Net Express

データベース アクセス

Micro Focus NetExpress™

データベース アクセス

Micro Focus®

第 2 版
1998 年 10月

序文

本書はリレーショナル データベースにアクセスするために埋め込み SQL を使用した COBOL アプリケーションの作成方法について述べています。 NetExpress には以下の機能があります。

- OpenESQL

  OpenESQL は、COBOL プログラム内に SQL 文を埋め込んで、ODBC ベースのデータ ソースをアクセスできるようにする 埋め込み SQL プリプロセッサです。

- DB2 ECM

  DB2 External Checker Module (ECM) は、NetExpressと一緒に提供される組込みプリプロセッサの新型であり、Micro Focus COBOL コンパイラと親和性があるように設計されています。 DB2 ECM は、埋め込み SQL 文を変換して、適切な DB2 データベース サービス を呼び出します。 これは、以下のものと一緒に使用します。

  - IBM Software Development Kit (SDK) for Windows 95/NT Version 2.1
  - IBM DB2 for Windows 95/NT Single User Version 2.1
  - IBM Distributed Database Connection Services (DDCS) for Windows NT Version 2.3
  - IBM DB2 ユニバーサル・データベース バージョン 5
  - IBM DB2 Connect Version 5

- 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 プリコンパイラとともに動くように設計されている組込みプリプロセッサです。 次のものと一緒に使用します。

  - Oracle Pro*COBOL Version 1.8

  上記のプリコンパイラを以前のバージョンの Micro Focus COBOL 製品と一緒に使用されていて、アプリケーションを NetExpress に移行しようとされているか、または UNIX プラットホームでアプリケーションを作成しようとされる場合は、COBSQL をお使い下さい。 他の形で 埋め込み SQL の開発をされている場合は、OpenESQL をお勧めします。

備考

- OpenESQL のアプリケーションは UNIX プラットホームでは動きません。
- COBSQL の使用は、標準の手続き型 COBOL プログラムに対してのみサポートされています。COBSQL は、

オブジェクト指向の COBOL (OO プログラム) 構文やネストを含むプログラムでは使用できません。

対象読者

本書は埋め込み SQL を使用してリレーショナル データベースをアクセスする COBOL アプリケーションを作成したり、修正しようとされる方を対象としています。

OpenESQL や ODBC を使われる場合は、その双方とも熟知されていることを想定しています。DB2 ECM を使われる場合には、DB2 をよくご存知であることを想定しています。COBSQL を使われる場合には、アクセスしようとする Oracle もしくは Sybase のデータベースをご存知であることを前提としています。

本書の使用方法

第 1 部 - はじめに

「第 1 章 はじめに」では、埋め込み SQL の簡単な概要と、埋め込み SQL アプリケーションの作成の仕方を説明しています。

「第 2 章 ホスト変数」では、ホスト変数、ホスト配列、インジケータ変数、インジケータ配列とその使用方法について説明しています。

「第 3 章 データ型」では、サポートされているデータ型と、データベースで使われているデータ型と正しい COBOL picture 句との対応方法について説明しています。

「第 4 章 カーソル」では、カーソルの宣言とオープンの仕方、およびそれを使用してデータを検索する方法について説明しています。

「第 5 章 データ構造体」では、SQLCA (SQL 通信領域) と SQLDA (SQL 記述子領域) について説明しています。

「第 6 章 動的 SQL」 では、動的 SQL の準備と実行の仕方について説明しています。

「第 7 章 SQL 言語リファレンス」 は、埋め込み文のリファレンスです。

第 2 部 - OpenESQL

「第 8 章 OpenESQL」 では、OpenESQL の概要と機能を説明しています。

「第 9 章 OpenESQL アシスタント」 は、すばやく、関単に SQL 文を作成するための OpenESQL アシスタントの使用方法のチュートリアルです。SQL クエリーをうまく作成できた時には、OpenESQL アシスタントはそのクエリーをプログラム中に挿入します。また、CONNECT 文やホスト変数宣言といった必要な補助コードも追加することができます。

第 3 部 - DB2

「第 10 章 DB2」では、DB2 ECM とその使用方法を説明しています。

第 4 部 - COBSQL

「第 11 章 COBSQL」では、COBSQL プリプロセッサの使用方法を説明しています。

表記規則

本書では次の書体や規則が使用されています。

- ユーザーが入力するテキストは、次のように表記します。

  `cat script_name | more`

  斜体テキストは、コマンドの一部として入力する変数を表します。

- コマンド行やコード例でオプション入力するテキストは、角かっこ ([]) で囲みます。次の式では、オプションで入力する単語 NOT がない場合、*column_name* は pattern_value のように指定し、NOT がある場合、*column_name* が *pattern_value* 以外であるように指定します。

  column_name [NOT] LIKE *pattern_value*

- 特定のモデルやオペレーティング システムだけに適用する項や段落については、段落のすぐ前に太字斜体の項目名を記載します。たとえば、次のようになります。

  OpenESQL
  この段落は OpenESQL に対してのみ適用され、DB2 や COBSQL には適用されません。

目次

第1章　はじめに

備考：このドキュメントでは、SQL 構文や戻されるエラー メッセージ、および本製品の COBOL 環境以外での SQL の使用方法の詳細は説明していません。これらの情報については、使用しているデータベースの付属ドキュメントをご覧ください。

1.1 概要

NetExpress には 3 種類の SQL プリプロセッサ（OpenESQL、DB2 ECM、COBSQL）が含まれており、埋め込み SQL ステートメントによって COBOL プログラムからリレーショナル データベースにアクセスすることが可能です。

- OpenESQL

  OpenESQL は、COBOL プログラム内に記述された埋め込み SQL ステートメントからリレーショナル データベースへの ODBC ドライバ経由のアクセスを可能にするプリプロセッサです。

- DB2 ECM

  DB2 外部チェッカー モジュール（ECM）は、Micro Focus COBOL コンパイラとの連携を重視した新しいタイプの統合型プリプロセッサです。埋め込み SQL ステートメントを対応する DB2 データベース サービスに変換します。DB2 ECM は次の各ツールと共に使用できます。

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

- COBSQL

  COBSQL は、リレーショナル データベース ベンダーが提供している COBOL プリコンパイラ向けの統合型プリプロセッサです。COBSQL は次の各ツールと共に使用できます。

  - Sybase Open Client Embedded SQL/COBOL ver. 11.1
  - Oracle Pro*COBOL ver. 1.8

旧バージョンの Micro Focus COBOL で上記のいずれかのプリコンパイラを使用して開発したアプリケーションを現バージョンの NetExpress に移行したり、UNIX プラットフォーム向けアプリケーションを開発する場合には、COBOL プリプロセッサとして COBSQL を選択してください。それ以外の開発で埋め込み SQL を使用する場合には、通常は OpenESQL を使用します。

備考

- OpenESQL では、UNIX 向けのアプリケーションは開発できません。
- COBSQL を使用できるのは、標準的な手続き型の COBOL プログラムだけです。オブジェクト指向の COBOL 構文やネストを含むプログラムには使用できません。

1.2 埋め込み SQL

上記の各プリプロセッサは、いずれも COBOL プログラム内に記述された埋め込み SQL ステートメントを取り込み、対応するデータベース呼び出しに変換します。

COBOL プログラム内に埋め込み SQL ステートメントを記述する際には、その前後を次の 2 つのキーワードで囲む必要があります。

```
EXEC SQL

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 ステートメントは、必要に応じて複数の行に続けて記述できます。その際の改行は COBOL 標準のルールに従います。なお、`EXEC SQL` と `END-EXEC` の間には記述できるのは埋め込み SQL ステートメントだけであり、通常の COBOL コードは記述できません。

ほとんどのデータベース ソフトウェア製品には、埋め込み SQL ステートメントに関する情報を網羅した SQL リファレンス ドキュメントが付属しています。ただし、リファレンスの記載情報を十分に理解するには、次に挙げるような基本的なデータベース操作を、対応するステートメントを使用して実行できることが前提になります。

操作	SQL ステートメント
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り込み	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り込み	OPEN, FETCH, CLOSE

次に一覧する埋め込み SQL ステートメントは、便宜上追加された INSERT、DELETE(SEARCHED)、および UPDATE(SEARCHED) を除き、いずれも標準の SQL ステートメントとはやや異なる機能を持つか、あるいは標準ステートメントへの追加として位置付けられます。

ステートメント	機能
BEGIN DECLARE SECTION	ホスト変数宣言セクションの開始を示します。
CALL	ストアド プロシージャを実行します。
CLOSE	OPEN ステートメントで開始された行単位のデータ取り込みを終了します。
COMMIT	トランザクションをコミット（確定）します。
CONNECT	データベース接続を確立します。
DECLARE CURSOR	行単位のデータ取り込み用のカーソルを定義します。
DELETE (POSITIONED)[1]	カーソル位置の行を削除します。
DELETE (SEARCHED)	テーブルから検索条件に合致する行を削除します。
DESCRIBE	SQLDA データ構造体にデータを設定します。
DISCONNECT[2]	1 つまたはすべてのデータベースとの接続を解除します。
END DECLARE SECTION	ホスト変数宣言セクションの終了を示します。
EXECSP	ストアド プロシージャを実行します。

ステートメント	機能
EXECUTE	名前に関連付けられた SQL ステートメントを実行します。
EXECUTE IMMEDIATE	指定されたホスト変数に格納された SQL ステートメントを実行します。
FETCH	指定されたカーソルの次の行を結果セットから取り込みます。
INCLUDE	アプリケーションで使用する特定の SQL 構造体を定義します。
INSERT	テーブルやビューにデータを追加します。
OPEN	指定カーソルを基準として行単位のデータ取り込みを開始します。
PREPARE	SQL ステートメントを名前に関連付けます。
QUERY ODBC[3]	ODBC データ ディクショナリへの照会を実行します。
ROLLBACK	現在のトランザクションをロールバックします。
SELECT INTO[1]	結果を 1 行取り込みます。（単独選択）
SET CONCURRENCY[3]	標準モード カーソルの並行オプションを設定します。
SET CONNECTION[3]	以降の SQL ステートメントに使用するデータベース接続を指定します。
SET OPTION[3]	照会処理オプションの値を割り当てます。
SET SCROLLOPTION[3]	標準モード カーソルのスクロール方法と行グループを設定します。
UPDATE (POSITIONED)[1]	カーソル位置の行データを変更します。
UPDATE (SEARCHED)	データの追加や修正によって既存行のデータを変更します。
WHENEVER	SQL ステートメント実行後のデフォルト処理として、CONTINUE、GOTO、PERFORM のいずれかを指定します。

オンライン ヘルプには、上記の各埋め込み SQL ステートメントの構文が使用例とともに詳しく説明されています。

備考

1. これらの埋め込みステートメントは、対応する標準 SQL ステートメントと同じ名前ですが、構文が拡張されています。構文についてはオンライン ヘルプをご覧ください 。

2. DISCONNECT ステートメントは、COBSQL で Oracle データベースにアクセスしている場合には無効です。

3. これらのステートメントは COBSQL ではサポートされていません。

1.2.1 大文字と小文字の扱い

プログラム内に記述した埋め込み SQL ステートメントのキーワードでは、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
```

上記の 3行は、いずれも同じステートメントとして認識されます。

カーソルやステートメント、および接続の名前は大文字と小文字が区別されます。したがって、これらの変数を参照する際には、宣言時の記述に合わせる必要があります。たとえば、C1 で宣言したカーソルは、かならず C1 で参照します。（c1 では参照できません。）

その他のテーブルや列などの項目名については、データベースの設定によって大文字と小文字の扱いが決定されます。

テーブルや列の名前などの SQL 識別子にはハイフン（-）は使用できません。

1.3 アプリケーションの作成

埋め込み SQL ステートメントを含む COBOL プログラムをコンパイルする際には、適切なコンパイラ ディレクティブを指定して、埋め込み SQL ステートメントを対応する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 ディレクティブを指定します。詳細については、*OpenESQL* の章を参照してください。

- DB2 ECM

  DB2 コンパイラ ディレクティブを指定します。詳細については、*DB2* の章を参照してください。

- COBSQL

  PREPROCESS"COBSQL" コンパイラ ディレクティブを指定します。詳細については、*COBSQL* の章を参照してください。

1.3.1 インターネット アプリケーション ウィザード

NetExpress のインターネット アプリケーション ウィザードを使用すれば、SQL データベースにアクセスする Web アプリケーションを容易に作成できます。

インターネット アプリケーション ウィザードの詳細については、オンライン ブック （インターネット アプリケーション）をご覧ください。

1.4 複数のプログラム モジュール

埋め込み SQL ステートメントを使用した複数のソース ファイルを個別にコンパイルした場合には、同じ実行可能ファイルにリンクするか、または独立したダイナミック リンク ライブラリ（.dll ファイル）として同じ実行可能ファイルから呼び出せばデータベース接続を共有できます。同じプロセスのいずれかのモジュールが最初に CONNECT ステートメントで確立したデータベース接続が、別の CONNECT ステートメントで切り替えられるまで共有されます。

OpenESQL
個別にコンパイルした複数のモジュールを 1 つのプログラムにリンクする（または 1 つのプログラムから呼び出す）場合には、SQL コンパイラ ディレクティブの INIT オプションをいずれか 1 つのモジュールのみに記述します。その他のモジュールでは、INIT オプションで自動的に確立されたデータベース接続を共有するか、または CONNECT ステートメントで必要に応じて接続先データベースを切り替えます。

COBSQL
複数の INIT ディレクティブが検出された場合、COBSQL は最初の INIT ディレクティブのみを処理し、それ以降の同ディレクティブを無視します。

OpenESQL/DB2
ステートメント名は、それが記述されたプログラム モジュール内のみで有効です。（プログラム モジュールとは、1 つのファイルにコンパイルされるコードのまとまりを意味します。） したがって、あるモジュール内に記述したステートメントを他のモジュールで使用することはできません。

OpenESQL/DB2
1 つのアプリケーションで同じ名前を持つカーソルを複数使用することはできません。

第2章 ホスト変数

ホスト変数とは、COBOL プログラム内で定義され、ODBC データソースとの値の受け渡しに使用されるデータ項目のことです。ホスト変数は COBOL プログラムの File Section、Working-Storage Section、Local-Storage Section、および Linkage Section で定義し、1 ～ 48 の範囲の任意のレベル番号を割り当てます。レベル 49 は VARCHAR 型の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、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、ホスト変数を同じ名前の表や列と区別します。

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 ODBC データソースに転送されるデータを指定します。

- 出力ホスト変数

  ODBC データソースから取り込まれたデータを格納します。

次の文では、:book-id が入力ホスト変数、:book-title が出力ホスト変数であり、前者は検索する本の ID、後者は検索結果を格納します。

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

ホスト変数は、埋め込み SQL 文の内、ODBC で動的パラメータを使用できる部分、つまり疑問符 (?) をパラメータマーカーとして使用できる部分で使用できます。

2.1 ホスト変数の宣言

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

次に例を示します。

```
EXEC SQL
  BEGIN DECLARE SECTION
```

```
END-EXEC

01 id           pic x(4).

01 name          pic x(30).

EXEC SQL

   END DECLARE SECTION

END-EXEC


display "ID 番号を入力してください： "

accept id.


* この文は、データ項目 "id" に格納された

* 数値と同じ社員番号を持つ社員の名前を取り込み、

* データ項目 "name" に格納します。


EXEC SQL

   SELECT emp_name INTO :name FROM employees

      WHERE emp_id=:id

END-EXEC
```

OpenESQL および DB2

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することができます。

ホスト変数の宣言では、以下の点に留意してください。

- ホスト変数名には COBOL におけるデータ項目の命名規約に準拠します。
- ホスト変数の宣言は、COBOL のデータ項目を宣言できる部分であれば記述できます。
- ホスト変数名には下線 (_) は使用できません。

備考

- ホスト変数に 27 文字以上の名前を使用すると、問題が発生する可能性があります。ホスト変数として使用するデータ項目には、半角 26 文字以下の名前を付けてください。
- 複数のデータ項目を単一のホスト変数として使用することも可能です。

2.2 ホスト配列

COBSQL

COBSQL を使用している場合、Oracle データベースでは、ここで記述するホスト配列の情報はすべて有効です。Sybase データベースを使用している場合は、ホスト配列を使用できるのは SELECT 文または FETCH 文のどちらか片方の出力変数としてだけです。

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。このような配列を、「ホスト配列」 と呼びます。

ホスト配列は、INSERT、UPDATE、または DELETE の入力変数として使用したり、SELECT 文や FETCH 文の INTO 句で出力変数として使用することが可能です。これらの文でホスト変数を SELECT 文、FETCH 文、DELETE 文、INSERT 文、UPDATE 文などで使用すると、大量のデータを扱うことができます。

備考

OpenESQL および DB2

- 1 つの SQL 文中で、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 文中のホスト変数はすべてホスト配列である必要があります。

COBSQL

- 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することができます。この場合にだけ、ホスト配列と通常のホスト変数を併用できます。

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 TO 'datasourcename' USER 'user.pwd'
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 行分が取り出されます。SELECT 文が 25 行以上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出すことが可能です。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。

同一 SQL 文中で複数のホスト配列を使用する場合は、各配列の次元をそろえる必要があります。

2.2.1 FOR 句

COBSQL

COBSQL を使用している場合、ここに記述する FOR 句の情報は Oracle データベースの場合だけに有効です。Sybase データベースを利用している場合には適用されません。

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することができます。FOR 句は、UPDATE 文、INSERT 文、DELETE 文などで配列の一部のみを処理する場合に特に有効です。

FOR 句では整数型のホスト変数を使用します。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、SELECT 文により 10 行分を取り出します。この 10 という数値は :maxitems で指定した値です。

処理する配列要素数は、最も小さいホスト配列 (またはインジケータ配列) の次数と FOR 句の変数値のいずれか小さい方の値に等しくなります。

FOR 句に指定された変数の値がゼロ以下の場合には、行の処理は実行されません。

2.2.2 処理済み行数の確認

SQLCA 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 文については、処理済みの行の累計値が記録されます。

DB2
DB2 では、次の内容が SQLERRD(3) に記録されます。

- 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 が呼び出された場合は、データベースが更新可能なら 1、データベースが読取り専用なら 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。

DB2
DB2 では、次の内容が SQLERRD(4) に記述されます。

- PREPARE が呼び出されて成功した場合は、文の処理に必要とされる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 が呼び出された場合は、下位のクライアントからの 1 フェーズ コミットなら 0、1 フェーズ コミットなら 1、1 フェーズ、読取り専用コミットなら 2、2 フェーズ コミットなら 3。

DB2
DB2 では、次の内容が SQLERRD(5) に記述されます。

- 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。
    - 活動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

フィールドにマイナスの値が記述されます。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、サーバー認証なら認証型値 0、クライアント認証なら 1、DB2 コネクト使用の認証なら 2、DCE セキュリティ サービス認証なら 3、不特定の認証には 255。

