

# Micro Focus® Net Express®

## データベースアクセス

第 7 版  
2006 年 1 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。

ここでは、SQL に関する知識があることを前提としています。また、必要なクライアントソフトウェアの構成方法を含め、使用するリレーショナルデータベースに精通していることを前提としています。

はじめに



# 1: はじめに

ここでは、データベースアクセス機能の概要を説明します。

注：

ここでは、SQL 構文、返されるエラーメッセージ、または COBOL 環境以外の SQL の使用方法については説明しません。これらの詳細は、各データベースベンダが提供しているマニュアルを参照してください。

## 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により、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。次のプリプロセッサが使用できます。

Net Express に付属のプリプロセッサの 1 つを使用できます。

- OpenESQL (ODBC 3.0 準拠のドライバを使用する場合)
- DB2 ECM (DB2 データベースを使用する場合)
- COBSQL (リレーショナルデータベースベンダが提供している COBOL プリプロセッサを使用する場合)

COBSQL なしのデータベースベンダが提供しているプリプロセッサを使用できません。この場合は、COBSQL を使用するときに行うようにデータベースソフトウェアで COBOL 環境を作成する必要があります。詳細は、『COBSQL』の章にある『準備』の節を参照してください。

以降では、Net Express に付属の 3 つのプリプロセッサを説明します。

## OpenESQL

OpenESQL は、ODBC が使用可能なデータソースにアクセスする COBOL アプリケーションで埋め込み SQL を利用できる統合プリプロセッサです。

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。

Server Express の OpenESQL は、Micro Focus Net Express を使用して開発したアプリケーションと互換性があります。そのため、Net Express の OpenESQL アシスタントを使用して開発したアプリケーションは、簡単に UNIX プラットフォームに移植できます。

アプリケーションが異なるリレーショナルデータベースシステムで設計されている場合、ま

または、次回に採用するリレーショナルデータベースシステムが決定していないような場合は、OpenESQL を使用します。

OpenESQL の詳細は、『[OpenESQL](#)』の章を参照してください。

## Oracle

Oracle データベースを使用する場合、または、SQL 指令 TARGETDB に ORACLEOCI を設定してコンパイルする場合は、アプリケーションは実行時に ODBC 呼び出しではなく OCI 呼び出しを行います。そのため、そのアプリケーションは ODBC ドライバを必要としません。詳細は、『[OpenESQL](#)』の章にある『[Oracle OCI サポート](#)』の節を参照してください。

## DB2 ECM

DB2 ECM (external checker module; 外部チェッカーモジュール) は、このシステム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 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 の詳細は、『[DB2](#)』の章を参照してください。

## 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。

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。

COBSQL の詳細は、『[COBSQL](#)』の章を参照してください。

---

注：COBSQL プリプロセッサは、標準的な手続き型 COBOL プログラムのみサポートされません。オブジェクト指向 COBOL の構文 (OO プログラム) と入れ子を使用するプログラムでは、COBSQL を使用できません。

---

## 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。

はじめに

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

完全な構文の説明は、その使用例とともに、各埋め込み SQL 文の[オンラインヘルプ](#)に記載されています。

## 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。

カーソル名、文名、および接続名の大文字小文字は、変数が宣言されたときに使用したものと一致する必要があります。たとえば、C1 というカーソルを宣言する場合には、常に C1 (c1 ではなく) で参照しなければなりません。

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。

## 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。

## 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

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

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

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。

## 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、『[CGI ベースのアプリケーション](#)』マニュアルを参照してください。

## 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。

### 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。

アプリケーション内では、カーソル名は一意である必要があります。

## COBSQL

COBSQL では、INIT 指令を複数回指定すると、2回目以降の指令が無視されます。

---

Copyright © 2006 Micro Focus (IP) Ltd. All rights reserved.

---

ホスト変数



## 2: ホスト変数

ここでは、ホスト変数およびホスト変数とデータベースとの値の受け渡しの方法について説明します。

### 概要

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
```

### 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。ただし、集団項目は WHERE 句では使用できません。

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は、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。

---

## 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。

## 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：

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。

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には、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。

## 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、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。

---

## 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。

## 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。

## 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.
```

## 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
```



ホスト変数

```
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。

---

Copyright © 2006 Micro Focus (IP) Ltd. All rights reserved.

---

 はじめに

データ型 

## 3: 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タ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。

### 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。

### 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。

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
```



## データ型

```
VAR emp-comm IS DISPLAY(8,2)
END-EXEC.
EXEC SQL
    END DECLARE SECTION
END-EXEC.
```

## 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。

## 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は、『Programming with INFORMIX-ESQL/COBOL』マニュアルを参照してください。

## 整数データ型

### 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(2) 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。

## OpenESQL

SQL(DBMAN=ODBC) 指令でコンパイルする場合は、OpenESQL では TINYINT データ型をサポートしていません。かわりに SMALLINT を使用してください。

## 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。

## 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、整数型のホスト変数を 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、整数型のホスト変数を 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。

## 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せん。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。

# 文字データ型

## 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。

## VARCHAR

### 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。

### 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 通りで宣言できません。

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

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。

### 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。

### 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 データ項目の詳細は、『Informix SQL ガイド』のマニュアルを参照してください。

## Unicode

OpenESQL では、PIC N(n) フィールドを使用して Unicode データ型をサポートします。詳細は、『OpenESQL』の章にある『[OpenESQL の Unicode サポート](#)』およびヘルプトピックの『[SQL/COBOL データ型マッピング](#)』を参照してください。

## 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。

## 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。

## 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
```

## OpenESQL

OpenESQL は、どのリレーショ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 {dyyyy-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 {thh: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 {tsyyyy-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
```

## DB2

DB2 では、TIMESTAMP データ型の最大長は 26 文字です。

## COBSQL

### 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されています。

## 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 関数の詳細は、Sybase の『SQL Server Reference Manual: Volume 1 Commands, Functions and Topics』を参照してください。

## 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。

## バイナリデータ型

### 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。

### 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いる場合は、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。

### COBSQL

#### 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。詳細は、Oracle のマニュアルを参照してください。

## 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 として定義できません。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。

---

## 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 データ項目の詳細は、『Informix SQL ガイド』を参照してください。

## 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

---

## BINARY

構文：

```
SQL [TYPE] [IS] BINARY(n)
```

例：

```
01 hv-name SQL TYPE IS BINARY(n)
```

生成後

```
01 hv-name pic x(n).
```

---

## CHAR-VARYING

構文：

```
SQL [TYPE] [IS] CHAR-VARYING(n)
```

一般規則：

1. CHAR-VARYING データは、OpenESQL に SQL\_VARCHAR として渡されます。
2. 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ます。
3. 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。

例：

```
01 hv-name SQL TYPE IS CHAR-VARYING(n)
```

生成後

```
01 hv-name pic x(n).
```

---

## DATE

構文：

```
SQL [TYPE] [IS] DATE
```

一般規則：

1. DATE データは、YYYY-MM-DD の形式である必要があります。

例：

```
01 hv-name SQL TYPE IS DATE
```

生成後

```
01 hv-name pic x(10).
```

---

## DATE-RECORD

構文：

```
SQL [TYPE] [IS] DATE-RECORD
```

例：

```
01 hv-name SQL TYPE IS DATE-RECORD
```

生成後

```
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 および TIMESTAMP-RECORD

OpenESQL は、TIMESTAMP カラムデータをより簡単に処理できるように TIMESTAMP および TIMESTAMP-RECORD SQL TYPE をサポートします。TIMESTAMP および TIMESTAMP-RECORD は、固定された日付時刻形式に編成されたデータが必要になります。すべてのタイムスタンプ情報を含む作業場所内の単一のデータ項目として生成するために TIMESTAMP を使用します。次に示すタイムスタンプの各要素用のデータ項目を含む集団項目として生成するために TIMESTAMP-RECORD を使用します。

年  
月  
日  
時  
分  
秒  
小数部の秒

構文：

### 書き方 1 - TIMESTAMP

```
SQL [TYPE] [IS] TIMESTAMP
```

## 書き方 2 - TIMESTAMP-RECORD

SQL [TYPE] [IS] TIMESTAMP-RECORD

一般規則：

1. TIMESTAMP データは、YYYY-MM-DD HH:MM:SS の形式である必要があります。
2. TIMESTAMP-RECORD ホスト変数を使用してデータを挿入するには、生成されたフィールド名内で有効なデータを渡す必要があります。
3. 小数部の秒は、9 桁までサポートされます。ただし、この値は、DBMS や ODBC ドライバによって異なります。たとえば、Oracle および DB2 UDB では、6 桁までサポートされます。Microsoft SQL Server では、3 桁までです。詳細は、DBMS または ODBC ドライバのマニュアルを参照してください。
4. 小数部のデータは左桁寄せで渡され、データ項目の定義された桁数に含まれている必要があります。たとえば、678 の小数部の秒値は、9 桁で定義されたデータ項目に 678000000 として渡されます。
5. SELECT 文または FETCH 文からデータが戻される場合は、小数部の秒は右桁寄せになります。
6. Oracle を使用している場合は、国や地域によって日付時刻形式が異なるため、Oracle の NLS\_TIMESTAMP\_FORMAT パラメータで定義します。OpenESQL で定義された形式は、NLS\_TIMESTAMP\_FORMAT で指定された値と一致する必要があります。

