデバッグファイルの提供を求める場合があります。これらのデバッグファイルは、追加の DB2 コンパイラ指令を指定して作成します。これらの指令のいくつかを指定すると、独自でデバッグする場合に役に立ちます。 次の指定があります。

指令	作成ファイル	ファイル内の情報

CHKECM(CTRACE)	ecmtrace.txt	このファイルには、EXEC SQL 文を置き換えるために生成されるコードを示す擬似 COBOL コードが含まれます。このコードは、IBM DB2 COBOL プリコンパイラから出力されるものと同じです。
CHKECM(TRACE)	ecmtrace.txt	このファイルには、DB2 ECM とコンパイラ間で受け渡しされる情報に関する詳細情報が含まれます。無効な構文を生成するエラーが発生した場合に、このファイルを使用して、問題が発生した箇所を分離できます。
DB2(CTRACE)	sqltrace.txt	このファイルには、IBM プリコンパイラサービスに渡される情報の詳細なリストと、その結果が含まれます。このファイルは、エラーが DB2 ECM のみでなく DB2 システムソフトウェアのバグに関連する場合にも役に立ちます。
ECMLIST	*program-name*.lst	このファイルは、EXEC SQL 文を置き換えるために生成されるコードを示す擬似 COBOL コードを含む、標準の COBOL リストファイルです。CHKECM(CTRACE) 指令と LIST 指令を使用してプログラムをコンパイルする必要があります。

9.10 リンク方法

アプリケーションにリンクするには、次の手順を実行します。

1. Net Express プロジェクトを開き、「**ビルドタイプ**」を「**一般リリースビルド**」に設定します。
2. **.exe** または **.dll** ファイルを右クリックします。
3. [**ビルド設定 ...**] を選択して、[**リンク**] タブをクリックします。
4. 「**カテゴリ**」を「**高度な設定**」に設定します。
5. 「**これらの LIB とリンクする**」編集ボックスで、次のように入力します。

   ```
   db2api.lib
   ```

9.11 バインディング

DB2 コンパイラ指令の NOACCESS オプションを使用する場合や、コンパイルしたマシン以外のマシンでアプリケーションを実行しようとする場合は、実行する前にアプリケーションを特定のデータベースにバインドしてください。この場合には、BIND オプションを使用してバインドファイル作成し、DB2 BIND コマンドを使用してプログラムをデータベースにバインドします。詳細については、使用している SQL システムのマニュアルを参照してください。
環境変数 HCOBND を指定することにより、現在のソースディレクトリ以外のディレクトリにバインドファイルを格納するよう DB2 ECM に指示できます。次に例を示します。

```
HCOBND=d:¥production¥binds
export HCOBND
```

HCOBND 環境変数で指定したディレクトリは、この環境変数の設定が解除またはリセットされるか、または、次の例のように DB2 BIND 指令オプションを使用して特定のバインドファイル名を指定するまではすべてのバインドファイルに使用されます。

```
DB2(bind=d:¥test¥test1.bnd)
```

DB2 BIND **指令オプションは、**HCOBND **環境変数を上書きします。**

第 10 章：DB2 用の SQL オプション

ここでは、Net Express を使用して PC で DB2 アプリケーションを作成および維持する方法を説明します。

10.1 概要

SQL オプションを使用する場合は、メインフレームや LAN ベースのデータベースシステムにアクセス要求しなくても、Net Express IDE で、埋め込み SQL 文を含むプログラムをコンパイル、デバッグ、および実行できます。
SQL オプションを使用する場合は、PC の EBCDIC 環境をメインフレームの EBCDIC 環境と完全に互換させることができます。詳細は、『*SQL オプション NLS 環境*』を参照してください。
SQL オプションは、一般に普及してる XDB データベースシステムと同じ技術を使用しています。XDB データベースはメインフレーム DB2 データベースとまったく同じように動作しますが、SQL オプションは PC で実行されます。パーソナル XDB サーバは PC にインストールされ、DB2 アプリケーション用の完全な自己格納式の開発およびテスト環境を提供します。
ワークグループ環境で開発している場合は、システム管理者は SQL オプションを構成して、パーソナル XDB サーバに加え、複数の LAN ベースの共有 XDB サーバにアクセスできるようにします。この 2 つの階層構成により、アプリケーションのテストに関するいくつかの追加のオプションを利用できます。たとえば、次のような処理ができます。

- メインフレームテストデータの完全なコピーまたはサブセットを共有の XDB サーバで維持できます。ワークグループのメンバはこのデータに対して直接テストする。または、データを他のユーザから分離する必要がある場合はパーソナル XDB サーバに必要なテーブルをインポートできます。
- テストデータを、共有の XDB サーバとワークグループメンバのパーソナル XDB サーバに複製できます。この場合は、共有 XDB サーバは統合テストに使用し、ワークグループメンバはパーソナル XDB サーバに対して開発テストやユニットテストを実行できます。
- 共有 XDB サーバとパーソナル XDB サーバにデータを分散させることができます。たとえば、容量の大きい、より静的なテーブルは共有 XDB サーバに残し、揮発性の高いテーブルは PC にエクスポートできます。

また、適切な構成では、SQL オプションの接続機能を使用してメインフレームの DB2 データベースにシームレスにアクセスできます。この 3 階層構成では、すべての開発およびテストオプションにアクセスできるので、使用環境に合わせて効率的なワークフローを自由に作成できます。たとえば、次のことを実行できます。

- すべてのテストにメインフレームの DB2 テスト環境を使用する。
- メインフレームを統合テストに使用し、ワークグループメンバはパーソナル XDB サーバまたは共有 XDB サーバで開発およびユニットテストを実行する。
- テストデータを 3 つのすべての階層に分散し、大容量テーブルはメインフレームに残し、他のデータは共有 XDB サーバで保持する。この場合には、ワークグループメンバは、分離が必要な場合にパーソナル XDB サーバに特定のデータをダウンロードできます。

10.2 SQL オプションの構成要素

DB2 用の SQL オプションは、次のような埋め込み SQL プリコンパイラと多数の構成ツールおよびグラフィックデータユーティリティで構成されます。

- *XDB サーバ*
- *サーバ構成ユーティリティ*
- *サーバ管理オプション*
- *SQL ウィザード*
- *移行ユーティリティ*
- *宣言ジェネレータユーティリティ*
- *オプションユーティリティ*
- *バインドユーティリティ*
- *ゲートウェイプロファイルユーティリティ*
- *Password Expiration Manager ユーティリティ*
- *SQL オプションプリプロセッサ*

これらすべての構成要素には、Net Express IDE の [**ツール**] メニューと [**オプション**] メニューの [**SQL For DB2**] をクリックしてアクセスできます。SQL オプションツールのヘルプは、構成要素または [**ヘルプ**] メニューから表示できます。このメニューから、エラーメッセージヘルプと詳細な SQL リファレンスにもアクセスできます。
オプションユーティリティは、[**オプション**] メニューで [**SQL For DB2**]、[**クライアント**] の順にクリックして検索できます。

10.3 XDB サーバ

XDB サーバは、すべてのデータベース操作を実行し、DB2 システムをエミュレートするサーバ構成要素です。パーソナル XDB サーバは、DB2 用の SQL オプションをインストールするときにユーザのマシンにインストールされます。このローカルコピーの XDB サーバがデフォルトサーバになります。ローカルの XDB データベースに対してクライアントプリケーションをデバッグまたは実行する前に、XDB サーバが実行されている必要があります。パーソナル XDB サーバを起動するには、次の 3 つの方法のうちのどれかを使用します。

- [**ツール**] メニューで [**SQL For DB2**]、[**サーバの起動**] の順にクリックする。
- プロジェクトを開くと自動的に XDB サーバが起動するようにプロジェクトを構成する。
- Windows NT を使用している場合は、パーソナル XDB サーバを NT サービスとしてインストールし、自動的に起動する。

メインフレームの DB2 データに直接アクセスする場合は、最初にシステム管理者が多くの SQL 接続構成要素をインストールおよび構成する必要があります。構成が完了した後に、システム管理者はメインフレームデータへの接続方法をユーザに通知できます。DB2 のリンク機能の詳細については、[**ヘルプ**] メニューの[**SQL For DB2**]、[**DB2 にリンク**] の順にクリックして参照します。

10.3.1 サーバ構成ユーティリティ

サーバ構成ユーティリティを使用して、パーソナル XDB サーバの構成を変更できます。たとえば、次の構成を変更できます。

- XDB サーバ名
- XDB サーバセキュリティを使用可能にするかどうか
- XDB サーバにアクセスするときに使用するプロトコル

- 他のユーザが XDB サーバを使用できるようにするかどうか
- XDB サーバで使用可能なシステムリソースの量

10.3.2 サーバ管理オプション

[**ツール**] メニューの [**SQL For DB2**] オプションを使用して、XDB サーバへの接続を制御できます。
[**サーバの起動**] を使用して、パーソナル XDB サーバを実行できます。
[**ログオン**] では、クライアントのセキュリティがオンの場合に、XDB サーバに対話形式でログオンできます。クライアントのセキュリティは、オプションユーティリティの [**セキュリティ**] タブで制御します。セキュリティがオンになっている XDB サーバにアクセスするときには、クライアントのセキュリティをオンにする必要があります。
XDB サーバのセキュリティがオンになっているときに、多数の SQL オプションユーティリティを個別にログオンしないで使用する場合は、通常 [**ログオン**] を使用します。XDB サーバのセキュリティがオフの場合は、明示的にログオンする必要はありません。
[**ログオフ**] を使用する場合は、一度実行したログオンを取り消すことができます。たとえば、セキュリティがオンになっている XDB サーバにログオンしてからログオフした場合は、次に XDB サーバにアクセスするときには、AuthID とパスワードを入力しないとアクセスできません。XDB サーバのセキュリティがオフの場合は、明示的にログオフする必要はありません。
パーソナル XDB サーバを停止するには次の 2 つの方法があります。

- [**ツール**] メニューで [**SQL For DB2**]、[**サーバの停止**] の順にクリックする。
- Windows NT を使用している場合は、[コントロール パネル] で [**Micro Focus XDB Server for NX**] をダブルクリックし、[**停止**] をクリックする。

10.4 SQL ウィザード

SQL ウィザードは、XDB および DB2 データベースを簡単に作成、維持およびクエリーできるようにするグラフィックユーティリティです。SQL ウィザードでは、次の処理が実行できます。

- *システムのセキュリティと優先順位の管理*
- *位置、テーブル、およびクエリーの管理*
- *クエリーの作成と実行*
- *テーブルへのデータの直接入力*
- *データのインポートとエクスポート*
- *バッチスクリプトの実行*

10.4.1 システムのセキュリティと優先順位の管理

SQL オプションには、データベースシステムに必要なセキュリティと管理の機能が用意されています。ただし、使用する目的によっては、セキュリティ機能を使用可能にする必要がない場合もあります。このため、パーソナル XDB サーバとそのクライアントツールおよびユーティリティは、セキュリティ機能がオフの状態でインストールされます。
セキュリティは、サーバとクライアントでそれぞれ個別に制御されます。XDB サーバのセキュリティをオンにする場合は、セキュリティがオンになっているクライアントからのみアクセスできます (ここでいうクライアントとは、XDB 構成ツールまたはグラフィカルユーザユーティリティのどれかを指します。ユーザが開発するクライアントアプリケーションでは、CONNECT 文による通常の方法で権限を処理する必要があります)。パスワードを処理する DB2 アプリケーションを開発している場合は、セキュリティをオンにしてユーザ認証が有効化されているテスト環境で作業する場合もあります。

SQL ウィザードの [**管理**] メニューで、システムセキュリティを管理します。セキュリティは、次の 3 つのレベルで制御できます。

- すべてのユーザ
- 名前付きのユーザグループ
- 個々のユーザ

XDB サーバに対してセキュリティがオフになっている場合は、次の理由からすべてのユーザが事実上のスーパーユーザとなります。

- パスワードが不要である。
- すべてのユーザが、オプションユーティリティの [**接続**] タブで指定した共通の AuthID を共有する。
- すべてのユーザが、クライアントユーティリティとサーバユーティリティの両方に対するアクセス権をもつ。

サーバセキュリティのステータスは、サーバ構成ユーティリティ ([**オプション**] メニューで [**SQL For DB2**]、[**XDB サーバ**] の順にクリックしてアクセスします) を使用して設定します。セキュリティが使用可能になると、ユーザは有効な AuthID と必要に応じてパスワードを使用してログオンする必要があります。ユーザの実際の AuthID は、共有の AuthID を置き換えます。つまり、ユーザは、GRANT 文と REVOKE 文を使用してデータベースオーナーから適切なアクセス特権が付与されていない場合は、データベースにはアクセスできません。
スーパーユーザのステータスが割り当てられた AuthID は、ユーザ、グループ、および優先順位の設定を変更できます。この変更は、XDB サーバ上のすべてのデータベースに影響します。最初は、INSTALL という名前の AuthID がスーパーユーザとして設定されています。この AuthID でログオンした場合は、必要に応じて [**管理**] メニューの [**ユーザ**] をクリックして、各ユーザに適切なステータスを割り当てて、追加のスーパーユーザを作成できます。
標準ユーザは、[**管理**] メニューで、自分のパスワードのみ変更できます。標準ユーザの所有する特定のデータベースやテーブルのアクセス特権を変更する場合は、[**ファイル**] メニューで [**新規作成**]、[**SQL**] の順にクリックして、SQL 文の GRANT と REVOLE を使用してアクセス特権を変更します。テーブル作成者の AuthID はそのテーブルのオーナーと見なされます。
システムセキュリティの管理に加え、スーパーユーザは個々のユーザまたはユーザグループに優先順位を割り当てることができます。優先順位では、スーパーユーザが設定した基準に従ってユーザまたはグループで使用できる処理リソースの量を制御します。

注：SQL オプションをインストールした時点では、デフォルトの AuthID は TUTORIAL に設定されています。この AuthID にはユーザ特権が割り当てられていますが、システムテーブルへのアクセスは許可されません。つまり、パーソナル XDB サーバでセキュリティをオンにした場合は、この AuthID でクライアントユーティリティにログオンできません。
この場合は、デフォルトのスーパーユーザ AuthID の INSTALL を使用してログオンする必要があります。TUTORIAL の AuthID に、スーパーユーザのステータスを設定する。または、システムテーブルに対する適切な権限を TUTORIAL に付与できます。INSTALL スーパーユーザ AuthID にはパスワードが割り当てられていません。XDB サーバのセキュリティをオンにした直後にパスワードを割り当てる必要があります。

10.4.2 位置、テーブル、およびクエリーの管理

SQL ウィザードの「カタログ参照」ウィンドウを使用する場合は、XDB サーバのシステムテーブルを含む (この場合は AuthID に適切な権限が必要です)、位置、テーブル、およびクエリーを管理できます。

「カタログ参照」ウィンドウを開くには、SQL ウィザードの [**表示**] メニューのエントリをクリックする。または、ツールバーの をクリックします。「カタログ参照」には、[**位置**]、[**テーブル**]、および [**クエリー**] の 3 つのタブがあります。
スーパーユーザは、[**位置**] タブで位置を管理できます。スーパーユーザは、位置の作成、変更、および削除の他に、現在アクティブな位置の設定を実行できます。他のユーザは、使用可能な位置のリスト表示のみできます。
[**テーブル**] タブには、位置、AuthID、テーブル、およびカラムが階層形式で表示されます。このページでは、テーブルの作成、オープン、変更、および削除の他に、テーブルの定義や指標、主キー、外部キー、ビュー、別名、および同義語を表示できます。システムテーブルを編集できるのは、スーパーユーザのみです。
[**クエリー**] タブには、位置、AuthID、およびクエリー階層下にある現在のサーバにアクセス可能な保存されているすべてのクエリーが表示されます。このページでは、クエリーの作成、オープン、実行、および削除ができます。

10.4.3 クエリーの作成と実行

SQL クエリーを作成するには、SQL 文を SQL ウィンドウ (SQL ウィザードツールバーで をクリック) に直接入力する。または、「クエリーデザイン」ウィンドウ (ツールバーの をクリック) で指示クエリーを使用します。指示クエリーを作成する場合は、SQL を理解する必要はありません。「クエリーデザイン」ウィンドウは、次の 2 つの領域に分割されます。

- スキーマビュー。このビューには、クエリーに関連するテーブルと、それらのテーブル間の関係が表示されます。ウィンドウのこの領域は、テーブル表示領域と呼ばれます。
- QBE (query-by-example) ビュー。このビューには、フィルタを適用してクエリーを実行した結果を示すカラムが表示されます。ウィンドウのこの領域は、クエリー条件グリッドと呼ばれます。

指示クエリーを作成する場合は、最初にテーブルとカラムを選択します。そして、選択したカラムに条件とソート基準を適用して、最終的な結果集合を取得します。このメニューでは、結合、計算されたカラムの作成、minimum、maximum、average などの組込み関数の使用などを行うこともできます。
指示クエリーを作成するときには、等価の SQL 文が SQL ウィザードで作成されます。これらの SQL 文を参照するには、 をクリックします。SQL 文は直接編集できますが、変更内容は「クエリーデザイン」ウィンドウに表示されません。「クエリーデザイン」ウィンドウに戻るには、 をクリックします。指示クエリーを保存してから SQL ウィンドウでクエリーを編集することをお奨めします。 をクリックする場合は、いつでもクエリーを実行し、結果を参照できます。