2.3 インジケータ変数

埋め込み SQL では、インジケータ変数を使用してデータベースに NULL 値を保存したり、データベースから NULL 値を取り込むことができます。インジケータ変数は、常に次のように定義します。

```
pic S9(4) comp-5
```

2.3.1 NULL 値

COBOL とは異なり、SQL では NULL を格納できる変数をサポートしています。NULL はエントリが存在しないことを意味し、通常は値が不明であるか、または定義されていない状態を示します。NULL を使用すると、数値列のゼロや文字列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から区別することができます。たとえば、価格列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。価格が無効か、または未設定です。

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。ホスト変数の値が NULL の場合、対応するインジケータ変数の値は -1 になります。一方、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。次の例では、埋め込み UPDATE 文中のホスト変数 (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 の値が -1 の場合、UPDATE 文を実行すると ODBC ドライバは次のように解釈します。

```
EXEC SQL

  UPDATE closeoutsale

     SET temp_price = null, listprice = :oldprice

END-EXEC
```

インジケータ変数は検索条件では使用できません。例えば、次のような 埋め込み SQL 文は無効です。

```
EXEC SQL
   DELETE FROM closeoutsale
      WHERE temp_price = :saleprice:saleprice-null
END-EXEC
```

NULL 値の検索には is null 関数を使用できます。次に使用例を示します。

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

2.3.2 データの切り捨て

インジケータ変数は、上記とは別の用途に使用されることもあります。データベースからホスト変数にデータを取り込む際に、格納先のホスト変数のサイズを超過するデータは一部が切り捨てられます。このような切り捨てが行われると SQLCA データ構造体に警告フラグ (sqlwarn1) が設定され、インジケータ変数はデータベース内の切り捨て前のデータサイズに設定されます。

2.3.3 インジケータ配列

COBSQL
COBSQL を使用している場合は、ここに記述するインジケータ配列の説明は Oracle データベースだけに有効です。

インジケータ配列はインジケータ変数と同様、次の用途に使用できます。

- NULL 値の割り当て
- NULL 値の検出

- データの切り捨ての検出

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

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

第3章 データ型

SQL では、COBOL とは異なるデータ型を使用します。

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ソースごとに異なり、これらのデータ型をすべて実装することはほとんどありません。

3.1 データ型の変換

SQL 文を埋め込んだ COBOL プログラムでは、ホスト変数を COBOL プログラム変数としてだけでなく、SQL データベース変数としても使用するため、OpenESQL により COBOL のデータ型を適切な SQL のデータ型に変換する必要があります。データ型を正しく変換するには、ホスト変数と COBOL PICTURE 句を正しく宣言することが前提になります。また、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も必要です。

次のセクションでは、さまざまな SQL データ型と、対応するホスト変数の宣言方法を説明します。

COBSQL
Oracle で COBSQL を使用している場合、データベース エンジン が変換の一種を行って、COBOL データ型からデータベース データ型に変換することができます。通常、数字または整数データ型のホスト変数は次のように定義してください。

```
PIC S9(..)..COMP..
```

文字またはテキスト データ型は次のように定義してください。

```
PIC X(...).
```

Oracle では、特定のホスト変数にデータベース データ型を定義することができます。これはデータ型が複雑な場合に便利です。次のように行われます。

```
EXEC SQL

BEGIN DECLARE SECTION

END-EXEC.

*

* データ項目を Oracle データ型の DISPLAY として定義します

*

01 emp-comm pic s9(6)v99 DISPLAY SIGN LEADING SEPARATE
```

```
*

EXEC SQL

   VAR emp-comm IS DISPLAY(8,2)

END-EXEC.

EXEC SQL

   END DECLARE SECTION

END-EXEC.
```

ホスト変数のデータベース 型の定義についての詳細は、ご使用のデータベース ベンダが提供する COBOL プリコンパイラ マニュアルを参照してください。

3.2 整数データ型

3.2.1 TINYINT

SQL の 1 バイトの整数データ型。COBOL では PIC s9(4) COMP-5 と宣言します。

DB2
DB2 では TINYINT データ型をサポートしていません。

3.2.2 SMALLINT

SQL の 2 バイトの整数データ型。COBOL では BINARY、COMP、COMP-X、COMP-5、および COMP-4 を使用して宣言します。OpenESQL で処理できるのは、符号付き SMALLINT のみです。符号なしの SMALLINT はサポートされていません。

例えば、次の各定義を使用すると、ホスト変数を SMALLINT 型に直接変換することができます。

```
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

現在 OpenESQL は、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。

COBSQL
オラクルでは、ホスト変数を `shortint1` または `shortint2` として、または次のように定義するのが最適です。

```
03 shortint7    PIC S9(4) COMP-5.
```

これらは Oracle データ型の NUMBER(38) にマップされます。

3.2.3 INT

SQL の 4 バイトの整数データ型。COBOL では、BINARY、COMP、COMP-X、COMP-5 、および COMP-4 を使用して宣言します。OpenESQL で処理できるのは、符号付き INTのみです。符号なしの INT はサポートされていません。

例えば、次の各定義を使用すると、ホスト変数を INT 型変数に直接変換することができます。

```
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 では、符号付き整数をサポートしていますが、符号なし整数はサポートしていません。

COBSQL
Oracle では、整数ホスト変数を次のように `longint1`、`longint2` として、または次のように定義するのが最適です。

```
03 longint7    PIC S9(8) COMP-5.
```

これらは Oracle データ型の NUMBER(38) にマップされます。

3.2.4 BIGINT

SQL の 8 バイトの整数データ型。COBOL では PIC S9(18) COMP-3 と宣言します。

OpenESQL

OpenESQL では、SQL データ型 BIGINT から変換した値を格納するホスト変数として使用する COBOL データ項目の最大サイズは、PIC S9(18) です。ただし、BIGINT 型変数には、PIC S9(18) データ項目に格納可能な最大値を超過するデータが取り込まれる可能性があるため、コード内でデータの切り捨てをチェックする必要があります。

DB2

DB2 では BIGINT データ型をサポートしていません。

COBSQL

Oracle では BIGINT をサポートしていません。

3.3 文字列データ型

3.3.1 固定長文字列

ドライバによって最大長が定義された固定長文字列。SQL データ型では CHAR。COBOL では PIC X(*n*) と宣言します。*n* には 1 から 最大長 までの整数が入ります。

対応する COBOL での宣言例を次に示します。

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

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

COBSQL

Oracle では Oracle データ型 CHAR(*n*) にマップされます。サポートされる最大の固定長文字列は 255バイトです。

3.3.2 可変長文字列

OpenESQL および DB2

可変長文字列。SQL データ型で VARCHAR。COBOL では、次の 2 通りの方法で宣言できます。

- 固定長文字列 (PIC X(*n*))
- レベル 49 の基本項目 2 つから構成されるグループ項目。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 を使用して宣言します。もう 1 つの項目は PIC X(*n*) で宣言し、実際のデータを格納します。*n* には整数が入ります。

次に宣言例を示します。

```
03 varchar1.

   49 varchar1-len       pic 9(4) comp-5.

   49 varchar1-data      pic x(200).
```

```
03 Longvarchar1.

   49 Longvarchar1-len    pic 9(4) comp.

   49 Longvarchar1-data   pic x(30000).
```

SQL の CHAR、VARCHAR、または LONG VARCHAR 型変数にコピーされた文字列が、これらのデータ型に定義されたデータ長より長い場合には、変数に格納できない部分が切り捨てられ、SQLCA データ構造体の SQLWARN1 フラグが設定されます。一方、定義されたデータ長より短い文字列には、空白文字が付加されます。

COBSQL

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.
```

次に Oracle はデータ項目 USERNAME を次のようなグループ項目に展開します。

```
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 グループ名 USERNAME を使用してください。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.
```

これにより Oracle データ型 VARCHAR(*n*) または VARCHAR2(*n*) にマップされます。Oracle は、すべてのラージ文字列項目にデータ型 LONG を与えます。

3.4 概数データ型

REAL： 概数を格納する 32 ビット浮動小数点データ型。COBOL では COMP-2 として宣言します。

FLOAT/DOUBLE： 概数を格納する 64 ビット浮動小数点データ型。COBOL では COMP-2 で宣言します。

次に宣言例を示します。

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

For example:

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

OpenESQL
OpenESQL では、埋め込み SQL の単精度浮動小数点がサポートされていないため、32 ビットおよび 64ビット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 (REAL) を COMP-1 として、倍精度浮動小数点 (FLOAT または DOUBLE) を COMP-2 としてサポートします。

DB2
DB2 バージョン 2.1 のみが倍精度浮動小数点 (FLOAT または DOUBLE) を COMP-2 としてサポートします。

COBSQL
Oracle では 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 Oracle データ型 NUMBER にマップされます。

3.5 真数データ型

DECIMAL/NUMERIC：ドライバで指定された精度 (桁数) と小数部の桁数まで格納する真数データ型。

COBOL では、COMP-3、PACKED-DECIMAL、または NUMERIC USAGE DISPLAY で宣言します。

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 では、これらはデータ型 NUMBER(*p*,*s*) にマップされます。

3.6 日付および時刻データ型

COBOL には、日付データや時刻データ専用のデータ型はありません。したがって、SQL の日付列や時刻列は文字列に変換されます。

SQL のタイムスタンプ値を COBOL で出力するホスト変数を PIC X(*n*) と定義した場合、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、*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。

例えば、次のようになります。

```
1994-05-24 12:34:00.000
```

DB2
DB2 では、TIMESTAMP データ型の最長は 26 文字です。

COBSQL
Oracle データ項目には一意のデータ定義があり、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機能があります。これらの機能は次のとおりです。

- TO_CHAR

  Oracle の日付形式から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。

両機能は、変換する項目と日付、時刻、または日時マスクを指定し、データ項目に適用します。次に例を示します。

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

```
exec sql

  insert into emp (ename, TO_DATE(hiredate, 'DD-MM-YYYY'))

        values   (:ename, :hiredate)

end-exec.
```

これは Oracle データ型 DATE にマップされます。DATE データ型の詳細については、Oracle 「*SQL Language Reference Manual*」 を参照してください。これらの機能を Oracle SQL 文内で使用する方法について詳しく説明されています。

3.7 バイナリ データ型

OpenESQL

SQL には、BINARY、VARBINARY、および IMAGE の3種類のバイナリ データ型があります。これらのデータ型の変数は、COBOL では PIC X(*n*) として表されます。データ変換は実行されません。 データベースから取り込んだデータのサイズが格納先の COBOL フィールドより大きい場合は、フィールドに格納できない部分が切り捨てられ、SQLCA データ構造体の SQLWARN1 フィールドに "W" が設定されます。一方、データ長がフィールドより短い場合には、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 型の列にデータを挿入するには、動的 SQL 文を使用する必要があります。

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 を使用してください。IBM ODBC ドライバを使用している場合は、IBM 互換データ型の代わりに 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、非常に大きい列 (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。

COBSQL

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違いは、文字データにはコードセット変換が行われますが、バイナリ データには何も行われません。

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 の使用には制限があります。詳細に付いては Oracle のマニュアルを参照してください。

3.8 SQL - COBOL 間のデータ型変換の概要

次の表は、OpenESQL で SQL とCOBOL 間のデータ変換を行う場合に使用する対応関係を示します 。

SQL のデータ型	COBOL の PICTURE 宣言	備考
SQL_CHAR(*n*)	PIC X(*n*)	
SQL_VARCHAR(*n*)	PIC X(*n*)	
SQL_LONGVARCHAR	PIC X(*max*)	*max* = 32K
SQL_DECIMAL(*p*,*s*)	PIC 9(*p*-*s*)V9(S) COMP-3	*p* = 精度 (桁数)
		s = 小数部の桁数
SQL_NUMERIC(*p*,*s*)	PIC 9(*p*-*s*)V9(S) COMP-3	
SQL_SMALLINT	PIC S9(4) COMP-5	
SQL_INTEGER	PIC S9(9) COMP-5	
SQL_REAL	COMP-2	
SQL_FLOAT	COMP-2	
SQL_DOUBLE	COMP-2	
SQL_BIT	PIC S9(4) COMP-5	
SQL_TINYINT	PIC S9(4) COMP-5	
SQL_BIGINT	PIC S9(18) COMP-3	
SQL_BINARY(n)	PIC X(n)	
SQL_VARBINARY(*n*)	PIC X(n)	
SQL_LONVARBINA	PIC X(*max*)	
SQL_DATE	PIC X(10)	*yyyy-mm-dd*
SQL_TIME	PIC X(8)	*hh:mm:ss*
SQL_TIMESTAMP	PIC X(26)	*yyyy-mm-dd hh:mm:ss.ffffff*

第4章 カーソル

SELECT 文の実行結果として複数行のデータ (結果集合) を抽出するプログラムを書く場合は、カーソルを宣言して使用する必要があります。

カーソル機能を使用すると、複数行のデータを一度に取り出し、その結果集合内の指定位置で更新や削除を実行できます。カーソルと呼ぶ理由は、画面上のカーソルが現在位置を示すように、結果集合内の現在位置を示すためです。

次に示すコード例では、DECLARE 文でカーソル (Cursor1) を SELECT 文に関連付け、OPEN 文で Cursor1 をオープンします。カーソルをオープンすると、関連付けた SELECT 文が実行されます。続いて FETCH 文により 2 つの列 (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) に格納します。FETCH 文はループしており、取り込むデータがなくなるまで繰り返し実行されます。取り込み完了後、Cursor1 をクローズします。

```
EXEC SQL DECLARE Cursor1 CURSOR FOR

   SELECT au_fname, au_lname FROM authors

END-EXEC

...

EXEC SQL

   OPEN Cursor1

END-EXEC

...

perform until sqlcode not = zero

   EXEC SQL

      FETCH Cursor1 INTO :first_name,:last_name

   END-EXEC

      display first_name, last_name

end-perform

...

EXEC SQL
```

```
   CLOSE Cursor1
END-EXEC
```

4.1 カーソル宣言

カーソルを使用するには、先に宣言しておく必要があります。宣言には DECLARE CURSOR 文を使用し、この文で、カーソル名のほか、SELECT 文 (または PREPARE 文で定義した SELECT 文の名前) を指定します。

カーソル名には、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。例えば、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
```

この例の DECLARE 文では、入力ホスト変数 (last-name) を使用して SELECT 文を指定しています。OPEN 文でカーソル (Cur1) をオープンすると入力ホスト変数 last-name の値が読み取られ、指定された SELECT 文が実行されます。

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

この例では、DECLARE 文が PREPARE 文で定義した SELECT 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することができます。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、COBSQL は DECLARE CURSOR 文に対してアニメーション ブレークポイントを生成します。

4.1.1 オブジェクト指向 COBOL 構文

オブジェクト指向 (OO) プログラム内では、データ項目を宣言するのに有効な場所であればどこでもカーソルを宣言することができます。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することができますが、その場合には、カーソルをオブジェクトストレージで宣言する必要があります。

備考 : 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。

4.2 カーソル オープン

宣言したカーソルを使用するには、先にオープンしておく必要があります。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 文が PREPARE 文で定義した文を参照しており、この参照文にパラメータ マーカーが含まれる場合には、パラメータ マーカーの値を提供するホスト変数 (または SQLDA 構造体の名前) を OPEN 文で指定する必要があります。次に例を示します。

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

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

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

COBSQL
Oracle データベースでは、カーソルをクローズする前に再オープンし、SELECT 文を再評価することができます。ただし、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起こります。MODE プリコンパイラ 指令の詳細については、*「ORACLE プリコンパイラ・プログラマーズ・ガイド」* を参照してください。

4.3 カーソルによるデータの取り込み

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。次に例を示します。

```
perform until sqlcode not = 0

EXEC SQL

   FETCH Cur1 INTO :first_name

END-EXEC

end-perform
```

カーソルが結果集合の最後に到達すると、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。互いに異なる型のカーソル、カーソルが読めるロックされたデータ、およびカーソルがデータに置くロックの詳細については、後述の *「カーソル オプション」* の項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 に置かれる値に影響を与えます。MODE プリコンパイラ指令の詳細については、*「ORACLE プリコンパイラ・プログラマーズ・ガイド」* を参照してください。.