---

注：OpenESQL 形式と Oracle の NLS\_TIMESTAMP\_FORMAT パラメータ値と一致しない場合は、Oracle はエラーメッセージを生成します。

NLS\_TIMESTAMP\_FORMAT の値を確認するためには、Oracle の SQLPLUS ユーティリティを使用します。具体的な操作は次のとおりです。

- OpenESQL で定義している値を NLS\_TIMESTAMP\_FORMAT 環境変数に設定します。
- PIC X(29) のホスト変数を準備し、NLS\_TIMESTAMP\_FORMAT で定義された形式内の文字列を渡します。
- 適切な形式を指定するために INSERT 文の TO\_TIMESTAMP 関数を使用します。次に例を示します。

```
exec sql
  insert into mf_datetime
    (col_a
     ,col_date
     ,col_timestamp
    ) values
    (:mf-col-a
```



```

        ,TO_DATE(:mf-col-date, 'YYYY-MM-DD')
        ,TO_TIMESTAMP(:mf-col-timestamp, 'YYYY-MM-DD HH24:
MI.SS.FF')
    )
end-exec

```

SQLPLUS ユーティリティおよび NLS\_TIMESTAMP\_FORMAT パラメータの詳細は、Oracle のマニュアルを参照してください。

7. SQL Server が日付や時刻の値を格納する場合は、桁によって小数部の最終桁の上下で丸めが発生する可能性があります。次に例を示します。

渡される値	SQL Server から返される値
01/01/98 23:59.59.999	1998-01-02 00:00:00.000
01/01/98 23:59.59.995	1998-01-01 23:59:59.997
01/01/98 23:59.59.996	1998-01-01 23:59:59.997
01/01/98 23:59.59.997	1998-01-01 23:59:59.997
01/01/98 23:59.59.998	1998-01-01 23:59:59.997
01/01/98 23:59.59.992	1998-01-01 23:59:59.993
01/01/98 23:59.59.993	1998-01-01 23:59:59.993
01/01/98 23:59.59.994	1998-01-01 23:59:59.993
01/01/98 23:59.59.990	1998-01-01 23:59:59.990
01/01/98 23:59.59.991	1998-01-01 23:59:59.990

例：

#### 書き方 1 - TIMESTAMP

```
01 hv-name SQL TYPE IS TIMESTAMP
```

生成後

```
01 hv-name pic x(29).
```

#### 書き方 2 - TIMESTAMP-RECORD

```
01 hv-name SQL TYPE IS TIMESTAMP-RECORD
```

生成後

```

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.
```

## 書き方 1 および 2

### ホスト変数の定義：

```
01 mf.  
    03 mf-col-a                pic s9(09)  comp-5.  
    03 mf-col-date            sql type date.  
    03 mf-col-timestamp       sql type timestamp.  
    03 mf-col-tsrec           sql type timestamp-record.  
01 ws-char-ts                pic x(29).
```

### フィールドの初期化：

```
move 1                to          mf-col-a  
move "2005-03-31"     to          mf-col-date  
move "2005-04-15 13:45:56.456123"  
    to                mf-col-timestamp  
move 2005             to          mf-col-tsrec-year  
move 04               to          mf-col-tsrec-month  
move 16               to          mf-col-tsrec-day  
move 16               to          mf-col-tsrec-hour  
move 55               to          mf-col-tsrec-min  
move 58               to          mf-col-tsrec-sec  
move 678000000       to          mf-col-tsrec-frac
```

### INSERT 文：

```
exec sql  
    insert into mf_datetime  
        (col_a  
         ,col_date  
         ,col_timestamp  
        ) values  
        (:mf-col-a  
         ,:mf-col-date  
         ,:mf-col-timestamp  
        )  
end-exec
```

## 4: カーソル

ここでは、カーソルの使用方法について説明します。

### 概要

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

```

## 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 文が実行されません。

```

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。

## オブジェクト指向構文

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場所節で宣言する必要があります。

---

注：

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。

---

## 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が含まれる場合には、パラメータマーカ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。

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。

## 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は、『[カーソルオプション - OpenESQL](#)』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の詳細は、『[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。

## カーソルのクローズ

カーソルの使用の終了後、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の詳細は、『[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。

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。

## 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。

#### SET SCROLLOPTION

SET SCROLLOPTION 文は、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。並行制御を有効にするために、カーソルをオープンする前に SET CONCURRENCY 文を使用します。

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。

## 位置づけ UPDATE 文と DELETE 文

位置づけ UPDATE 文と DELETE 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。

## COBSQL

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。

# 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。

---

---

## 5: データ構造体

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。

### 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 SQLCA (SQL communications area; SQL 通信領域) に返されます。

Oracle、Sybase、および Informix では、SQLCA のバージョンがそれぞれ異なります。Oracle、Sybase、および Informix の SQLCA にはどれも、SQLCODE、SQLERRML、SQLERRMC、および SQLWARN フィールドが存在します。フィールドのサイズや位置は、プリコンパイラの種類によって異なる場合があります。

SQLCA データ構造体のレイアウトの詳細は、ヘルプトピックの『[SQLCA データ構造体](#)』を参照してください。

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。

### 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 値の詳細は、ヘルプトピックの『[SQLCODE 値](#)』を参照してください。

#### 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。Oracle 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。

## SQLSTATE 変数

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 値の詳細は、ヘルプトピックの『[SQLSTATE 値](#)』を参照してください。

## 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。

## 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

シヨンで SQLWARN フラグの値を確認する必要があります。

W - 警告が生成されました。

空白文字 - 警告は生成されませんでした。

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、ヘルプトピックの『[SQLCA データ構造体](#)』を参照してください。

## 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)

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".
```

## 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。

## 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、次の 2 つの部分で構成されています。

SQLERRML - エラーメッセージの長さが保持されます

SQLERRMC - 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.
```

## 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。



## DB2

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。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。

### SQLERRD(3)

DB2 では、SQLERRD(3) に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)

DB2 では、SQLERRD(4) に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、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)

DB2 では、SQLERRD(5) に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。

## 記述子領域 (SQLDA)

SQLDA (SQL descriptor area; SQL 記述子領域) は、プリコンパイラごとに一意です。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 で使用される SQLDA は、Oracle SQLDA と互換性がありません。

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は、ホスト変数ではなく、SQL 記述子領域 (SQLDA) を使用します。

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。詳細情報には、カラム名、データ型、長さ、または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が含まれます。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) を使用することもできます。

静的 SQL 文の SQLDA は使用できませんが、カーソル FETCH 文の SQLDA は使用できます。

## COBSQL

Oracle、Sybase、および Informix では、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。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。そのため、次の構文を使用してプログラムにインクルードできません。

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

## Oracle

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 コピーファイルを使用するには、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。

Oracle では SQLDA が提供されません。SQLDA に関する内容や ORACA コピーファイルの詳細は、『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。

## Sybase

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。

## Informix

Informix では SQLDA コピーファイルは提供されません。Informix プリコンパイラのマニュアルでは、Informix の動的 SQL を使用するために定義するデータ項目のレイアウトが説明されています。

## OpenESQL

SQLCA 構造体は、システムの基本インストールディレクトリにある source ディレクトリ内の sqlca.cpy ファイルで定義されています。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 の詳細は、ヘルプトピックの『[SQLDA データ構造体](#)』を参照してください。

## 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。 32 ビット - $\text{sqln} * 44 + 16$ として計算されます。 64 ビット - $\text{sqln} * 56 + 16$ として計算されます。

## 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。

## FETCH 文

SQLDA 構造体を使用して FETCH 文を実行するには、次の処理を実行します。

1. 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。
2. アプリケーションは、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)。
3. 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。

## 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。

## 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
```



## 6: 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 文と呼びます。

### 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
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。

#### 文を 1 回実行する

このタイプの動的 SQL 文では、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。

#### 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。

#### 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。

## 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。変数の型と個数の両方、または一方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。

## 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

### プレースホルダ

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 Vn 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。その例を次に示します。

```
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。

### PREPARE 文の物理的な位置

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。

## 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。

## EXECUTE IMMEDIATE 文

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
```

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。

## 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
```

## 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 文の詳細は、『[カーソル](#)』の章を参照してください

## 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 の標準的なストアー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 :param1, :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 :param1, :param2
end-exec
```

## 例

次に、データソース「SQLServer 2000」を使用してストアードプロシージャ「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 MFSQLMESSAGE TEXT 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 dsq12 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
```

```
*> stop run
```

```
exit.
```

## 7: 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。

### 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 (data source name; DSN) の設定

#### 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 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。

DSN の割り当て方法の詳細は、「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。

---

注：64 ビットバージョンの Windows 上で実行している場合に、32 ビットアプリケーション用の DSN を作成する場合は、32 ビットバージョンの ODBC データソースアドミニストレータユーティリティを実行する必要があります。このユーティリティを実行するコマンドは `odbcad32` です。odbcad32.exe ファイルは、`windowsdir¥syswow64` フォルダ内にあります。

---

### 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。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 (UNIX は除く) の WITH PROMPT パラメータ

SQL (CONNECTIONPOOL) 指令

## COBOL プログラムの移行方法

メインフレームから移行する場合には、有効な移行ツールとして OpenESQL を使用できます。OpenESQL は、わかりやすい移行機能を提供し、しかも効率的です。OpenESQL の機能を次に示します。