10.4.4 テーブルへのデータの直接入力

SQL ウィザードには、スプレッドシートのようなテーブルビューと、データを直接入力および編集できるフォームビューがあります。一度に多くのレコードを参照する場合はテーブルビューを使用し、一度に 1 つずつレコードを参照する場合はフォームビューを使用します。
テーブルを編集するには適切なアクセス権限が必要です。システムテーブルを編集できるのはスーパーユーザのみです。標準ユーザは、自分の AuthID または従属するグループの GroupID を使用して作成したテーブルを編集できます。また、標準ユーザは、GRANT 文を使用してアクセス権を特別に付与されたテーブルも編集することができます。どの場合でも、[**レコード**] メニューの [**編集を許可**] を選択して編集を使用可能にする必要があります。

「カタログ参照」では、いつでもテーブルを開いたり作成したりできます。詳細については、『*位置、テーブル、およびクエリーの管理*』を参照してください。テーブルビューは自動的に開いて、クエリーの結果を表示します。

10.4.5 データのインポートとエクスポート

SQL オプションでは、さまざまな形式のデータをインポートおよびエクスポートできます。インポートとエクスポートの設定を指定するには、SQL ウィザードの [**ファイル**] メニューで [**新規作成**] の [**インポート**]。または、[**新規作成**] の [**エクスポート**] をクリックします。これらの設定を .imp または .exp ファイルに保存し、後からルーチンデータ変換のためのバッチファイルとして実行できます。
SQL オプションのインポート機能を使用して、別のファイル形式で準備されたテストデータに対してプログラムを実行します。たとえば、Microsoft Access データベース、Microsoft Excel または Lotus 1-2-3 のスプレッドシートに保存されている適切なテストデータを、区切りの ASCII ファイルまたは固定フィールドの ASCII ファイルにエクスポートできます。エクスポートしたこれらのファイルは、XDB サーバにインポートできます。
XDB データベースには、次のどのファイルのデータでもインポートできます。

- 自由形式の ASCII ファイル (区切り)

  区切り (自由形式) ASCII ファイルからデータをインポートします。

- 固定形式の ASCII ファイル (カラム状)

  カラム状 (固定形式) の ASCII ファイルからデータをインポートします。

- dBASE ファイル

  dBASE II、III、および III+ ファイルからデータをインポートします。dBASE III ファイルと同じ形式の dBASE IV ファイルは、dBASE III オプションを使用してインポートできます。

- DBMAUI

  IBM の DB2 から、IBM の DBMAUI 機能を使用してダウンロードされたデータをインポートします。

- DSNTIAUL

  IBM の DB2 から、IBM の DSTINAUL 機能を使用してダウンロードされたデータをインポートします。

ゲートウェイ接続をしていない場合は SQL オプションのエクスポートを機能を使用して XDB データベースからメインフレーム DB2 データベースにデータを転送したり、テーブルを作成するために必要な DDL コマンドを作成したりします。
XDB データベースのデータは、次のどのターゲットにでもエクスポートできます。

- 自由形式の ASCII ファイル (区切り)

  区切り (自由形式) の ASCII ファイルにデータをエクスポートします。

- **固定形式の** ASCII **ファイル** (**カラム状**)

  カラム状 (**固定形式**) **の** ASCII **ファイルにデータをエクスポートします。**

- dBASE **ファイル**

  データを dBASE II **ファイルと** dBASE III **ファイルにエクスポートします。**

- DBMAUI

  データを DBMAUI **ファイルにエクスポートします。**DBMAUI **ファイルは、**DB2 **システムに読み込むことができます。**

- DSNTIAUL

  データを DSNIAUL **ファイルにエクスポートします。**DSNTIAUL **ファイルは、**DB2 **システムに読み込むことができます。**

- WordPerfect Mail Merge **ファイル**

  データを WordPerfect 4.2 **と** 5.0 **の** 2 **次マージファイルにエクスポートします。**

- SQL **文**

  テーブルを再作成し、データをそのテーブルに挿入して、テーブルの指標を作成するために使用できる DDL **コマンドを生成します。**COMMENT、WHERE、**および** SYNONYM **の文をインクルードできます。**

10.4.5.1 インポート / エクスポート時の NLS に関する考察事項

データをインポートまたはエクスポートするときは、文字集合の変換に注意が必要です。特に次の場合には注意してください。

- **データに、**ASCII **文字集合の前半部分に含まれない区別的発音** (**アクセント**) **文字などの文字が含まれる場合。**
- **さまざまな** EBCDIC **コードページを使用してデータを作成した場合。**

SQL **ウィザードを使用して** EBCDIC **データをインポートまたはエクスポートする場合は、**DSNTIAUL **データ形式を使用することをお奨めします。この形式を使用する場合は、**EBCDIC **から** ANSI **へまたは** ANSI **から** EBCDIC **への適切な変換を指定できます。**
「コードページ」ウィンドウでエントリを 1 **つ選択して、適切な変換を指定します。たとえば、スウェーデン語の** EBCDIC **テーブルをエクスポートするとします。エクスポート時には、**ANSI **から** EBCDIC **への変換「1252 - 278」を選択します。または、スペイン語の** EBCDIC **テーブルをインポートする場合は、**EBCDIC **から** ANSI **への変換「284 - 1252」を選択します。**

10.4.6 バッチスクリプトの実行

SQL **ウィザードを使用して、次の種類のバッチスクリプトを実行できます。**

- 指示クエリー
- SQL スクリプト (DDL と DML を含む)
- インポートファイル
- エクスポートファイル

バッチスクリプトを実行するには、SQL ウィザードで [**ファイル**] メニューの [**バッチの実行**] をクリックします。
他のいくつかの SQL オプションユーティリティでもバッチファイルを作成できます。また、対応するユーティリティで、これらのバッチファイルを実行することもできます。たとえば、次のようなユーティリティが使用できます。

- 移行ユーティリティを使用して、.mig ファイルを作成および実行します。
- 宣言ジェネレータユーティリティを使用して、.dge ファイルを作成および実行します。

10.5 移行ユーティリティ

システム管理者によって SQL オプションのメインフレーム接続機能が構成されている場合は、移行ユーティリティを使用して、XDB 位置間および XDB 位置と DB2 サブシステム間でデータをインポートまたはエクスポートできます。
移行ユーティリティは、中間ファイルの転送と指標および外部キーの手動編集を排除することで、位置間でのデータのコピープロセスを簡略化します。移行ユーティリティを使用する場合は、1 つの手順でキー、指標およびテーブルをコピーできます。また、SELECT 文を使用してデータを抽出することで、テーブルのサブセット (選択されたカラムと行) もコピーすることができます。
移行ユーティリティの参照整合性機能を使用する場合は、データの依存関係を自動的に検出し、テーブルを正しい順序でコピーできます。また仮の「影響分析」レポートを作成することもできます。このレポートには、移行を実行する前に移行の影響が記述されます。特定のテーブルを定期的にコピーする場合は、その指定内容を .mig ファイルに保存できます。このファイルをバッチモードで実行する場合は、手動で再指定しなくても移行を実行できます。
移行ユーティリティの詳細については、[**ヘルプ**] メニューで [**SQL For DB2**]、[**移行**] の順にクリックしてください。

10.6 宣言ジェネレータユーティリティ

アプリケーションプログラムから DB2 データにアクセスするには、クエリー内の SQL 文とプログラム間の共通のデータ項目となるホスト変数を作成する必要があります。プログラムでのデータ項目の宣言は、SQL 文で使用する宣言名と同じ名前にする必要があります。また、データ項目宣言は、関連するカラムの関連 DB2 データ型に準拠する必要があります。
宣言ジェネレータユーティリティを使用する場合は、ホスト変数を宣言するコピーブックの作成プロセスを自動化できます。コピーブックを生成するテーブルを指定する場合は、宣言ジェネレータによって、XDB システムで使用されてるデータ名とデータ型に一致するデータ宣言を含むコピーブックが作成されます。宣言ジェネレータは、対話形式で使用することもバッチモードで使用することもできます。また、テーブルごとに個別のコピーブックを作成することも、多くのテーブルに対する宣言を含む 1 つのコピーブックを作成することもできます。
宣言ジェネレータを開くには、[**ツール**] メニューで [**SQL For DB2**]、[**宣言ジェネレータ**] の順にクリックします。

10.7 オプションユーティリティ

DB2 用の SQL オプションを構成したり、SQL オプションと XDB および DB2 データベースサーバとの接続を構成する場合には、オプションユーティリティを使用します。変更できる設定は、次のカテゴリに分類されます。

- 接続

  クライアントアプリケーション (SQL ウィザードまたはユーザが開発したプログラム) とそのアプリケーションが使用する XDB または DB2 データベース間の接続を構成します。

- パス

  SQL ウィザードと他の SQL オプションファイルの検索位置または書き込み先を指定します。

- 形式

  日付、時刻、および数字フィールドの形式を指定します。

- マルチユーザ

  ネットワーク環境内のマルチユーザ XDB サーバに関するオプションを指定します。たとえば、分離レベルや自動コミットを制御できます。

- 参照

  「カタログ参照」ウィンドウに表示するデータベースオブジェクトを指定します。

- クエリー

  クエリーを処理するためデフォルトのオプションを指定します。たとえば、修飾されていないテーブル名を使用するかどうか、およびカルテシアン積を許可するかどうかを指定できます。これらのオプションは、クエリーの編集時に特定のクエリーに対して上書きできます。

- クエリーの実行

  SQL ウィザードでクエリーを実行するオプションを指定します。たとえば、クエリー結果をテーブルビューとフォームビューのどちらに表示するかを制御できます。また、返される行数を制御できます。

- SQL

  さまざまな SQL 操作パラメータ (互換性およびソートシーケンスを含む) のデフォルトを指定します。たとえば、DB2 または SQL/DS の互換性、およびソートシーケンスとして ASCII と EBCDIC のどちらを使用するかを設定できます。また、エスケープ文字や SysAuthID も設定できます。

- セキュリティ

セキュリティオプションを指定します。クライアントのセキュリティをオンに設定したり (使用する XDB サーバのセキュリティがオンに設定されている場合に必要です)、ユーザパスワードの実施基準を制御したりできます。

オプションユーティリティには、現在のすべての構成設定が一覧表示される「**概要**」ページがあります。[**概要**] タブではオプションを変更できませんが、参照用に印刷することはできます。
オプションユーティリティを開くには、[**オプション**] メニューで [**SQL For DB2**] の [**クライアント**] をクリックします。

10.8 バインドユーティリティ

バインドユーティリティでは、DRDA を使用してアクセスするリモートの DB2 位置に静的 SQL パッケージを作成できます。ただし、作成先の位置が DB2 Link ゲートウェイに適切に登録されていること、および AuthID に作成先の位置でのバインド権限があることが必要です。
バインドユーティリティは、SQL プリコンパイラで作成された **.dbr** ファイルに保存されているデータベース要求モジュール (DBRM) を処理することで、リモートシステムに静的 SQL パッケージを作成します。プリコンパイラはプログラムごとに個別の **.dbr** ファイルを作成します。そのファイルには、プログラム内の各埋め込み SQL 文に対する DBRM エントリが 1 つずつ作成されます。
デフォルトの SQL プリコンパイラ設定を使用している場合は、アプリケーションをバインドする必要はありません。デフォルト設定では、プログラム内の埋め込み SQL 文がリモートシステムに対して動的に実行されます。リモートの DB2 データにアクセスしても、アプリケーションを変更するたびに (アプリケーションをデバッグしているときなど) バインドを繰り返したくない場合は、埋め込み SQL 文を動的に実行すると便利です。オプションで、プリコンパイラの設定を変更して、静的な SQL データベース要求モジュール作成し、実装目的でアプリケーションをバインドすることもできます。関連の指令 (DBRM、LOCATION、COLLECTION-ID、および AUTOBIND) の詳細については、[**ヘルプ**] メニューで [**SQL For DB2**]、[**COBOL プリコンパイラ**] の順にクリックし、[**SQL オプションプリプロセッサ**]、[**SQL オプションプリプロセッサ指令**] の順に選択してください。
ある位置に存在するパッケージは、集合識別子、プログラム識別子、バージョンレベルおよび一貫性トークンで識別されます。一般的に、集合識別子は、1 つ以上のプログラムで構成される単一のアプリケーションで使用するパッケージをグループ化するために使用します。プリコンパイラは、プログラムファイル名のルートからパッケージのプログラム識別子を作成します。バージョンラベルを指定する。または、デフォルト値 (01) を受け入れることができます。現在のパッケージはその一貫性トークンに基づいて識別され、プリコンパイルの開始時のタイムスタンプを基に作成されるので、バージョンラベルを指定する必要はありません。DB2 アプリケーションを再コンパイルするたびにタイムスタンプが変更されるので、静的 SQL パッケージを使用している場合には、DB2 ホストの位置にアプリケーションを再バインドする必要があります。コンパイルするたびにアプリケーションを再バインドする手間を省くために、パーソナル XDB サーバの EBCDIC の位置に保存されているデータに対してデバッグすることもできます。そして、目的の安定レベルに到達した後に、メインフレームデータに対してアプリケーションをテストできます。
バインドユーティリティでは、アプリケーションに対して多数のオプション (アプリケーションのバージョン番号や分離レベルなど) を指定できます。これらのオプションはファイルに保存して後から使用することができます。
バインドユーティリティを開くには、[**ツール**] メニューで [**SQL For DB2**]、[**バインド**] の順にクリックします。詳細については、[**ヘルプ**] メニューで [**SQL For DB2**]、[**バインド**] の順にクリックしてください。

10.9 ゲートウェイプロファイルユーティリティ

メインフレームにアクセスするアプリケーションを実行する前に、ゲートウェイプロファイルユーティリティを使用して XDB Link を構成する必要があります。このユーティリティでは、次のことが実行できます。

- ワークステーションとメインフレームデータソース間の接続の定義
- データを正しく変換し、クライアント変換を制御するために必要な設定の構成

ゲートウェイプロファイルユーティリティを開くには、[**オプション**] メニューで [**SQL For DB2**]、[**XDB Link**] の順にクリックします。このユーティリティの詳細については、[**ヘルプ**] メニューで [**SQL For DB2**]、[**DB2 にリンク**] の順にクリックしてください。

10.10 Password Expiration Manager ユーティリティ

Password Expiration Manager (PEM) ユーティリティは、次の操作を実行するときに使用できるプログラムです。

- 期限が満了した MVS パスワードの更新。
- 任意の時点での MVS パスワードの変更。
- DB2 ログオンの失敗の理由の識別。MVS は、ユーザ確認に失敗する場合に、クライアントワークステーションに一般的なセキュリティエラーを返しますが、Password Expiration Manager ユーティリティでは、失敗の原因をすぐに特定できます。失敗の典型的な理由として、無効なユーザ ID、無効なパスワード、パスワードの期限切れ、および無効なユーザ ID が挙げられます。

Password Expiration Manager ユーティリティを開くには、[**ツール**] メニューで [**SQL For DB2**]、[**PEM**] の順にクリックします。このユーティリティの詳細については、[**ヘルプ**] メニューで [**SQL For DB2**]、[**DB2 にリンク**] の順にクリックし、[**PEM ユーティリティ**] を選択してください。

注：Password Expiration Manager ユーティリティを使用するには、XDB Link Host Option for PEM または IBM が提供する CICS の PEM 機能をメインフレームで実行している必要があります。また、データベース管理者はゲートウェイプロファイルユーティリティを使用して、XDB Link がインストールされているマシンで PEM の位置を定義する必要があります。
これらの手順の実行方法の詳細については、XDB Link Host Option を使用している場合は、『***Link Configuration Guide***』を参照してください。IBM の PEM 機能を使用している場合は、『***Link User's Guide***』にある『*XDB Link for PEM(APPC) の構成方法*』のトピックを参照してください。

10.11 SQL オプションプリプロセッサ

SQL プリコンパイラでは、COBOL プログラマは埋め込み SQL を含むアプリケーションを作成し、完全な機能のリレーショナルデータベースシステムに対してそのアプリケーションをテストできます。
COBOL プログラムをコンパイルおよびデバッグするには、COBOL プログラムが特定の要件を満たしている必要があります。これらの要件については、『*データベースアクセス*』マニュアルの第 1 ～ 6 章および『***SQL Option Preprocessor***』オンラインヘルプで説明しています。『***SQL Option Preprocessor***』オンラインヘルプを参照するには、[**ヘルプ**] メニューで [**SQL For DB2**]、[**COBOL プリコンパイラ**] の順にクリックしてから、「**SQL オプションプリプロセッサ**」を選択してください。

10.11.1 SQL 通信領域 (SQLCA)

埋め込み SQL を含むすべての COBOL プログラムでは、作業場所節で SQL 通信領域 (SQLCA) またはフィールド SQLCODE を定義する必要があります。通常、この定義は、Net Express に付属の SQLCA コピーブックをインクルードして実行します。SQLCA 構造体の詳細については、『*SQL 解説書*』を参照してください。『*SQL 解説書*』を表示するには、[**ヘルプ**] メニューで [**SQL For DB2**]、[**SQL 解説書**] の順にクリックしてください。

10.11.2 SQL 記述子領域 (SQLDA)

動的 SQL を含む COBOL プログラムでは、SQL 記述子領域 (SQLDA) を定義する必要があります。通常、この定義は、Net Express に付属の SQLDA コピーブックをインクルードして実行します。ただし、Net Express に組み込まれた SQLDA コピーブックが通常使用する構造体と一致しない場合があります。この場合は、別のコピーブックを作成する必要があります。
SQLDA には動的 SQL クエリーに関する情報が保存されます。SQLDA は、クエリーに適切な量の領域を割り当てるために必要です。
動的 SQL の詳細については、『*動的 SQL*』の章を参照してください。SQLDA 構造体の詳細については、『*SQL 解説書*』を参照してください。『*SQL 解説書*』を表示するには、[**ヘルプ**] メニューで [**SQL For DB2**]、[**SQL 解説書**] の順にクリックしてください。