4.4 カーソルのクローズ

カーソルを使用したデータの取り込みが完了したら、使用したカーソルを CLOSE 文でクローズします。次に例を示します。

```
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は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詳細については、*「ORACLE プリコンパイラ・プログラマーズ・ガイド」* を参照してください。

COBSQL
カーソルをクローズすると、ORACLE クライアントはそのカーソルに関連する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。次のプリコンパイラ オプションがカーソルの割り当て解除を制御します。HOLD_CURSOR、MAXOPENCURSORS および RELEASE_CURSOR です。プリコンパイラ指令の詳細については、*「ORACLE プリコンパイラ・プログラマーズ・ガイド」* を参照してください。

4.5 カーソル オプション

OpenESQL
この項で述べるカーソル オプションについての説明は、OpenESQL だけに適用されます。

カーソルの動作やパフォーマンスは、次に挙げる OpenESQL 文を使用して調整することができます。

- SET SCROLLOPTION

  カーソルの結果集合に含まれる行の選択方法を指定します。

- SET CONCURRENCY

  並行アクセスの実行時には、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。並行制御を有効化するには、カーソルをオープンする前に SET CONCURRENCY 文を使用します。

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

4.6 UPDATE 文 : 位置づけ および DELETE 文 : 位置づけ

UPDATE 文 : 位置づけ および DELETE 文 : 位置づけは、検索条件句の代わりに WHERE CURRENT OF 句を記述し、カーソルと併用することができます。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は、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 SQL 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ことがあります。

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。

4.7 カーソルの使用

カーソルは大量のデータを扱う場合に便利ですが、カーソルを使用する際には留意する点があります。データ並行性、データ整合性、およびデータ一貫性です。

データの整合性を確保するために、データベース サーバーはいくつかのロック方法を実装することができます。ロックを取得しないデータ アクセスの型、共有ロックを取得する型、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することはできますが、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することができません。

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 は読込み専用カーソルだけに使用できます。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む可能性があります。コミットされていないデータを読むのは危険です（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 "ダーティー リード" と呼ばれます。データベースによっては、ダーティー リードは行えません。

- レベル1

  レベル 1 は、読込み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と、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、排他ロックをデータに置きます。

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。代表的な例は次の通りです。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。しかし、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 サーバーが問題を発見し、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 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。適用可能なロックの型と動作については、マニュアルに記述されています。マニュアルには、データがロックされる物理レベルについても記述されています。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スキャンするカーソルを使用する場合には、ロックされたデータのアクセス可能性を減少させるので、注意が必要です。

第5章 データ構造体

OpenESQL では、SQLCA と SQLDA の 2 つのデータ構造体を使用します。

5.1 SQL 通信領域

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情報が SQL 通信領域 (SQLCA) に格納されます。

SQLCA には 2 つの変数 (sqlcode および sqlstate) のほか、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。データベースとクライアント アプリケーション間のデータ交換の方法としては、将来は、SQLSTATE 変数の方が SQLCA よりも好ましい方法になりますが、これはまだ実現されていません。

5.1.1 sqlcode 変数

埋め込み SQL 文が正常に実行できたかどうかを確認する手段としては、sqlcode 変数の値をチェックする方法が最も一般的です。sqlcode には、次の表に一覧したいずれかの値が格納されます。

SQLCODE	メッセージ	意味/備考
0	空白	正常に実行されました。
0	パラメータ マーカーとホスト変数 (またはユーザー SQLDA) の数が一致しません	CONNECT など特殊な埋め込み SQL 文のみに使用されます。
1	空白	警告。SQLWARN をチェックしてください。
-1	自動接続に失敗しました	SQL(INIT) が使用され、CONNECT の自動実行でエラーが発生しました。SQL(INIT) を使用するプログラムは、起動後ただちに SQLCODE をチェックする必要があります。
100	これ以上該当する行はありません	FETCH 文でこれ以上データを取り込めない場合の戻り値

SQLCODE	メッセージ	意味/備考
10000	(OpenESQL ランタイムモジュールによるエラー検出後に、ODBC 呼び出しの SQLERROR から最初に戻されるエラーメッセージ)	ODBC エラーメッセージの詳細については、ODBC ドライバのマニュアル (または他の ODBC リファレンス) を参照してください。SQLSTATE には、ODBC のエラー状況によって異なるコードが格納されます。
	ODBC エラーを取り込めません	ODBC エラーが発生しましたが、エラーの詳細情報が取り込めません。通常、極端なメモリ不足など、実行時に重大な問題が発生したことを示します。
-19085	ODBC カタログ問合せが無効です	QUERY ODBC 文のパラメータが無効です。
19101	文が長すぎます	
19199	PREPARE 文 (または EXECUTE IMMEDIATE 文) で ESQL のキーワードが検出されました	
19313	ホスト変数が不足しています	
19413	データの 10 進変換でオーバーフローが発生しました	
19501	宣言されたカーソルがありません	
19514	カーソルが定義されていません	
-19701	接続名が NULL かまたは見つかりません	このエラー (-19701) と次のエラー (-19702) は、プログラムが、存在しない接続を参照したときに発生します。原因としては、CONNECT 文が正常に実行される前 (またはすべての接続が解除された後) に埋め込み SQL 文の実行を試みた場合などが挙げられます。
-19702	接続名が見つからないか、または存在しない接続を解除しようとしました	上記の説明 (-19701) を参照してください。
19703	接続を確立できませんでした	
19707	接続名が重複しています	

SQLCODE	メッセージ	意味/備考
19822	ユーザー SQLDA の初期化が不適切です	
19957	文のテキストが見つからないか、または空白です	
-20000	この埋め込み SQL の機能はサポートされていません	COBOL コンパイラで正常に処理される埋め込み SQL 構文の中には、OpenESQL ランタイムモジュールでサポートされていない構文が含まれる可能性があります。そのような構文を含む文を実行しようとすると、このエラーが発生します。

COBSQL および DB2

COBSQL と DB2 では、他にも正の SQLCODE 値を出力することがあります。これは、SQL 文を実行したが警告を生成した、ということを意味します。

COBSQL

SQLCODE を設定できる正の値の範囲の詳細については、ORACLE の エラー メッセージ マニュアルを参照してください。

COBSQL

値 +100 は「データが見つからない」の ANSI 標準です。Oracle では「データが見つからない」に違う値を戻すことがあります。Oracle で、「データが見つからない」について値 +100 を戻すには、Oracle プリコンパイラ 指令 MODE=ANSI 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 文を扱う方法にも影響を与えます。Oracle プリコンパイラ MODE 指令の詳細については、*「ORACLE プリコンパイラ・プログラマーズ・ガイド」* を参照してください。

COBSQL

SQLCODE がゼロの場合でも、警告が生成された可能性があります。`sqlwarn` フラグの値をチェックして、警告のタイプを判別してください。Oracle では、データベース サーバーがアプリケーションに警告を戻したときには、常に `sqlwarn0` がセットされます。

5.1.2 sqlstate 変数

SQLSTATE 変数

DB2

DB2 ユニバーサル・データベースでは SQL-92 準拠の SQLSTATE 値を戻します。DB2 バージョン 2.1 では異なります。

`sqlstate` 変数は SQL-92 標準に導入され、将来のアプリケーションの推奨機構です。`sqlstate` 変数は、次の 2

つの構成からなっています。

- 最初の 2 文字は `class` 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 標準またはその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。

SQLSTATE 値の詳細は、オンライン ファイルに記述されています。ヘルプ ファイル索引の "SQLSTATE" を参照してください。

備考　関数が正常終了したかどうかは、通常、SQLCODE の値 (正常終了の場合 0) で確認できますが、SQLSTATE の値 (正常終了の場合 00000) で確認することも可能です。

SQLSTATE	エラー
01000	一般的な警告
01002	接続解除エラー
01004	データの一部が切り捨てられました
01006	権限が無効化されていません
01S00	接続文字列属性が無効です
01S01	行にエラーが含まれています
01S02	オプション値が変更されました
01S03	更新または削除された行はありません
01S04	複数の行が更新または削除されました
01S05	取り消しは SQL_CLOSE オプションを指定した SQLFreeStmt として処理されました
01S06	結果集合から最初の行セットが戻る前にデータを取り込もうとしました

SQLSTATE	エラー
07001	パラメータ数が正しくありません
07006	制約付きデータ型属性違反です
07S01	使用するデフォルトパラメータが無効です
08001	データソースに接続できません
08002	接続が使用中です
08003	接続が閉じています
08004	データソースによって接続が拒否されました
08007	トランザクション処理中に接続エラーが発生しました
08S01	通信リンクエラー
21S01	挿入値リストが列リストと合致しません
21S02	導出表の次数が列リストと合致しません
22001	文字列データの右端が切り捨てられました
22002	必要なインジケータ変数が記述されていません
22003	数値データが有効範囲を超えています
22005	代入エラー
22008	日時フィールドでオーバーフローが発生しました
22012	ゼロ除算エラー
22026	文字列データの長さが一致しません

SQLSTATE	エラー
23000	整合性制約違反です
24000	カーソル状態が無効です
25000	トランザクション状態が無効です
28000	権限指定が無効です
34000	カーソル名が無効です
37000	構文エラーまたはアクセス違反が検出されました
3C000	カーソル名が重複しています
42000	構文エラーまたはアクセス違反が検出されました
70100	操作が取り消されました
IM001	ドライバがサポートしない関数です
IM002	データソース名が見つからず、デフォルトドライバが指定されていません
IM003	指定されたドライバをロードできませんでした

SQLSTATE	エラー
IM004	ドライバの SQLAllocEnv でエラーが発生しました
IM005	ドライバの SQLAllocConnect でエラーが発生しました
IM006	ドライバの SQLSetConnect オプションでエラーが発生しました
IM007	データソースまたはドライバが指定されていません。ダイアログは許可されませんでした
IM008	ダイアログエラー
IM009	変換用の .DLL ファイルをロードできませんでした
IM010	データソース名が長すぎます
IM011	ドライバ名が長すぎます
IM012	DRIVER キーワードの構文エラー
IM013	トレースファイル エラー
S0001	実表またはビュー表はすでに存在します
S0002	実表が見つかりません
S0011	索引はすでに存在します
S0012	索引が見つかりません
S0021	列はすでに存在します
S0022	列が見つかりません
S0023	列のデフォルトがありません
S1000	一般エラー
S1001	メモリ割り当てエラー

SQLSTATE	エラー
S1002	列番号が無効です
S1003	プログラムタイプが有効範囲を超えています
S1004	SQL データ型が有効範囲を超えています
S1008	操作が取り消されました
S1009	引数の値が無効です
S1010	関数シーケンスエラー
S1011	今回の操作は無効です
S1012	指定されたトランザクション処理コードが無効です
S1015	カーソル名がありません
S1090	文字列またはバッファの長さが無効です
S1091	記述子のタイプが有効範囲を超えています
S1092	オプションのタイプが有効範囲を超えています
S1093	パラメータ番号が無効です
S1095	関数のタイプが有効範囲を超えています
S1096	情報のタイプが有効範囲を超えています
S1097	列のタイプが有効範囲を超えています
S1098	範囲のタイプが有効範囲を超えています
S1099	NULLABLE タイプが有効範囲を超えています
S1100	UNIQUE オプションのタイプが有効範囲を超えています
S1101	精度オプションのタイプが有効範囲を超えています
S1103	方向オプションが有効範囲を超えています

SQLSTATE	エラー
S1105	パラメータのタイプが無効です
S1106	取り込みタイプが有効範囲を超えています
S1107	行の値が有効範囲を超えています
S1108	並行オプションが有効範囲を超えています
S1109	カーソルの位置が無効です
S1110	ドライバの完了方法が無効です
S1111	ブックマーク値が無効です
S1C00	無効なドライバです
S1T00	タイムアウトになりました

5.1.3 警告フラグ

文を実行すると警告が生成されることがあります。警告のタイプを確認するには、アプリケーションで sqlwarn フラグの値を確認する必要があります。このフラグには、特定の警告が発生すると「W」、その他の場合には空白文字が設定されます。

5.1.4 WHENEVER 文

埋め込み SQL 文を実行するたびに sqlcode (または sqlstate) の値を明確に確認するには、非常に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 文のステータスをチェックします。

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、次の各状況に対して 3 つのデフォルトアクション (CONTINUE、GOTO、PERFORM) のいずれかを登録できます。

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

WHENEVER 文にデフォルトアクションを設定すると、それまでに同じ状況に対して設定したすべての WHENEVER 文は無効になります。

COBSQL
Oracle では、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。

COBSQL
Oracle では、`sqlwarn0` が 'W' に設定されると 'SQLWARNING' 句がトリガされます。

ある特定の条件に対する WHENEVER 文は、その条件に対する以前の WHENEVER 文 をすべて置き換えます。

WHENEVER 文のスコープは、実行シーケンス内の論理位置ではなく、ソース 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、パラグラフ C ではなく パラグラフ 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 "First item not found".
B.
   EXEC SQL
```

```
      WHENEVER NOT FOUND PERFORM C.
   END-EXEC.
C.
   DISPLAY "Second item not found".
```

5.2 SQL 記述子領域 (SQLDA)

SQLDA は各プリコンパイラに固有です。Oracle SQLDA は、OpenESQL や DB2 で使用されている SQLDA とは互換性がなく、その逆も互換性がありません。

COBSQL
Oracle では、プログラムで動的 SQL を使用している場合だけに SQLDA が必要です。

COBSQL
Oracle では、プログラムに次のような構文を使用して SQLDA を記述することはできません。

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

SQLDA は、標準 COBOL コピーファイルとして定義する必要があります。

COBSQL
Oracle ではそのほかに ORACA というコピーファイルが、動的 SQL で使用できます。このコピーファイルをプログラムに使用するには、次の構文を使用してください。

```
EXEC SQL
   INCLUDE ORACA
END-EXEC
```

ORACA を使用する前には、Oracle プリコンパイラ オプションで ORACA=YES を設定してください。Oracle プリコンパイラ オプション設定の詳細については、*「ORACLE プリコンパイラ・プログラマーズ・ガイド」* を参照してください。

COBSQL
Oracle では SQLDA を提供していませんが、レイアウトの定義については *「ORACLE プリコンパイラ・プログラマーズ・ガイド」* で説明されています。

指定するパラメータの数やデータ型がコンパイル時点で不明な場合には、ホスト変数の代わりに SQL 記述子領域 (SQLDA) を使用できます。

SQLDA には、入力パラメータや出力列の詳細情報が格納されます。この情報には、各入力パラメータ (または出力パラメータ) に対応する列の名前、データ型、データ長、および実際のデータバッファへのポインタが含まれます。SQLDA は、通常パラメータマーカーと併用して定義済みの SQL 文への入力値を指定しますが、DESCRIBE 文 (またはPREPARE 文の INTO オプション) と併用して、定義済みの SELECT 文からデータを受け取ることもできます。

SQLDA は静的 SQL 文とは併用できませんが、カーソル FETCH 文との併用は可能です。

OpenESQL
SQLDA 構造体は、COBOL システムのインストールディレクトリ内の source サブディレクトリに含まれる sqlda.cpy ファイルに記述されており、COBOL プログラムのデータ部 (DATA DIVISION) に次の文を記述するとインクルードできます。

```
EXEC SQL

   INCLUDE SQLDA

END-EXEC
```

SQLDA の詳細については、5.2.3 SQLDA データ構造体を参照してください。

5.2.1 SQLDA の使用方法

SQLDA 構造体を使用するには、その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	SQLDA 構造体が保持できる SQLVAR エントリの最大数に設定します。
SQLDABC	SQLDA の最大サイズ。 SQLN * 44 + 16 で計算します。

5.2.1.1 PREPARE 文および DESCRIBE 文

DESCRIBE 文 (または INTO 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列の名前やデータ型などの情報を入力することができます。

これらの文を実行するには、その前に SQLN フィールドと SQLDABC フィールドを上記のように初期化する必要があります。

文を実行すると、定義済み文のパラメータ数が SQLD フィールドに格納されるほか、各パラメータに対応する SQLVAR レコードの SQLTYPE と SQLLEN に値が格納されます。

SQLN フィールドの値を特定できない場合には、SQLN を 1、SQLD を 0 に設定して DESCRIBE 文を実行することも可能です。この場合、SQLDA 構造体に列の詳細情報を転記できませんが、結果集合に含まれる列数が SQLD に挿入されます。

5.2.1.2 FETCH 文

SQLDA 構造体を使用して FETCH 文を実行するには、その前にアプリケーション内で上記のように SQLN フィールドと SQLDABC フィールドを初期化し、さらに列データの格納先とな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 レコードの一部です。) インジケータ変数を使用する場合には、SQLIND を対応するインジケータ変数のアドレスに設定する必要があります。

データ型フィールド (SQLTYPE) とデータ長フィールド (SQLLEN) には、PREPARE INTO 文 (または DESCRIBE 文) で取り込まれた値が格納されます。これらの値は FETCH 文の実行前にアプリケーション内で上書きすることが可能です。

5.2.1.3 OPEN 文と 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値を設定する必要があります。この場合、各変数の SQLN、SQLD、SQLDABC、SQLTYPE、SQLLEN、SQLDATA フィールドなども含めて設定します。SQLTYPE フィールドの値が奇数の場合、SQLIND フィールドにインジケータ変数のアドレスを設定します。

5.2.2 DESCRIBE 文

PREPARE 文の後に、 DESCRIBE 文を実行すると、定義された文によって戻された列のデータ型、データ長、および名前を列ごとに取り込むことができます。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 INTO オプションを使用すると、両方の操作を同時に実行することができます。

```
EXEC SQL

   PREPARE stmt1 INTO :sqlda FROM :stmtbuf

END-EXEC
```

5.2.3 SQLDA データ構造体

SQLDA 構造体の構成を次に示します。

```
01 SQLDA sync.

    05 SQLDAID           PIC X(8) VALUE "SQLDA  ".
```

```
05 SQLDABC           PIC S9(9) COMP-5 value 0.
05 SQLN              PIC S9(4) COMP-5 value 0.
05 SQLD              PIC S9(9) COMP-5 value 0.
05 SQLVAR OCCURS 0 to 1489 TIMES DEPENDING ON SQLD.
   10 SQLTYPE        PIC S9(4) COMP-5.
   10 SQLLEN         PIC S9(4) COMP-5.
   10 SQLDATA        USAGE POINTER.
   10 SQLIND         USAGE POINTER.
   10 SQLNAME.
      15 SQLNAMEL PIC S9(4) COMP-5.
      15 SQLNAMEC PIC X(30).
```

次の表は、SQLDA 構造体の内容を示しています。

フィールド	内容
SQLDAID	文字列「SQLDA」
SQLCABC	SQLDA 構造体の長さ (SQLN * 44 + 16 で計算します)
SQLN	割り当てた SQLVAR エントリの合計数。入力パラメータ (または出力列) と同数になります。
SQLD	使用する SQLVAR エントリ数
SQLVAR	グループ項目。SQLVAR のレコード数は SQLD の値によって決定されます。
SQLTYPE	列またはホスト変数のデータ型を表す数値。および NULL 値を使用できるかどうかを示す数値 (有効値については、次のセクションを参照してください)。
SQLLEN	列から取り込まれる値のデータ長。データが 10 進数型の場合 (金額を含む)、SQLLEN は、2 つの部分に分けられ、最初のバイトに精度、次のバイトに小数点以下の桁数が格納されます。
SQLDATA	FETCH、OPEN、および EXECUTE のホスト変数のアドレス。アプリケーションで設定しま

	す。DESCRIBE および PREPARE 文には、SQLDATA は使用しません。
SQLIND	FETCH、OPEN、および EXECUTE に関連づけられたインジケータ変数がある場合、そのアドレスを設定します。NULL 値を設定できない列については、このフィールドを定義しません。NULL 値を設定できる列については、データが NULL の場合 SQLIND の値は -1、NULL 以外の場合 0 になります。DESCRIBE および PREPARE 文では、SQLIND は使用しません。
SQLNAME	列の名前フィールドとデータ長フィールドから構成されるグループ項目。FETCH、OPEN、および EXECUTE では使用されません。
SQLNAMEL	名前の列のデータ長
SQLNAMEC	列の名前。導出列の場合には、選択リスト内での元の位置を示す ASCII の数字定数が格納されます。

5.2.3.1 SQLTYPE の有効値

奇数のコード値は、NULL 値が設定できることを示します。

(1) COBOL プログラム内で PREPARE INTO (または DESCRIBE) 文を実行して取り込むことができるデータ型

(2) 動的 SQL を使用して、アプリケーションで設定できるデータ型

(3) COBOL ホスト変数用にサポートされているデータ型

コード	データ型	SQL データ型	COBOL データ型
384/385 (2)	日付文字列 (10 バイト)	*date*	PIC X(10)
386/387 (1)	日付	*date*	
388/389 (2)	時刻文字列 (8 バイト)	*time*	PIC X(8)
390/391 (1)	時刻	time	
392/393 (2)	タイムスタンプ文字列 (26 バイト)	*timestmp*	PIC X(26)
394/395 (1)	タイムスタンプ	timestamp	

コード	データ型	SQL データ型	COBOL データ型
404/405 (1)	可変長ラージバイナリ	*Long varbinary*	01 NAME
			49 PIC LEN S9(9) COMP-5
			49 PIC VAL X(n)
408/409 (1)	可変長ラージ文字列	*Long varchar*	01 NAME
			49 PIC LEN S9(9) COMP-5
			49 PIC VAL X(n)
444/445 (1,2)	バイナリ	*Binary*	PIC X(n)
446/447 (1,2)	可変長バイナリ	*varbinary*	01 NAME
			49 PIC LEN S9(9) COMP-5
			49 PIC VAL X(n)
452/453 (1,2,3)	固定長文字列	*char*	PIC X(n)
480/481 (1,2,3)	浮動小数点 (8 バイト)	*float*/*double*	COMP-2
482/483 (1,2)	浮動小数点 (4 バイト)	*real*	COMP-1
484/485 (1,2,3)	10 進数	*decimal*,*numeric*/*bigint*	PIC 9(n)V9(m) COMP-3
496/497 (1,2,3)	整数 (4 バイト)	*Integer*	PIC S9(9) COMP-5
500/501 (1,2,3)	整数 (2 バイト)	*Smallint*	PIC S9(5) COMP-5
502/503 (1,2)	整数 (1 バイト)	*tinyint*	PIC S9(9) COMP-5