複合指令である BEHAVIOR は、COBOL 環境で定義されます。BEHAVIOR は、MAINFRAME と ANSI のどちらかが定義されます。どちらの値でも、OpenESQL が次に示す内容を決定します。

- 不定 COBOL カーソル (明示的に読み取り専用または更新可能としてカーソルが定義されていない) の最適な DBMS カーソル
- COBOL カーソル (読み取り専用および更新可能) の最適な DBMS カーソル選択
- 適切な分離レベル

プログラムに関連付けられた EXEC SQL 文の変更なしで、上記に挙げた BEHAVIOR 特性を上書きします。

トレース指令である TRACELEVEL は、特定のアプリケーションで使用される DBMS カーソルのタイプを示し、パフォーマンスと効率を相対的に報告する実行時統計を提供します。

詳細は、『SQL コンパイラ指令』の節にある [BEHAVIOR](#) と [TRACELEVEL](#)、『[デモンストレーションアプリケーション](#)』の節にあるデモアプリケーション behavior.app を参照してください。



ださい。

## SQL コンパイラ指令

埋め込み SQL 文を含むプログラムをコンパイルする場合には、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。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。

他のコンパイラ指令が指定できるところであればどこにでも SQL 指令を指定できます。指定できる箇所の例を次に示します。

\$SET 文

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

Net Express IDE の「詳細指令」画面使用

コマンド行

```
cob -V testconn.cbl -C"anim SQL(DBMAN==ODBC)"
```

IDE から SQL 指令を設定する方法の詳細は、ヘルプトピックの『[SQL コンパイラ指令オプションの設定](#)』を参照してください。

注：複数の SQL 指令を指定できますが、お奨めしません。複数の SQL 指令オプションを指定すると、特定のオプションが原因で競合を起こし、意図しないキャンセルや上書きが発生します。

## SQL コンパイラ指令オプション

SQL コンパイラ指令は、オプションのセットで構成されています。詳細は、ヘルプトピックの『[SQL コンパイラ指令](#)』を参照してください。使用可能なオプションの詳細は、ヘルプトピックの『[SQL コンパイラ指令オプション](#)』を参照してください。

## 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お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 SQL 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)	<b>ecmtrace.txt</b>	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 コードです。このファイルは、OpenESQL ODBC プリコンパイラからの出力ファイルと等価です。

CHKECM(TRACE)	<b>ecmtrace.txt</b>	ODBC ECM とコンパイラの間で受け渡される情報の詳細。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)	<b>sqltrace.txt</b>	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。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	<b><i>program-name.lst</i></b>	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。

## 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、次のどちらかに該当する場合は CONNECT 文を使用します。

- 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さまざまなデータソースにアクセスする。
- プログラムが複数のデータベースにアクセスする。

### 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、アプリケーション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。

## 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の詳細は、ヘルプトピックの『[キーワード](#)』を参照してください。

## 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](#)』の節を参照)。アプリケーションを使用して、ODBC データソースを設定します。

Net Express の基本インストールディレクトリのディレクトリ source には、sqlca.cpy と sqllda.cpy のコピーファイルが提供されており、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適切なファイルが存在することを確認してください。

環境	ファイル
32 ビット	<b>odbcrw32.dll</b>
64 ビット	<b>odbcrw64.dll</b>

## デモンストレーションアプリケーション

システムの基本インストールディレクトリにある `openesql` 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

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があります。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。

次のアプリケーションが用意されています。

#### behavior.app

データソース名 LocalServer に接続するには、Microsoft SQL Server ODBC ドライバを使用します。これを行うには、ヘルプトピックの『[ODBC データソース名の設定](#)』を参照してください。

behavior プログラムは、behavior.cbl と behavsub.cbl によって、表を作成し格納します。表の作成後に、BEHAVIOR 指令が同じ不定 COBOL カーソル宣言である、読み取り専用 (behavior.cbl) と 更新可能 (behavsub.cbl) をどのように作成するかを確認します。

これは、プロジェクトディレクトリのトレースファイル OpenESQLTrace.processID.log で検証できます。ログファイルは、behavior.cbl の TRACELEVEL 指令で生成されます。

#### 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。

#### 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。テスト順序は次のとおりです。

1. 接続
2. テストテーブルの削除
3. テストテーブルの作成
4. 行の挿入
5. コミット
6. 行の更新
7. 読み取りと検証
8. ロールバック
9. 読み取りと検証
10. テストテーブルの削除
11. 接続の解除
12. 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。

## 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。

---

## データ型

ヘルプトピックの『[SQL/COBOL データ型マッピング](#)』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。

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。

日付 {d 'yyyy-mm-dd'}

時刻 {t 'hh:mm:ss'}

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

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.
```

または、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 )
```

## SQLCA の使用方法

SQLCA データ構造体は、システムの基本インストールディレクトリ内の source ディレクトリにある 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。

---

## 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：ストアー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ー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。

ストアー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 ストアー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的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ー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 storedProcecedureCall
```

この場合には、ストアー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。ストアー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。ストアー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)。

## 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、次のディレクトリに格納されています。

Examples¥Net Express IDE¥odbcsql

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。

## 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 MFSQLMESSAGE TEXT 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.

## 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 を使用するときに行う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。

### 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。

---

## 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。

## 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、CONNECT 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。

## 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があります。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。

## 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、MS 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. cbldsqlwrapper
        inherits from olebase.
object section.
class-control.
        cbldsqlwrapper is class "cbldsqlwrapper"
*> 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 theInterfaceId    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 theInterfaceId
                        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 theInterfaceId
    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>Path - ライブラリはこの（完全なパスの）  
位置にあります。

```

*>-----
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 ESQAction         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
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-ReportSQLExceptionAndExit
```

```
end-if
```

```
    .
```

C-SetAutoCommitOff.

```
EXEC SQL
```

```
    SET AUTOCOMMIT OFF
```

```
END-EXEC
```

```
if sqlcode zero
```

```
    move z"Set Autocommit Off failed " to ESQLAction
```

```
    perform Z-ReportSQLExceptionAndExit
```

```
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-ReportSQLExceptionAndExit
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-ReportSQLExceptionAndExit
end-if

```

```

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

X-Rollback.

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

Y-Disconnect.

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

Z-ReportSQLExceptionAndExit.

```
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。

## 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。

## 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。

---

## 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
```

---

注：アプリケーションが静的 SQL を使用している場合は、標準データ型を返します。ただし、アプリケーションが動的 SQL を使用している場合は、上記の新しいデータ型を返します。