10.11.3 オブジェクト指向 COBOL 構文のサポート

SQL オプションプリプロセッサは、オブジェクト指向 COBOL の構文 (OO プログラム) と動作するように拡張されました。ただし、次のような制限に注意する必要があります。

- METHOD 内で EXEC SQL WHENEVER 文を使用する場合には、SQL 文をもつ同一 CLASS 内でコード化された追加 METHOD は、先行する定義済み WHENEVER 文内で参照される節をもつ必要があります。これを行わない場合は、節が定義されていないことを示すコンパイルエラーが発生します。この制限は、他に EXEC SQL WHENEVER 文を定義することで回避できます。

10.11.4 セキュリティ

SQL オプションのセキュリティ機能は、システムへのアクセスを制御し、テーブルオブジェクトに保存されたデータを保護します。サーバとクライアントの両方に対してセキュリティオプションをオンにしている場合は、**VALIDATE** プリプロセッサ指令を使用してコンパイルする場合や埋め込み SQL を含むプログラムを実行するときにユーザ ID およびパスワードを入力する必要があります。

10.11.5 DSNTIAR 機能

SQL オプションの DSNTIAR 機能では、SQL 戻りコードを文字列エラーメッセージに変換します (メインフレームの IBM DSNTIAR 機能に似ています)。この機能を起動する方法の詳細については、『*SQL Option Preprocessor*』オンラインヘルプを参照してください。『*SQL Option Preprocessor*』オンラインヘルプを参照するには、[**ヘルプ**] メニューで [**SQL For DB2**]、[**COBOL プリコンパイラ**] の順にクリックしてから、「**SQL オプションプリプロセッサ**」を選択してください。

10.11.6 移行に関する考察事項

このバージョンの Net Express では、SQL オプションプリプロセッサの機能が変更され、古いタイプのプリプロセッサアーキテクチャではなく、外部コンパイラモジュール (ECM) インターフェイスと呼ばれる

新しいタイプのプリプロセッサインターフェイスを使用するようになりました。ECM は、Net Express コンパイラおよびデバッガとの統合が強化されています。
最も大きな相違として、新しいプリプロセッサでは EXEC SQL から始まる文のみ渡されるという点が挙げられます。古いプリプロセッサは、プログラム内のコード全体を参照する必要がありました。
他の相違は、プリプロセッサ指令の数が削減されたという点です。これは、コンパイラとの統合が強化されたことと、コンパイルされるプログラムがデフォルトでメインフレームの DB2 互換性を目標にしていることがその理由です。

10.11.6.1 プリプロセッサの起動方法

以前の Workbench 製品または Mainframe Express バージョン 1.1 から移行するには、SQL オプションプリプロセッサを起動する、次の構文を変更します。
`p($xdbdir¥xdb)` 追加のプリプロセッサ指令 `end-p`
これを、次のように変更します。
`XDB(`追加のプリプロセッサ指令`)`
たとえば、VALIDATE と FILLSYSCAT 指令でプログラム TEST1 をコンパイルする、次のコマンドを変更します。

```
Cobol test1 p($xdbdir¥xdb) validate fillsyscat end-p gnt;
```

これを、次のように変更します。

```
Cobol test1 xdb(validate fillsyscat) gnt;
```

この変更は、バッチファイルを使用してプログラムをコンパイルするすべてのユーザに影響します。

10.11.6.2 注釈内の指令

XDB ソフトウェアの初期のバージョンでは、$$XDB を記述してプリコンパイラ指令を注釈内に記述することもできました。この機能は現在でも使用できますが、新しい SQL オプションプリプロセッサでそれらの指令を選択するには COMPILER 指令 **DIRECTIVES-IN-COMMENTS** を指定する必要があります。

10.11.6.3 あいまいな参照

同じ名前の COBOL 変数を複数回定義し、ホスト変数名を完全に修飾しない場合には、新しいプリプロセッサはあいまいな参照のフラグを設定し、次のメッセージを表示します。
`SQ0408S <variable-name>` が一意ではないため、修飾する必要があります。

10.11.6.4 デバッグファイル

古いプリプロセッサには、テキストファイル XDB.$$$ を生成する DEBUG 指令がありました。この XDB.$$$ には、EXEC SQL 文を SQL オプションの CALL 文に変換する際の変更内容が一覧表示されていました。このファイルは、問題を特定する場合に非常に便利でした。現在のプリプロセッサ指令にはこの機能はありません。そのかわりに、ECM インターフェイスは、問題解決のためにデバッグファイルを作成するかどうかユーザに確認した上で、COMPILER 指令とプリプロセッサ指令を使用してデバッグファイルを生成します。

指令	作成ファイル	機能
XDB(CTRACE)	sqltrace.txt	SQL 文と、EXEC SQL 文によって生成されるすべてのタスクの実行前と実行後の情報、および SQLCA 情報。プリコンパイラエラーは正数の SQLCODE として返されます。SQL オプションサーバエラー (VALIDATE が使用されてる場合) は、サーバが設定した内容

		を変更しないで SQLCODE フィールドに返されます。
CHKECM(CTRACE)	ecmtrace.txt	EXEC SQL 呼び出しを置き換えるために ECM で生成される COBOL コード。古い DEBUG ファイルとこのファイルの大きな相違は、コードを COBOL ファイルで実行し、このファイルに EXEC SQL 文と、ECM で生成された変数のみをインクルードする場合は、文の切り取りや貼り付けを行う必要があるという点です。
CHKECM(TRACE)	ecmtrace.txt	ECM とコンパイラ間で受け渡しされるすべての情報が一覧表示される非常に詳細なファイル。コンパイラによって ECM で問題が発生したことを示すエラーメッセージが生成された場合は、テキストファイルで定数 ECM-FATAL を検索します。これにより、通常はエラーが生成された文を特定できます。
ECMLIST	*program-name*.lst	このファイルは、EXEC SQL 文を置き換えるために生成されるコードを示す擬似 COBOL コードを含む、標準の COBOL リストファイルです。CHKECM(CTRACE) 指令と LIST 指令を使用してプログラムをコンパイルする必要もあります。

同じプログラムをコンパイルして、3 つのデバッグファイルを生成するコマンド行エントリを、次に示します。

```
Cobol test1 xdb(validate fillsyscat ctrace) chkecm(ctrace)
    omf(gnt);
```

10.11.6.5 SQL 文の最大数

新しいプリプロセッサは、1 パスプリコンパイラです。古いプリプロセッサも 1 パスプリコンパラでしたが、あらかじめ読み取ることができました。このため古いプリプロセッサでは、プログラム内の EXEC SQL 文の数を把握して、作業場所節で各 EXEC SQL 文の変数を定義できました。新しいプリプロセッサでは常に、750 個の EXEC SQL 文を保存する記憶域が事前に割り当てられます。プログラムで次のエラーが発生することがあります。

SQ0299s 内部 EXEC SQL 文テーブルがオーバーフローしています。 - 指令 MAXSQL を使用してデフォルト値を上書きしてください。

このエラーが発生した場合は、プリプロセッサ指令 MAXSQL を使用して、プログラムで処理できる SQL 文の最大数を増加できます。

10.11.6.6 IDE での追加指令の設定

新しいプロジェクトを作成したり新しいプログラムを既存のプロジェクトに追加したりする場合は、そのプログラムの [**ビルド設定**] をクリックし、[**コンパイル**] タブをクリックしてから、[**高度**] をクリックして、SQL オプションプリプロセッサ指令を設定できます。「ESQL プリプロセッサ」ドロップダウンリストから「**XDB**」を選択してから、「**指令**」ドロップダウンリストから追加する指令を選択します。指令オプションの簡単な説明が表示されます。

10.11.7 XDB 指令

コンパイラ指令 **XDB** には、次のオプションがあります。

APOST	AUTHID	AUTOBIND

AUTOCLOSE	BLOCKING	COLLECTION-ID
CONCAT	COPY	CTRACE
DATE	DB2	DB2CLOSE
DB2VER	DBERROR	DBRM
DECLARE	DEFAULT-CHAR	DIRECTIVES
EXIST-CHECK	FILLSYSCAT	GENSQLCA
GRANT-EXECUTE	HYPHEN-IN-CURSOR	IGNORE-NESTED
LIBINCLUDE	LOCATION	MAXSQL
NEVERCLOSE	NOFOR	NOT
OMF	OPTIMIZE	PKGSET
QUOTE	SAVE-RETURN-CODE	SQLDA-VER
SQLDS	STRICT-DB2	TIME
VALIDATE	VALIDATE-ERR-LVL	VALIDATE-LOGIN
XDBFUNCS		

『***SQL Option Preprocessor***』オンラインヘルプでは、各指令の構文とオプションおよび古い指令について詳しく説明しています。DB2 などの一部のオプションはデフォルト値です。ほとんどのオプションには代替 (反対の) 構文があり、オプションで設定される条件をオフにすることができます。各オプションのデフォルト構文には下線が付いています。SQL オプションの指令オプションでは、大文字と小文字を区別しません。『***SQL Option Preprocessor***』オンラインヘルプを参照するには、[**ヘルプ**] メニューで [**SQL For DB2**]、[**COBOL プリコンパイラ**] の順にクリックしてから、「**SQL オプションプリプロセッサ**」を選択してください。

10.11.8 エラーメッセージ

SQL オプションプリプロセッサで生成されるエラーメッセージは、すべて SQnnnnl という形式になります。この「nnnn」はエラーメッセージの番号、「l」はその重大度を表します。
『***SQL Option Preprocessor***』オンラインヘルプでは、各エラーメッセージと考えられる原因について説明しています。

10.11.9 リンク方法

アプリケーションにリンクするには、次の手順を実行してください。

1. Net Express プロジェクトを開き、「**ビルドタイプ**」を「**一般リリースビルド**」に設定します。
2. **.exe** または **.dll** ファイルを右クリックします。
3. 未解決記号 _XDBINTRF エラーが表示された場合は、次の操作を実行してください。
    1. [**ビルド設定 ...**] を選択して、[**リンク**] タブをクリックします。
    2. 「**カテゴリ**」を「**高度な設定**」に設定します。

3. 「**これらの LIB とリンクする**」の編集ボックスで、次のように入力します。

   ```
   xdbintrf.lib
   ```

4. 次に、オブジェクトを再度リンクします。
5. これでも問題が解決されない場合は、LIB 環境変数に SQL オプションのディレクトリが含まれているか確認してください。これを確認するには、[**プロジェクト**] メニューを選択し、[**プロパティ**]、[**IDE**]、[**インポート**] の順にクリックします。LIB 環境変数が見つかるまで表示されているリストボックスを下方向にスクロールします。

10.11.10 アプリケーションの分散

ロジックにアクセスする SQL オプションを含む **.exe** または **.dll** を使用して、他のマシンにアプリケーションを分散するには、次の **.dll** をインクルードする必要があります。

- **xdbintrf.dll**
- **xdbcrun.dll**

10.12 SQL オプション NLS 環境

SQL オプションでは、さまざまなメインフレーム EBCDIC コードページすべてをエミュレートする XDB の位置を作成できます。SQL ウィザードまたは COBOL プログラムを使用してこれらの位置にアクセスする場合には、これらの位置のデータは、メインフレームのデータと同じように表示されます。メインフレームや Net Express の IDE (またはその両方) と完全な互換性を確保するためには次の 5 つの手順を実行する必要があります。

1. エミュレートする EBCDIC コードページを決定します。

   通常、メインフレームのコードページを使用します (どのコードページかわからない場合は、XDB Link を使用して確認することができます)。メインフレームがない場合は、どのコードページでも選択できます。

2. このコードページに対するローカルの EBCDIC 位置を作成します。

   SQL ウィザードの「**位置の作成**」ダイアログボックスを使用します。「**ソートシーケンス**」ボックスで必要な EBCDIC コードページを選択します。詳細については、[**ヘルプ**] メニューで [**SQL For DB2**]、[**SQL ウィザード**] の順にクリックしてください。

3. プロジェクトの各国語サポート (NLS) の設定が、指定した EBCDIC 位置と同じであるか確認します。

   Net Express で COBOL プログラムをコンパイルした場合は、そのプログラムを特定の EBCDIC コードページに関連付けることができます。この方法の詳細については、[**ヘルプ**] メニューの [**ヘルプトピック**] をクリックし、[**目次**] タブをクリックしてから、「**開発環境**」、「**データファイルの処理**」、「**Configurable Codesets**」、「**方法**」を順に選択してください。

4. XDB Link を使用している場合は、メインフレームにアクセスするように XDB Link を構成します。

ゲートウェイプロファイルユーティリティを使用して PC ソフトウェアに対する DB/2 の位置を定義してから、(ローカルのワークステーション構成で) 単一のワークステーションコードページを 911 として指定します。次に、SQL ウィザードを使用して、メインフレーム上の DB/2 位置にアクセスします。cpg_info.txt という名前のファイルが mfsql¥bin フォルダに作成され、メインフレーム上の DB/2 位置の EBCDIC コードページが識別されます。次の表で、この EBCDIC コードページ番号を定義します。

Net Express NLS の設定	コードページ番号
31 オランダ語	37
33 フランス語	297
34 スペイン語	284
39 イタリア語	280
43 オーストリアドイツ語	273
44 英語 (英国)	285
45 デンマーク語	277
46 スウェーデン語	278
47 ノルウェー語	277
49 ドイツ語	273
351 ポルトガル語	37
358 フィンランド語	278
437 英語 (米国)	37
500 インターナショナル	500

5. ローカルの EBCDIC 位置にメインフレームからのデータを入力します。

   メインフレームで XDB Link を使用している場合は、2 つの方法でこれを行うことができます。移行ユーティリティを使用する。または、SQL ウィザードを使用します。

   - 移行ユーティリティを使用するには、メインフレーム上の DB/2 の位置を入力元の位置とし、ローカルの EBCDIC 位置を入力先の位置とします。
   - SQL ウィザードを使用するには、最初にメインフレームデータを DSNTIAUL 形式でエクスポートしてから、SQL ウィザードにインポートします。DSNTIAUL 形式を使用する場合は、エクスポートとインポートの両方に対してコードページ変換を指定する必要があります。エクスポートの場合は、PC の ANSI コードページから EBCDIC コードページへの変換を使用し、インポートの場合は、EBCDIC コードページから ANSI への変換を行います。たとえば、スウェーデン語の EBCDIC DB/2 位置から抽出するとします。エクスポートの場合は、ANSI から EBCDIC への変換「1252 - 278」を選択し、インポートの場合は「278 - 1252」を選択します。

   メインフレームで XDB Link がない場合は、メインフレームからのデータを DSNTIAUL 形式のファイルに抽出してから、SQL ウィザードのインポート機能を使用してローカルの EBCDIC 位置にデータをインポートし、適切な EBCDIC から ANSI コードページへの変換を実行します。

XDB EBCDIC データベースは、マシンのデフォルトの OEM 文字集合 (通常、米国ではコードページ 437、他のほとんどの地域ではコードページ 850) を使用して保存されます。テーブルや指標などのデータベースオブジェクトの名前もこの文字集合を使用して妥当性検査されます。オプションで、名前にアクセント文字を使用する場合などに、次の文字集合のどれかを使用してデータベースオブジェクトを妥当性検査することを指定できます。

MS-DOS コードページ	文字集合
437	US Latin 1
850	インターナショナル Latin 1
857	トルコ語
863	カナダのフランス語
865	北欧の言語
860	ポルトガル語

XDB サーバで、名前確認のためにサポートされているコードページの 1 つを使用することを指定するには、mfuser¥config フォルダ内の xdb.ini ファイルに次の 2 行を追加する必要があります。

[SERVER]
XDBCP=*codepage*

codepage は、必要なコードページの 3 桁の番号です。

位置を EBCDIC 位置として作成している場合 (たとえばデフォルトの位置 MAINTAIN) には、データは EBCDIC に自動変換されてからクライアントアプリケーションに渡されます。PC の ASCII データとメインフレームの EBCDIC データ間の変換は、移行ユーティリティを使用している場合には DB2 Link ゲートウェイで処理されます。SQL ウィザードでデータのインポートおよびエクスポート機能を使用している場合は、DSNTIAUL データ形式を選択し、適切なコードページ変換を選択することで、使用する変換を制御できます。詳細については、『*インポート / エクスポート時の NLS に関する考察事項*』を参照してください。

EBCDIC 変換とデータファイルコンバータの詳細については、『*データファイルの変換*』の章を参照してください。

10.13 既存 XDB データの使用方法

XDB をすでに使用している場合は、DB2 用の SQL オプションを使用してそのデータベースにアクセスできます。ただし、それらのデータベースが作成された位置が互換することが前提になります。

10.14 ヒント

- 新しい DML コマンドをプログラムに追加したくても正しい構文がわからない場合は、SQL ウィザードの「クエリーデザイン」ウィンドウを使用します。クエリーを作成してから、正しい結果が得られるまでクエリーを実行します。そして、SQL ウィンドウを開き、プログラムの EXEC SQL 文と手続き部末尾の文の間に SQL 文をコピーします。
- テーブルの作成に必要なプログラムを記述しているときに、適切な構造体のテストテーブルがある場合は、SQL ウィザードの*エクスポート*機能を使用して、プログラム用の DDL コマンドを生成します。
- WITH DEFAULT 制約を使用してテストテーブルに日付カラムを作成する場合は、IDE の日付指定機能を使用してシステム日付を行単位で変更できます。この場合は、XDB サーバを再起動する必要はありません。