第6章 動的 SQL

アプリケーションのコンパイル時に、ソースコードに完全に記述されている SQL 文を静的 SQL 文と呼びます。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。例えば、アプリケーションの実行時に、エンドユーザーが任意の SQL 文を入力する場合などです。このような文を動的 SQL 文と呼び、アプリケーションの実行時に組み立てます。

動的 SQL 文には、次のように 4 つの型があります。

1. 文を一度実行する。
2. 同じ文を複数回実行する。
3. 既定の選択基準を使用して、既定のデータのリストを選択する。
4. いづれかの選択基準を使用して、いづれかの量のデータを選択する。

クエリーは、3 と 4 の型を使用して処理します。1 および 2 の型ではデータは戻されません。実行結果の成功、または失敗だけが戻されます。

1 の型では、動的 SQL 文は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。

2 の型の動的 SQL 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。2 の型の場合は、文を準備してから実行する必要があります。

3 の型の動的 SQL は、ホスト変数の数と型が分かっている SELECT 文です。はじめに文を準備して、結果を保持するためにカーソルを宣言します。次に、カーソルをオープンし、カーソルからデータが取り出され、最後にカーソルをクローズします。

4 の型の動的 SQL はコードするのが一番難しく、変数の型が実行時にのみ解決される、または変数の数が実行時にのみ解決される、またはこの両方です。この型の SQL 文のシーケンスは、通常、次の通りです。文を準備し、文にカーソルを宣言します。次に、使用する変数を記述し、記述した変数を使用してカーソルをオープンします。次に、取り出す変数を記述し、その記述内容を使用して変数を取りだして、最後に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 はホスト変数を指名できるので、ホスト変数を記述する必要はありません。

6.1 動的 SQL 文の定義

動的 SQL 文を実行するには、先に PREPARE 文で定義する必要があります。

PREPARE 文では、動的 SQL 文を含む文字列を記述し、この文に名前を関連付けます。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、値の代わりにパラメータマーカー (?) 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、文の実行時に指定します。

PREPARE 文で定義した SQL 文は、次のいずれかの方法で実行します。

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

COBSQL
Oracle では、プレース ホルダとして疑問符を使用しません。ホスト変数注釈を使用します。便宜上、プレース ホルダは V*n* と名づけられ、 *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。読みやすいように、同じプレース ホルダを複数回使用することができますが、文の実行時 (またはカーソルを使用している場合は、オープン時) には各プレース ホルダに対してホスト変数が 1 つづつ必要です。次に例を示します。

```
string "update ordtab " delimited by size

         "set order_no = :v1, "

            "line_no = :v2, "

            "cust_code = :v3, "

            "part_no = :v4, "

            "part_name = :v5, "

            "order_val = :v6, "

            "pay_value = :v7 "

         "where order_no = :v1 and "

            "line_no = :v2 and "

            "cust_code = :v3 " delimited by size

       into Updt-Ord-Stmt-Arr

end-string
```

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


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


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

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

COBSQL
Oracle プリコンパイラを使用する場合は、PREPARE 文の物理的位置が重要です。PREPARE 文は、EXECUTE 文または DECLARE 文よりも前に記述しなければなりません。

6.2 動的 SQL 文の実行

PREPARE 文で定義した SQL 文は、EXECUTE 文で個別に実行できます。

備考 : EXECUTE 文では、結果を戻す SQL 文は実行できません。

定義済み文にパラメータマーカーが含まれている場合、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値を指定するか、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。定義済みの文に含まれるパラメータマーカー数は、SQLDA 構造体のメンバー数 (using descriptor :sqlda) またはホスト変数の数 (using :hvar) と一致する必要があります。

```
move "INSERT INTO publishers VALUES (?,?,?,?)" to stmtbuf

EXEC SQL
   PREPARE stmt1 FROM :stmtbuf
```

```
END-EXEC

...

EXEC SQL

   EXECUTE stmt1 USING :pubid,:pubname,:city,:state

END-EXEC.
```

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

6.2.1 ダイレクト実行

パラメータ マーカーを含まない動的 SQL 文は、PREPARE と EXECUTE を順次実行する代わりに EXECUTE IMMEDIATE を使用すると、すぐに実行できます。次に例を示します。

```
move "DELETE FROM emp WHERE last_name = 'Smith'" to stmtbuf

EXEC SQL

   EXECUTE IMMEDIATE :stmtbuf

END-EXEC
```

COBSQL
EXECUTE IMMEDIATE を使用すると、文が実行されるたびに構文検査が再度行われます。文が何回も使用されるような場合には、文を PREPARE して、必要なときに EXECUTE するほうが効率的です。

6.3 動的 SQL 文とカーソル

結果を戻す動的 SQL 文は、EXECUTE 文では実行できません。この場合、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。次に例を示します。

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

次に、OPEN 文でカーソルをオープンすると、PREPARE で定義した文が実行されます。

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

PREPARE で定義した文にパラメータ マーカーとして疑問符 (?) が含まれている場合、対応するパラメータ値を提供するホスト変数 または SQLDA 構造体 を OPEN 文で指定する必要があります。

カーソルを開くと、FETCH 文を使用してデータを取り出すことができます。次に例を示します。

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

FETCH 文の詳細については、*「カーソル」* の章を参照してください。

最後に CLOSE 文を使用してカーソルをクローズします。

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

CLOSE 文の詳細については、*「カーソル」* の章を参照してください。

第7章 SQL言語リファ レンス

この章は各SQL埋め込み文のリファレンスです。 すべてのSQL埋め込み文の前にはキーワードEXEC SQL、後にはキーワードEND-EXECがそれぞれ必要です。 この章では次の文を取り扱います。

文	説明
BEGIN DECLARE SECTION	ホスト変数宣言節の始めを示す。
CALL	ストアド プロシージャを実行する。
CLOSE	OPEN文で始まった指定されたカーソルに対する一度に1行ずつの検索を終わりにする。
COMMIT	現行トランザクション中で行われた更新を永続的にする。
CONNECT TO	与えられたユーザ名とパスワードを使って特定のデータベースと接続する。
DECLARE CURSOR	一度に1行ずつのデータ検索のためにカーソルを定義する。
DECLARE DATABASE	データベースの名前を宣言する。
DELETE (位置付け)	カーソルが現在ある位置の行を取り除く。
DELETE (検索)	標準SQL文。
DESCRIBE	SQLDAデータ構造に書き込む。
DISCONNECT	1つまたはすべてのデータベースとの接続を取り消す。
END DECLARE SECTION	ホスト変数宣言節の終わりを示す。
EXECSP	ストアド プロシージャを実行する。
EXECUTE	準備するSQL文を実行する。
EXECUTE IMMEDIATE	指定されたホスト変数に入っているSQL文を実行する。
FETCH	指定されたカーソルについて、結果セットから次の行を獲得する。

文	説明
INCLUDE	特定のSQLデータ構造をアプリケーションで使用するために定義する。
INSERT	データベースに行を追加するための標準SQL文。
OPEN	指定されたカーソルについて一度に1行のデータ検索を始める。
PREPARE	ホスト変数に入っている文字列からSQL文を受け付け、文を名前と関連付ける。
QUERY ODBC	SELECT文と同様に、結果セットを渡す。
ROLLBACK	現行トランザクションでなされた更新をキャンセルする。
SELECT DISTINCT (DECLARE CURSOR 使用)	FETCH文を使用したデータ行の抽出を可能にする。
SELECT INTO	結果の1行を検索する（単選択とも呼ばれる）。
SET CONCURRENCY	標準モードカーソルの同時性オプションを指定する。
SET CONNECTION	以後のSQL文に使用するデータベース接続を指定する。
SET OPTION	クエリー処理オプションの値を指定する。
SET SCROLLOPTION	以後の標準モードカーソル宣言のためのスクロール方法を指定する。
UPDATE (位置付け)	カーソルが現在ある位置の行を変更する。
UPDATE (検索)	標準SQL。
WHENEVER	SQLの実行後に取るべきデフォルトアクションを指定する。

注. SQL埋め込み文のキーワードの大文字と小文字はプログラムの中では区別されません。 例えば、EXEC SQL CONNECT TO、exec sql connect to、Exec Sql Connect Toは等価です。 カーソル名、文名、接続名の大文字と小文字の区別は、変数の宣言に使用される大文字と小文字の区別と同じです。 例えば、カーソルをC1と宣言した場合、変数c1は固有の変数として取り扱われます。 テーブル名や列名などの他の語の大文字と小文字を区別するかどうかは、サーバの設定によって決まります。

7.1 BEGIN DECLARE SECTION

ホスト変数宣言節の始めを示します。

構文

```
BEGIN DECLARE SECTION
```

コメント

- BEGIN DECLARE SECTION文は、COBOLの変数宣言ができる場所ならどこでも使用でき、COBOLにおける通常の範囲規定に従います。COBOLの宣言節の終わりを識別するにはEND DECLARE SECTIONを使用します。
- 宣言節は入れ子にできません。
- 宣言した変数は次の規則に従ってください。
- 変数は、SQLではなくCOBOLで宣言しなければなりません。
- SQL文によって参照された変数は、（SQLコンパイラ指令のDB2オプションを使用しないかぎり）そのSQL文より前の宣言節でインクルードしなければなりません。
- 衝突を防ぐために、宣言節の中の変数は、たとえコンパイル単位が異なっていても、外部の宣言節やその他の宣言節と同じであってはなりません。
- 宣言節の中でデータ構造が定義されている場合、ホスト変数には最下レベルの項目（PIC句付き）だけが使用できます（SQLコンパイラ指令のDB2オプションを指定した場合、COBOLレコード構造をホスト変数として使用できます）。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC

01 emp-id       pic x(4).

01 emp-name     pic x(30).

EXEC SQL END DECLARE SECTION END-EXEC
```

7.2 CALL

ストアド プロシージャの実行に使用されます。

構文

```
{FOR :row_count} {:result_hvar =}
```

```
CALL stored_procedure_name {(parameter[,...])}
```

ただし

row_count

整数型のホスト変数。結果およびパラメータのホスト変数がすべて同じサイズの行数で、すべての行を使用するとは限らない場合は、使用される行数を指定します。CALLがDECLARE CURSOR文の一部である場合は、FOR句を使用することはできません。

result_hvar

手続きの結果を受け取るホスト変数。

stored_procedure_name

ストアド プロシージャの名前。

parameter

リテラル定数、キーワードCURSORまたはフォームのホスト変数パラメータです。

[keyword=] :param_hvar [IN | INPUT | INOUT | OUT | OUTPUT]

ここで、

Keyword	キーワードパラメータ用の正式なパラメータです。
param_hvar	ホスト変数です。
IN	入力パラメータを指定します。
INPUT	入力パラメータを指定します。
INOUT	入力/出力パラメータを指定します。
OUT	出力パラメータを指定します。
OUTPUT	出力パラメータを指定します。

CURSORは、結果セットを返すOracle 8ストアド プロシージャに対してだけ使用されます。CURSORを使用すると、対応するパラメータがアンバウンドされます。

ストアド プロシージャの詳細については、オンラインブックを参照してください。

例

```
EXEC SQL
```

```
   CALL myProc(param1,param2)
END-EXEC

EXEC SQL
   :myResult = CALL myFunction(namedParam=:paramValue)
END-EXEC

EXEC SQL
   CALL getDept(:empName IN, :deptName OUT)
END-EXEC

EXEC SQL
   DECLARE cities CURSOR FOR CALL locateStores(:userState)
END-EXEC
```

7.3 CLOSE

OPEN文によって始まった指定されたカーソルに対する一度に1行のデータ検索を終わりにし、カーソル接続を閉じます。

構文

```
CLOSE cursor_name
```

ただし

cursor_name

前に宣言され開かれたカーソルを指定します。

コメント

- CLOSE文は未処理の行を廃棄し、カーソルが持っていたロックを解放します。カーソルは、閉じる前に宣言し開いておかなければなりません。すべての開いたカーソルは、プログラムの終了時に自動的に閉じます。

例

```
EXEC SQL
DECLARE C1 CURSOR FOR
     SELECT ID, NAME, DEPT, JOB, YEARS, SALARY, COMM
         FROM STAFF
END-EXEC


EXEC SQL
    OPEN C1
END-EXEC


perform until sqlcode not = zero
*   SQLCODE will be zero as long as it has successfully fetched data
  EXEC SQL
      FETCH C1 INTO :idnum, :nme, :dept, :job,
                 :years, :salary, :comm
  END-EXEC
  if sqlcode = zero
    display idnum, nme, dept, job, years, salary, comm
  end-if
end-perform.


EXEC SQL
   CLOSE C1
END-EXEC.
```

7.4 COMMIT

現行トランザクション中でなされた更新をすべてデータベース中で永続的に更新します。

構文

```
COMMIT  [{WORK | TRAN | TRANSACTION}]  [RELEASE]
```

ただし

- WORK、TRAN、TRANSACTIONは書いても書かなくても影響はありません。構文チェックのみなされます。
- RELEASEが書かれていて、トランザクションが正常にCOMMITされると、現在の接続は切断されます。

例

```
* Ensure that multiple records are not inserted for a
* member of staff whose staff_id is 99
 EXEC SQL
    DELETE FROM staff WHERE staff_id = 99
 END-EXEC

* Insert dummy values into table
 EXEC SQL
    INSERT INTO staff
    (staff_id
    ,last_name
    ,first_name
    ,age
    ,employment_date)
    VALUES
    (99
    ,'Lee'
```

```
      ,'Phil'
      ,19
      ,'1992-01-02')
    END-EXEC

    IF SQLCODE NOT = ZERO
      DISPLAY 'Error: Could not insert dummy values.'
      DISPLAY SQLERRMC
      DISPLAY SQLERRML
      EXEC SQL DISCONNECT ALL END-EXEC

      STOP RUN
    END-IF

    EXEC SQL
      COMMIT
    END-EXEC

   * Check it was committed OK
    IF SQLCODE = ZERO
      DISPLAY 'Error: Could not commit values.'
      DISPLAY SQLERRMC
      DISPLAY SQLERRML
      EXEC SQL DISCONNECT CURRENT END-EXEC
      STOP RUN
```

```
 END-IF

 DISPLAY 'Values committed.'

* Delete previously inserted data
 EXEC SQL
    DELETE FROM staff WHERE staff_id = 99
 END-EXEC

 IF SQLCODE NOT = ZERO
    DISPLAY 'Error: Could not delete dummy values.'

    DISPLAY SQLERRMC
    DISPLAY SQLERRML
    EXEC SQL DISCONNECT ALL END-EXEC
    STOP RUN
 END-IF

* Check data deleted OK, commit and release the connection
 IF SQLCODE NOT = ZERO
    DISPLAY 'Error: Could not delete values.'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
    EXEC SQL DISCONNECT ALL END-EXEC
    STOP RUN
```

```
    END-IF

    EXEC SQL
       COMMIT WORK RELEASE
    END-EXEC

   * Check data committed OK and release the connection.
    IF SQLCODE NOT = ZERO

       DISPLAY 'Error: Could not commit and release.'
       DISPLAY SQLERRMC
       DISPLAY SQLERRML
       EXEC SQL DISCONNECT CURRENT END-EXEC
    END-IF

    DISPLAY 'Values committed and connection released.'
```

7.5 CONNECT

与えられたユーザ名とパスワードを使って特定のデータベースに接続します。

構文

```
CONNECT TO [data source] [AS db_name]
  USER [user [.password] [WITH [NO] PROMPT]
CONNECT user [{IDENTIFIED BY password | '/' password]
  [AT db_name] [USING data source] [WITH [NO] PROMPT]
CONNECT WITH PROMPT
CONNECT RESET [db_name]
```

ただし

data source

ODBCデータソースの名前を指定します。データソースを省略した場合、デフォルトのODBCデータソースが使用されます。データソースは文字定数またはホスト変数で指定できます。

name

接続の名前を指定します。接続名は最大30文字で、英数字とファイル名に有効な任意の記号が使用できます。 最初の文字は数字以外の文字でなければなりません。 接続名にはSQL埋め込みのキーワード、CURRENT、ALLを使用しないでください。これらの名前は無効です。

user

指定したデータソースに有効なユーザーIDを指定します。

password

指定したユーザーIDに有効なパスワードを指定します。

RESET

RESETを指定すると、指定された接続を切断します。CURRENT、DEFAULTまたはALLを指定できます。

コメント

- オプションには、文字リテラルまたはホスト変数をいれることができます。1つの接続だけを使用する場合、接続の名前を与える必要はありません。複数の接続を使用する場合、各接続に名前を指定しなければなりません。
- 1つの接続だけを使用する場合は、接続に名前をつける必要はありません。複数の接続を使用するときは、各接続に名前をつける必要があります。 接続名はプロセス内でグローバルです。名前の付いた接続は、別々にコンパイルし、単一の実行可能モジュールにリンクしたプログラムの間で共有できます。
- CONNECTに成功すると、すべてのデータベーストランザクションは、CONNECT RESETが最後に宣言されているとき以外は、現在の接続となります。別の接続を使用するには、SET CONNECTION文を使用する必要があります。
- WITH PROMPTが使用されている場合は、ODBCランタイムモジュールは、エントリまたはランタイム時の詳細接続の確認のために、プロンプトを表示します。

例

```
EXEC SQL
```

```
    CONNECT TO :svr USER :usr
END-EXEC


EXEC SQL
    CONNECT TO "lsv_server.pubs" AS C1 USER "sa"
END-EXEC
```

または

```
EXEC SQL
    CONNECT TO "server.db1" USER $U¥INTEGRATED
END-EXEC
```

備考

SQLコンパイラ指令のINITオプションが使用されている場合は、データベースへの暗黙的接続がランタイム時に行われます。この場合は、明示的CONNECT文を実行する必要はありません。

7.6 DECLARE CURSOR

データの一度に1行の検索のためにカーソルを定義します。

構文