---

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 INCLUDE 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  
 \* 影響を受ける行の数。

```

    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
* stop run
  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。

---

Copyright © 2006 Micro Focus (IP) Ltd. All rights reserved.

---

## 8: OpenESQL アシスタント

ここでは、OpenESQL アシスタントの使用と構成について説明します。

### 概要

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

COBOL プログラムへの SQL クエリーの挿入

COBOL プログラムへの補助コードの作成と挿入

### OpenESQL アシスタントオプションの設定方法

OpenESQL アシスタントを使用して SQL クエリーを作成する前に、OpenESQL アシスタントの動作を決定するオプションと SQL の生成方法を指定します。これらは、「OpenESQL アシスタント構成オプション」ダイアログボックスで行います。このダイアログボックスでは、次の設定ができます。

オーナー名を使用してツリービューやクエリー内の表名で修飾

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、ツリービュー内のすべてのテーブルが表示されます。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、ツリービューで表示されるテーブル名の後ろにかっこで囲まれたオーナー名のテーブル名を設定できます。クエリー処理も、オーナー名で修飾されたテーブル名を返します。

テーブル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われません。すべてのテーブル名とカラム名を引用符識別子で囲むことができます。

SQLCODE の確認のかわりに SQLSTATE の確認を使用

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 文を生成するように変更できます。

Web サービス用のインターフェイスコードを生成



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 文も作成できます。

テーブル名の順序でツリービュー内に項目を一覧表示

デフォルトでは、ツリービュー内に一覧表示される項目の順序は、システムカタログから返される順序です。これを、テーブル名の順序で一覧表示できます。この順序にすると、項目の一覧表示にはオーナー名は考慮されません。

大文字、小文字、または混在文字でコードを生成

デフォルトでは、SQL クエリーやコピーブックは大文字と小文字の組み合わせが使用されます。すべて小文字または大文字で生成することもできます。

結果行の最大数を設定

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。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。そのため、この返される行の最大数を指定できます。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 を使用

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。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できます。

ホスト変数に SQL TYPE を使用

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ます。これにより、SQL プリコンパイラは、ホスト変数の用途をより的確に判断できます。以前の方法でも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にも生成できます。

3 つの方法の 1 つを使用して、コピーブックの構造体名を生成

デフォルトでは、コピーブックの構造体名を「DCLtablename」として生成します。テーブル名のための構造体名や有効な COBOL 名での構造体名にもできます。アンダスコアはすべてハイフンに変換されます。

3 つの方法の 1 つを使用して、ホスト変数を生成

デフォルトでは、カラム名とテーブル名の接頭語との組み合わせを使用してホスト変数が作成されます。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。常にカラム名 (接頭語なし) のみを使用してホスト変数を生成できます。また、アルファベット文字の接頭語も使用できます。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文字の接頭語でホスト変数が生成されます。

---

注：生成されたホスト変数名が文字数が 32 文字以上の場合や、名前に無効な文字が含まれる場合は、指定方法に関わらず、FLD が前に付いたカラム番号としてホスト変数名が生成されます。

---

### 3 つの方法の 1 つを使用して、インジケータ変数を生成

デフォルトでは、コピーブックの最後にインジケータ変数を生成します。各ホスト変数の最後にも生成できます。また、インジケータ変数を生成させないこともできます。

### データソースへ接続するログオン詳細

デフォルトでは、データソースへの接続時に、ユーザ ID およびパスワードが要求されます。常に同じデータソースに接続する場合や、データソースへの接続時に常に同じログオン情報を使用する場合は、接続するたびに自動的に使用されるユーザ ID およびパスワードを指定できます。

### ツリービューに表示されるデータを限定

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、リストの作成応答時間が長くなります。このリストの作成応答時間を短くするためには、返されるデータを限定してこのリストの内容を制限できます。これを行うには、修飾子、テーブル名、オーナー、またはこれらの組み合わせを指定します。これらの指定で、次のワイルドカード文字が使用できます。

- 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。

たとえば、XYZ で始まるテーブルのみをリストに限定表示する場合には、「XYZ %」と指定します。

---

### 注：

- 指定した結果で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。
- OpenESQL アシスタントは、Unicode をサポートします。次に示すオプションによって Unicode データの表示が異なります。

「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。

「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のどちらかをチェックすると、同じクエリーは Unicode 文字列以外の結果が表示されません。

---

## 方法

# OpenESQL アシスタントの使用方法

OpenESQL アシスタントを使用すると、SQL クエリーを簡単に設計および作成でき、COBOL コードに SQL クエリーを埋め込みます。クエリーを指定すると、OpenESQL アシスタントは必要なコードを生成します。それらをプログラム内に埋め込む前に、適切な結果の作成を確認するためにクエリーをテストすることもできます。また、COBOL 内のクエリーをサポートするために必要な補助コードを生成または埋め込むことができます。

## OpenESQL アシスタントの起動方法

Net Express を起動したときに、OpenESQL アシスタントのドッキング可能なウィンドウを表示するように構成できます。また、必要なときに OpenESQL アシスタントを簡単に起動できます。

OpenESQL の構成によっては、ユーザ名とパスワードを入力する必要があります。

## 方法

## データソースへの接続方法

OpenESQL アシスタントで SQL を作成する前に、アクセスしたいデータを含むデータソースへ接続する必要があります。ワークステーション上のカタログされたすべてのデータソースは、OpenESQL アシスタントのツリービューに一覧表示されます。一度に接続できるデータソースは 1 つです。接続した後は、データソース内のすべてのデータ名はツリービューに表示されます。

一度データソースへ接続すると、データソースのプロパティを参照できます。「データソースのプロパティ」ダイアログボックスには、次の情報が含まれます。

- 使用された、識別子を囲む引用文字
- テーブル名の最大サイズ
- カラム名の最大サイズ

これらの情報は、データソースでサポートされるテーブルとカラムを確認できます。テーブルまたはカラムの長さ 0 は、表示されません。たとえば、検索条件を設定する場合に、長さ 0 のテーブルまたはカラムを指定できません。

## 方法

## クエリーのビルド方法

OpenESQL を使用してクエリーをビルドする方法は、次の複数の手順を実行します。

クエリーするテーブルを決定します。  
作成するクエリーの型を選択します。  
クエリーするカラム (1 つまたは複数) を選択します。  
データ結果を制限するために検索条件を指定します。  
検索または表示されたデータ順序を規定します。  
結合テーブルを指定します。

以後では、これらの手順について説明します。

## テーブルおよびクエリーの型の選択方法

1 回の操作で 1 つのテーブルを選択できます。または、データソース内のすべてのテーブルを選択できます。「生成するクエリーのタイプを選択」ダイアログボックスでテーブル上で実行したいクエリーの型を選択します。このダイアログボックスには、選択したテーブル上で実行できるクエリーの型が一覧表示されます。選択するクエリーの型によっては、デフォルトのカーソル名も変更できます。また、クエリーをストアードプロシージャとして生成し、ストアードプロシージャ (stored procedure; SP) 名も指定できます。

選択されたテーブルは、テーブルアイコン  にチェックマークが付いて、ツリービューに表示されます。

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。

SELECT クエリーを選択すると、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ます。次に例を示します。

```
SELECT FROM Customer A
```

最初に選択したテーブルは、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。

各カラム名の前にも、別名 (A.CustID、A.Company など) が付きます。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。

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 (\$) が使用されることがあります。OpenESQL アシスタントは、システムテーブルの名前に生成されたテーブル名を規約違反にするような特殊文字を検出すると、自動的にカラム名やテーブル名を引用符で囲みます。

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、OpenESQL 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を生成します。

## 方法

### カラムの選択

テーブルおよびクエリーの型を選択すると、クエリーを処理する 1 つ以上のカラムを選択できます。

選択されたカラムは、カラムアイコン  にチェックマークが付いて、ツリービューに表示されます。

## 方法

### カラム用の集約関数の指定方法

SELECT (カーソル) または SELECT DISTINCT (カーソル) クエリーのどのカラムでも集約関数を指定できます。集約関数は、カラムから返されるデータの特定な情報を取得します。

OpenESQL は、返されるデータの各グループに対して、次の情報を返します。

集約関数	返されるデータの各グループに対する値
SUM	すべての値の合計が返されます (数字カラムのみ)
AVG	すべての値の平均が返されます (数字カラムのみ)
MAX	最大値が返されます
MIN	最小値が返されます
COUNT	レコード数が返されます

## 方法

### 検索条件の指定方法

OpenESQL アシスタントの [検索条件] タブで検索条件を指定する場合は、クエリーで返される行を制限できます。OpenESQL アシスタントは、指定する条件に基づいた WHERE 句を生成します。

## 方法

### 取り出すデータのソート方法

OpenESQL アシスタントの [ソート] タブでソートするカラムを指定して、クエリーから返される行の順序を決定できます。OpenESQL アシスタントは、指定する順序に基づいた ORDER BY 句を生成します。

## 方法

### 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。OpenESQL アシスタントは、最初に一致するカラムを使用して結合テーブルを作成します。一致するカラムが見つからない場合は、自動的に結合は生成されません。この場合は、検索条件機能を使用して結合を指定できます。

この機能は、SELECT クエリーのみで使用可能です。

## 方法

## クエリーでの作業

ここでは、いくつかの OpenESQL アシスタントの機能とクエリーの作成を拡張する推奨処理を説明します。

### カラムの詳細の表示方法

OpenESQL アシスタントの [詳細] タブは、テーブルに含まれるカラムの詳細情報を表示します。[詳細] タブには、選択されたテーブルの次に示すカラムの内容が表示されます。

#### 「カラム名」

テーブルの各カラム名。各カラム名の前に、別名が付くことに注意してください。たとえば、テーブルの CustID という名前は、A.CustID のように「カラム名」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。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、カラムのデータ型を使用して生成された COBOL PICTURE 句と一致する必要なホスト変数がすべて宣言されます。詳細は、『[補助コード](#)』の節を参照してください。

#### 「精度」

数字カラムの総桁数またはテキストカラムのカラム長。

#### 「位取り」

カラムの数値を丸める桁数。

#### 「ホスト変数」

カラムについて生成されたホスト変数の値。ホスト変数名は、次の形式で生成されます。

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

#### 「インジケータ変数」

カラムについて生成されたインジケータ変数の値。インジケータ変数名は、次の形式で生成されます。

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

## 方法



## 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を解除すると、OpenESQL は現在のクエリーを破棄します。他のデータソースへ接続する前に、現在のデータソースの接続を解除する必要があります。

### 方法

#### クエリーの終了

クエリーを終了すると、OpenESQL は生成されたコードは破棄され、新規クエリーを作成できます。

### 方法

#### SELECT クエリーからテーブルの削除

OpenESQL アシスタントで作成されるほとんどのクエリーは、1 つのテーブルのみにアクセスします。クエリーからテーブルを削除すると、OpenESQL はクエリーを閉じて、作成されたコードは失われます。ただし、1 つ以上のテーブルを結合する SELECT クエリーを作成した場合は、クエリーを閉じずに安全にテーブルを削除できます。SELECT クエリーからテーブルの削除を行うと、削除を確認するダイアログが表示されます。確認後に、OpenESQL はクエリーからテーブルに関係する生成されたコードを削除します。また、テーブルに属するカラムに関係するすべてのコードも削除されます。

### 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

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、配列取り込みを処理できます。

### 方法

## SQL クエリーのテスト方法

プログラムコードで動作させる前に、OpenESQL アシスタントでクエリーをテストできます。これを行うには、OpenESQL アシスタントからクエリーを実行します。すべての SELECT 文の結果は、[実行結果] タブに自動的に表示されます。他のクエリーの型は、メッセージボックスに次に示す情報が表示されます。

実行されるクエリーの名前。

返される行数。

クエリーの実行が成功したかどうか。

クエリーが正しく実行されるために、追加の指定が必要になる場合があります。OpenESQL アシスタントがコードを実行できない場合は、正しくするように求められます。

---

注：OpenESQL アシスタントでクエリーを実行すると、データベースに対して UPDATE 文、INSERT 文、および DELETE 文が実行され、内容が変更されます。

---



## 方法

### 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、COBOL プログラムに追加できます。

## 方法

### 補助コード

OpenESQL で生成された SQL クエリーは、COBOL プログラムから正しく実行されるために、クエリーを除く追加のコードが必要になる場合があります。たとえば、DB2 UDB で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。クエリーを埋め込むと、OpenESQL アシスタントで補助コードを生成して埋め込みます。

### EXEC SQL 補助コードの生成方法

OpenESQL アシスタントは、次に示す EXEC SQL 補助コードを生成できます。

#### CONNECT 文

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

#### SQLCA 宣言

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

#### 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。OpenESQL アシスタントでは、INCLUDE 文を自動生成し、プログラムに必要な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します。

#### 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します。

#### ROLLBACK 文

ROLLBACK 文を自動生成し、直前の SQL 文 (または複数の 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します。

#### 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ます。

入力インターフェイス LINKAGE

出力インターフェイス LINKAGE

## 方法

### 補助コードの埋め込み方法

SQL クエリーを埋め込む方法と同じように、補助コードを埋め込みます。

## 方法

### ストアードプロシージャ

ここでは、ストアードプロシージャの生成方法とOpenESQL アシスタントを使用してストアードプロシージャをテストする方法を説明します。

---

注：ストアー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 文をもつストアー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、簡単なストアードプロシージャと、これらのストアー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。

---

### ストアードプロシージャクエリーの生成方法

Microsoft は、ストアー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ー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ます。

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、クエリーをストアー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と、ストアードプロシージャおよび、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 文を生成します。

OpenESQL アシスタントでは、ストアードプロシージャを削除し、作成する 2 つの SQL 文を生成します。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。

OpenESQL アシスタントでは、クライアントプログラムに記述するコードも生成し、ストアードプロシージャを呼び出します。選択するクエリーの型によっては、ストアードプロシージャを呼び出す DECLARE CURSOR 文や SQL CALL 文のみを生成することがあります。

## 方法

### ストアードプロシージャクエリーのテスト方法

ストアードプロシージャクエリーを作成した後に、他のクエリーをテストした方法と同じように、ストアードプロシージャクエリーをテストできます。ただし、OpenESQL アシスタントでは、ストアードプロシージャ用に複数の SQL 文が作成されるので、複数回クエリーを実行する必要があります。クエリーが実行されるたびに、次の SQL 文が実行されます。

クエリーを最初に実行するとき、DROP PROCEDURE 文が実行され、次のメッセージが表示されます。

ストアードプロシージャ '*spname*' が存在していないため削除できません。

このメッセージは意図的なものであり、DROP PROCEDURE を複数回実行すると、異なるメッセージが表示されます。

次にクエリーを実行すると、CREATE PROCEDURE 文が実行されます。文の実行が成功したことを示すメッセージが表示されます。

定数値が必要な検索条件をもつストアードプロシージャの CALL 文を実行する場合は、定数値を入力するよう求められます。

すべての SQL 文が実行されると、[実行結果] タブに結果が表示されます。

正しい結果になるまで、この処理を繰り返します。

## 方法

### プログラムへのストアードプロシージャの埋め込み方法

この方法は、プログラムへ標準的な SQL クエリーを埋め込む方法と似ています。

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。プログラムがクライアントプログラムである場合は、DROP PROCEDURE 文と CREATE PROCEDURE 文は必要ありません。

注：前述の方法でクエリーをテストする場合は、ストアー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。ストアー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、ストアードプロシージャはデータソースに保存されません。

## 方法

## 9: DB2

ここでは、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 ECM (external compiler module; 外部コンパイラモジュール) は、このシステム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。

### 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』の章にある『[文字データ型](#)』の節を参照してください。

### 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 では新型でないため、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).

## 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。



## ユーザ定義関数

ユーザ定義関数 (user defined function; 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。

---

## 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。



## 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。

## 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。

## 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。

## 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
```