- SQL オプションでは、エラーのない状態でプログラム手続きの末尾に到達するとデータベースへの変更が自動的にコミットされます。エラーが発生した場合や、手続きの末尾に到達する前にデバッグを停止する場合は、かわりにロールバックが実行されます。COBOL アプリケーションを実行する場合は、LOGOFF 指令を使用する必要はありません。

第 11 章 : ストアドプロシージャ

ここでは、ストアドプロシージャの概要と、OpenESQL と DB2 でのストアドプロシージャの動作について説明します。
ストアドプロシージャは、SQL 文を実行できるコンパイル済みのプログラムです。
アプリケーションプログラムをクライアント / サーバ環境で使用する場合や次の問題のどちらかが適用される場合は、ストアドプロシージャを使用する必要があります。

- アプリケーションがホスト変数にアクセスするときにセキュリティと整合性を保証したい場合。
- アプリケーションによって一連の SQL 文が実行され、多数のネットワーク送受信操作が作成されるため、CPU と経過時間が過剰に消費されてしまう場合。

11.1 OpenESQL ストアドプロシージャ

OpenESQL では、次の 2 つの文をストアドプロシージャとともに使用できます。

- CALL

  OCBC ストアドプロシージャの呼び出しに対する一般的なサポートを提供します。

- EXECSP

  Micro Focus Embedded SQL Toolkit for Microsoft SQL Server との下位互換性を提供します。

ストアドプロシージャでは、次の処理を実行できます。

- 入力パラメータの受け入れ
- 出力パラメータの返し
- 入力パラメータの受け入れと出力パラメータの返し
- 位置指定パラメータやキーワードパラメータの使用
- 結果の返し
- 結果集合の返し
- パラメータ配列による呼び出し

注：ストアドプロシージャの機能はデータベースベンダごとに大きく異なり、各ベンダは上記の機能を部分的に提供しています。このため、データベース間でのストアドプロシージャの呼び出しは、OpenESQL 文に比べて汎用性がありません。

ストアドプロシージャの呼び出し時には、コンマで区切られたリストとしてパラメータが渡されます。これらのパラメータはかっこで囲むこともできます。パラメータには、ホスト変数、定数、または CURSOR キーワードが使用できます。CURSOR キーワードで渡されたパラメータはバインドが解除されるため、結果集合を返す Oracle 8 ストアドプロシージャのみで使用してください。
パラメータがホスト変数の場合は、その後にパラメータタイプを表す語句 (IN、INPUT、INOUT、OUT、OUTPUT) を指定できます。パラメータタイプを指定しない場合は、INPUT が使用されます。
ホスト変数パラメータは、正式なパラメータ名と等号記号をホスト変数の直前に指定することで、キーワードパラメータとして渡すこともできます。

```
EXEC SQL CALL myProc (keyWordParam = :hostVar) END-EXEC
```

移植性を重視する場合には、ホスト変数のみをパラメータに指定し、定数パラメータは使用しないでください。また、呼び出しに渡されたパラメータは通常の位置指定パラメータとキーワードパラメータのどちらかのみとして扱い、これらの両方として処理しないでください。サーバによっては両方のパラメータがサポートされていますが、その場合でもキーワードパラメータは常にすべての位置指定パラメータの後に渡される必要があります。キーワードパラメータはコードの記述の明確化に役立ち、サーバがデフォルトのパラメータ値とオプションパラメータをサポートする場合に効果的です。
結果集合を返すストアドプロシージャの呼び出しは、カーソル宣言で使用する必要があります。次に例を示します。

```
EXEC SQL
    DECLARE cursorName CURSOR FOR storedProcecureCall
```

この場合には、ストアドプロシージャは他のタイプのカーソルと同様に、カーソルの OPEN 処理と結果集合行の FETCH 処理によって呼び出されます。
現バージョンの OpenESQL は、単一の結果集合のみをサポートしています。
ODBC パラメータは、Oracle 配列パラメータと異なります。パラメータ配列を使用する場合は、配列の各要素に同じ文を繰り返し実行した場合と同じ結果を得られます。ストアドプロシージャ呼び出しで 1 つのパラメータを配列として渡すと、他のすべての引数も同じ数の要素を含む配列として渡されます。ストアドプロシージャでは、これらのパラメータの各「行」で呼び出された場合と同様の処理が実行されます。パラメータで渡される行数は、呼び出しの直前に **FOR :hvar** を指定することによって、配列の上限サイズを超えない数に制限することができます (**:hvar** は、渡すべき行の数を含む整数型のホスト変数です)。

注：Net Express のオンラインヘルプには、CALL 文と EXECSP 文の両方の構造体と例が記載されています ([**ヘルプ**] メニューの [**ヘルプトピック**] をクリックします。[**索引**] タブをクリックし、[**CALL**] または [**EXECSP**] をクリックします)。

11.2 DB2 ストアドプロシージャ

前述したとおり、ストアドプロシージャとは、SQL 文を実行できるコンパイル済みのプログラムです。ストアドプロシージャは、ローカルまたはリモートの DB2 ユニバーサルデータベースサーバに保存されます。ローカルの DB2 ユニバーサルデータベースサーバアプリケーションまたはリモートの DRDA アプリケーションでは、SQL 文の CALL を使用してストアドプロシージャを呼び出します。
ストアドプロシージャを使用して、アプリケーションの数多くの SQL 文を、DB2 サブシステムまたは DB2 ユニバーサルデータベースサーバに対する単一メッセージに組み込むことで、一連の SQL 文を 1 回の送受信操作で処理してネットワークトラフィックを削減できます。

注：DB2 for OS/390 で DB2 コネクトを使用してストアドプロシージャを実行している場合は、IBM DB2 のマニュアルでストアドプロシージャを実行するために必要な他の手順を参照してください。

11.2.1 ストアドプロシージャの処理

ストアドプロシージャを呼び出して実行するには、次の操作を実行します。

1. Net Express 実行可能ディレクトリ (**base¥bin**) を PATH 文に追加します。

   または

COBOL ランタイム .dll (シングルスレッドのランタイムを使用している場合は cblrtss.dll) を、ストアドプロシージャを実行するディレクトリにコピーします。この操作は、DB2 で COBOL ストアドプロシージャを実行できるようにするために必要です。

2. ストアドプロシージャを記述し準備します。この手順については、『ストアドプロシージャの作成と準備』を参照してください。
3. ストアドプロシージャを呼び出すアプリケーションを記述して、準備します。そのアプリケーションの SQL 文 CALL は、呼び出すストアドプロシージャと同じパラメータリストと連結規則を使用する必要があります。この手順については、『ストアドプロシージャを使用するアプリケーションの作成と準備』を参照してください。
4. CREATE PROCEDURE コマンドを実行して、DB2 ユニバーサルデータベースサーバに対してストアドプロシージャを定義します。これにより、行が適切な 1 つ以上のシステムテーブルに保存されます。詳細については、『DB2 ユニバーサルデータベースでのストアドプロシージャの定義』を参照してください。
5. ストアドプロシージャをコンパイルします。詳細については、『DB2 ユニバーサルデータベースでのストアドプロシージャのコンパイル』を参照してください。
6. ストアドプロシージャをデバックし、テストします。詳細については、『DB2 ユニバーサルデータベースでのストアドプロシージャのデバッグ』を参照してください。

11.2.2 ストアドプロシージャの作成と準備

ストアドプロシージャは、DB2 ユニバーサルデータベースサーバのアドレス領域で実行されるアプリケーションプログラムです。ストアドプロシージャには、アプリケーションに通常含まれている多くの文を含めることができます。ストアドプロシージャは、複数のプログラムで構成することができます。ストアドプロシージャでは、入れ子のストアドプロシージャや他のプログラムを呼び出すことができますが、その場合は制限があります。詳細については、『*DB2 Universal Database アプリケーション開発ガイド*』を参照してください。
ストアドプロシージャを呼び出すアプリケーションプログラムは、SQL 文をサポートする言語であればどの言語でも記述できます。ストアドプロシージャは、C、JAVA、COBOL などの多くの言語で記述できます。また、ANSI SQL 99 標準の永続ストアドモジュールに準拠する SQL プロシージャ言語も使用できるようになりました。
ストアドプロシージャを 1 つの言語、たとえば JAVA で記述して、クライアントでは別の言語、たとえば COBOL を使用できます。言語が異なる場合には、DB2 はクライアントとストアドプロシージャ間で値を透過的に渡すため、各プログラムでは、CREATE PROCEDURE 文で定義された形式で値を取得します。

11.2.2.1 ストアドプロシージャの機能

ストアドプロシージャの動作は、他の SQL アプリケーションと似ています。
ストアドプロシージャには、次の制限が適用されます。

- C または COBOL のストアドプロシージャは、Windows サーバで実行するためにビルドする場合には、動的リンクライブラリ (DLL) にする必要があります。
- SQL プロシージャ言語を使用する場合は、SQL 文によって C プログラムが生成されるので、C コンパイラがインストールされている必要があります。
- 次の SQL 文はストアドプロシージャで使用できません。
  - CONNECT
  - DISCONNECT
  - SET CONNECTION

SQL 文の詳細リストについては、『*DB2 Universal Database アプリケーション開発ガイド*』を参照してください。

11.2.2.2 ストアドプロシージャの準備

ストアドプロシージャを DB2 ユニバーサルデータベースサーバで実行するためには、事前に次の作業を実行する必要があります。

1. 埋め込み SQL のマニュアルでストアドプロシージャの作成手順を参照して、アプリケーションを準備します。
2. DB2 ユニバーサルデータベースサーバに対してストアドプロシージャを定義します。『DB2 ユニバーサルデータベースでのストアドプロシージャの定義』を参照してください。
3. ストアドプロシージャ DLL または JAVA ルーチンを、DB2 ユニバーサルデータベースサーバマシンの、CREATE PROCEDURE で指定したディレクトリに保存します。このディレクトリを指定しない場合は、DB2 ユニバーサルデータベースがインストールされているデフォルトの sqllib¥function サブディレクトリが使用されます。他のオプションについては、『*DB2 Universal Database アプリケーション開発ガイド*』および『*DB2 Universal Database SQL リファレンス*』を参照してください。

11.2.2.3 アプリケーションでのストアドプロシージャの処理方法

標準のストアドプロシージャには、操作処理や論理処理を行う 2 つ以上の SQL 文が含まれます。この例では、CALLSTPR という名前のアプリケーションをワークステーションのクライアントで実行し、ストアドプロシージャ GETEMPSVR を呼び出します。次の処理が行われます。

1. ワークステーションのアプリケーションは、DB2 ユニバーサルデータベースサーバへの接続を確立します。
2. SQL 文の CALL によって、アプリケーションでストアドプロシージャ GETEMPSVR を実行することを、DB2 ユニバーサルデータベースサーバに通知します。呼び出し側アプリケーションは、必要なパラメータを提供します。
3. DB2 ユニバーサルデータベースサーバは、システムテーブルで、ストアドプロシージャ GETEMPSVR と関連付けられた行を検索します。
4. DB2 ユニバーサルデータベースサーバは、要求に関する情報をストアドプロシージャに渡します。
5. ストアドプロシージャは、SQL 文を実行します。
6. ストアドプロシージャは出力パラメータに値を代入して、終了します。
7. 呼び出し側アプリケーションに制御が戻り、アプリケーションは出力パラメータを受け取ります。

アプリケーションは他のストアドプロシージャを呼び出したり、他の SQL 文を実行したりできます。アプリケーションの設計者は、ストアドプロシージャでの作業の COMMIT 処理をサーバで実行するか、または、単一のトランザクションとしてクライントで実行するかを決定します。

11.2.3 ストアドプロシージャを使用するアプリケーションの作成と準備

ストアドプロシージャを呼び出して、パラメータのリストをそのプロシージャに渡すには、SQL 文の CALL を使用します。アプリケーションプログラムは、複数のストアドプロシージャを呼び出せます。サーバに接続したアプリケーションは、ストアドプロシージャへの呼び出しを SQL 文とともにサーバに送信できます。

11.2.3.1 SQL 文 CALL の実行

CALL 文を使用して、アプリケーションで一連の SQL 文をそれぞれ実行します。

11.2.3.1.1 例 1

『アプリケーションでのストアドプロシージャの処理方法』での説明に従ってストアドプロシージャを呼び出すには、アプリケーションで次の文を使用します。

```
EXEC SQL
        CALL GETEMPSVR (:V1, :V2)
END-EXEC
```

CALL 文でホスト変数を使用する場合は、それらのホスト変数を先に宣言する必要があります。

11.2.3.1.2 例 2

最初の例は、入力パラメータに NULL 値を使用できないことが仮定になります。NULL 値を使用できるようにするには、次のように文を作成します。

```
EXEC SQL
        CALL GETEMPSVR (:V1 :IV1, :V2 :IV2)
END-EXEC
```

:IV1 と :IV2 は、パラメータのインジケータ変数です。

11.2.3.1.3 例 3

整数または文字列の定数値や NULL 値をストアドプロシージャに渡すには、次のように文を記述します。

```
EXEC SQL
        CALL GETEMPSVR (2, NULL)
END-EXEC
```

11.2.3.1.4 例 4

ストアドプロシージャの名前にホスト変数を使用するには、次のように文を記述します。

```
EXEC SQL
        CALL :PROCNAME (:V1, :V2)
END-EXEC
```

11.2.3.1.5 例 5

ストアドプロシージャ名を GETEMPSVR と仮定します。ホスト変数 PROCNAME は、値 GETEMPSVR を含む 254 文字以内の文字変数です。ストアドプロシージャ名は不明でも、パラメータリスト規則が判明している場合にはこの方法を使用してください。
個々のホスト変数ではなく、1 つの構造体でパラメータを渡すには、次のように文を記述します。

```
EXEC SQL
        CALL GETEMPSVR USING DESCRIPTOR :ADDVALS
END-EXEC
```

ADDVALS は、SQLDA の名前です。

11.2.3.1.6 例 6

SQLDA を使用するストアドプロシージャにホスト変数名を使用するには、次のように文を記述します。

```
EXEC SQL
```

```
        CALL :PROCNAME USING DESCRIPTOR :ADDVALS
END-EXEC
```

この形式は特に柔軟性があります。この形式では、1 つの CALL 文を使用してさまざまなパラメータリストをもつ複数の異なるストアドプロシージャを呼び出すことができるからです。
クライアントプログラムでは、SQL CALL 文を作成する前に、ホスト変数 PROCNAME にストアドプロシージャ名を代入して、SQLDA ADDVALS にパラメータ情報をロードする必要があります。
上記の各 CALL 文の例では、SQLDA を使用しています。SQLDA を明示的に指定しない場合には、プリコンパイラは、パラメータリスト内の変数に基づいて SQLDA を生成します。
CALL 文は、アプリケーションプログラムからのみ実行できます。CALL 文を動的に使用できません。

11.2.3.2 パラメータ規則

アプリケーションが CALL 文を実行する場合には、DB2 ユニバーサルデータベースサーバは、文で指定されたパラメータと値を使用してストアドプロシージャのパラメータリストを作成します。DB2 ユニバーサルデータベースサーバは、システムテーブルからパラメータに関する情報を取得します。詳細については、『DB2 ユニバーサルデータベースでのストアドプロシージャの定義』を参照してください。パラメータは、次のタイプのどれかに定義されます。

- IN

  入力専用パラメータ。ストアドプロシージャに値を渡します。

- OUT

  出力専用パラメータ。ストアドプロシージャから呼び出し側プログラムに値を返します。

- INOUT

  入出力パラメータ。ストアドプロシージャに値を渡したり、ストアドプロシージャから値を返したりします。

ストアドプロシージャで 1 つ以上の出力専用パラメータを設定できなかった場合は、DB2 ユニバーサルデータベースサーバが、ストアドプロシージャへの入力時に設定された値を使用して出力パラメータを呼び出し側プログラムに返します。COBOL は 3 つのパラメータリスト規則をサポートします。他の言語は他の規則をサポートします。パラメータリスト規則は、CREATE ORICEDURE 文で定義されたパラメータ形式に基づいて選択されます。

パラメータ形式	説明
SIMPLE	SIMPLE (または GENERAL) を使用して、呼び出し側プログラムが、入力パラメータ (IN または INOUT) として NULL 値をストアドプロシージャに渡さないようにします。ストアドプロシージャは、CALL 文で渡される各パラメータの変数を宣言する必要があります。
SIMPLE WITH NULLS	SIMPLE WITH NULLS (または GENERAL WITH NULLS) を使用して、呼び出し側プログラムでストアドプロシージャに渡されるパラメータに NULL 値を使用できるようにします。次の規則が適用されます。 • インジケータ変数を、呼び出し側プログラムの CALL 文の各パラメータの後に記述する必要が

	あります。この場合は、次の形式のどちらかを使用します。 o ホスト変数 :インジケータ変数 または o ホスト変数 INDICATOR :インジケータ変数 • ストアドプロシージャは、CALL 文で渡される各パラメータの変数を宣言する必要があります。 • ストアドプロシージャは、CALL 文で渡される各パラメータのインジケータ変数を含む NULL インジケータ構造体を宣言する必要があります。 • エントリ時には、ストアドプロシージャは入力パラメータに関連付けられたすべてのインジケータ変数を調べて、NULL 値を含むパラメータを判断する必要があります。 • 終了時には、ストアドプロシージャは、出力変数に関連付けられたすべての変数に値を代入する必要があります。呼び出し側に対して NULL 値を返す出力変数のインジケータ変数には、負数を代入する必要があります。それ以外の場合は、インジケータ変数にはゼロ (0) の値を代入する必要があります。
DB2SQL	CALL 文のパラメータに加え、次の引数がストアドプロシージャに渡されます。 • CALL 文の各入力パラメータに対する NULL インジケータ • DB2 に返される SQLSTATE • ストアドプロシージャの修飾名 • ストアドプロシージャの特定名 • DB2 に返される SQL 診断文字列