```
[AT db_name] DECLARE cursor_name CURSOR FOR
        {select_stmt | stored_procedure_name | prepared_stmt_name}
```

ただし

db_name

DECLARE DATABASEを使用して宣言されているデータベースの名前。

cursor_name

以後の文でカーソルを識別するためのカーソル名。カーソル名は最大30文字で、COBOL識別子に関する通常の規定に従います。 最初の文字は数字以外の文字でなければなりません。

select_stmt

任意の有効なSQL SELECT文またはQUERY ODBC文です。

stored_procedure_name

ストアド プロシージャの名前。

prepared_stmt_name

PREPAREされたSQL SELECT文またはQUERY ODBC文です。

コメント

- DECLARE CURSORは、カーソル名を指定されたSELECT文と関連付け、FETCH文を使ってデータ行を検索できるようにします。
- 2つの別々のコンパイルしたプログラムが同じカーソルを共有することはできません。特定のカーソルを参照する文は、すべて同時にコンパイルしなければなりません。
- DECLARE CURSOR文は、最初のカーソル参照より前になければなりません。SELECT文は、カーソルが開いたときに実行されます。SELECT文には次の規則が適用されます。
  - INTO句またはパラメータマーカをいれることはできません。
  - 前に宣言節で識別した入力ホスト変数をいれることができます。
  - ODBCドライバによっては、位置決め更新または位置決め削除を実行する場合、SELECT文にFOR UPDATE句をいれる必要があることがあります。

例

```
EXEC SQL DECLARE C1 CURSOR FOR

   SELECT last_name, first_name FROM staff
END-EXEC

EXEC SQL DECLARE C2 CURSOR FOR
   QUERY ODBC COLUMNS TABLENAME 'staff'
END-EXEC
```

7.7 DECLARE DATABASE

名前のついたデータベースを宣言します。

構文

```
DECLARE db_name DATABASE
```

ただし

db_name

データベースに関連する名前。識別子である必要があり、ホスト変数であってはなりません。引用符を含むことはできません。

コメント

- DECLARE DATABASE文は、データベースの名前を宣言します。DECLARE db_name文を宣言してから、CONNECT ... AT db_name文を使用する必要があります。

備考

EXECUTE IMMEDIATE、またはPREPAREとEXECUTEと一緒に、DECLARE DATABASEを使用することはできません。

7.8 DELETE (位置付け)

カーソルが現在ある行を削除します。

構文

```
DELETE  FROM  table_name  WHERE CURRENT OF cursor_name
```

ただし

table_name

SELECT FROMオプションで使用するのと同じテーブル名です（DECLARE CURSORを参照）。

cursor_name

前に宣言された開いているフェッチされたカーソルを指定します。

コメント

- ODBCは位置付け削除をサポートします。位置付け削除は、拡張構文において、カーソルを使って最後にフェッチされた行を削除します。ODBC 1.0では主要構文でしたが、ODBC 2.0では拡張構文に移行しまし

た。すべてのドライバが位置づけ削除をサポートしているわけではありません。しかし、OpenESQLは位置づけ更新および位置づけ削除をしやすいように、ODBCカーソル名をCOBOLカーソル名と同じように設定します。

- ODBCドライバによっては、カーソルが使用するSELECT文は、位置付け削除を可能とするために「FOR UPDATE句」を含む必要があることがあります。
- 位置付け削除ではホスト配列を使用できません。
- 標準SQL文で使用されるもう1つの形式は、検索削除といいます。

例

```
EXEC SQL DECLARE C1 CURSOR FOR
    select au_fname,au_lname from authors for browse
END-EXEC

EXEC SQL OPEN C1 END-EXEC

perform until sqlcode not = zero

    EXEC SQL FETCH C1 INTO :fname,:lname END-EXEC

    if sqlcode = zero
        display fname " " lname
        display "Delete?" accept reply
        if reply = "y"
         EXEC SQL
           DELETE FROM AUTHORS WHERE CURRENT OF C1
         END-EXEC
         display "delete sqlcode="sqlcode
```

```
          end-if

       end-if


  end-perform.
```

7.9 DELETE (検索)

検索基準を満たすテーブル行を削除します。DELETEは標準SQL文です。

構文

```
        [FOR :host_integer] DELETE [FROM] {table_name | view_name}

                  [WHERE search_conditions]
```

ただし

:host_integer

処理される配列要素の最大数を指定します。 PIC S9(4) COMP-5で宣言されなければなりません。

FROM

オプションのキーワードです。ANSI SQL 92に適合するために必要です。

table_name

削除操作のためのターゲットテーブルです。

view_name

削除操作のためのターゲットビューです。

WHERE

削除する行を識別する標準SQLのWHERE句です。

search_conditions

標準SQLのWHERE句に使用できる任意の有効な式です。

コメント

- 詳細な構文については、ODBCドライバに付属のマニュアルを参照してください。

- WHERE句には、単一のホスト変数とホスト配列を混在させることはできません。ホスト変数の1つが配列である場合は、すべて配列である必要があります。
- DELETE（検索）文は、検索条件に一致するテーブルの行を削除します。WHERE句を使用しない場合、名前のついたテーブル内の行すべてが削除されます。

例

```
EXEC SQL
    DELETE FROM authors WHERE auid = '12345'
END-EXEC
```

7.10 DESCRIBE

SQLDAデータ構造にデータを書き込みます。

構文

```
DESCRIBE prepared_stmt_name INTO :sqlda_struct
```

ただし

prepared_stmt_name

PREPAREされたSQL文です。

INTO :sqlda_struct

書き込むべき出力のSQLDAデータ構造を指定します。

コメント

- DESCRIBE文は、準備した動的SQL文に関する情報を提供します。 この文は、指定された準備する文によって返されるデータ型、長さ、列名を指定されたSQLDAデータ構造に書き込みます。
- DESCRIBEは、SQLDA構造のsqldフィールドに列の数を挿入します。 非選択文を準備した場合、sqldは0に設定されます。 DESCRIBEを呼び出す前にSQLDAデータ構造の次のフィールドをアプリケーションによって初期化しなければなりません。

  sqln

  構造が持つことができるsqlvar（列記述子）項目の最大数に設定しなければなりません。

  sqldabc

SQLDAの最大サイズ。sqln * 44 + 16によって求められます。

sqlnを0に設定した場合、列記述子項目は構築されませんが、sqldは必要な項目の数に設定されます。DESCRIBE文はINTO句付きのPREPARE文と似た動作をします。

例

```
EXEC SQL INCLUDE SQLDA END-EXEC


EXEC SQL BEGIN DECLARE SECTION END-EXEC
01 statement             pic x(80).
EXEC SQL END DECLARE SECTION END-EXEC


EXEC SQL
    DECLARE C1 CURSOR FOR stmt1
END-EXEC


move "select * from sysobjects" into statement


move 20 to sqln
compute sqldabc = 16+44*sqln
EXEC SQL
   PREPARE stmt1 FROM :statement
END-EXEC
EXEC SQL
   DESCRIBE stmt1 INTO :sqlda
END-EXEC
```

```
* The data structure "sqlda" now contains a description
* of the dynamic SQL statement.
 EXEC SQL
    OPEN C1
 END-EXEC

 EXEC SQL
    FETCH C1 USING DESCRIPTOR :sqlda
 END-EXEC
```

7.11 DISCONNECT

1つまたはすべてのデータベース接続を閉じます。

構文

```
DISCONNECT {name | ALL | CURRENT | DEFAULT}
```

ただし

name

接続名を指定します。

ALL

すべての接続（SQLコンパイラ指令のINITオプションを指定したときの自動接続を含む）を切断します。

CURRENT

現在の接続を切断します。 現在の接続は、CONNECT TOによって最後に確立された接続か、SET CONNECTON文によって設定される以後の接続です。

DEFAULT

省略時の接続を切断します。これは、接続名を指定しないCONNECT文による接続です。

コメント

- DISCONNECT文はデータベースとの接続を閉じます。さらにその接続について開いたすべてのカーソルは

自動的に閉じます。

例

```
EXEC SQL CONNECT TO "srv1.pubs" AS server1 USER "sa." END-EXEC

EXEC SQL CONNECT TO "srv2.pubs" AS server2 USER "sa." END-EXEC

EXEC SQL SET CONNECTION server1 END-EXEC

EXEC SQL SELECT name FROM sysobjects INTO :name END-EXEC

EXEC SQL SET CONNECTION server2 END-EXEC

EXEC SQL SELECT name FROM sysobjects INTO :name END-EXEC

EXEC SQL DISCONNECT server1 END-EXEC

EXEC SQL DISCONNECT server2 END-EXEC.


*The first SELECT will take place against the pubs

*database on server "srv1" The second SELECT will

*take place against the pubs database on server "srv2"
```

7.12 END DECLARE SECTION

BEGIN DECLARE SECTION文によって開始されたホスト変数宣言節を終了します。

構文

```
END DECLARE SECTION
```

コメント

- END DECLARE SECTION文は、BEGIN DECLARE SECTION文によって準備しなければなりません。

例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC

01 emp-id    pic x(4).

01 emp-name  pic x(30).

EXEC SQL END DECLARE SECTION END-EXEC
```

7.13 EXECSP

ストアド プロシージャを実行します。

構文

```
[FOR :row_count] EXECSP [:result_hvar = ] stored_procedure_name
        [(parameter [,...])] [WITH RECOMPILE]
```

ただし

row_count

整数型のホスト変数。結果およびパラメータのホスト変数がすべて同じサイズの配列で、要素すべてを使用するわけではない場合は、使用される要素数を指定します。EXECSPがDECLARE CURSOR文の一部である場合は、FOR句を使用できません。

result_hvar

手続き結果を受け取るホスト変数。

stored_procedure_name

ストアド プロシージャの名前。

parameter

リテラル定数またはフォームのホスト変数パラメータ。

[keyword=] :param_hvar [OUT | OUTPUT]

ここで、

Keyword	キーワードパラメータの正式なパラメータ名
param_hvar	ホスト変数
OUT	出力パラメータを指定
OUTPUT	出力パラメータを指定
WITH RECOMPILE	無視され、影響はありません。構文の互換性のためにあるだけです

コメント

- EXECSPは、ストアド プロシージャを実行するために使用されます。CALL文と代替え可能であり、Micro

Focus Embedded SQL Toolkit for Microsoft SQL Serverと下位互換性があります。可能な場合はいつでも、CALL文をEXECSP文に優先して使用できます。

例

```
EXEC SQL
   EXECSP myProc param1,param2
END-EXEC


EXEC SQL
   EXECSP :myResult = myFunction namedParam = :paramValue
END-EXEC


EXEC SQL
   EXECSP getDept :empName, :deptName OUT
END-EXEC


EXEC SQL
   DECLARE cities CURSOR FOR EXECSP locateStores :userState
END-EXEC
```

7.14 EXECUTE

PREPAREされたSQL文を実行します。

構文

```
EXECUTE prepared_stmt_name
 [using descriptor :sqlda_struct | using :hvar[,...]]
```

ただし

prepared_stmt_name

前にPREPAREされたSQL文です。

:sqlda_struct

入力値の記述が入っている前に宣言したSQLDAデータ構造です。

:hvar

1つまたは複数の入力ホスト変数です。

コメント

- EXECUTE文は動的SQL文を処理します。この文は、パラメータマーカがある場合はそれを値に置き換えてから、指定されたPREPAREするSQL文を実行します。(PREPAREする文は、PREPARE文を使って作成します。)結果を返さない文だけが許されます。
- PREPAREする文にマーカが入っている場合、EXECUTE文にはホスト変数と同じ数のusing :hvarオプションか、アプリケーションによってすでに書き込まれたSQLDAデータ構造を識別するusing descriptor :sqlda_structオプションが入っていなければなりません。
- PREPAREする文の中のパラメータマーカの数は、sqldata項目の数(using descriptor :sqlda)またはホスト変数(using :hvar)と一致していなければなりません。

例

```
MOVE "INSERT INTO publishers VALUES(?,?,?,?)" TO stmtbuf.


EXEC SQL
   PREPARE st FROM :stmtbuf
END-EXEC.


DISPLAY "Enter publisher id" ACCEPT pubid.
DISPLAY "Enter publisher name" ACCEPT pubname.
DISPLAY "Enter city" ACCEPT city.
DISPLAY "Enter state" ACCEPT state.


EXEC SQL
```

```
    EXECUTE st USING :pubid, :pubname, :city, :state
  END-EXEC.
```

7.15 EXECUTE IMMEDIATE

指定されたホスト変数に入っているSQL文を実行します。

構文

```
        EXECUTE IMMEDIATE :stmt_hvar
```

ただし

:stmt_hvar

文字列ホスト変数です。

コメント

- EXECUTE IMMEDIATE文には、パラメータマーカまたはホスト変数をいれることはできません。この文は結果を返すことはできません。この文の結果は廃棄されます。さらにこの文に、SQL埋め込みに対して排他的なSQLキーワードをいれることもできません。
- 行が返される場合、Sqlcodeが+1に設定されます。

例

```
   EXEC SQL
      DELETE FROM staff WHERE staff_id = 99
   END-EXEC


   * Put the required SQL statement in prep.
    MOVE   "insert into staff (staff_id, last_name, first_name ,age,
   -      "employment_date) VALUES (99, 'Lee', 'Phillip', 19, '1992-
   -      "01-02')" TO prep


   * Note EXECUTE IMMEDIATE does not require the statement to be
```

```
* prepared
 EXEC SQL
    EXECUTE IMMEDIATE :prep
 END-EXEC

* Check it worked...
 IF SQLCODE = ZERO
    DISPLAY 'Statement executed OK.'
 ELSE
    DISPLAY 'Error: Statement not executed.'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
    EXEC SQL DISCONNECT ALL END-EXEC
    STOP RUN
 END-IF

* Run through the same procedure again, this time deleting the
* values just inserted
 MOVE "delete from staff where staff_id = 99" TO prep

 EXEC SQL
    EXECUTE IMMEDIATE :prep
 END-EXEC

 IF SQLCODE = ZERO
```

```
      DISPLAY 'Statement executed OK.'
   ELSE
      DISPLAY 'Error: Statement not executed.'
      DISPLAY SQLERRMC
      DISPLAY SQLERRML
      EXEC SQL DISCONNECT ALL END-EXEC
      STOP RUN
   END-IF
```

7.16 FETCH

指定されたカーソルについてOPEN文によって生成された結果セットから次の行を獲得します。

構文

```
[FOR :host_integer] FETCH cursor_name
  [using descriptor :sqlda_struct | INTO :hvar [,...]]
```

ただし

:host_integer

処理される配列要素の最大数を指定します。 PIC S9(4) COMP-5で宣言されなければなりません。

cursor_name

前に宣言し開いたカーソルです。

:sqlda_struct

前にDESCRIBE文によって書き込まれたSQLDAデータ構造で、出力値のアドレスが入っています。 このオプションは、準備するSELECT文によって宣言されたカーソルといっしょにだけ使用します。 (SELECT文はPREPARE文を使って準備します。)

:hvar

データを受け取るための1つまたは複数のホスト変数を識別します。

コメント

- FETCH文はカーソルの結果セットから次の行を検索し、その行にある列の値を対応するホスト変数（またはSQLDAデータ構造で指定されたアドレス）に書き込みます。これ以上フェッチする行がない（結果セットの終わりに達した）場合、sqlcodeの値は100となりsqlstateの値は"02000"となります。
- OPEN cursor_name文はFETCH文より前にあり、カーソルはFETCHの実行中に開いてなければなりません。また、ホスト変数のデータ型は対応するデータベース列のデータ型と互換性がなければなりません。
- 列の数がホスト変数の数より少ない場合、Sqlwarn3の値がWに設定されます。エラーが起こった場合、それ以上の列は処理されません（処理済みの列はもとに戻りません）。
- 別な方法として:hvar変数に、それぞれカーソル宣言文の選択リストにある列に対応する複数のフィールドが入ったCOBOLレコードを指定することができます。この形式を使用する場合、SQLコンパイラ指令のDB2オプションを指定しなければなりません（このフラグを使うと、PREPARE INTO文とDESCRIBE文がCOBOLコンパイラによって拒否されることに注意してください)。
- INTO句のホスト変数の1つが配列である場合、INTO句のホスト変数はすべて配列である必要があります。

例

```
EXEC SQL DECLARE C1 CURSOR FOR
   SELECT au_fname,au_lname FROM authors
END-EXEC


EXEC SQL OPEN C1 END-EXEC


perform until sqlcode not = 0
   EXEC SQL
      FETCH C1 INTO :fname,:lname
   END-EXEC
   if sqlcode not = 0
       display fname, lname
    end-if
```

```
    end-perform
    EXEC SQL
        FETCH LAST C1 INTO :fname,:lname
    END-EXEC
    perform until sqlcode not = 0
        EXEC SQL
            FETCH PRIOR C1 INTO :fname,:lname
        END-EXEC

        if sqlcode not = 0
            display fname, lname
        end-if
    end-perform

    EXEC SQL CLOSE C1 END-EXEC
```

7.17 INCLUDE

特定のSQLデータ構造をアプリケーションで使用するために定義します。

構文

```
INCLUDE {sqlca | sqlda | filename}
```

ただし

sqlca

SQL通信領域（SQLCA）データ構造がアクセスされることを示します。

sqlda

SQL記述子領域（SQLDA）データ構造がアクセスされることを示します。

filename

COBOLのCOPY文と同様、この時点でソースにインクルードされたファイルを示します。

コメント

- この文には、COBOLプログラム内で指定されたSQLデータ構造またはソースファイルの定義が入っています。SQLCAデータ構造はSQL埋め込みプログラムといっしょに使用しなければなりません。プログラムにSQLDAデータ構造を参照するSQL埋め込み文（例えば、PREPARE INTO文）がある場合、SQLDAデータ構造も使用しなければなりません。
- この文は対応する.cpyファイルを使用します。sqlca.cpyとsqlda.cpyは現在のディレクトリにあるか、COBCPY環境変数によってこれらのファイルが入ったディレクトリを指定しなければなりません。

例

```
EXEC SQL INCLUDE SQLCA END-EXEC

EXEC SQL INCLUDE SQLDA END-EXEC

EXEC SQL INCLUDE MYFILE END-EXEC
```

7.18 INSERT

テーブルまたは表示に新しい行を追加します。INSERTは標準Transact-SQL文です。

構文

```
[FOR :host_integer] [AT db_name] INSERT [INTO]
 {table_name | view_name} [(column_list)]
 {VALUES (constant_expression) [,...]}
```

ただし

:host_integer

処理される配列要素の最大数を指定します。 PIC S9(4) COMP-5で宣言されなければなりません。

db_name

DECLARE DATABASEを使用して宣言されているデータベースの名前です。

table_name

行を挿入するテーブルです。

view_name

行を挿入するビューです。

INTO

オプションのキーワードです。ANSI SQL 92に適合させるために必要です。

column_list

- データを追加する1つまたは複数の列を指定します。列の順序は任意ですが、入力データは列と同じ順序でなければなりません。
- 列リストが必要なのは、テーブルにある（すべてではなく）一部の列がデータを受け取るときだけです。列リストの項目はかっこで囲みます。 列リストを与えない場合、受け取るテーブルにあるすべての列が（CREATE TABLEの順序で）データを受け取るものとみなされます。
- 列リストによって、値を入力する順序が決まります。

VALUES

定数式のリストを導入します。

constant_expression

指示された列について定数またはヌル値を指定します。 値リストはかっこで囲み、明示的または暗示的な列リストと一致していなければなりません。 非数値定数は一重引用符または二重引用符によって囲みます。