## 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。

## NOT 演算 ( )

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。

## 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。

## SQL 通信領域

SQL 文を実行すると、プログラムの SQL 通信領域 (sql communications area; 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。

---

ヒント：ANSI 互換のために、SQLCA 全体を定義することをお奨めします。

---

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。

## 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。そのため、DB2(INIT=PROT) コンパイラ指令を機能させるには、非オブジェクト指向モジュールを含めて、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。

## 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。

## INIT DB2 指令オプション

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 オプションには、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 を指定することで選択できます。

INIT 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 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 オプションを指定することができます。INIT 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ル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。

## UDB-VERSION DB2 指令オプション

DB2 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 UDB 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 (program identifier; 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。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。
```

## 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](#)』の節を参照してください。

DB2 UBD バージョン 7.1 で導入されている API 呼び出しを使用する場合は、DB2 ECM は API 呼び出しを検証するために IBM モジュールを呼び出します。DB2 ECM が指令構文でエラーを検出できない場合は、「-104 SQLCODE」のエラーメッセージが表示されます。

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形式 (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。

## 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0-1 : 「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
```

## 自動コンパイル

コマンドファイルなどのバックグラウ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。

## DB2 コンパイラ指令

他のコンパイラ指令が指定できるのであればどこにでも DB2 コンパイラ指令を指定できます。指定できる箇所の例を次に示します。



## \$SET 文

```
$set db2(init=prot bind collection=myschema)
```

## Net Express IDE の「詳細指令」画面使用

### コマンド行

```
cob -Vt testconn.cbl -C"anim  
db2(init==prot bind collection==myschema)"
```

IDE から SQL コンパイラ指令を設定する方法の詳細は、ヘルプトピックの『[SQL コンパイラ指令オプションの設定](#)』を参照してください。

---

注：複数の SQL 指令は指定できませんので、1 つの指定箇所に対して 1 つの SQL 指令を指定する必要があります。

---

## DB2 コンパイラ指令オプション

DB2 コンパイラ指令は、オプションのセットで構成されています。詳細は、ヘルプトピックの『[DB2 コンパイラ指令](#)』を参照してください。使用可能なオプションの詳細は、ヘルプトピックの『[DB2 コンパイラ指令オプション](#)』を参照してください。

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よって依存します。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。

---

注：このリストは、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い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分からない場合は、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。

---

## 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。DB2 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。QUALFIX オプションの詳細は、『[DB2 コンパイラ指令オプション](#)』の節を参照してください。

実行時のエラー条件は、SQLCODE のゼロ以外の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は、『[SQL 通信領域](#)』の節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
```

## 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お問い合わせする必要が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)	<b>ecmtrace.txt</b>	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)	<b>ecmtrace.txt</b>	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)	<b>sqltrace.txt</b>	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。

## 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」の編集ボックスで、次のように入力します。

```
db2api.lib
```

## 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。

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

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。

## 10: 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。

## 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。

---

## 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グし、テストします。詳細は、『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。

## 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 を使用できません。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。

## 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 (dynamic link library; 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。

## 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。

## 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。

- 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。
- 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。
- ストアドプロシージャは、SQL 文を実行します。
- 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。
- 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アントで実行するかを決定します。

## 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。

### 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。

#### 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。

#### 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。

#### 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
```

#### 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
```

## 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。

## 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ます。

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。

## 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。

- ホスト変数 :インジケータ変数  
または
- 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

## 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

```

## 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;

```

## 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。

## 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<sup>1</sup>。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 コマンドでストアー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。

## DB2 ユニバーサルデータベースでのストアー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ードプロシージャの 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。

```

Project:db2_sp.APP - Net Express - [getempsvr.cbl]
File Edit Search Animate Project View Options Tools Window Help

01 P-SQLSTATE PIC X(5).
*> Declare the qualified procedure name (null term)
01 P-PROC PIC X(27).
*> Declare the specific procedure name
01 P-SPEC PIC X(18).
*> Declare SQL diagnostic message token
01 P-DIAG.
    49 P-DIAG-LEN PIC 9(4) USAGE BINARY.
    49 P-DIAG-TEXT PIC X(70).
PROCEDURE DIVISION using link-empno,
                        inot-sqlcode,
                        link-firstname,
                        link-lastname,
                        link-hiredate,
                        link-salary,
                        p-ind1,
                        p-sqlstate,
                        p-proc,
                        p-spec,
                        p-diag.