11.2.3.3 インジケータ変数による処理の高速化

出力パラメータに多くの記憶域が必要な場合は、記憶域全体をストアドプロシージャに渡すのではなく、SQL 文 CALL のすべての大きいサイズの出力パラメータに対してインジケータ変数を宣言します。インジケータ変数は、呼び出し側プログラムでストアドプロシージャに 2 バイトの領域のみを渡し、ストアドプロシージャから領域全体を受け取る場合に使用します。

注:SIMPLE WITH NULLS 連結規則を使用している場合は、すべてのパラメータに対してインジケータ変数を宣言する必要があります。このため、大きいサイズの出力パラメータに対して別のインジケータ変数を宣言する必要はありません。

大きいサイズの出力変数に関連付けられている各インジケータ値に負数を代入します。そして、CALL 文にインジケータ変数をインクルードします。この方法は、ストアドプロシージャの連結規則が SIMPLE または SIMPLE WITH NULL の場合に使用できます。
たとえば、SIMPLE 連結規則で定義されているストアドプロシージャ STPROC2 は、1 つの整数入力パラメータと、長さが 5000 の文字出力パラメータをもつと仮定します。この場合は、ストアドプロシージャに 5000 バイトの記憶域を渡すのはむだな領域を消費することになります。かわりに、これらの文を含む COBOL プログラムで、出力変数のストアドプロシージャに 2 バイトのみ渡し、ストアドプロシージャから 5000 バイトすべてを受け取ります。

```
INNUM           PIC S9(9) COMP
OUTCHAR PIC X(5000)
IND             PIC S9(4) COMP
.
.
.
MOVE -1                       TO IND
EXEC SQL CALL STPROC2(:INNUM, :OUTCHAR :IND)  END-EXEC
```

11.2.3.4 渡されたパラメータのデータ型の宣言

ストアドプロシージャは、渡された各パラメータを宣言する必要があります。[1] また、DECLARE PROCEDURE の PARMLIST カラムには、各パラメータに対する互換性のある SQL データ型宣言を含める必要があります。PARMLIST 文字列と対応する言語の宣言については、Net Express の『*データベースアクセス*』マニュアルで SQL データ型の表を参照してください。
次に例を示します。

```
CREATE PROCEDURE GETEMPSVR
 (IN  EMPNO CHAR(6),
  INOUT SQLCD    INT ,
  OUT FIRSTNME CHAR(12),
  OUT LASTNAME CHAR(12),
  OUT HIREDATE CHAR(10),
  OUT SALARY   DEC(9,2) )
  LANGUAGE COBOL
  EXTERNAL NAME 'GETEMPSVR!GETEMPSVR'
  PARAMETER STYLE DB2SQL;
```

11.2.3.5 制限

IBM 社は、ストアドプロシージャに関連するすべての SQL 構文のサポートを、サポートされるすべての言語に実装しているわけではありません。たとえば、結果集合を使用したり、DB2 ユニバーサルデータベースのワークステーションバージョンでその機能がサポートされる EXEC SQL 構文を使用したりする COBOL ストアドプロシージャは作成できません。これは、将来変更される可能性があります。サポートされる機能とその機能に対応する言語の詳細については、『*DB2 Universal Database SQL リファレンス*』および『*DB2 アプリケーション開発ガイド*』を参照してください。
ネイティブな DB2 プリコンパイラを使用する場合は、構造体、配列、またはベクトルパラメータはサポートされません。ただし、OpenESQL プリコンパイラと ODBC 接続を使用するとより柔軟な処理ができます。詳細については、Net Express の『*データベースアクセス*』マニュアルを参照してください。

11.2.4 DB2 ユニバーサルデータベースでのストアドプロシージャの定義

ストアドプロシージャは、定義されるまで使用できません [1]。定義するには、CREATE PROCEDURE コマンドを使用します。この場合は、DB2 コマンドプロンプトを使用するか、または、プログラム内にコマ

ンドを記述し、コンパイルして実行します。DB2 コマンドプロンプトを使用する場合は、最初に、ストアドプロシージャを実行する DB2 ユニバーサルデータベースサーバに接続します。
たとえば、次のようにして接続します。

```
C:> db2 connect to sample
```

DB2 コマンドプロンプトでコマンドを入力する (継続文字とコマンド区切りを含む) か、または、ANSI テキストファイルに CREATE PROCEDURE を記述できます。たとえば、テキストファイル **creproc.sql** に前述のコマンドが記述されている場合には、入力するコマンドは次のようになります。

```
C:> db2 -td; -vf creproc.sql
```

詳細は、次のとおりです。

- 「-td」オプションは、次の文字がコマンドの末尾である区切り文字であることを示します。この例では、セミコロン (;) になります。
- 「-vf」オプションは、次のトークンが、SQL コマンドスクリプトを含む処理ファイルであることを示します。

CREATE PROCEDURE 文は、ストアドプロシージャを一意に識別します。ストアドプロシージャを変更して、パラメータの追加や削除または機能の変更を行う場合は、DROP PROCEDURE コマンドを使用します。そして、CREATE PROCEDURE コマンドでストアドプロシージャを再度追加する必要があります。

---

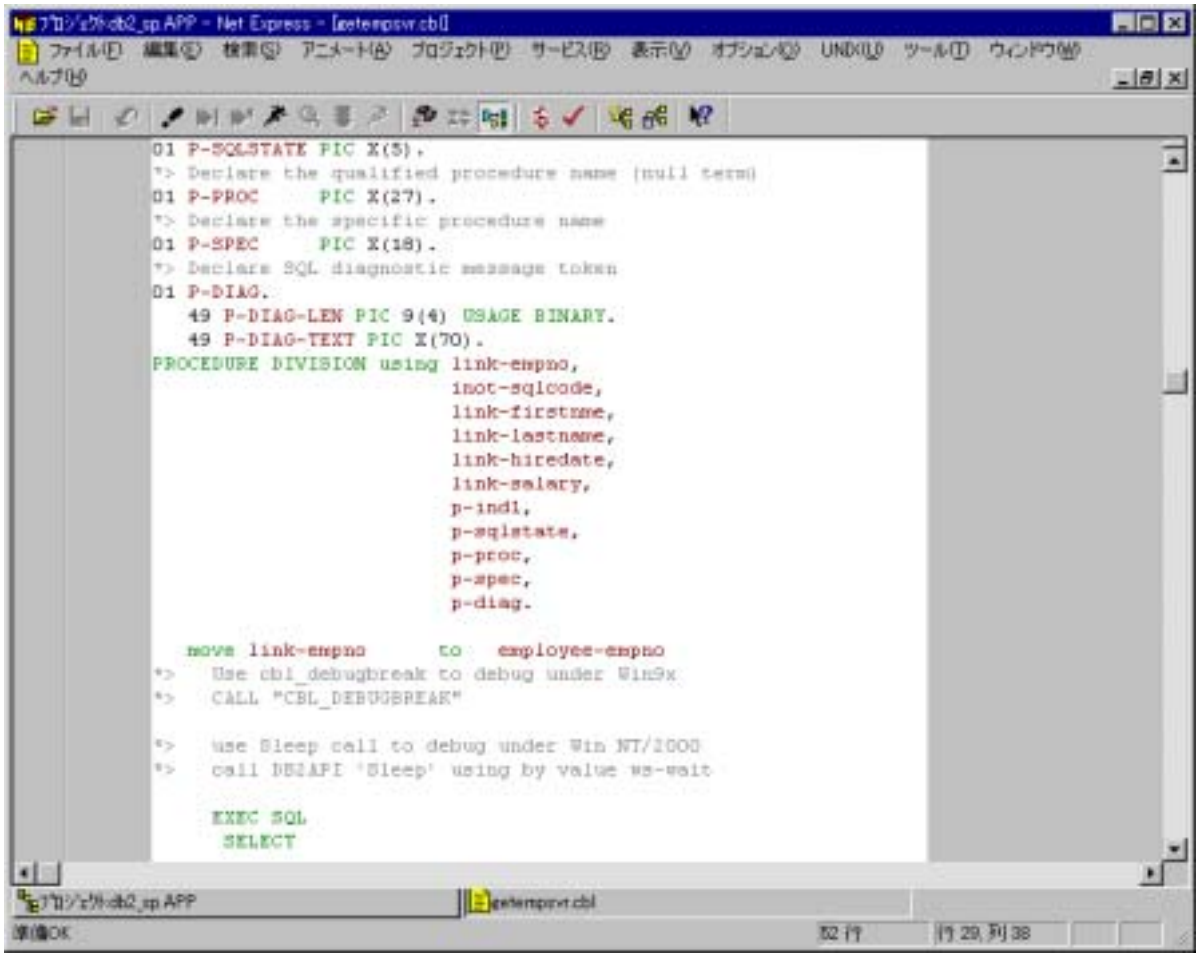
[1] DB2/2 が最初に開発された時点では、CREATE PROCEDURE 機能はサポートされていませんでした。また、CREATE PROCEDURE を実行しないで COBOL ストアドプロシージャを作成することも可能です。この方法の例と必要なパラメータリストは、DB2 ユニバーサルデータベースアプリケーション開発クライアントに組み込まれています。

11.2.5 DB2 ユニバーサルデータベースでのストアドプロシージャのコンパイル

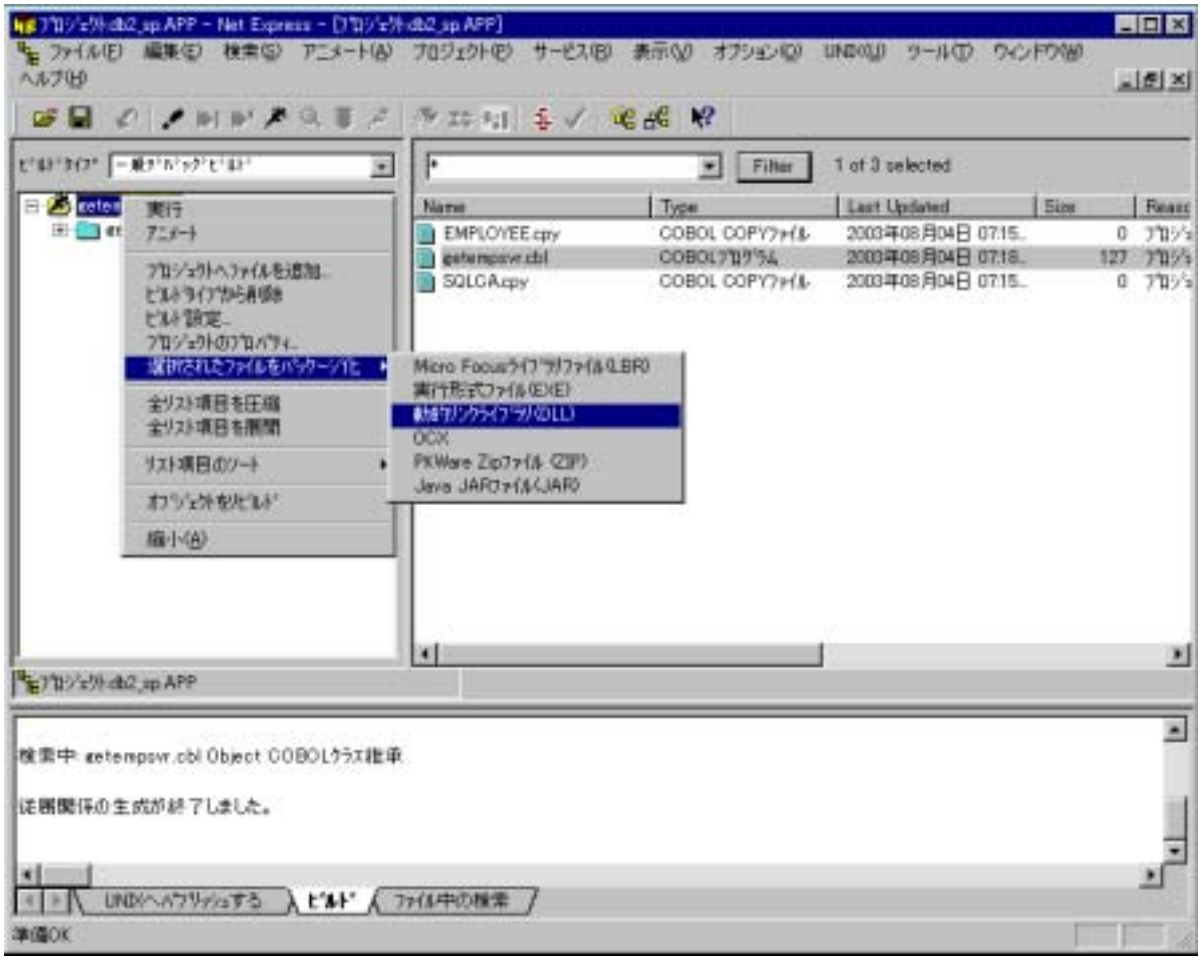
Net Express と DB2 ユニバーサルデータベースを使用して COBOL ストアドプロシージャをコンパイルするには、次の手順を実行します。

1. ストアドプロシージャとして使用するプログラムを、DB2 ユニバーサルデータベースプログラムと同じように、DB2 指令でコンパイルします。これは、プログラムに $SET 文を追加して行います。DB2 指令オプションの詳細については、Net Express の『***データベースアクセス***』マニュアルを参照してください。

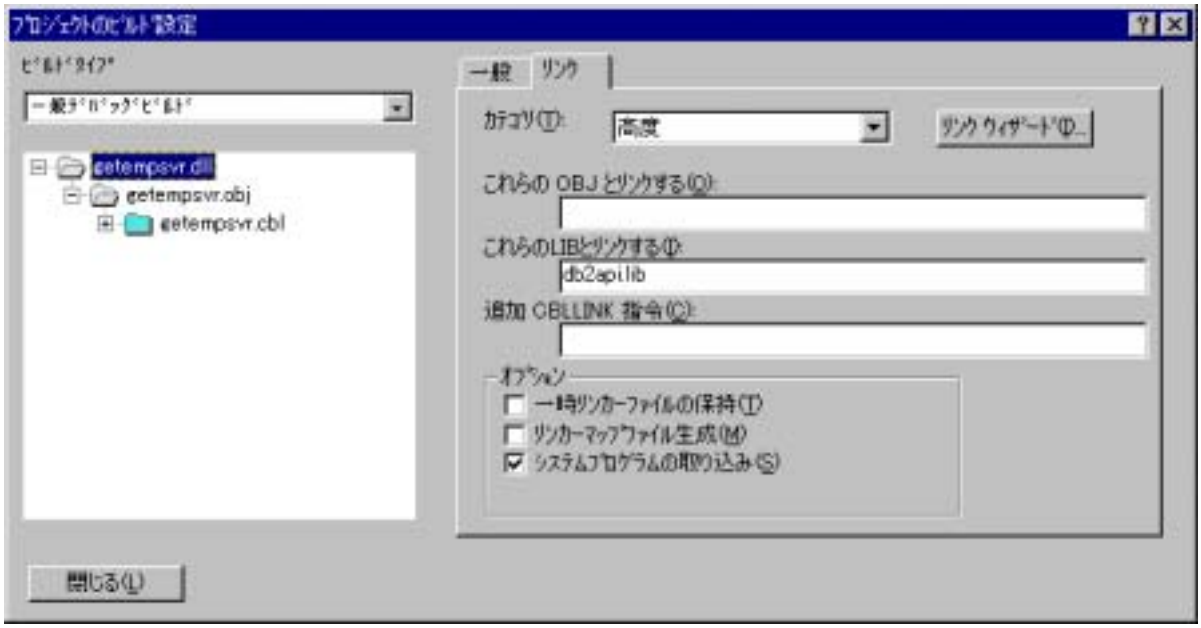
   プログラムにストアドプロシージャの CALL 文が含まれている場合は、CALL_RESOLUTION DB2 指令を指定してください。この指令を指定しない場合は、SQL0204 エラーが発生します。