コメント

- INSERT文は直接、ODBCドライバに渡されます。詳細な構文については、ODBCドライバに付属のマニュアルを参照してください。
- WHERE句のホスト変数が配列である場合は、INSERT文は各配列要素セットにつき1回ずつ実行されます。
- INSERT文は表に新しい行を追加します。既存の行の列値を変更するには、UPDATE文を使用します。
- 列がヌル値を許すように定義されている場合、列リストとVALUESリストの中の項目を省略できます。
- 1つの文の中でテーブルから行を選択し、同じテーブルに挿入することができます。

例

```
DISPLAY "Enter new staff member’s id:"


ACCEPT staff-id
```

```
DISPLAY "Enter new staff member’s last name:"
ACCEPT last-name

DISPLAY "Enter new staff member’s first name:"
ACCEPT first-name

DISPLAY "Enter new staff member’s age:"
ACCEPT age

DISPLAY "Enter new staff member’s employment date(yyyy/mm/dd):"
ACCEPT employment-date

EXEC SQL
   INSERT INTO staff
   (staff_id
  ,last_name
  ,first_name
  ,age
  ,employment_date)
   VALUES
   (:staff-id
  ,:last-name
  ,:first-name
  ,:age
```

```
    ,:employment-date)


  END-EXEC
```

7.19 OPEN

指定されたカーソルについて一度に1行のデータ検索を開始します。

構文

```
OPEN cursor_name

[using descriptor :sqlda | using :hvar[,...]]
```

ただし

cursor_name

前に宣言したカーソル。

:sqlda

アプリケーションによって前に構築されたSQLDAデータ構造を指定します。 SQLDAデータ構造には、各入力パラメータのアドレス、データ型、長さが入っています。 このオプションは、DECLARE文の中で準備するSQLを参照するカーソルといっしょにだけ使用します。

:hvar

SELECT文にあるパラメータマーカに対応する1つまたは複数の入力変数を指定します。このオプションは、DECLARE文の中で準備するSQLを参照するカーソルといっしょにだけ使用します。

コメント

- OPEN文は対応するDECLARE CURSOR文の中で指定されたSELECT文を実行し、FETCH文によって一度に1行ずつアクセスされる結果セットを生成します。
- カーソルが静的SELECT文（準備されていないもの）によって宣言された場合、SELECT文にはホスト変数（:hvar）をいれることができますが、パラメータマーカは使用できません。ホスト変数の現在値がOPEN文の実行時に置き換えられます。静的に宣言されたカーソルでは、OPEN文にusing :hvarオプションまたはusing descriptor :sqldaオプションをいれることはできません。
- カーソルを動的SELECT文（PREPAREされたもの）によって宣言した場合、SELECT文にはパラメータマーカをいれることができますが、ホスト変数は使用できません。パラメータマーカは、SELECT文の中で列値が許される場所ならどこでも使用できます。SELECT文にパラメータマーカがある場合、OPEN文は、ホス

ト変数と同じ数のusing :hvarオプション、またはアプリケーションによってすでに書き込まれたSQLDAデータ構造を識別するusing descriptor :sqldaオプションのどちらかを含む必要があります。

- using descriptor :sqldaオプションでは、プログラム変数の値はSELECT文にあるパラメータマーカに置き換えられます。これらのプログラム変数の値は、SQLDAデータ構造の中の対応するsqldata項目によってアドレスされます。
- SELECT文の中のパラメータマーカの数は、sqldata項目の数（using descriptor :sqlda）またはホスト変数（using :hvar）と一致していなければなりません。

例

```
*Declare the cursor...

 EXEC SQL
 DECLARE C1 CURSOR FOR
    SELECT staff_id, last_name
    FROM staff
 END-EXEC

 IF SQLCODE NOT = ZERO
    DISPLAY 'Error: Could not declare cursor.'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
    EXEC SQL DISCONNECT ALL END-EXEC
    STOP RUN
 END-IF

 EXEC SQL
    OPEN C1
```

```
    END-EXEC

    IF SQLCODE NOT = ZERO
       DISPLAY 'Error: Could not open cursor.'
       DISPLAY SQLERRMC
       DISPLAY SQLERRML
       EXEC SQL DISCONNECT CURRENT END-EXEC
       STOP RUN
    END-IF

    PERFORM UNTIL sqlcode NOT = ZERO
   *SQLCODE will be zero as long as it has successfully fetched data
       EXEC SQL
          FETCH C1 INTO :staff-staff-id, :staff-last-name
       END-EXEC
       IF SQLCODE = ZERO
          DISPLAY "Staff ID: " staff-staff-id
          DISPLAY "Staff member's last name: " staff-last-name
       END-IF
    END-PERFORM

    EXEC SQL
      CLOSE C1
    END-EXEC
```

```
IF SQLCODE NOT = ZERO
  DISPLAY 'Error: Could not close cursor.'
  DISPLAY SQLERRMC
  DISPLAY SQLERRML
END-IF
```

7.20 PREPARE

ホスト変数の中の文字列からSQL文を受け取り、文を名前と関連付けます。

構文

```
PREPARE stmt_name [INTO :sqlda] FROM :hvar
```

ただし

stmt_name

PREPAREする文の名前です。以後のEXECUTE文またはOPEN文や前のDECLARE CURSOR文に対して文を指定します。

:sqlda

書き込むべき出力SQL記述子領域（SQLDA）データ構造です。

:hvar

SQL文が入ったホスト変数です。

コメント

- PREPARE文は動的SQL文を処理します。
- PREPAREする文は次の2種類の目的に使えます。
  - DECLARE文の中でPREPAREする文を参照するカーソルを開くことができます。
  - PREPAREする文を実行できます。
- PREPAREする文をEXECUTE文によって使用する場合、結果を返すようなSQL文を:hvarにいれることはできません。
- 単選択文（SELECT INTO）は動的SQLでは許されないので、PREPAREできません。

- PREPAREを使用する場合、:hvarの中のSQL文にはホスト変数やコメントをいれることはできませんが、パラメータマーカは使用できます。またSQL文には、SQL埋め込みに対して排他的なSQLキーワードをいれることはできません。
- INTO :sqldaオプションは、DESCRIBEの機能をPREPARE内に併合します。したがって、次の2つの文は機能的に同じです。

```
EXEC SQL
   PREPARE stmt1 INTO :sqlda FROM :stmt-buf
END-EXEC
```

または

```
EXEC SQL
   PREPARE stmt1 FROM :stmt-buf
END-EXEC
EXEC SQL
   DESCRIBE stmt1 INTO :sqlda
END-EXEC
```

例

```
PROGRAM-ID. progname.

WORKING-STORAGE SECTION.

  EXEC SQL INCLUDE SQLCA END-EXEC
  EXEC SQL BEGIN DECLARE SECTION END-EXEC
  01 prep         PIC X(80).
  01 nme          PIC X(20).
  01 car          PIC X(20).
```

```
   01 n60          PIC x(5).
   EXEC SQL END DECLARE SECTION END-EXEC

PROCEDURE DIVISION.

   EXEC SQL CONNECT TO 'srv1' USER 'sa' END-EXEC
   IF SQLCODE NOT = ZERO
      DISPLAY 'Error: Could not connect to database.'
      DISPLAY SQLERRMC
      DISPLAY SQLERRMC

      STOP RUN
   END-IF

* Ensure attempt is not made to recreate an existing table...
    EXEC SQL DROP TABLE mf_table END-EXEC

* Create a table...
 EXEC SQL CREATE TABLE mf_table
           (owner         char(20)
           ,car_col       char(20)
           ,nought_to_60   char(5))
 END-EXEC

 IF SQLCODE NOT = ZERO
```

```
    DISPLAY 'Error: Could not create table'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
    EXEC SQL DISCONNECT CURRENT END-EXEC
    STOP RUN

 END-IF

* Insert an SQL statement into host variable prep...
 MOVE "insert into mf_table values(?,?,?)" TO prep

* Prepare the statement...
 EXEC SQL
    PREPARE prep_stat FROM :prep
 END-EXEC

 IF SQLCODE NOT = ZERO
    DISPLAY 'Error: Could not prepare statement'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
    EXEC SQL DISCONNECT CURRENT END-EXEC
    STOP RUN
 END-IF

 MOVE "Owner" TO nme
```

```
 MOVE "Lamborghini" TO car
 MOVE "4.9" TO n60

* Execute the prepared statement using the above host variables...

 EXEC SQL
    EXECUTE prep_stat USING :nme, :car, :n60
 END-EXEC

 IF SQLCODE NOT = ZERO
    DISPLAY 'Error: Could not execute prepared statement.'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
           EXEC SQL DISCONNECT CURRENT END-EXEC
    STOP RUN
 END-IF

* Finally, drop the now unwanted table...
 EXEC SQL
    DROP TABLE mf_table
 END-EXEC

 IF SQLCODE NOT = ZERO
    DISPLAY 'Error: Could not drop table.'
    DISPLAY SQLERRMC
```

```
   DISPLAY SQLERRML
   EXEC SQL DISCONNECT CURRENT END-EXEC

   STOP RUN
END-IF

DISPLAY 'All statements executed.'

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

7.21 QUERY ODBC

ODBCデータソース中のカタログ情報を取り出しカーソルに定義します。

構文

```
        QUERY ODBC {COLUMN | COLUMNS}
          [QUALIFIER qualifier-name] [OWNER owner-name]
          [TABLENAME table-name] [COLUMNNAME column-name]
        QUERY ODBC {DATATYPE | DATATYPES}
          [TYPE {datatype-name | BIGINT | BINARY | BIT |
          CHAR | DATE | DECIMAL | DOUBLE | FLOAT | INTEGER |
          LONG VARBINARY | LONG VARCHAR | NUMERIC | REAL |
          SMALLINT | TIME | TIMESTAMP | TINYINT | VARBINARY |
          VARCHAR}]
        QUERY ODBC {TABLE | TABLES} [QUALIFIER qualifier-name]
          [OWNER owner-name] [TABLENAME table-name]
          [TYPE tabletype-name]
```

ただし

qualifier-name

表を選択する際に使われる修飾子を指定するホスト変数、識別子または定数。 すべてのODBCドライバが修飾をサポートしている訳ではなく、サポートのしかたも異なります。 例えば、データソースが複数データベースをサポートしている場合、どのデータベースを参照するかを指定するために修飾が使われます。 また、ファイルベースのデータソースの場合には、ディレクトリのパス名を特定するために修飾が使われます。

owner-name

表を選択する際に、表のオーナーを指定するホスト変数、識別子または定数。 すべてのODBCドライバがオーナーをサポートしている訳ではありません。

table-name

クエリーに含める表の名前のパターンを指定するホスト変数、識別子または定数。

datatype-name

クエリーに含めるデータ型の名前のパターンを指定するホスト変数、識別子または定数。

tabletype-name

クエリーに含める表タイプの名前のパターンを指定するホスト変数、識別子または定数。

コメント

- QUERY ODBC文は、SELECT文と同様にクエリーの結果の集合を検索します。 従って、DECLARE / OPEN文 または DECLARE / PREPARE / OPEN文によってカーソルに結び付けられていなければなりません。
- 検索パターンは以下の文字の組み合わせからなります：
    - SQL識別子のための正しい文字のすべて
    - アンダースコア (_)：任意の一文字にマッチする
    - パーセント(%)：任意の文字列にマッチする
    - ODBCドライバが定めるエスケープ文字。アンダースコアまたはパーセントが、パターン内でワイルドカードとしてではなく、文字そのものを意味するようにさせる
- 検索パターンが指定されていない場合には、% が書かれたとみなされ、すべてのエンティティにマッチします。

- テーブルのクエリーは以下の規則に従ってなされます：

    - qualifier-name が % で、owner-name と table-name が空文字列であれば、クエリーの結果はデータソース中のすべての正しい修飾名となります。 この場合、クエリー結果の中で TABLE_QUALIFIER（下を参照）以外のすべての桁はNULLになります。

    - owner-name が % で、qualifier-name と table-name が空文字列であれば、クエリーの結果はデータソース中のすべての正しい修飾名となります。 この場合、クエリー結果の中で TABLE_OWNER（下を参照）以外のすべての桁はNULLになります。

    - tabletype-name が % で、 qualifier-name と owner-name と table-name が空文字列であれば、クエリーの結果はデータソース中のすべての正しい表タイプ名となります。 この場合、クエリー結果の中で TABLE_TYP（下を参照）以外のすべての桁はNULLになります。

    - tabletype-name が省略された場合にはすべてのタイプの表がクエリー結果中に返されます。tabletype-name を指定する場合には、'TABLE,VIEW' のようにコンマで区切られた表タイプのリストとして指定しなければなりません。

クエリー結果のデータ型

各クエリーは以下のデータ型で返されます：

- 桁のクエリー

TABLE_QUALIFIER	VARCHAR(128)	
TABLE_OWNER	VARCHAR(128)	
TABLE_NAME	VARCHAR(128) NOT NULL	
COLUMN_NAME	VARCHAR(128) NOT NULL	
DATA_TYPE	SMALLINT NOT NULL	ODBCデータ型コードを表わす定数については odbcext.cpy と odbc.cpy を参照
TYPE_NAME	VARCHAR(128) NOT NULL	列のデータ型のためのドライバの依存名
PRECISION	INTEGER	
LENGTH	INTEGER	桁の値の固有表現に必要なメモリのサイズ
SCALE	SMALLINT	
RADIX	SMALLINT	数字型の桁の場合、2進か10進かに応じて2または10

		となる。数字型でない桁の場合 NULL
NULLABLE	SMALLINT NOT NULL	
REMARKS	VARCHAR(254)	

- データ型のクエリー

TYPE_NAME	VARCHAR(128) NOT NULL	列のデータ型コードを表わすドライバ定義の名称
DATA_TYPE	SMALLINT NOT NULL	ODBCデータ型コードを指定する定数。定数値は odbcext.cpy と odbc.cpy に定義されている。
PRECISION	INTEGER	この型の桁の最大精度
LITERAL_PREFIX	VARCHAR(128)	この型の定数を書くために必要な前置文字
LITERAL_SUFFIX	VARCHAR(128)	この型の定数を書くために必要な後置文字
CREATE_PARAMS	VARCHAR(128)	この型の桁を生成するときに必要なパラメタ。 例えば、十進数型の場合の精度、スケールなど
NULLABLE	SMALLINT NOT NULL	
CASE_SENSITIVE	SMALLINT NOT NULL	文字データ型の場合、比較における大文字・小文字の区別の有無
SEARCHABLE	SMALLINT NOT NULL	SQL_UNSEARCHABLE, SQL_LIKE_ONLY または SQL_ALL_EXCEPT_LIKE (odbc_cpy中に定義される).
UNSIGNED_ATTRIBUTE	SMALLINT	数値型の場合、符号の有無
MONEY	SMALLINT NOT NULL	数値型の場合、金額型かどうか
AUTO_INCREMENT	SMALLINT	自動増加のデータ型かどうか
LOCAL_TYPE_NAME	VARCHAR(128)	各国語バージョンのデータ型名
MINIMUM_SCALE	SMALLINT	
MAXIMUM_SCALE	SMALLINT	

- 表のクエリー

TABLE_QUALIFIER	VARCHAR(128)	
TABLE_OWNER	VARCHAR(128)	
TABLE_NAME	VARCHAR(128)	
TABLE_TYPE	VARCHAR(128)	TABLE, VIEW, SYSTEM TABLE, GLOBAL TEMPORARY, LOCAL TEMPORARY, ALIAS, SYNONYM またはデータソース独自の型識別名 のいずれか
REMARKS	VARCHAR(254)	

例

```
EXEC SQL
   DECLARE tcurs CURSOR FOR QUERY ODBC TABLES
END-EXEC


EXEC SQL DECLARE C1 CURSOR FOR
   QUERY ODBC TABLES OWNER :tab-owner TABLETYPE 'TABLE,VIEW'
END-EXEC


MOVE 'staff' to tab-name
EXEC SQL DECLARE C2 CURSOR FOR
   QUERY ODBC COLUMNS TABLENAME :tab-name
END-EXEC


EXEC SQL DECLARE C3 CURSOR FOR
   QUERY ODBC DATATYPES
END-EXEC
```

7.22 ROLLBACK

現行トランザクション中でなされた更新をすべてキャンセルします。

構文

```
ROLLBACK {WORK | TRAN | TRANSACTION} [RELEASE]
```

RELEASEが書かれていて、トランザクションがロールバックに成功すると、現在の接続が切断されます。

例

```
EXEC SQL
  ROLLBACK
END-EXEC


EXEC SQL
  ROLLBACK WORK RELEASE
END-EXEC
```

7.23 SELECT DISTINCT (DECLARE CURSOR 使用)

カーソル名をSELECT DISTINCT文に関連付け、FETCH文を使用してデータ行を検索できるようにする。

構文

```
[AT db_name] DECLARE cursor_name CURSOR FOR
SELECT DISTINCT select_list FROM table_list [select_options]
```

ただし

db_name

DECLARE DATABASEを使用して宣言されているデータベースの名前。

cursor_name

カーソル名は、以降の文でカーソルを識別するために使用します。カーソル名には、有効なファイル名を長さ30文字まで含むことができます。最初の文字は英数字である必要があります。

select_list

抽出する列の名前。

table_list

select_listで指定した抽出する列を含む、表の名前。

select_options

抽出する列数を制限、または抽出する行の順序を指定するオプション。

コメント

- 行セット内の重複する行を削除するには、SELECT INTOではなくSELECT DISTINCTを使用します。
- DECLARE CURSOR文は、カーソル名をSELECT DISTINCT文と関連付け、FETCH文を使用してデータ行を抽出できるようにします。
- 別々にコンパイルされた2つのプログラムは、カーソルを共有できません。特定のカーソルを参照する文は、一緒にコンパイルする必要があります。
- SELECT DISTINCT文は、カーソルがオープンされた時に実行されます。次の規則はSELECT DISTINCT文に適用されます。
    - INTO句またはパラメータマーカを含むことができません。
    - 前もって宣言節で識別された入力ホスト変数を含むことができます。
    - 位置付け更新または位置付け削除が実行された場合は、ODBCドライバによっては、SELECT DISTINCT文はFOR UPDATE句を含む必要があることもあります。

例

```
01 age-array       pic s9(09) comp-5 occurs 10 times.
01 lname-array     pic x(32)         occurs 10 times.