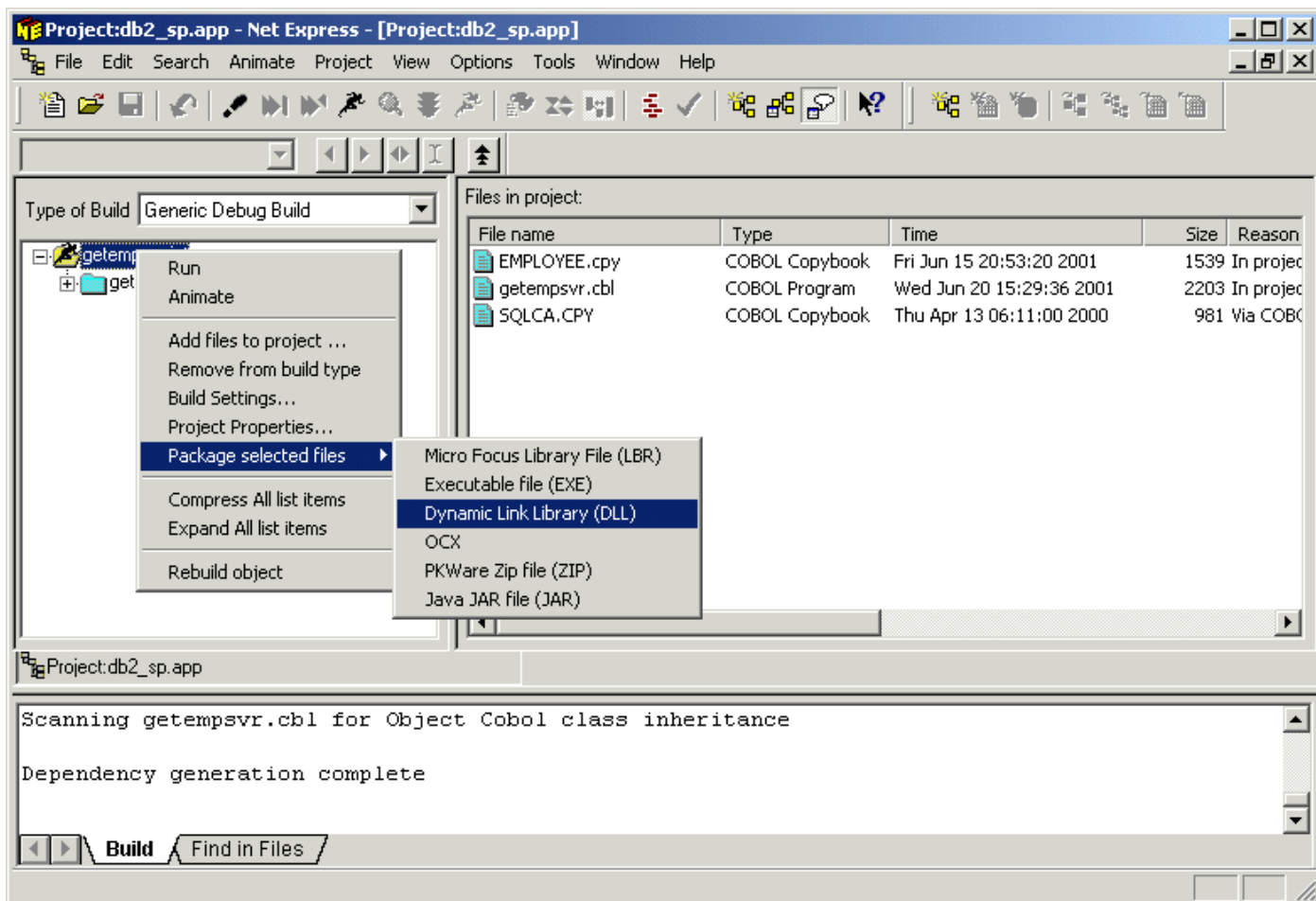
    move link-empno to employee-empno
*> Use cbl_debugbreak to debug under Win9x
*> CALL "CBL_DEBUGBREAK"

*> use Sleep call to debug under Win NT/2000
*> call DB2API 'Sleep' using by value ws-wait

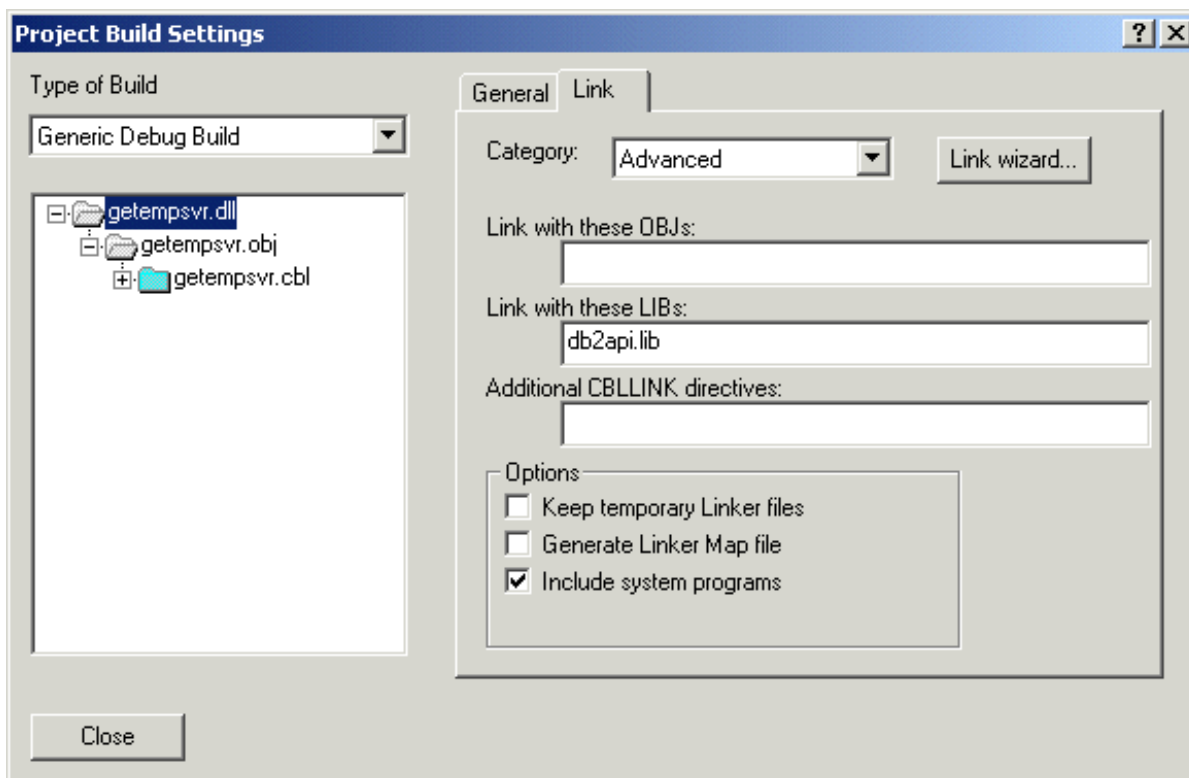
EXEC SQL
SELECT

```

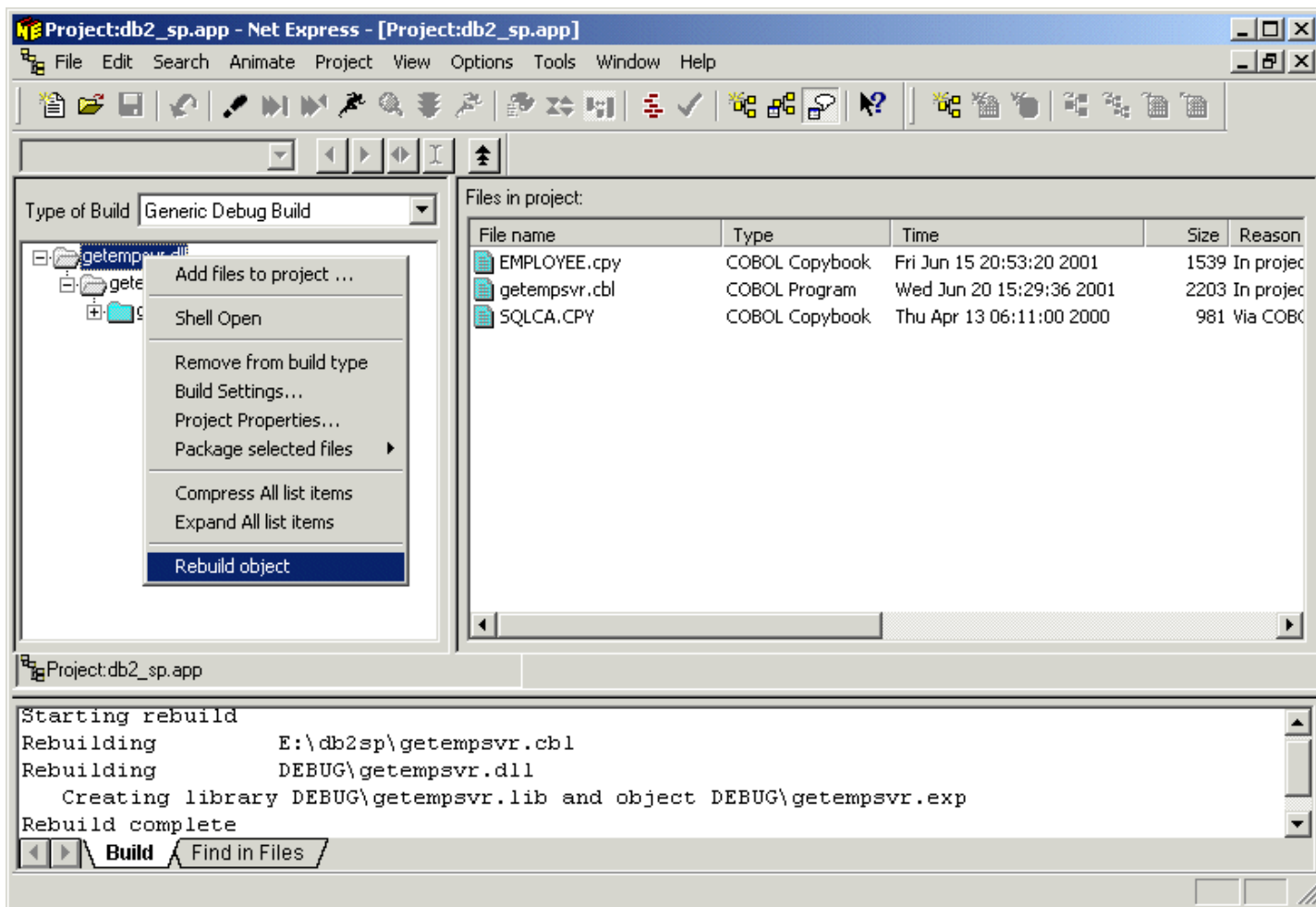
2. プログラムを Net Express プロジェクトに追加した後に、プログラムを選択し、そのプログラムを動的リンクライブラリ (.dll) としてパッケージ化します。



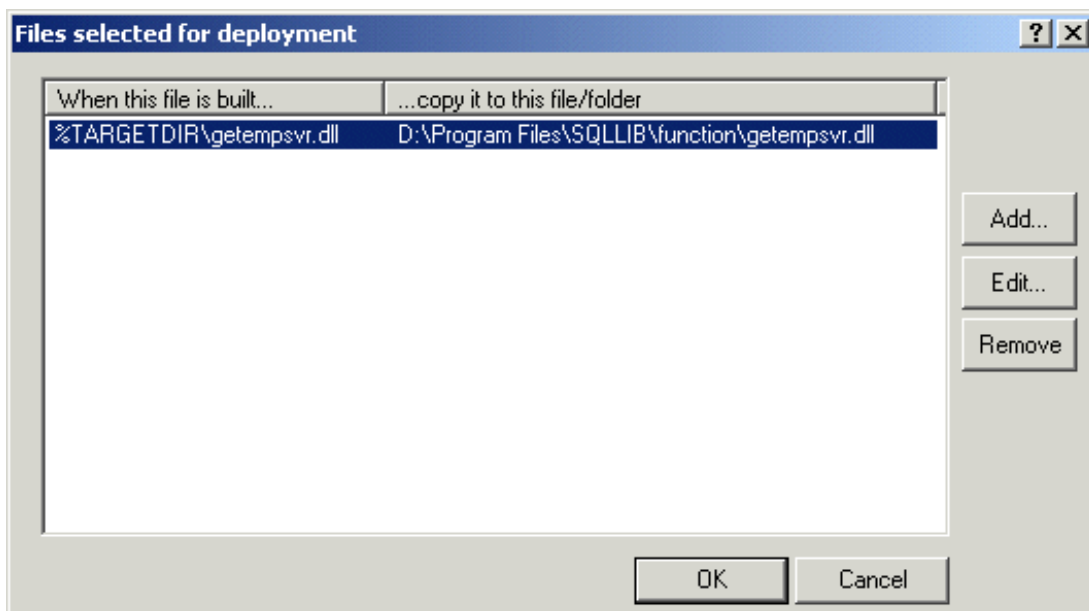
- プログラムのビルド設定で、[リンク] タブを選択し、ドロップダウンリストボックスから「高度な設定」カテゴリを選択します。「これらの LIB とリンクする」エントリフィールドで、db2api.lib を追加し、[閉じる] ボタンをクリックします。



- ストアー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。



- CREATE PROCEDURE の定義方法に応じて、ストアードプロシージャをテストするか、または、ストアードプロシージャを 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、[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。



## DB2 ユニバーサルデータベースでのストアードプロシージャのデバッグ

ストアー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、ストアードプロシージャコードをデバッグするプログラムは、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ー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 ストアー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、DB2 UDB バージョン 8 以前では db2dari プロセス、および DB2 UDB バージョン 8 以降では db2fmp 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 または db2fmp 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ため、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.
```