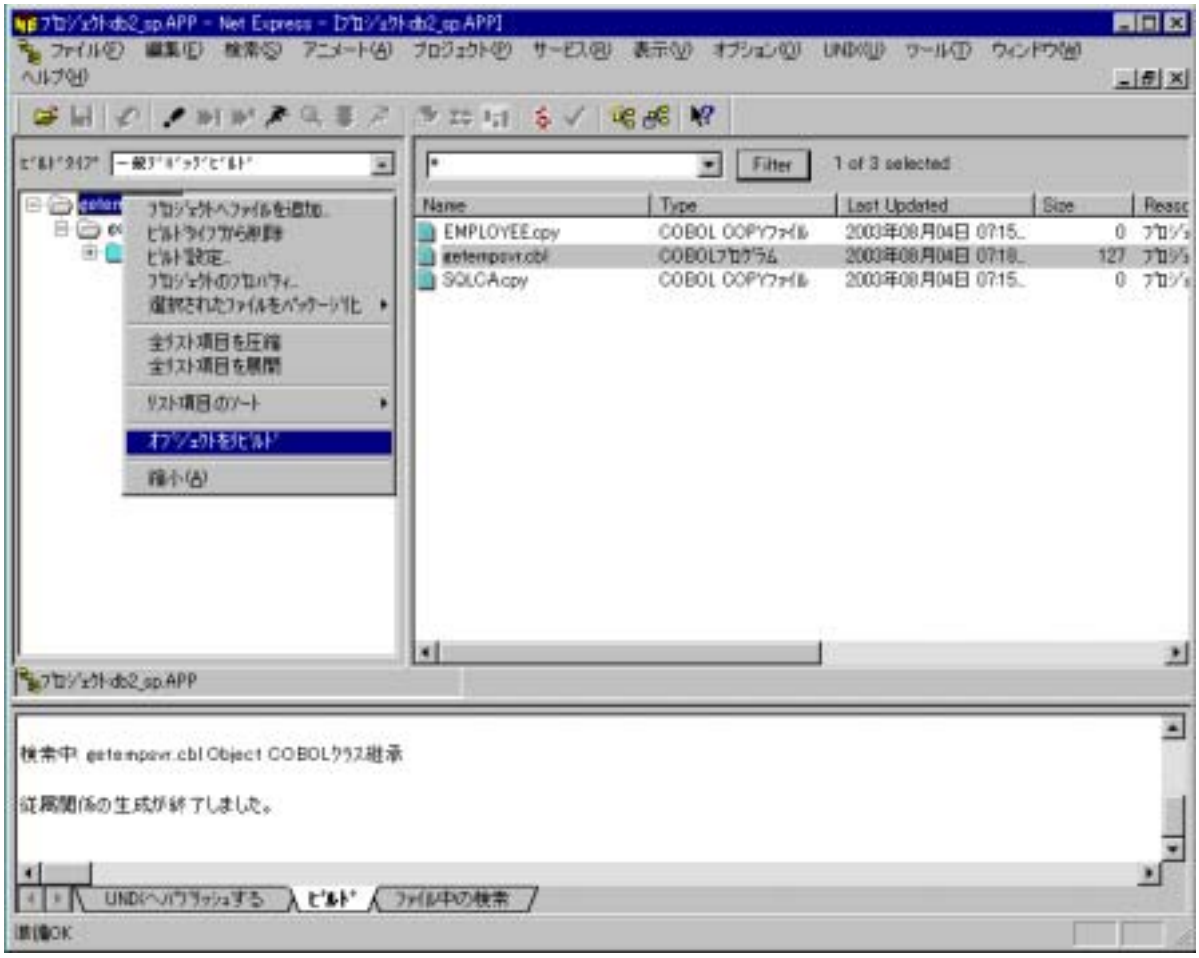
2. **プログラムを** Net Express **プロジェクトに追加した後に、プログラムを選択し、そのプログラムを動的リンクライブラリ (.dll) としてパッケージ化します。**



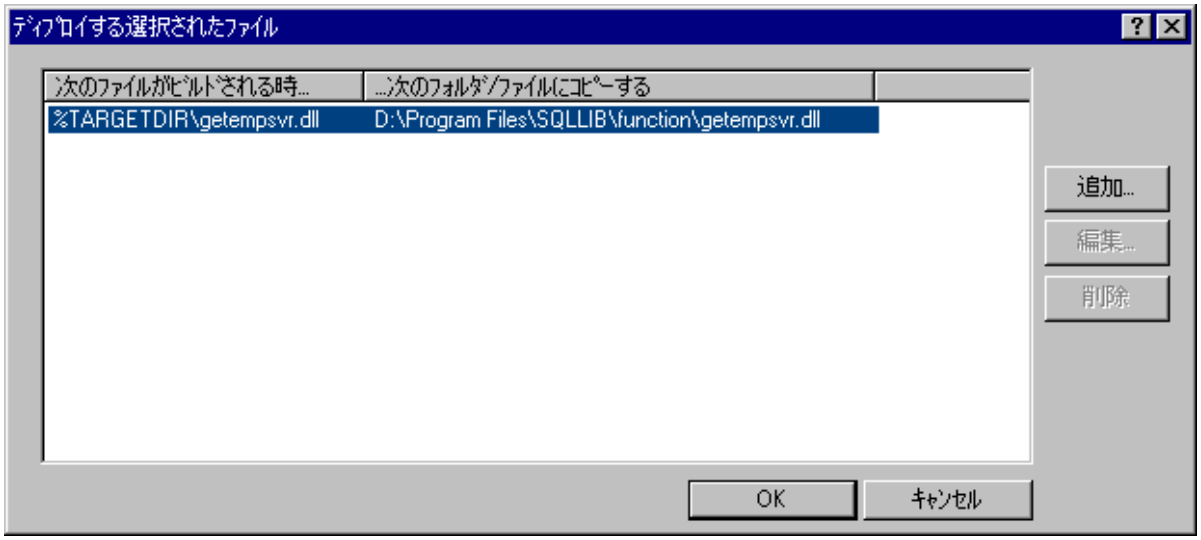
3. **プログラムのビルド設定で、[リンク] タブを選択し、ドロップダウンリストボックスから「高度な設定」カテゴリを選択します。「これらの LIB とリンクする」エントリフィールドで、db2api.lib を追加し、[閉じる] ボタンをクリックします。**



4. ストアドプロシージャをコンパイルおよびリンクするには、.dll ファイルを選択してから、コンテキストメニューで [オブジェクトをリビルド] を選択します。



5. CREATE PROCEDURE の定義方法に応じて、ストアドプロシージャをテストするか、または、ストアドプロシージャを sqllib¥function サブディレクトリにコピーできます。[オブジェクトをリビルド] コマンドを実行した後に、[プロジェクト] メニューで [ディプロイ] を選択してからファイルを選択し、コピー先を指定します。



11.2.6 DB2 ユニバーサルデータベースでのストアドプロシージャのデバッグ

ストアドプロシージャをデバッグすることはサーバを停止し、サーバコードを操作することを意味するため、ストアドプロシージャコードをデバッグするプログラマは、データベースサーバを使用する唯一のユーザであることが必要です。このため、可能な場合は使用していてるワークステーションのデータベースを使用して、DB2 ユニバーサルデータベースのストアドプロシージャをデバッグする必要があります。クライアントプログラムを記述しないで COBOL ストアドプロシージャをテストするには、次の方法を使用します。

- ストアドプロシージャを実行する IBM ストアドプロシージャビルダ。ストアドプロシージャが保存されているデータベースに接続して、実行するストアドプロシージャを選択します。ストアドプロシージャビルダでは、INPUT 値と INOUT 値の入力が求められます。その結果とエラーが表示されます。
- DB2 コマンドプロンプト。

予期した結果にならない場合は、COBOL ストアドプロシージャをデバッグします。Windows 9x で Net Express を実行している場合は、ストアドプロシージャに CBL_DEBUGBREAK の呼び出しを追加するのみで、文の実行時に Net Express デバッガが表示されます。
Windows NT/2000/XP で実行している場合には、DB2 ストアドプロシージャは、db2dari プロセスで実行されます。これは、CBL_DEBUGBREAK の呼び出しがそれ自体では機能しないことを意味します。「sleep」関数を使用してストアドプロシージャをコンパイルし、実行プロセスにアタッチする必要があります。CBL_DEBUGBREAK ウィンドウが表示された後に、「NO」で応答します。「YES」で応答すると失敗します。
sleep 関数が切れる前に、db2dari プロセスにアタッチする必要があります。sleep 関数が切れると、ストアドプロシージャはアニメートできません。アニメートできない場合は、次に説明する処理を最初から繰り返す必要があるので、sleep 値には大きい値を設定することが重要です。
sleep 値を設定するには、sleep 時間を定義する変数を作業場所節で定義します。sleep 関数の呼び出しは、CBL_DEBUGBREAK の呼び出しの前または後に記述します。
次に例を示します。

```
01  ws-wait                  pic 9(8) comp-5 value 30000.
```

プログラムに次の文を追加して、アニメートを開始する前にプログラムが実行されるようにします。

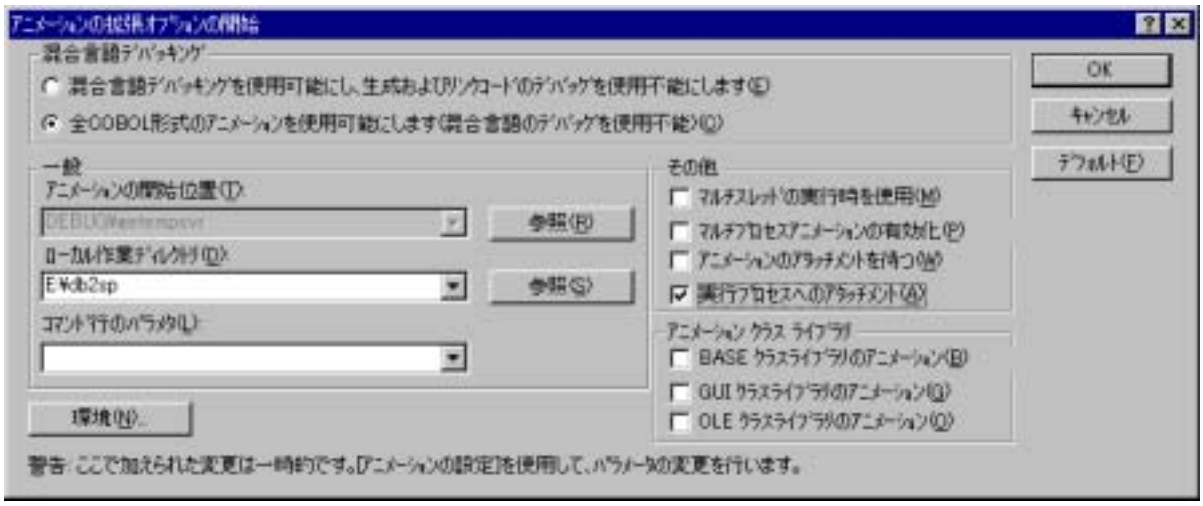
```
call DB2API 'Sleep' using by value ws-wait
```

DB2API は、特殊名段落で呼び出し規約 74 として定義されます。
ストアドプロシージャをアニメートするには、次の操作を実行します。

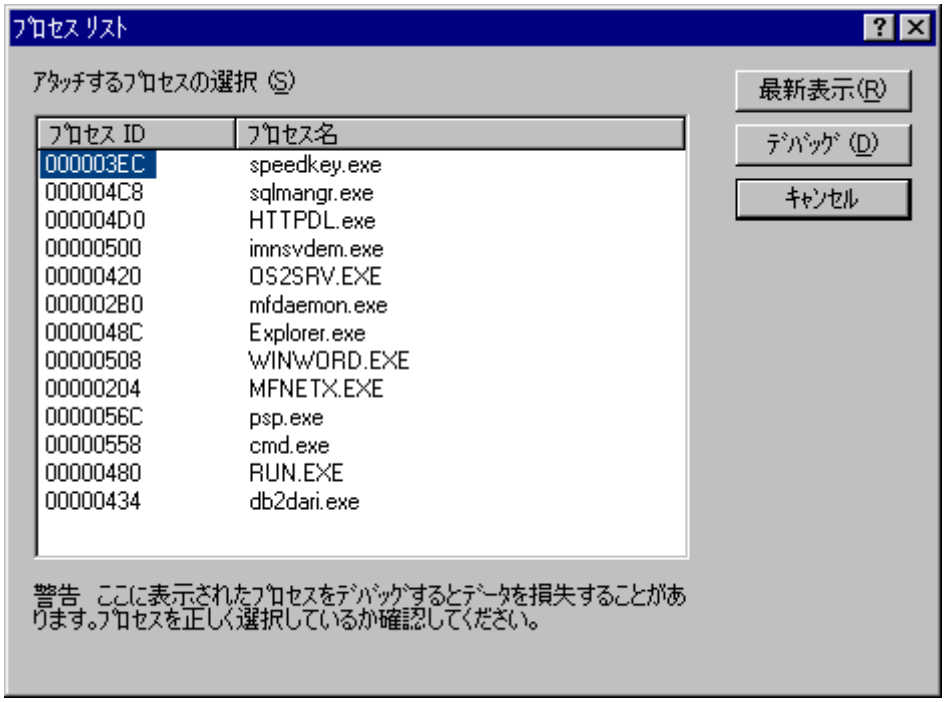
1. IBM ストアドプロシージャビルダを使用するか、または、クライアントプログラムを実行して、ストアドプロシージャを呼び出します。
2. Net Express を起動し、[**ステップ実行**] ボタンをクリックして、「アニメーションの起動」ウィンドウを表示します。

   デバッガを実行プロセスにアタッチするには、管理者権限が必要です。

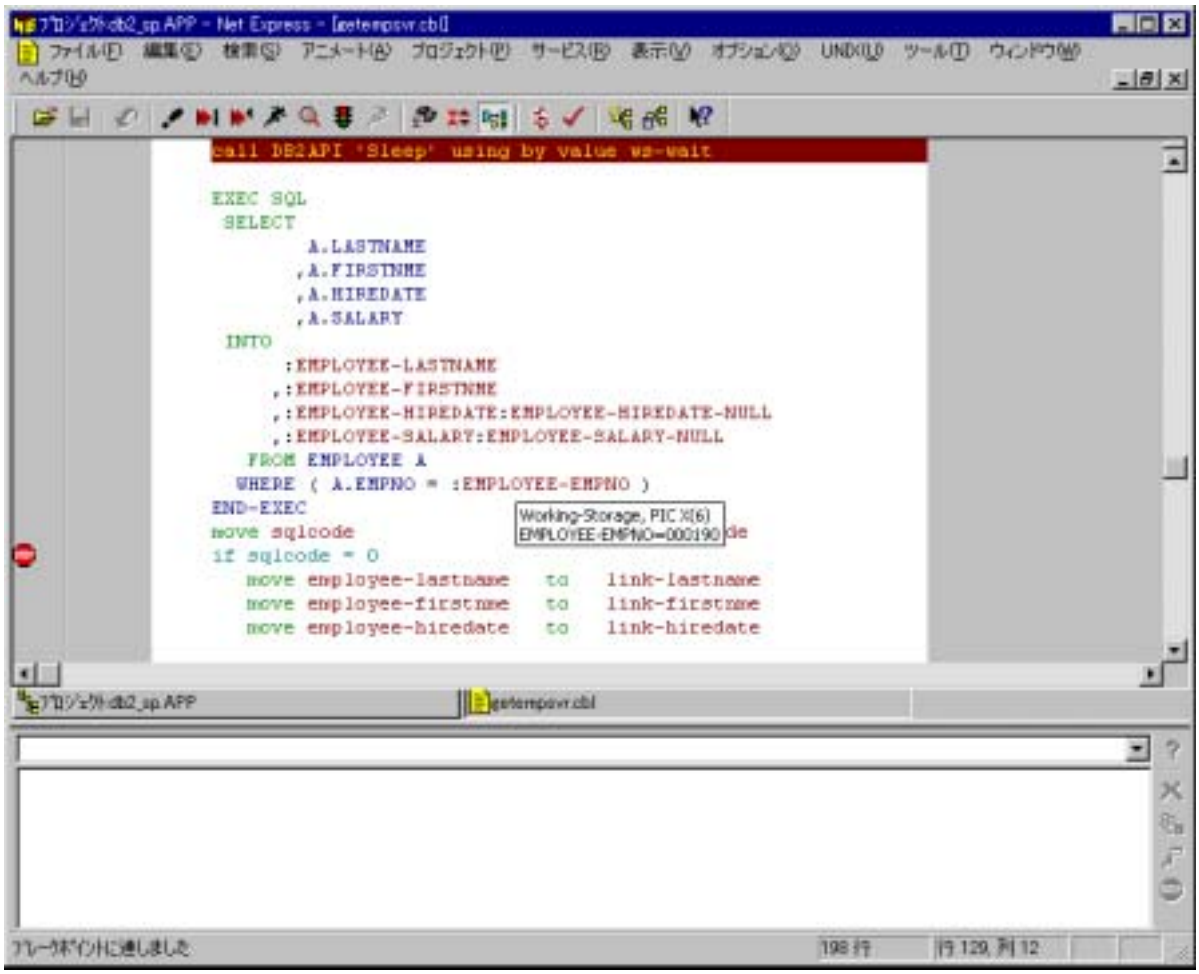
3. [**オプション**] ボタンをクリックします。
4. 「実行中プロセスへアタッチ」チェックボックスをチェックし、[OK] ボタンをクリックします。



5. **db2dari.exe** に関連付けられたプロセスを選択し、[**デバッグ**] ボタンをクリックします。
   db2dari プロセスが一覧表示されない場合は、[**最新表示**] ボタンをクリックします。



6. [**ブレーク**] ボタンをクリックします。デバッガが表示されます。
7. COBOL 文で、アニメートを開始するブレークポイントを設定し、[**実行**] ボタンをクリックします。待機時間の値を「1」に変更して、スリープタイマが切れるのを待機する必要がないようにします。



ストアドプロシージャのデバッグが完了したら、Animator を終了します。

ヒント：ストアドプロシージャが予期しない動作を行う場合は、連結節の各パラメータを調べて、ストアドプロシージャに渡されたパラメータが要求された形式になっているか確認してください。

第 12 章 : COBSQL

ここでは、COBSQL 統合プロセッサの使用方法について説明します。COBSQL は、リレーショナルデータベースベンダが提供している COBOL プリコンパイラとともに機能します。使用するプリコンパイラのリストについては、『*はじめに*』の章を参照してください。
SOURCEFORMAT"FREE" 指令でコンパイルされたプログラムで COBSQL を使用することはできません。プログラムが自由形式の場合は、OpenESQL を使用してください。