MOVE 5 TO staff-id
EXEC SQL
   SELECT DISTINCT last_name
      INTO :lname-array
```

```
        FROM staff
        WHERE staff_id  :staff-id
   END-EXEC


   EXEC SQL
      SELECT DISTINCT age
        INTO :age-array
        FROM staff
        WHERE first_name  'George'
   END-EXEC
```

7.24 SELECT INTO

結果の1行を検索します。SELECT INTO文は単選択とも呼ばれます。

構文

```
        [FOR :host_integer] [AT db_name] SELECT [select_list]
                INTO :hvar[,...] select_options
```

ただし

:host_integer

処理されるホスト配列要素の最大数。PIC S9(4) COMP-5と宣言される必要があります。

db_name

DECLARE DATABASEを使用して宣言されているデータベースの名前。

select_list

テーブルのデータを検索する部分を指定します。

:hvar

select_listの項目を受け取る1つまたは複数のホスト変数を指定します。

select_options

Transact-SQLのSELECT文（例えば、FROM句やWHERE句）で使用できる1つまたは複数の文や他のオプションを指定します。

コメント

- 単選択には、FROM句が入っている必要があります。
- SELECT INTO文は、結果の1行を検索し、select_listにある項目の値をINTOリストで指定されたホスト変数に割り当てます。受け取るホスト変数の数より多くの列を選択した場合、Sqlwarn3の値が'W'に設定されます。ホスト変数のデータ型と長さは、それに割り当てられた値と互換性がなければなりません。データが切り捨てられた場合、Sqlwarn1が'W'に設定されます。
- SELECT INTO文がデータベースから複数の行を返す場合、第1行以外の行はすべて廃棄され、Sqlwarn4が'W'に設定されます。第1行以外も戻したい場合は、カーソルを使用します。あるいは、INTO 句で配列項目を指定します。配列は、host_integerの値、配列の最大数または返された行数のうち、最小の値まで増やすことができます。
- INTO句のホスト変数のいずれか1つが配列である場合は、INTO句のホスト変数はすべて配列である必要があります。

例

```
EXEC SQL
   SELECT au_lname INTO :lname FROM authors
      WHERE stor_au_id=:auid
END EXEC


EXEC SQL
   SELECT stor_id INTO :storid:stornull FROM discounts
      WHERE discounttype=’initial customer’
END-EXEC
```

7.25 SET CONCURRENCY

標準モードカーソルの同時性オプションを設定します。

構文

```
SET CONCURRENCY {READONLY | LOCKCC | OPTCC | OPTCCVAL}
```

コメント

SET CONCURRENCY文では、以後開かれる標準モードカーソルのための同時性オプションを設定できます。

オプション	説明
READONLY	読み込み専用カーソルを指定します。データは変更できません。
LOCKCC	各行が入ったデータページがフェッチされるたびに更新目的ロックをかけます。開いたトランザクション内ではない場合、次のフェッチが実行されたときにロックは解放されます。開いたトランザクション内では、トランザクションが閉じたときにロックは解放されます。
OPTCC	タイムスタンプまたは値を使った楽観的同時性制御。カーソルによる行の変更は、前回のフェッチ後に行が変化していない場合だけ成功します。変更は、タイムスタンプを比較するか、タイプスタンプが使用できない場合はすべての非テキストの非イメージ値を比較することによって検出します。
OPTCCVAL	値を使った楽観的同時性制御。カーソルによる行の変更は、前回のフェッチ後に行が変化していない場合だけ成功します。変更は、すべての非テキストの非イメージ値を比較することによって検出します。

SET CONCURRENCY文はカーソルに同時性オプションを設定します。 次の場合を除いて、デフォルトはLOCKCCです。

- SQLコンパイラ指令のESQLVERSIONオプションが2.0に設定される場合、デフォルトはOPTCCです。
- LOCKCCがODBCドライバにサポートされていない場合、SET CONCURRENCYはREADONLYに設定されます（すべてのODBCドライバはこの設定をサポートするはずです)。

備考

- すべてのODBCドライバが、SET CONCURRENCY文をサポートしているとは限りません。
- 接続を確立してから、SET CONCURRENCYを使用する必要があります。
- Microsoft Accessドライバを使用している場合は、CONCURRENCYのデフォルトはREADONLYです。更新可能なカーソルをAccessで使用したい場合は、まず、SCROLLOPTIONをKEYSETに設定し、それからCONCURRENCYをLOCKCCに設定します。
- OPTCCまたはOPTCCVALオプションが使用され、最後のFETCH文の後に行が更新されている場合は、UPDATE WHERE CURRENT OF文は、値sqlcode -532 (sqlstate = "01001") で失敗する可能性があります。アプリケーションは、この状況を処理するコードを含む必要があります。

例

```
EXEC SQL SET CONCURRENCY READONLY END-EXEC
```

7.26 SET CONNECTION

現在の接続を名前のついた接続にします。

構文

```
SET CONNECTION {name | DEFAULT}
```

ただし

name

データベース接続の名前を指定します。nameの値は、前のCONNECT文で指定された接続名と一致していなければなりません。nameには、接続のリテラル名か、文字列値が入ったホスト変数の名前が使用できます。

DEFAULT

CONNECT文で接続名を省略して接続を確立した場合は、確立した接続をDEFAULTとして参照できます。

備考

- コンパイルモジュールにかかる接続をしている場合は、名前のついた接続を使用する必要があります。

例

```
EXEC SQL CONNECT TO "srv1.pubs" AS server1 USER "sa." END-EXEC
EXEC SQL CONNECT TO "srv2.pubs" AS server2 USER "sa." END-EXEC
EXEC SQL SET CONNECTION server1 END-EXEC
EXEC SQL SELECT name FROM sysobjects INTO :name END-EXEC
EXEC SQL SET CONNECTION server2 END-EXEC
EXEC SQL SELECT name FROM sysobjects INTO :name END-EXEC
EXEC SQL DISCONNECT server1 END-EXEC
EXEC SQL DISCONNECT server2 END-EXEC.
*The first SELECT will take place against the pubs
```

```
*database on server "srv1" The second SELECT will
*take place against the pubs database on server "srv2"
```

7.27 SET OPTION

さまざまなSQLオプションを設定できます。ODBCドライバすべてがサポートしているとは限りません。

構文

```
SET OPTION {querytime | logintime | application | host} value
```

ただし

value

リテラルまたはホスト変数の名前です。ホスト変数には、applicationまたはhostの場合は文字値、logintimeまたはquerytimeの場合は数値が入っていなければなりません。

コメント

SET OPTION文では、次のオプションの値を設定できます。

オプション	説明
querytime	OpenESQLに応答するのをプログラムが待つ秒数を設定します。デフォルトは0秒でいつまでも待ちます。 このオプションは、既存のネットワークのタイムアウト設定には優先しません。
logintime	CONNECT TO文に応答するのをプログラムが待つ秒数を設定します。 デフォルトは10秒です。0の場合はタイムアウトするまで待ちます。
application	CONNECT TO文が実行されたときにOpenESQLによってデータソースに渡されるLOGINREC構造の中にアプリケーション名を設定します。
host	CONNECT TO文が実行されたときにOpenESQLによってデータソースに渡されるLOGINREC構造の中にワークステーション名を設定します。

備考

- ODBCドライバすべてがSET OPTION文をサポートしているとは限りません。

例

```
EXEC SQL SET OPTION logintime 5 END-EXEC
```

```
EXEC SQL CONNECT TO "srv1.pubs" USER "sa." END-EXEC

*If we are unable to log in to the server "srv1" within five
*seconds, the CONNECT will time out and return to the program.

EXEC SQL SET OPTION querytime 2 END-EXEC

EXEC SQL SELECT name FROM sysobjects INTO :name END-EXEC

*If the SELECT statement does not respond within two seconds.
*the query will time out and return to the program.
```

7.28 SET SCROLLOPTION

標準モードカーソルに関するスクロール方法と行メンバシップを設定します。

構文

```
SET SCROLLOPTION {KEYSET | DYNAMIC | FORWARD | STATIC}
```

コメント

SET SCROLLOPTION文では、以後開かれる標準モードカーソルのスクロール方法を設定できます。

オプション	説明
KEYSET	キーセットカーソルでは、カーソルの結果セットの行のメンバシップと順序はカーソルが開かれたときに決まります。行を削除したり、WHERE句の基準を満たさなくなるように変更した場合、行はフェッチされなくなります。1つのテーブルに基づくカーソルによって行を挿入した場合だけ、行はカーソルの結果セットに現れます。カーソルの所有者による更新や他のユーザによる結果セット内の行に対する変更のコミットは可視です。
DYNAMIC	動的カーソルでは、カーソルの結果セットの行のメンバシップはフェッチ時に決まり、フェッチごとに変わることがあります。行を削除したり、WHERE句の基準を満たさなく

	なるように変更した場合、行はカーソルの結果セットから消えます。行を挿入したり、WHERE句の基準を満たすように変更した場合、行はカーソルの結果セットに現れます。カーソルの所有者による更新や他のユーザによる結果セット内の行に対する変更のコミットは可視です。
FORWARD	DYNAMICと等価ですが、将来変更される可能性があります。
STATIC	静的カーソルでは、結果セットは静的になります。メンバシップの変更、カーソルオープン後の結果セットの値または順序は、常に検出されるとは限りません。

次の場合を除いて、デフォルトはDYNAMICです。

- SQLコンパイラ指令のESQLVERSIONオプションが2.0に設定される場合、デフォルトはKEYSETです。
- ODBCドライバがDYNAMICをサポートしていない場合、SET SCROLLOPTIONはFORWARDに設定されます（すべてのODBCドライバはこの設定をサポートするはずです)。

ODBCドライバがサポートしていないオプションを設定しようとした場合は、エラー（-19512)になります。

備考

- すべてのODBCドライバが、SET SCROLLOPTION文をサポートしているとは限りません。
- SET SCROLLOPTIONを使用する前に接続を確立する必要があります。

例

```
EXEC SQL SET SCROLLOPTION DYNAMIC END-EXEC
```

7.29 UPDATE (位置付け)

カーソルが現在ある位置の行にあるデータを変更します。

構文

```
[AT db_name] UPDATE table_name SET column=expression[,...]
     WHERE CURRENT OF cursor_name
```

ただし

db_name

DECLARE DATABASEを使用して宣言されているデータベースの名前。

table_name

更新するテーブルを指定します。

column=expression

特定の列名の値を指定します。この値には式またはヌル値が使用できます。

cursor_name

前に宣言された開いているフェッチされたカーソルを与えます

コメント

ODBCは位置付け更新をサポートします。位置付け更新は、拡張構文において、カーソルを使用して最後に取り込まれた行を更新します（ODBC 1.0では主な構文にありましたが、ODBC 2.0では拡張構文に移行しました)。すべてのドライバが位置付け更新をサポートしているとは限りませんが、OpenESQLは、位置付け更新および位置付け削除をしやすいように、ODBCカーソル名をCOBOLカーソル名と同じように設定します。

ODBCドライバによっては、位置付け更新を可能にするには、カーソル名が使用するSELECT文がFOR UPDATE句を含む必要がある場合もあります。

標準SQL文で使用されるUPDATEのもう1つの形式は、検索更新といいます。

位置付け更新ではホスト配列を使用できません。

例

```
EXEC SQL CONNECT TO 'srv1' USER 'sa' END-EXEC


EXEC SQL DECLARE C1 CURSOR FOR
   SELECT last_name, first_name
   FROM staff
   FOR UPDATE
END-EXEC


EXEC SQL
   OPEN C1
END-EXEC
```

```
PERFORM UNTIL SQLCODE NOT = ZERO
   EXEC SQL
       FETCH C1 INTO :fname,:lname
   END-EXEC
   IF SQLCODE = ZERO
      DISPLAY fname " " lname
      DISPLAY "Update?"
      ACCEPT reply
      IF reply = "y"
         DISPLAY "New last name?"
         ACCEPT lname
         EXEC SQL
            UPDATE staff
            SET last_name=:lname WHERE CURRENT OF c1
         END-EXEC
         DISPLAY "update sqlcode=" SQLCODE
       END-IF
   END-IF
END-PERFORM

EXEC SQL DISCONNECT ALL END-EXEC
STOP RUN.
```

7.30 UPDATE (検索)

新しいデータを追加するか、既存のデータを変更することによって、既存の行にあるデータを変更します。UPDATE

は標準Transact-SQLのUPDATE文です。

構文

```
[FOR :host_integer] [AT db_name] UPDATE {table_name | view_name}
  SET [column=expression [,...]] [WHERE search_conditions]
```

ただし

:host_integer

処理されるホスト配列要素の最大数。PIC S9(4) COMP-5で宣言されなければなりません。

db_name

DECLARE DATABASEを使用して宣言されているデータベースの名前。

table_name

更新するテーブルを指定します。

view_name

更新するビューを指定します。

column=expression

特定の列名の値を指定します。この値には式、選択文、またはヌル値が使用できます。

search_conditions

標準SQLのWHERE句の後に続く任意の有効な式を指定します。

コメント

UPDATEは標準SQL文で、ODBCドライバに直接、渡されます。詳細な構文については、ODBCドライバに付属のマニュアルを参照してください。

WHERE句を指定しない場合は、名前の付いた表の行すべてが更新されます。

WHERE句またはSET句で使用されるホスト変数の1つが配列である場合は、WHERE句で使用されるホスト変数はすべて配列である必要があります。

例

```
EXEC SQL
```

```
    BEGIN TRANSACTION
    IF EXISTS (SELECT * FROM authors
              WHERE au_fname = "John" AND au_lname = "Doe")
         UPDATE authors
             SET au_fname = "Jonathan"
                WHERE au_lname = "Doe"
    ELSE
         DELETE FROM authors
             WHERE au_lname = "Doe"
    COMMIT TRANSACTION
END-EXEC
```

7.31 WHENEVER

埋め込みSQL文を実行した後に取るべきデフォルトアクションを指定します。

構文

```
WHENEVER {NOT FOUND | SQLERROR | SQLWARNING}
        {CONTINUE | GOTO stmt_label | PERFORM label }
```

ただし

NOT FOUND | SQLERROR | SQLWARNING

発生するエラーレベルを指定します。

GO TO stmt_label

特定のエラーレベルが検出されたときにジャンプするプログラム内の場所を識別します。

PERFORM label

特定のエラーレベルが検出されたときにPERFORMされるプログラム内の節または段落を識別します。

CONTINUE

ソースプログラム内の次の文を実行します。

コメント

- WHENEVER文は、SQL埋め込み文を実行した後に、次の各条件（NOT FOUND、SQLERROR、SQLWARNING）に応じてデフォルトアクションを指定します。

条件	sqlcode
エラーなし	0
NOT FOUND	100
SQLWARNING	+1
SQLERROR	（負）

- WHENEVER文の範囲は、実行順序ではなく、ソースプログラム内の文の位置に関係します。デフォルトでは、3つの条件すべてについてCONTINUEです。

例

```
EXEC SQL WHENEVER sqlerror PERFORM errormessage1 END-EXEC


EXEC SQL
   DELETE FROM staff
   WHERE staff_id = ’hello’
END-EXEC


EXEC SQL
   DELETE FROM students
   WHERE student_id = ’hello’
END-EXEC


EXEC SQL WHENEVER sqlerror CONTINUE END-EXEC
```

```
EXEC SQL
   INSERT INTO staff VALUES (’hello’)
END-EXEC

DISPLAY ’Sql Code is: ’ SQLCODE

EXEC SQL WHENEVER sqlerror PERFORM errormessage2 END-EXEC

EXEC SQL
   INSERT INTO staff VALUES (’hello again’)
END-EXEC

STOP RUN.

errormessage1 SECTION.
   display "SQL DELETE error: " sqlcode

   EXIT.