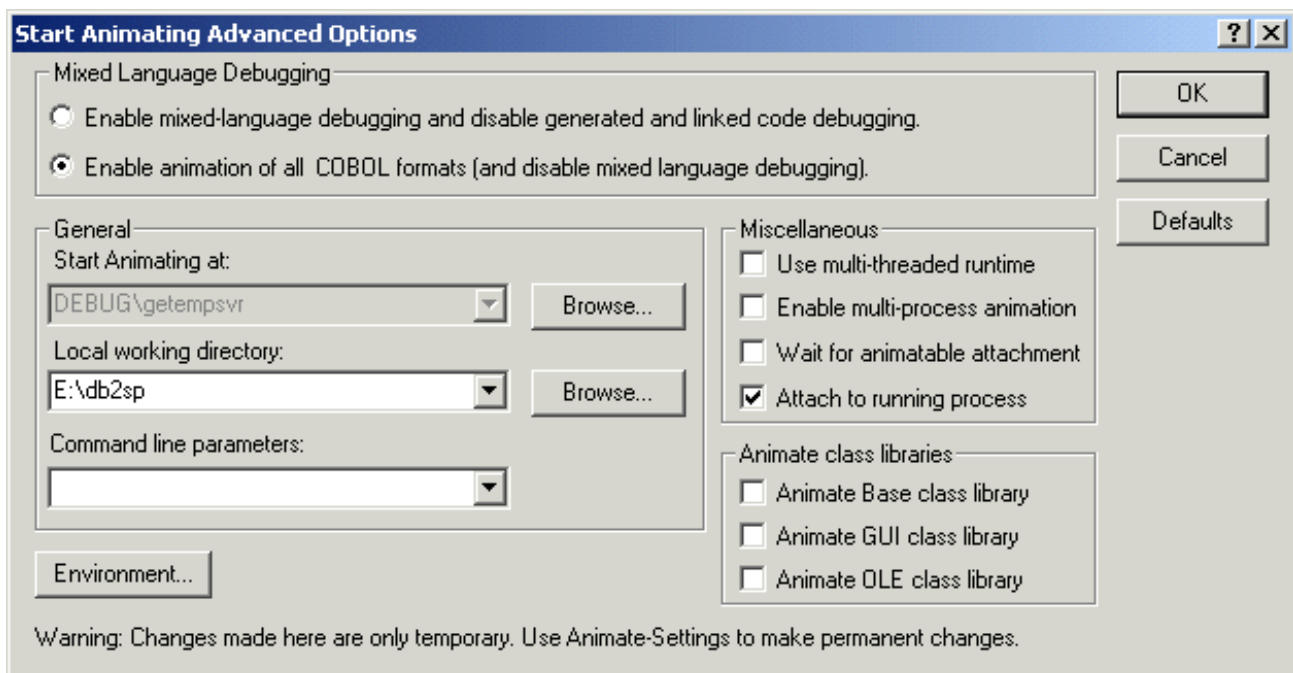
プログラムに次の文を追加して、アニメートを開始する前にプログラムが実行されるようにします。

```
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。

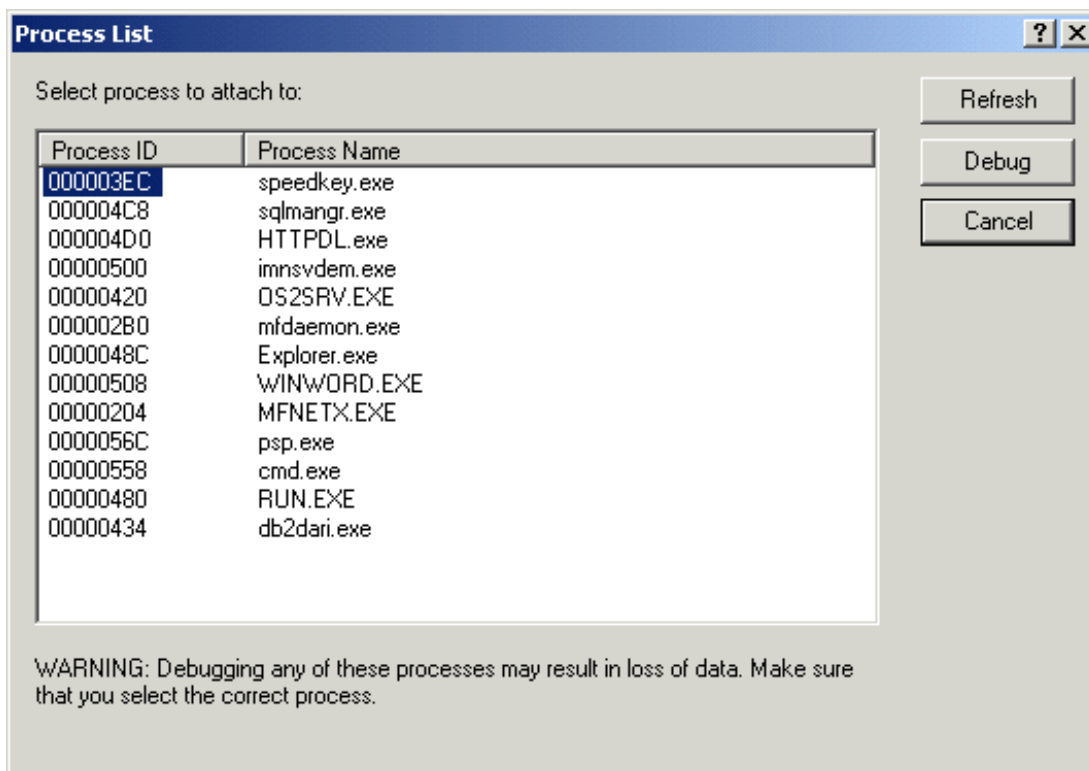
ストアドプロシージャをアニメートするには、次の操作を実行します。

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. 「実行中の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。

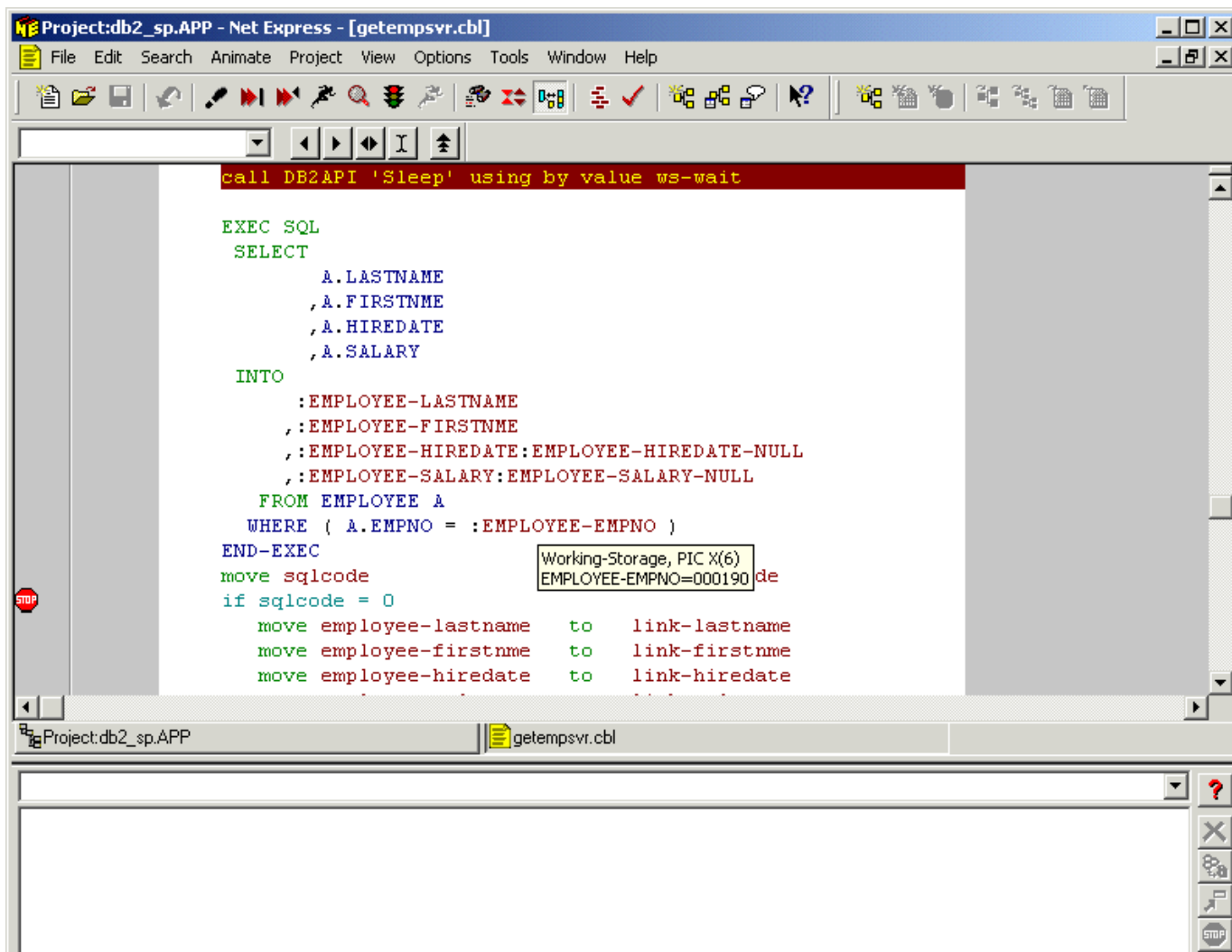




- DB2 UDB バージョン 8 以前では db2dari.exe、または DB2 UDB バージョン 8 以降では db2fmp.exe に関連付けられたプロセスを選択し、[デバッグ] ボタンをクリックします。db2dari または db2fmp プロセスが一覧表示されない場合は、[最新表示] ボタンをクリックします。



- [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。
- 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。



Control+Break or Break button pressed.

122 lines

Ln 52, Col 49



ストアドプロシージャのデバッグが完了したら、アニメータを終了します。

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。

Copyright © 2006 Micro Focus (IP) Ltd. All rights reserved.

DB2

COBSQL

## 11: 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。

すでに COBSQL 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このシステムに移行する場合は、COBSQL を使用してください。他の埋め込み SQL アプリケーション開発には OpenESQL をお奨めします。

### 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 文
```

```
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実行コードが表示され、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。

---

注：SOURCEFORMAT"FREE" コンパイラ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。

---

### 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。

例：