12.1 概要

COBOL プログラムに次の形式で埋め込み SQL 文を記述すると、Oracle、Sybase、または Informix のデータベース管理システム (DBMS) の SQL 機能を利用することができます。

```
EXEC SQL
   SQL statement
END-EXEC
```

また、埋め込み SQL 文を含むプログラムは、COBOL コンパイラでコンパイルする前に Oracle、Sybase、または Informix のプリコンパイラで処理する必要があります。この処理によって、埋め込み SQL 文が対応するデータベースサービス呼び出しに変換されます。さらに、ソースコードには COBOL ホスト変数をデータベースシステム側の SQL 変数名にバインドするコードが追加されます。
この方法の利点は、各データベースルーチンの呼び出し形式に精通している必要がありません。また、欠点はプログラムをアニメートする場合はプリコンパイラの出力コードが表示され、埋め込み SQL 文を含む元のソースコードを見ることができません。この問題は COBSQL を使用すると回避できます。
COBSQL はサードパーティのスタンドアローンプリコンパイラと Micro Focus COBOL の環境の間を統合するインターフェイスとして機能します。COBSQL を使用すると、EXEC SQL 文を含むプログラムをアニメートでき、プリコンパイラが生成するコードではなく、元のソースコードを表示できます。
ここでは、Oracle、Sybase、または Informix プリコンパイラと COBSQL を 組み合わせて、プログラムのコンパイルとアニメートを実行する方法を説明します。

12.2 操作方法

COBSQL を使用するには、プログラムのコンパイル時に PREPROCESS"COBSQL" コンパイラ指令を指定します。この指令以降のすべての指令が、コンパイラから COBSQL に渡されます。コンパイラ指令は $SET 文でプログラム内に記述できます。Net Express の「**ビルド設定**」ダイアログボックスで設定することもできます。
COBSQL に渡す指令を終了するには、COBOL の ENDP 指令を使用する必要があります。これを行うには、プロジェクト設定または Net Express のビルド設定の末尾に、次の行を追加します。

```
Preprocess(cobsql) csqltype=database_product end-c
    comp5=yes endp;
```

database_product には、Oracle、Sybase、または Informix のどれかを指定します。たとえば、Oracle の場合は、次の行を追加します。

```
Preprocess(cobsql) csqltype=oracle end-c comp5=yes endp;
```

注: Net Express のビルド設定では、セミコロン (;) に後に続く指令はすべて無視されます。このため、ビルド設定に上記の行を追加する場合は、設定の末尾にこの行を置く必要があります。

Net Express の「ビルド設定」ダイアログボックスで指令を設定する場合には、システムによってデフォルトの COBOL 指令が追加されるため、必ず上記の行を記述する必要があります。

プロジェクト設定と Net Express の「ビルド設定」ダイアログボックスのどちらで設定した場合でも、END-C と ENDP を含む行は次のように処理されます。

- END-C の前に記述された指令が COBSQL に渡されます。
- END-C と ENDP の間に記述された指令は、COBSQL を通じてプリコンパイラに渡されます。
- ENDP の後に記述された指令は COBOL コンパイラに渡されます。そのため、ENDP 指令を記述しない場合は、コンパイラ指令は COBOL コンパイラではなく COBSQL に渡されます。

12.2.1 指令の指定方法

COBSQL に渡す指令は、コンパイラ指令と同じように指定できます。ただし、COBSQL 指令の前には、必ず PREPROCESS"COBSQL" を記述する必要があります。
COBSQL 指令は、Net Express の標準指令ファイル (**cobol.dir**) に記述することもできます。

注:

- **cobol.dir** の各行には 1 つ以上のコンパイラ指令を指定できます。
- **cobol.dir** ファイルの各コンパイラ指令は 1 つの行に記述します。1 つの指令を複数行に分けて記述しないでください。
- **cobol.dir** ファイルで P(COBSQL) または PREPROCESS(COBSQL) が検出されると、コンパイラは ENDP に達するまで、その行の残りをプリプロセッサに渡します。
- プリプロセス文と、それ以降の ENDP までの全オプションは単一のコンパイラ指令として処理されます。そのため、プリプロセス文とすべてのオプションは、**cobol.dir** ファイルの単一行に記述する必要があります。

また、COBSQL 指令とプリコンパイラ指令を **cobsql.dir** ファイルに記述することもできます。このファイルは、現在のディレクトリまたは、$COBDIR で指定されたディレクトリに格納する必要があります。COBSQL は、現在のディレクトリか、または、$COBDIR で指定されたディレクトリで **cobsql.dir** ファイルを検索します。**cobsql.dir** ファイルが見つかった時点で、検索は終了します。**cobsql.dir** ファイルが現在のディレクトリに存在する場合には、$COBDIR で指定されたディレクトリは検索されません。

注:

- **cobsql.dir** の各行には 1 つ以上の COBSQL 指令を指定できます。
- **cobsql.dir** ファイルは COBOL コンパイラでは処理されないため、COBOL コンパイラ指令は指定しないでください。
- **cobsql.dir** では、各 COBSQL 指令は 1 つの行に記述します。1 つの指令を複数行に分けて記述しないでください。
- **cobsql.dir** ファイルで END-C、END、または END-COBSQL が検出されると、その行の残りをデータベースプリコンパイラに渡します。
- データベースプリコンパイラに渡すオプションは、すべて **cobsql.dir** の単一行に記述してください。

COBSQL は、**cobsql.dir** ファイルを処理し、各指令は「**ビルド設定**」ダイアログボックスで設定します。多くの指令は、直前に NO を記述すると逆の効果になります。たとえば、DISPLAY の逆の効果は NO DISPLAY です。次の指令リストでは、NO が使用できる指令にはアスタリスク (*) が付けられています。デフォルトでは、すべての指令が NO 付きで設定した状態です。

一部の指令は短縮名が指定できます。次の指令リストでは、該当する指令のすぐ下に短縮名が示されています。
また、COBOL コンパイラによって COBSQL に渡すことができる指令もあります (『*COBOL 指令*』を参照)。この方法を使用すると、使用頻度の高い指令を何度も指定する手間が省けます。COBOL コンパイラから取り込み可能な指令は、COBSQL 指令より前に処理されます。
次に例を示します。

```
cobol -v -k testprog p(cobsql) csqlt=ora makesyn end-c
    comp5=yes mode=ansi endp omf(gnt) list();
```

- `end-c` で終了する COBSQL 指令は、`csqlt=ora` と `makesyn` です。
- `endp` で終了するプリコンパイラ指令 (この場合は、Pro*COBOL) は、`comp5=yes` と `mode=ansi` です。
- コンパイラ指令は、`omf(gnt)` と `list()` です。

12.2.2 COBSQL 指令

次に、COBSQL 指令のリストを示します。

指令	説明
COBSQLTYPE CSQLT	使用するプリコンパイラ (ORACLE、ORACLE8、SYBASE、または INFORMIX-NEW) を指定します。たとえば、COBSQLTYPE=ORACLE と指定します。
CSTART* CST	COBSQL が実行時にデータベースサポートモジュールをロードします。
CSTOP* CSP	COBSQL が、アプリケーションが異常終了したときにロールバックを実行できる実行停止モジュールをロードします。
DEBUGFILE* DEB	デバッグファイル (**.deb**) を作成します。
DISPLAY* DIS	プリコンパイラの統計を表示します。この指令は、最初に COBSQL が正しくスタンドアローンのプリプロセッサを呼び出しているかを確認するときにのみ使用します。
END-COBSQL END-C END	COBSQL 指令の末尾を示します。残りの指令は、プリコンパイラに渡されます。
KEEPCBL	プリコンパイルされたソースファイル (**.cbl**) を保存します。
MAKESYN	すべての COMP ホスト変数を COMP-5 ホスト変数に変換します。MAKESYN と NOMAKESYN のどちらも設定されていない場合は (デフォルト)、COMP または COMP-4 として定義されている (ホスト変数のみではなく) すべての変数が COMP-5 に変換されます。
NOMAKESYN	COMP-5 変数および COMP-5 ホスト変数の変換を実行しません。
SQLDEBUG	Micro Focus 社が COBSQL のデバッグに使用するファイルを数多く作成します。これらのファイルには、プリコンパイラからの出力ファイル (通常は **.cbl** 拡張子)、プリコンパイラが作成するリストファイル (通常は **.lis** 拡張子)、そして **.sdb** 拡張子をもつ COBSQL デバッグファイルがあります。さらに KEEPCBL と TRACE をオンにします。
TRACE*	トレースファイル (**.trc**) を作成します。
VERBOSE	プログラムの処理中にすべてのプリコンパイラメッセージを表示し、ステータスを更新します。この指令は、最初に COBSQL が正しくスタンドアローンのプリプロセッサ

を呼び出しているかを確認するときのみに使用します。

12.2.3 COBOL 指令

次に、COBOL 指令のリストを示します。

指令	説明
BELL*	エラーが発生したときにベルを鳴らすかどうかを制御します。
BRIEF*	SQL エラー番号とともにエラーテキストを表示するかどうかを制御します。
CONFIRM*	受け付けられた指令、および拒否された指令を表示します。
LIST*	プリコンパイラリストファイル (.lis) を保存します。
WARNING*	報告する SQL エラーの最低重大度を指定します。

12.3 COBSQL アプリケーションのビルド方法

COBSQL アプリケーションの出荷パッケージに同梱すべきデータベースサポートモジュールは、このマニュアルでは説明していません。次の説明は、必要なすべてのサポートモジュールがエンドユーザのマシンにインストール済みで、マシンがデータベースサーバと通信できる状態に正しく設定されていることを仮定しています。
COBSQL アプリケーションをリンクする場合は、インポートライブラリ csqlsupp.lib を使用します。これにより、COBSQL で挿入された呼び出しが COBSQL の初期化モジュールと実行停止モジュールに変換されます。呼び出しの変換には csqlsupp.dll モジュールが使用されるため、COBSQL アプリケーションにはこのモジュールを同梱する必要があります。この汎用サポートモジュールは、Oracle と Sybase アプリケーションにも必要です。
アプリケーションのリンク時に、使用するライブラリの 1 つとして csqlsupp.lib を指定します。ライブラリ csqlsupp.lib は、NetExpress¥base¥lib ディレクトリに格納されています。また、csqlsupp.dll モジュールは NetExpress¥base¥bin ディレクトリに格納されています。
アプリケーションを主プログラム (.exe ファイル) と複数の DLL ファイルとして実装する場合には、CSTART または CSTOP COBSQL 指令でコンパイルしたモジュールのみに csqlsupp.lib をリンクする必要があります。
すべてのプログラムを CSTART または CSTOP でコンパイルした場合には、すべてのモジュールに csqlsupp.lib をリンクする必要があります。csqlsupp.lib をリンクしたモジュールは、本来のサイズよりわずかに大きくなります。アプリケーションの実行時には、単一の csqlsupp.lib のみがロードされます。
主プログラムのみを CSTART と CSTOP でコンパイルした場合には、主プログラムのみに csqlsupp.lib ライブラリをリンクする必要があります。つまり、CSTART または CSTOP COBSQL 指令でコンパイルしたプログラムのモジュールには、必ず csqlsupp.lib をリンクする必要があります。

12.4 CP プリプロセッサを使用したコピーファイルの展開

コピーファイルを操作するために COBOL 内で使用されるすべての方法が、データベースプリコンパイラで利用できるわけではありません。また、COBSQL 自体はインクルードされたコピーファイルを処理できません。この問題は、Micro Focus Copyfile Preprocessor (CP) を使用することによって解決できます。
CP は、他のプリプロセッサ (たとえば、COBSQL) にコピーファイルを処理する機構を提供するために開発されたプリプロセッサです。CP は、COBOL コンパイラと同じ規則でコピーファイルを処理するた

め、すべてのコピーファイル関連のコンパイラ指令は自動的に取り込まれ、COBCPY 環境変数を使用してコピーファイルが検索されます。次の文があります。

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

CP は、この文も展開します。CP の詳細については、Net Express のオンラインヘルプを参照してください (ヘルプファイルの索引で「CP」を選択します)。

Oracle では .pco および .cob の拡張子、Sybase では .pco および .cbl の拡張子、Informix では .eco、.cob、および .mf2 の拡張子を使用します。

Oracle および Sybase:

CP でコピーファイルを解決し、正しく文をインクルードするためには、Sybase と Oracle で次の COBOL コンパイラ指令を使用します。

```
copyext (pco,cbl,cpy,cob) osext(pco)
```

Informix:

Informix の場合は、次の指令を使用します。

```
copyext (eco,mf2,cbl,cpy,cob) osext(eco)
```

COBSQL は、データベースプリコンパイラの起動前に CP を呼び出し、コピーファイルを展開します。この時点ですべてのコピー関連コマンドが解決されるため、データベースプリコンパイラでは、単一のソースファイルのみを処理することになります。

CP は、アニメートの実行中にコピーファイルを表示できる利点があります。

CP は `INCLUDE SQLCA` 文を検出する場合に、次の処理を実行します。

- 現在のディレクトリ内で **sqlca.*ext*** ファイルを検索します。***ext*** は、OSEXT および COPYEXT コンパイラ指令が設定するコピーファイルの拡張子を示します。この拡張子は、デフォルトでは **.cbl** または **.cpy** です。
- COBCPY パスで **sqlca.*ext*** を検索します。
- サンプルの **sqlca.cpy** ファイルが、Net Express 基本インストールディレクトリの **source** ディレクトリに提供されています。データベースベンダが提供する SQLCA ファイルが COBCPY パスに存在しない場合には、この Micro Focus デモンストレーションバージョンの SQLCA が使用されます。ただし、このサンプルファイルをそのまま使用すると、作成したプログラムが正しく実行できない可能性があります。

注:ファイル **sqlca.cpy** を使用すると、プログラムの実行時にエラーが発生する可能性があります。

CP プリプロセッサの SY 指令を指定する場合は、SQLCA インクルードファイルの CP 展開を防止できます。次に使用例を示します。

```
preprocess"cobsql" preprocess"cp" sy endp
```

Sybase のプリコンパイラは SQLCA の展開機能を備えているため、Sybase のコードを処理する場合は、必ず CP の SY 指令を使用してください。

Oracle は、COMP 変数または COMP-5 変数のどちらでもコードを作成できるため、それぞれに対応するコピーファイルが 2 つ用意されています。標準の **sqlca.cob**、**oraca.cob**、および **sqlda.cob** のすべてに COMP データ項目が格納されています。**sqlca5.cob**、**oraca5.cob**、および **sqlda5.cob** には COMP-5 データ項目がそれぞれ格納されています。Oracle の comp5=yes 指令を使用する場合には、COBSQL の MAKESYN 指令を使用して SQLCA 内の COMP 項目を COMP-5 項目に変換する必要があります。

CP によるコピーファイル検索でエラーが発生した場合には、OSEXT および COPYEXT コンパイラ指令が正しく設定されているかどうかを確認してください。COPYEXT を先に設定し、最初のエントリとしてソースファイルの拡張子 (**.pco** や **.eco** など) を指定します。

コンパイラ指令の設定に問題がない場合は、コピーファイルが現在のディレクトリまたは COBCPY で指定されたディレクトリに存在することを確認します。

COBSQL と CP を組み合わせて使用すると、インクルードされたコピーファイル内の SQL エラーが正しく報告されます。CP を使用しない場合は行数が不正となり、エラーが表示されないか、または、正しくない行に表示されます。

12.5 各国語サポート (NLS)

COBSQL のエラーメッセージの表示に使用する言語は LANG 環境変数で指定します。NLS の詳細については、ヘルプファイルの索引で「**NLS**」を選択してください。LANG 環境変数の設定については「**LANG**」を選択します。
ほとんどのデータベースクライアントは、いくつかの NLS 機能を含んでいますが、LANG 環境変数の設定条件は COBOL システムの場合と異なります。そのため、環境変数のかわりに COBLANG を使用することをお奨めします。
COBLANG の設定は、COBOL システムのみに影響します。データベースクライアント側では LANG 変数を使用できます。COBLANG を正しく実行するためには、**mflang*nn*.lbr** (*nn* は COBLANG の設定値) が Net Express の **bin** ディレクトリで使用可能であることが必要です。たとえば、COBLANG=05 (英国 NLS メッセージ) にすると、**NetExpress¥Base¥Bin** ディレクトリに **mflang05.lbr** ファイルを作成します。
COBLANG の設定は、COBSQL のエラーメッセージのみに影響します。データベースプリコンパイラのエラーメッセージは、COBSQL で処理されないため、COBLANG の設定も反映されません。

12.6 例

ここでは、COBSQL を使用してプログラムをコンパイルする際に Net Express のコマンドプロンプトに入力するコマンドの例を、Oracle プリコンパイラと Sybase プリコンパイラについて説明します。

12.6.1 Oracle

```
cobol sample.pco anim nognt preprocess(cobsql)
   cstart cstop CSQLT=ORA end-c comp5=yes endp;
```

12.6.2 Sybase

```
cobol example1.pco confirm preprocess(cobsql)
   cstop csp cobsqltype=sybase preprocess(cp) sy endp;
```

12.7 トラブルシューティング

問題発生時には、次の各項目をチェックしてください。

- 基本的なネットワーク接続

  SQL 以前に、クライアントとサーバ間の通信状態をチェックします。TCP/IP ネットワークでは、ワークステーションとサーバとの間で双方向に ping コマンドを実行します。ホスト名が動作しない場合は、IP アドレスそのものを試してみます。PC の各種プロトコルを使用しているネットワークでは、ネットワークドライブのマウントやメッセージ送信を試みてください。