```
cobol testprog p(cobsql) COBSQL 指令 END-C プリコンパイラオプション ENDP
コンパイラ指令;
```

```
cobol testprog p(cobsql) csqlt=ora makesyn end-c xref=yes mode=ansi
endp 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、xref=yes と mode=ansi です。
- ・コンパイラ指令は list() です。

## 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、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。

PREPROCESS 指令と、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、PREPROCESS 指令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 環境変数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 環境変数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またはコマンド行で設定します。

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。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 testprog p(cobsql) csq1t=ora makesyn end-c
xref=yes mode=ansi endp list();
```

`end-c` で終了する COBSQL 指令は、`csq1t=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、`xref=yes` と `mode=ansi` です。

コンパイラ指令は `list()` です。

## COBSQL 指令

COBSQL 指令を次に示します。

---

### COBSQLTYPE

使用するプリプロセッサを指定します。

構文：

```
COBSQLTYPE={ORACLE | ORACLE8 | SYBASE | INFORMIX-NEW}
```

プロパティ：

同義語： CSQLT

---

### CSTART

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。

構文：

```
[NO]CSTART
```

プロパティ：

デフォルト： NOCSTART

ト：

同義語： CST

コメント：

この指令は、UNIX では使用できません。



---

## CSTOP

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。

構文：

[NO]CSTOP

プロパティ：

デフォルト： NOCSTOP

同義語： CSP

コメント：

この指令は、UNIX では使用できません。

---

## DEBUGFILE

デバッグファイル (.deb) を作成します。

構文：

[NO]DEBUGFILE

プロパティ：

デフォルト： NODEBUGFILE

同義語： DEB

---

## DISPLAY

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のみ使用します。

構文：

[NO]DISPLAY

プロパティ：

デフォルト： NODISPLAY

同義語： DIS

---

## END-COBSQL

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。

構文：

END-COBSQL

END-C

END

プロパティ：

デフォルト 指定できません。

ト：

同義語： END-C, END

---

## 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。

構文：

KEEPCBL

---

## 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。

構文：

[NO]MAKESYN

プロパティ：

デフォルト NOMAKESYN

ト：

---

## SQLDEBUG

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 (通常は拡張子 .cbl)、プリコンパイラが作成するリストファイル (通常は拡張子 .lis)、そして 拡張子 .sdb をもつ COBSQL デバッグファイルがあります。さらに KEEP CBL と TRACE をオンにします。

構文：

SQLDEBUG

プロパティ：

デフォルト 指定できません。

ト：

同義語： SQLDEBUG

## TRACE

構文：

[NO]TRACE

プロパティ：

デフォルト： NOTRACE

ト：

同義語： 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。

構文：

VERBOSE

プロパティ：

デフォルト： 指定できません。

ト：

## COBOL 指令

コンパイラが COBSQL に渡す COBOL 指令を次に示します。

指令	説明
[NO]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。
[NO]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。
[NO]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。
[NO]LIST	プリコンパイラリストファイル (.lis) を保存します。
[NO]WARNING	報告する SQL エラーの最低重大度を指定します。

## 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、`Net Express¥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 `Net Express¥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 `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。

## 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。CP は、次のような文を展開します。

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

CP の詳細は、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 と 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には、次の Oracle と Sybase 用の COBOL コンパイラ 指令を使用します。

```
copyext (pco,cbl,cpy,cob) osextpco)
```

Informix:

Informix の場合は、次の指令を使用します。

```
copyext (eco,mf2,cbl,cpy,cob) osextpco)
```

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、COBOL システム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、このサンプルの SQLCA が使用されます。

---

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。

## 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 (national locale support; 各国語サポート) の詳細は、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 変数を使用できません。COBLANG を正しく実行するためには、mflangnn.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、Net Express¥Base¥Bin ディレクトリに mflang05.lbr ファイルを作成します。

COBSQL のエラーメッセージファイル cobsq1.lng は、多くの異なる言語に翻訳され、COBOL NLS メッセージファイルとともに保存されます。現在の LANG 設定用のエラーメッセージファイル cobsq1.lng が存在しない場合には、デフォルトのエラーメッセージファイルが使用されます。

---

注：COBSQL は、データベースプリコンパイラのエラーメッセージを処理しません。

---

## 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。

### Oracle

```
cobol sample.pco anim nognt preprocess(cobsq1)
  cstart cstop CSQLT=ORA end-c comp5=yes endp;
```

### Sybase

```
cobol example1.pco confirm preprocess(cobsq1)
  cstop csp cobsq1type=sybase preprocess(cp) sy endp;
```

## 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ます。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を送付する。

- 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

## 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に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。

## 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.



- 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
- 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。

### Oracle 8 および 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)
```

生成された .cbl ファイルに変数を追加するために、Pro\*COBOL の作業場所節に変数を 1 つ以上定義する必要があります。

## Sybase を使用する際の考察事項

Sybase を使用する前に、次のようにシステムを変更する必要があります。

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-severity-number-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。

**severity** 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。

**number** Sybase エラーに割り当てられる 4 桁の一意のエラー番号。

**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-警告-2009-%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勧めします。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。

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。

## 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) osex(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。