- SQL ネットワークソフトウェア

SQL ネットワークソフトウェアと通常のネットワークソフトウェア間で通信が正常に実行されているかどうかをチェックします。多くの SQL ベンダは、SQL ネットワークの設定を確認する ping ユーティリティを提供しています。

- SQL による対話テスト

  SQL ネットワークに問題がない場合は、SQL 文による対話テストを行います。多くの SQL ベンダは、キーボードから SQL を入力し、その結果を表示する簡易ユーティリティを提供しています。また、SQL 対話テストに使用できるサンプルデータベースも提供しています。

- スタンドアローンプリコンパイラ

  スタンドアローンプリコンパイラの動作を妥当性検査します。アイコンまたはコマンド行でプリコンパイラを起動し、COBOL コードが正しく生成されることを確認してください。正常に動作するサンプルアプリケーションが、プリコンパイラに提供されています。

- プリプロセス済みアプリケーション

  プリプロセス済みのアプリケーションが正しく動作をすることを確認します。展開後のプログラムを COBOL コンパイラで処理し、生成されたアプリケーションを実行します。

- 最小限の指令による COBSQL のテスト

  最小限の指令で COBSQL の機能をテストします。Net Express でテスト用プロジェクトを作成し、サンプルプログラムと同じディレクトリに SQLCA コピーファイルを格納した後、プリコンパイラを実行して動作をチェックします。

ここまでの対処をしても問題が発生する場合は、サポート窓口にお問い合わせください。サポート窓口で、問題の原因を突き止められるように、次の作業を行ってください。

- COBSQL 指令 SQLDEBUG を使用する。
- 次の内容を zip 形式でサポート窓口に送付する。
  - 元のソース
  - 展開されたソース (データベースプリコンパイラによって生成)
  - トレースファイル (拡張子が .trc のファイル)
  - データベースリストファイル (通常は拡張子が .lis のファイル)
  - コマンド行デバッグファイル (拡張子が .sdb のファイル)
  - プロジェクトファイル (プロジェクト名と同じ名前の拡張子が .app のファイル)
  - 使用した COBSQL 指令の設定

12.7.1 一般的な問題点

上記の各項目を確認しても原因を特定できない場合には、さらに次の項目を確認してください。

- 最新のバージョン

  使用するすべての製品が最新バージョンであることを確認します。

- COMP と COMP-5 の競合

ベンダ提供のマニュアルとサンプルアプリケーションを確認します。

- 構成

  PATH および他の環境変数の設定と構成ファイルの内容を確認します。

- 指令

  デフォルトでは、COBSQL はデータベースプリコンパイラに渡すコマンド行を表示しません。コマンド行を表示するには、SQLDEBUG 指令を設定します。コマンド行の表示は、プリコンパイラでコマンド行エラーが検出されたときに必要になります。コマンド行エラーの原因としては、プリコンパイラに不正な指令を渡した場合やプリコンパイラコマンド行の長さが上限を超えている場合などが考えられます。

- メモリ

  プリコンパイラが異常終了する場合には、COBSQL は次のエラーメッセージを表示することがあります。

  `* CSQL-F-021:` プリコンパイラは完了しませんでした。 `--` 終了します。

  オペレーティングシステムによるデータベースプリコンパイラの実行時に、必要なメモリ領域を確保できなかった可能性があります。

- 出力ファイルの不在

  プリコンパイラの出力ファイルが見つからない場合には、COBSQL は次のエラーメッセージを表示することがあります。プリコンパイラで出力ファイルが生成されなかった可能性があります。出力ファイルが生成されなかった原因として、プリコンパイラで重大なエラーが発生したことが考えられます。

  `* CSQL-E-024:` ファイル *filename* の `I/O` が検出されました
  `* CSQL-E-023:` ファイルステータス `3 / 5`

  filename は、データベースプリコンパイラが生成された出力ファイル名です。

- 展開後のソースファイルのデータ不足

  プリコンパイラが正常に実行され、COBSQL がエラーメッセージ「展開後のソースファイルのデータ不足」を報告する場合には、データベースプリコンパイラが生成したソースファイルと元のソースファイルとの間で、一部の行を対応付けできなかったことを示します。

  このメッセージは、プログラムに SQL 文が含まれていない場合にも表示されます。一般的に、データベースプリコンパイラは、SQL 文が見つからないと出力ファイルの作成を中断するため、このメッセージが表示されます。

12.7.2 Oracle を使用する際の考察事項

Oracle を使用する際に発生する問題をトラブルシューティングするときには、次の点に注意してください。

- 次に挙げる Oracle プリコンパイラのオプションは、Oracle アプリケーションの動作やメモリ消費に大きな影響を与えます。そのため、これらの指令の設定を変更する場合には、事前に Oracle のマニュアルに十分に目を通してください。
- DBMS
- HOLD_CURSOR
- MAXOPENCURSORS
- MODE
RELEASE_CURSOR

- Oracle プログラムでは、ALTER SESSION コマンドによって OPTIMIZER_GOAL を変更することが可能です。これによって、アプリケーションのパフォーマンスに重大な影響を与える場合があります。この構文を使用する前に、特定の SQL 文よりも一般的なアプリケーションの最適化を考慮してください。特定の文のパフォーマンスを変更する必要がある場合は、HINTS も使用できます。ALTER SESSION、OPTIMIZER_GOAL、および HINTS の詳細については、Oracle のマニュアルを参照してください。
- Oracle では、埋め込み SQL 文内に配列を使用してネットワークのアクセスを減少できます。これにより、アプリケーションで「バッチ」SQL コマンド (単一の SQL 文による複数行のデータ処理) を実行できます。詳細については、『*ホスト変数*』の章にある『*ホスト配列*』を参照してください。

  アプリケーションは通常、1 つの SQL 文で 1 行のデータを取り込みますが、配列を使用すれば複数行の取り込みが可能になります。Oracle のプリコンパイラには、配列による複数行の取り込み例を示すサンプルプログラム (通常は、**sample3.pco**) が付属しています。配列の詳細については、『***ORACLE プリコンパイラガイド Pro*Cobol サプリメント***』マニュアルを参照してください。

- Pro*Cobol による BINARY ホスト変数の扱いは、COMP の PICTURE 句で定義された変数と同様です。そのため、Pro*Cobol で comp5=yes 指令を使用する場合は、BINARY 変数を COMP-5 に変換できます。
- Pro*Cobol では、SQLCA コピーファイルが現在のディレクトリに存在する場合を除き、必ず INCLUDE 指令で同ファイルをインクルードする必要があります。これは、SQLCA コピーファイルの位置を知る必要があるためです。インクルードしない場合は、*filename*.pco ファイルにエラーメッセージが出力されます (*filename* は 8 文字の文字列です)。
- Pro*Cobol の実行時には、さまざまな有益な情報が生成されます。COBSQL VERBOSE 指令を使用すると、これらの情報の一部を COBSQL の実行時に表示できます。

  Pro*Cobol が生成する全情報を取り込むには、xref=yes 指令を使用します。この指令を Pro*Cobol 構成ファイル ***$ORACLE_HOME*¥PRO*xx*¥pcbcfg.cfg** に追加できます。詳細は、次のとおりです。

xx	Pro*COBOL のバージョン (たとえば、Oracle 8.0 の場合は PRO80)
$ORACLE_HOME	マシン上の Oracle インストール時のルートディレクトリ

- COBSQL DISPLAY 指令を使用すると、現在の Pro*Cobol 指令を画面に表示できます。指令に続いて Pro*Cobol で生成された統計情報が表示されます。
- COBOL LIST 指令を使用すると、COBSQL はプリコンパイラから収集された情報を COBOL チェッカーに渡し、COBOL リストの下部に表示します。

12.7.3 Oracle Pro*COBOL 8.x 以降を使用する際の考察事項

Pro*COBOL 8.0 のサポートが COBSQL に追加されました。

12.7.3.1 指令

Oracle 8 で COBSQL を使用する場合は、次の指令を使用してください。

指令	説明
CBL2ORA8 C28	Oracle 8 特定のサポートモジュール ora8prot および ora8lib の呼び出しを行います。これらのモジュールはどちらも csqlsupp.dll にビルドされます。
COBSQLTYPE CSQLT	EXEC SQL プリプロセッサ。ORACLE8 および ORA8 オプションを使用して、COBSQL とともに Pro*COBOL 8.x 以降を使用します。

12.7.3.2 移行

Pro*COBOL 1.x から 8.x 以降にプログラムを移行する場合は、次の点に注意してください。

- データ部での注意点
  - すべての SQL 文をピリオドで終了します。
  - ANSI 注釈のみサポートされます。どのカラムも *> で開始される注釈はサポートされていません。
- 入れ子プログラムのサポート

  挿入されたすべての変数を GLOBAL として定義します。EBCDIC から ASCII への変換をサポートする COBSQL によって挿入されたデータ項目が含まれます。

- Oracle 8 は、Oracle COMP から COMP-5 への変換で問題が発生する可能性があるため、宣言節を必要としません。

  Oracle 指令 DECLARE_SECTION=NO が設定されている場合 (デフォルト) は、COMP、BINARY、または COMP-4 のすべてのデータ項目が COMP-5 に変換されます。

  宣言節への項目の変換を制限するには、次のどちらかを設定します。

  - DECLARE_SECTION=YES または MODE=ANSI

    または

  - Pro*COBOL 指令 COMP5=NO および COBSQL 指令 MAKESYN
- Pro*COBOL 8.x 以降でサポートされる特別なデータ形式は、次のとおりです。
  - PACKED-DECIMAL

これらのデータ項目は、COMP-3 データ項目と同じ方法で扱われます。

- COMP-4
- PIC 9(4) COMP / COMP-4 / BINARY / COMP-5
- PIC 9(4) USAGE DISPLAY
- PIC s9(4) USAGE DISPLAY SIGN TRAILING
- PIC s9(4) USAGE DISPLAY SIGN TRAILING SEPARATE
- PIC s9(4) USAGE DISPLAY SIGN LEADING
- PIC s9(4) USAGE DISPLAY SIGN LEADING SEPARATE

この形式は、以前は Oracle の DISPLAY データ型としてサポートされていました。

12.7.3.3 Micro Focus COBOL

Pro*COBOL 8.x 以降では、次に示す Micro Focus COBOL の言語の拡張機能、データ定義および節見出しが拒否されます。

- すべてのポインタデータ形式
- レベル-88 の変数
- Value 句の定数
- 外部フォームとして定義された項目
- 局所場所節
- スレッド局所場所節

この問題を解決するためには、これらの項目を、Pro*COBOL によって開かれないコピーブックに配置する必要があります。ただし、Pro*COBOL の起動前にコピーブックを展開する CP を使用している場合には、動作しません。CP を呼び出してコピーブックを展開する htmlpp を使用する場合に、問題が発生する可能性があります。そのため、COBSQL の前に htmlpp を起動する必要があります。
たとえば、次のコンパイル行は動作します。

```
COBOL -k PROG P(HTMLPP) PREPROCESS(COBSQL) CSQLT=ORACLE8
```

一方、次の行は動作しません。

```
COBOL -k PROG PREPROCESS(COBSQL) CSQLT=ORACLE8 P(HTMLPP)
```

12.7.4 Sybase を使用する際の考察事項

Sybase を使用する前には、Sybase に付属のコピーファイル cobpub.cbl と sybhesql.cbl ですべての COMP データ項目を COMP-5 に変更する必要があります。
Sybase を使用する際に発生する問題をトラブルシューティングするときには、次の点に注意してください。

- COBSQL で CP を使用しないでコピーファイルを検索する場合は、Sybase プリコンパイラが生成するコピーファイルが完全修飾されていることが必要です。Sybase プリコンパイラは、拡張子の検索を行いません。
- クライアントでのメッセージの報告に使用する正しい言語を検索するときに、Sybase プリコンパイラに問題がある場合は、Sybase のファイル locales.dat が正しく設定されているか確認してください。

  クライアントのオペレーティングシステムがデフォルト設定で構成されていて、Sybase で各国語サポートエラーが報告される場合は、LANG 環境変数を使用して、locales.dat ファイルの設定を上書きしてください。

たとえば、Windows NT クライアントで問題が発生し、locales.dat ファイルで次のような Windows NT の設定が行われている場合の例を説明します。

```
[NT]
locale = default, us_english, iso_1
locale = enu, us_english, iso_1
locale = fra, french, iso_1
locale = deu, german, iso_1
```

この場合に、英語用に LANG を設定するには、次のようにします。

```
LANG=enu
```

- Sybase エラーメッセージを識別することが困難な場合があります。COBSQL でそれらのエラーメッセージを識別するには、esql.loc ファイルを変更します。テキストエディタを使用して esql.loc ファイルを編集し、メッセージのレイアウトを次のように変更します。

  SYB-*number-type-text*

  パラメータの内容は、次のとおりです。

SYB-	変更された Sybase エラーメッセージであることを COBSQL に示す文字列。
number-	Sybase エラーに割り当てられる 4 桁の一意のエラー番号。
type-	エラーの重大度を示します。Sybase メッセージには、一般的エラーや致命的エラーではなく、警告のみのエラーもあります。
text	Sybase のエラーメッセージ。

  たとえば、esql.loc ファイルの一般的なエントリは、次のようになります。

  9 = M_PRECLINE, "警告 %1 行目のクエリーの確認中に誤りを検出しました。"

  これは、次のように変更されます。

  9 = M_PRECLINE, "SYB-2009-警告-%1 行目のクエリーの確認中に誤りを検出しました。"

  esql.loc ファイルは変更する前にコピーしておくことをお奨めします。変更バージョンを使用すると、COBSQL であらゆる Sybase エラーメッセージを検出できます。

esql.loc の位置は、使用する言語やコードページによって異なります。これは、locales.dat ファイルで定義されます。AIX プラットフォームのデフォルト言語の定義が次のような場合の例を説明します。

```
[aix]
locale = C, us_english, iso_1
locale = En_US, us_english, iso_1
locale = en_US, us_english, iso_1
locale = default, us_english, iso_1
```

このデフォルトの言語は、iso_1 コードページを使用した us_english なので、使用する esql.loc のコピーは次のようになります。

```
/sybase home/locales/messages/us_english/iso_1/esql.loc
```

sybase home は、Sybase クライアントのインストール先ディレクトリです。

Sybase でのさまざまなエラーメッセージファイルの使用および検索方法については、Sybase の『*Client Reference Manual*』を参照してください。

- Sybping を異なる Sybase サーバで使用する場合は、各サーバに定義されたサービスが必要になります。Sybase 製品の sqledit には、必要な情報の入力方法が示されます。
- Sybase への接続時に、サーバの名前が指定されていない場合は、Sybase により DSQUERY という名前の環境変数が検索されます。これは、プロジェクトの環境変数セクション内で設定できます。設定する場合は、異なるプロジェクトが異なる Sybase サーバに接続できるようになります。
- Sybase プリコンパイラの実行時には、さまざまな有益な情報が生成されます。COBSQL VERBOSE 指令を設定する場合は、これらの情報の一部を COBSQL の実行時に表示できます。
- COBSQL の DISPLAY 指令を使用する場合は、Sybase プリコンパイラによって作成された統計情報が画面に表示されます。
- COBOL LIST 指令を使用する場合には、COBSQL はプリコンパイラから収集された情報を COBOL チェッカーに渡し、COBOL リストの下部に表示します。
- Sybase プリコンパイラは、SQL 文を含まないプログラムを受け入れ、出力ファイルを作成します。つまり、プロジェクト内のプログラムはすべて正常にコンパイルされます。ただし、これはまた、Sybase プリコンパイラが、すべての作業場所節項目と、Sybase SQL 文を含む COBOL サポートコードを Sybase 以外のプログラムに挿入することも意味します。これによってプログラムがより大きくなり、パフォーマンスに影響を及ぼす場合があります。
- Windows NT で Sybase System 11 を使用する場合は、Sybase System 11 Server に別の Sybase COBOL プリコンパイラを使用できます。このため、Sybase プリコンパイラのインストール時には注意が必要です。Open Client のサポートファイルが独自のバージョンで提供されるため、これらの 2 つの製品のサポートファイルのバージョンが異なると、問題が発生する可能性があります。このような状況の場合は、Sybase System 11 COBOL プリコンパイラをインストールする前に、Sybase のサポート窓口にお問い合わせすることをお奨めします。
- サーバがインストールされているマシンに Sybase をインストールする場合 (またはその逆の場合) は、特に注意が必要です。Sybase クライアントのみをインストールする場合は、問題は発生しません。

12.7.5 Informix を使用する際の考察事項

- Informix は、eco、cob、および mf2 のファイル拡張子を使用します。CP でコピーブックを解決し、正しく文をインクルードするには、次の COBOL コンパイラ 指令を使用します。

  ```
  copyext(eco,mf2,cob,cpy,cbl) osext(eco)
  ```

- UNIX では、COBSQL が Informix プリコンパイラを起動するため、COBSQL を使用する前に、INFORMIXCOB、INFORMIXCOBTYPE、および INFORMIXCOBDIR のすべての環境

変数を設定する必要があります。これらの環境変数の詳細については、『*Informix COBOL/ESQL Programmers Guide*』を参照してください。

- Informix は、リストファイルのみにエラーメッセージを生成します。通常、エラーメッセージにはエラーが発生した行が示されます。COBSQL がエラーメッセージから行番号を特定できない場合は、エラーは報告されません。

- COBSQL を Informix で正しく実行するためには、COBSQL を使用する前に、INFORMIXDIR 環境変数を設定する必要があります。