errormessage2 SECTION.
   display "SQL INSERT error: " sqlcode
   EXIT.
```

第8章 OpenESQL

OpenESQL は、COBOL プログラム内に記述された埋め込み SQL ステートメントからリレーショナル データベースへの ODBC ドライバ経由のアクセスを可能にするプリプロセッサです。

備考：OpenESQL では、UNIX 向けのアプリケーションは開発できません。

8.1 ODBC ドライバとデータソース名

ODBC を使用するには、次の操作を行う必要があります。

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

8.1.1 ODBC ドライバのインストール

NetExpress のインストール開始時点で、表示される一覧から「ODBC ドライバ」を選択することによって、ODBC ドライバのインストールを明示的に選択する必要があります。

備考：ほとんどの ODBC ドライバは、対応するデータベース用のクライアント ソフトウェアがインストールされていなければ動作しません。

8.1.2 データソース名の設定

Net Express の ODBC 機能を使用するには、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（95/98/NT）のコントロール パネルから開くことができます。コントロール パネルで ODBC (32ビット) （または ODBC (16ビット)）アイコンをダブルクリックします。

「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表示されます。このダイアログボックスで「システム DSN」タブをクリックします。NetExpress の一部としてインストールされた ODBC ドライバが NetExpress ... という名前で一覧されます。NetExpress では Microsoft Access ドライバもインストールされ、「*NetExpress Microsoft Access*」という名前で一覧に表示されます。

DSN の割り当て方法については、「ODBC データ ソース アドミニストレータ」ダイアログボックスで [ヘルプ] ボ

タンをクリックし、表示されるオンライン ヘルプをご覧ください。

備考：これらの ODBC ドライバは、NetExpress アプリケーションの開発には使用できますが、ユーザに配布することはできません。ODBC ドライバの詳しいライセンス情報については、『*入門書*』の「*主要機能の一覧*」をご覧ください。

8.2 SQL コンパイラ 指令

プログラムのコンパイル時には、SQL コンパイラ 指令と関連オプションを指定する必要があります。これらを指定すれば、埋め込み SQL ステートメントがプリプロセッサによって対応するデータソースの関数呼び出しに変換されます。プログラムから呼び出す ODBC ドライバは、アクセスするデータソースによって決定されます。

SQL コンパイラ 指令のオプションは、次の 2 通りの方法で指定できます。

- プログラム内に $SET ステートメントで記述する。
- NetExpress の「指令の詳細」画面で指定する。
  この画面を表示するには、[プロジェクト] メニューから [ビルド設定] を選択し、表示されるダイアログボックスでプログラムを選択した後、「コンパイル」タブを選択して [詳細] ボタンをクリックします。

備考：上記の 2 つの方法を併用することはできません。いずれかの方法のみを使用してください。

SQL コンパイラ 指令に指定できるオプションを次に一覧します。

オプション	機能
ESQL Preprocessor	OpenESQL を使用する場合には、このオプションを OpenESQL に設定します。
[NO]ANSI92ENTRY	OpenESQL の動作を SQL ANSI 92 のエントリ レベル規格に準拠させます。

オプション	機能
[NO]AUTOCOMMIT	各 SQL ステートメントを個別のトランザクションとして処理し、実行後ただちにコミットします。このオプションを指定せずにトランザクションに対応した ODBC ドライバを使用する場合には、ステートメントをトランザクションの一部として明示的にコミット（またはロールバック）する必要があります。
[NO]CHECK	各 SQL ステートメントをコンパイル時にデータベースに送信します。コンパイル時のステートメント チェックを指定すると、DB と PASS の指定も必要になります。
CONNECTIONPOOL=[DRIVER | ENVIRONMENT | NONE]	ODBC 3.0 の接続プールを有効化します。接続プールを有効化すると、アプリケーションで閉じた接続がドライバ マネージャでは一定時間にわたって維持されるため、アプリケーションで同じ接続を再使用するときに接続を再確立する手間が省けます。接続プールは、特定の ODBC 環境を対象として設定するか、または各ドライバ単位で設定することができます。詳細については、ODBC のドキュメントをご覧ください。このオプションのデフォルト値は NONE で、省略すると接続プールは無効になります。接続プールは、接続を開閉する回数が多いアプリケーションのみに効果を発揮します。なお、Microsoft Transaction Server（MTS）など、接続プール機能を独自に実装した環境もあります。このオプションは、MTS を使用しない環境で ISAPI アプリケーションのパフォーマンスを向上させる場合などに効果的です。
[NO]CURSORCASE	ESQLVERSION の値が 2.0 であれば CURSORCASE、その他の場合には NOCURSORCASE がデフォルト値です。CURSORCASE を指定するとカーソル名の大文字と小文字が区別され、NOCURSORCASE では区別されません。旧バージョンの OpenESQL では、カーソル名の大文字と小文字は区別されています。
[NO]DB	接続するデータソースの名前。このオプションは、INIT オプションや CHECK オプションと共に使用します。

オプション	機能
[NO]DETECTDATE	デフォルト値は NODETECTDATE。DETECTDATE を指定すると、OpenESQL は ODBC エスケープ シーケンスで文字ホスト変数をチェックし、次の時間データを取り出します。 含む文字ホスト変数の値をチェックします。 {d<データ>} - 日付 {t<データ>} - 時刻 {ts<データ>} - タイムスタンプ 検出された値はバインドされます。文字カラムとしては使用されません。この処理は、サーバが日付表記文字列をサポートしていない場合（Microsoft Access など）に必要になります。また、汎用的なアプリケーションにも有効です。ただし、{d や {t、または {ts で開始するデータを含む文字カラムが存在する場合には問題が発生する可能性があります。
[NO]ESQLVERSION	OpenESQL の構文レベルを設定します。
[NO]INIT	プリプロセッサにデータベース接続用のコードを自動生成させます。INIT を指定する場合には、DB オプションと PASS オプションも指定する必要があります。
[NO]NIST	OpenESQL の動作を NIST による SQL ANSI 92 エントリ レベル規格に準拠させます。
ODBCTRACE= [ALWAYS | NEVER | USER]	USER （デフォルト値）は、ODBC トレース処理をコントロール パネルで制御する設定です。ODBC コントロール パネルからトレース対象のファイルを指定できます。ALWAYS は ODBC トレース処理を指令で制御する設定であり、IDE でのアプリケーション開発に適しています。ODBC コントロール パネルの設定に関わらず、この設定では常にトレース ファイル（MFSQLTRACE.LOG）が生成されます。通常の開発では、カレント プロジェクトの Debug ディレクトリ（プロジェクトのビルド設定によっては Release ディレクトリ）がトレース ファイルの格納先になります。NEVER を設定すると、アプリケーションのトレースは実行されず、コントロール パネルの設定も無視されます。ODBC トレース ファイルには重要な情報が含まれる場合があるため、情報の安全性を重視する場合には NEVER を使用します。

オプション	機能
[NO]PARAMARRAY	すべての入力パラメータに ODBC 配列バインドを使用します（ドライバが同機能をサポートしている場合のみ）。デフォルト値は PARAMARRAY。
[NO]PASS	データソースへの接続時に使用するログイン名。このオプションは INIT オプションや CHECK オプションと共に使用します。
[NO]RESULTARRAY	すべての出力パラメータに ODBC 配列バインドを使用します（ドライバが同機能をサポートしている場合のみ）。デフォルト値は RESULTARRAY。
[NO]TARGETDB	データベースの最適化オプション。Microsoft SQL Server と Oracle 8.0 に有効です。
THREAD=[SHARE | ISOLATE]	ISOLATE を指定すると、接続やカーソルがスレッドごとに生成されます。この設定は、マルチスレッドのアプリケーション サーバ環境（IIS/ISAPI など）に使用します。一方、SHARE を指定した場合には、接続やカーソルはすべてのスレッドを対象として生成されます。あるスレッドでコード内に記述した CONNECT ステートメントを実行した後、別のスレッドで同じステートメントを実行すると、最初のスレッドですでに接続が開かれているためエラーになります。ISOLATE を指定した場合には、各スレッドで個別に接続を開くことができます。このオプションのデフォルト値は SHARE です。

8.3 データ ソース

NetExpress をインストールすると、2 つのデータソース名（DSN）が自動的に作成されます。これらのデータソース名（*NetExpress Sample1* と *NetExpress Sample2*）は、それぞれ NetExpress の一部として ¥demo¥smpldata ディレクトリにインストールされた Access のデモンストレーション データベース（demo.mdb と sample.mdb）を示します。

8.4 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、その前に対象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。

- CONNECT ステートメントによる接続（通常はこの方法を使用してください。）

  通常、コンパイル時にデータソース名を特定できない場合や、複数のデータベースにアクセスするプログラムに使用します。

- 自動接続

  通常、コンパイル時に指定したデータベースのみにアクセスするプログラムに使用します。SQL コンパイラ 指令の INIT オプションを指定すると、SQL コンパイラの DB オプションで指定されたデータソースに 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コードが、コンパイラによってプログラムの開始部分に挿入されます。

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、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 ステートメントを使用します。

自動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自動実行させるには、SQL コンパイラ 指令の INIT=PROT オプションを指定します。

8.5 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。該当する単語は、プログラム内で他の用途に使用すべきではありません。キーワードはオンライン ヘルプに一覧されています。ヘルプの索引で「OpenESQL」を選択してください。

8.6 アプリケーションのビルド

OpenESQL を使用して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。

- プログラム内の各 SQL ステートメントを 2 つのキーワード（EXEC SQL と END-EXEC）で囲む。
- 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する。

NetExpress のインストール ディレクトリの source サブディレクトリには、2 つのコピーファイル（sqlca.cpy および sqlda.cpy）がインストールされており、通常の方法でプログラムにインクルードできます。

NetExpress で .exe ファイルを作成する際には、必要なオブジェクト ファイルはすべて自動的にリンクされます。

備考：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必要な.dll ファイル（odbcrw32.dll および _sqlodbc.dll） が存在することを確認してください。

8.7 デモンストレーション アプリケーション

NetExpress をインストールすると、odbcesql ディレクトリにさまざまなデモンストレーション アプリケーションがコピーされます。odbcesql ディレクトリは、NetExpress のインストール ディレクトリの demo サブディレクトリ内に生成されています。

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 ドライバをインストールしておく必要があります。NetExpress のインストール時に、Microsoft Access ドライバを含む複数の ODBC ドライバが自動的にインストールされます。また、Access のデモンストレーション データベースが 2 つ ¥demo¥smpldata ディレクトリにコピーされ、対応するデータソース名が生成されます。デモンストレーション アプリケーションでは、*NetExpress Sample2* というデータソース名のデモンストレーション データベースを使用できます。

OpenESQL を使用するデモンストレーション アプリケーションを次に示します。これらのアプリケーションは、いずれかも処理の進行状況をコンソールに表示し、照会結果を表示する場合もあります。処理中にエラーが発生すると、エラー メッセージを表示して終了します。

- connect.app

  データソースとユーザ名、およびパスワードの入力を求めます。データソース名として「NetExpress Sample2」、ユーザ名として「admin」をそれぞれ入力し、パスワードは空欄のまま Return 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た後、「SQL データソース」ダイアログボックスが表示されます。このダイアログボックスの「マシン データソース」リストから「NetExpress Sample2」を選択し、[OK] をクリックすると、「ログイン」ダイアログボックスが表示されます。このダイアログボックスにユーザ名として「admin」を入力し、パスワードは空欄のまま [OK] をクリックすると 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

  デモンストレーション 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」をそのまま顧客コードとして入力すると、当該する顧客レコードの 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 Return キーを押すと、地域の入力が求められます。メッセージに表示された「CA」をそのまま入力すると、その地域の顧客が一覧され、さらに地域の入力が求められます。ここで Return 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、最初にデータソース名とユーザ名の入力が求められます。データソース名として「NetExpress Sample2」、ユーザ名として「admin」をそれぞれ入力すると、次の順序でテストが実行されます。

接続
テスト テーブルのドロップ
テスト テーブルの作成
行の挿入
コミット
行の更新
読み取り/検証
ロールバック
読み取り/検証
テスト テーブルのドロップ
接続の解除
テスト テーブルの作成

2 番目のテストではエラー メッセージが表示されることがありますが、これは意図的に生成されたものであり、プログラムの実行には影響しません。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とテストは失敗です。

- whenever.app

  接続失敗時のエラー メッセージを表示するサンプルです。起動後に表示される「SQL データソース」ダイアログボックスで「NetExpress Sample2」を選択して [OK] をクリックすると、「ログイン」ダイアログボックスが表示されます。このダイアログボックスにログイン名として「admin」を入力し、パスワードは空欄のまま [OK] をクリックすると、意図的にエラーが生成され、2 つのエラー メッセージが出力されます。

- catalog.app

  起動後に表示される「SQL データソース」ダイアログボックスで「NetExpress Sample2」を選択して [OK] をクリックすると、「ログイン」ダイアログボックスが表示されます。このダイアログボックスにログイン名として「admin」を入力し、パスワードは空欄のまま [OK] をクリックすると、3 種類のデータ ディクショナリ照会が実行され、結果が出力されます。

8.8 トランザクション管理

OpenESQL では、COMMIT ステートメントと ROLLBACK ステートメントによって ODBC のトランザクション管理機能を制御できます。ODBC では、各ステートメント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ますが、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。自動コミットを有効化するには、SQL コンパイラ 指令の AUTOCOMMIT オプションを指定してください。

備考：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ただちに確定される可能性があります。

8.9 データ型

OpenESQL は、SQL と COBOL との間で一定のルールに従ってデータ型を変換します。このルールは、SQL のデータ型と COBOL のデータ型の対応関係としてオンライン ヘルプに一覧されています。オンライン ヘルプの目次で、「*リファレンス*」、「*データベース アクセス*」、「*OpenESQL*」、「*データ型*」 を順次選択してください。

ODBC では、日付は *yyyy-mm-dd*、時刻は *hh:mm:ss* の形式で表記されます。これらの形式は、データソース側の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、データソース側の形式を使用できます。ほとんどのデータソースでは PIC X(26) を使用してください。次に例を示します。

```
01  mydate     PIC x(26).

...

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
```

ODBC のエスケープ シーケンスを使用することも可能です。ODBC では、日付、時刻、およびタイムスタンプの各リテラル用のエスケープ 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リテラルの表記方法は次のとおりです。

`{d 'yyyy-mm-dd'}` - 日付
`{t 'hh:mm:ss'}` - 時刻
`{ts yyyy-mm-dd hh:mm:ss[.f...]}` - タイムスタンプ

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

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

  01  date-field1     pic x(26).
  01  date-field2     pic x(26).
  01  date-field3     pic x(26).

  procedure division.
* データソースに接続。NetExpress に付属している
* デモンストレーション データソースの 1 つに接続します。
  EXEC SQL
     CONNECT TO 'NetExpress Sample1' USER 'admin'
  END-EXEC
* テーブルが存在すればドロップ。
  EXEC SQL
     DROP TABLE DT
  END-EXEC

* DATE、TIME、および TIMESTAMP の列を持つテーブルを作成。
* 注意： 専用の列が定義されているデータベースもあります。たとえば、
*      Access では上記の 3 つの列の代わりに DATETIME 列のみを使用します。
* 他のデータソースに DATE 列や TIME 列を作成する場合には、
* 当該データベースのドキュメントで列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 シーケンスを含むデータをテーブルに挿入**

```
EXEC SQL
   INSERT into DT values (1 ,
             {d '1961-10-08'},  *> 日付を指定
             {t '12:21:54'  },  *> 時刻を指定
             {ts '1966-01-24 08:21:56' } *> 日付と時刻（タイム
             )                          *> スタンプ）を指定
END-EXEC
```

* **挿入した値の取り込み**

```
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

```
  display '日付の設定値：' date-field1
  display '時刻の設定値：' date-field2
  display '注意：データソースは通常、日付に'
         ' デフォルト値を設定します。'
  display 'タイムスタンプの設定値：' date-field3
```

* テーブルを削除

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

* データソースとの接続を解除。

```
EXEC SQL
   DISCONNECT CURRENT
END-EXEC


Stop Run.
```

8.10 SQLCA の使用

SQLCA データ構造体は、NetExpress インストール ディレクトリの source サブディレクトリ内の 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、Data Division に次のステートメントを記述します。

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

このステートメント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、プログラムはその領域にはアクセスできません。ただし、`sqlcode` （または `sqlstate`）データ項目を別に宣言すれば、各 SQL EXEC ステートメントの実行後に SQLCA 内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 は文字データ項目 PIC X(*n*) として宣言します（*n* は任意の有効値）。ODBC エラー メッセージは、SQLCA メッセージ フィールドのサイズ（70 バイト）を超過することが少なくありません。そのため、MFSQLMESSAGETEXT データ項目は特に有効です。

備考：SQLCA や SQLCODE、SQLSTATE、および MFSQLMESSAGETEXT は、いずれもホスト変数として宣言する必要はありません。

8.11 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 odbcesql ディレクトリに格納されています。（odbcesql ディレクトリは、NetExpress のインストール ディレクトリの demo サブディレクトリ内にあります。）このプロジェクトを開いて必要に応じて再ビルドした後、[アニメート] メニューから [ステップ] 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。

8.12 ストアド プロシージャ

OpenESQL では、2 つのステートメント（CALL および EXECSP）をストアド プロシージャと組み合わせて使用で

きます。CALL は、ODBC ストアド プロシージャの呼び出しに使用します。EXECSP は、Micro Focus Embedded SQL Tookit for Microsoft SQL Server との互換性を維持する場合に使用します。

ストアド プロシージャでは次の処理を実行できます。

- 引数の入力
- 引数による値の戻し
- 引数の入力と引数による値の戻し
- 位置型引数やキーワード型引数の使用
- 結果の戻し
- 結果セットの戻し
- 引数配列による呼び出し

備考：ストアド プロシージャの機能はデータベース製品ごとに大きく異なり、各製品は上記の機能を部分的に実装しているに過ぎません。したがって、異なるデータベース製品間でのストアド プロシージャ呼び出しの移植性は、OpenESQL ステートメントに比べると大幅に限定されます。

ストアド プロシージャの呼び出し時には、カンマ（,）で区切られた一連の引数が渡されます。（各引数は括弧で囲まれる場合もあります。） 引数にはホスト変数やリテラル、および CURSOR キーワードが使用できます。なお、CURSOR で渡された引数はバインドが解除されるため、結果セットを戻す Oracle 8 ストアド プロシージャ以外には使用すべきではありません。

引数としてホスト変数を渡す場合には、その後に引数の種類（IN、INPUT、INOUT、OUT、OUTPUT）を指定できます。種類のデフォルト値は INPUT です。

ホスト変数は、直前に正規のパラメータ名と等号（=）を指定することによって、キーワード型引数として渡すことも可能です。次に例を示します。

```
EXEC SQL CALL myProc (keyWordParam = :hostVar) END-EXEC
```

移植性を重視する場合には、ホスト変数のみを引数に使用し、リテラルは使用しないでください。また、渡された引数は通常の位置型引数とキーワード型引数のいずれかのみとして扱い、これらの両方として処理すべきではありません。一部のサーバでは両方の引数がサポートされていますが、その場合でもキーワード型引数は常にすべての位置型引数の後に渡される必要があります。キーワード型引数はコード記述の明瞭化に役立ち、サーバがデフォルトの引数

値とオプション引数をサポートする場合に効果的です。

結果セットを戻すストアド プロシージャ呼び出しは、カーソル宣言で使用してください。次に例を示します。

```
EXEC SQL DECLARE cursorName CURSOR FOR storedProcecureCall
```

この場合、ストアド プロシージャは他のタイプのカーソルと同様、カーソルの OPEN 処理と結果セット行の FETCH 処理によって呼び出されます。

現バージョンの OpenESQL は、複数の結果セットには対応していません。

ODBC パラメータは Oracleの配列引数とは異なります。配列引数を使用すれば、配列の各要素に同じステートメントを繰り返し実行した場合と同じ結果が得られます。ストアド プロシージャ呼び出しで 1 つの引数を配列として渡すと、その他のすべての引数も同じ数の要素を含む配列として渡す必要があります。ストアド プロシージャでは、これらの引数の各要素ごとに呼び出された場合と同様の処理が実行されます。引数で渡す要素数は、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 は渡すべき要素の数を含む整数型のホスト変数です。）

第9章 OpenESQL アシスタント

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

- SQL 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 文の試験
- SQL の INSERT 文、UPDATE 文、および DELETE 文の作成

作成した SQL クエリーは、OpenESQL アシスタント を使用して NetExpress の COBOL コードに挿入することができます。適切なプロジェクトとプログラムを開くだけで、OpenESQL アシスタント により現在の挿入箇所に SQL クエリーを挿入することができます。また、SQL クエリーを挿入するときに必要となる補助コードを、OpenESQL アシスタント を使用して自動的に作成し、挿入する方法も選択できます。

本章では、チュートリアル形式で次の事項を説明します。

- OpenESQL アシスタント の起動方法
- データ ソースへの接続方法
- テーブルの選択方法
- 列の選択方法
- 列の選択解除方法
- テーブルに含まれる全列の選択方法
- テーブルの選択解除方法
- 列の詳細の表示方法
- 新規クエリーの作成方法
- 別のテーブルの選択方法
- クエリーの型の変更方法
- 別のデータソースへの接続方法
- SELECT 文によるクエリーの実行方法
- 検索条件の指定方法
- データソースからの接続解除方法
- 結合テーブルの作成方法

- プログラムへの埋め込み SQL 文の追加方法
- プログラムへの補助コードの追加方法
- OpenESQL アシスタント の終了方法

9.1 OpenESQL アシスタント の起動

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

図 9-1 OpenESQL アシスタント

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

9.2 データ ソースへの接続

OpenESQL アシスタント を起動すると、設定したすべての ODBC データ ソースのリストが表示されます。例えば、上記の図 9-1 では、OpenESQL アシスタント により、Excel_Files や Oracle_Database などの既存のデータ ソースが表示されています。

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

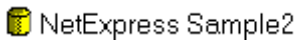

図 9-2 データ ソース アイコンとデータ ソース名

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

- ユーザー ID
- パスワード
- データベース名

データ ソースを全く設定していない場合は、NetExpress のインストール時に自動設定されたデータ ソース例を使用することができます。データ ソース例の 1 つである NetExpress Sample2 は、Microsoft Access データベース例の sample.mdb を指します。この sample.mdb は、NetExpress のインストール時に ¥demo¥smpldata¥access ディレクトリに格納されます。

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

9.3 テーブルの選択

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

図 9-3 テーブルの選択

テーブルを選択するには、テーブル名をダブルクリックします。その結果、必要なクエリーの型を選択するためのダイアログボックスが表示されます。この段階では選択するクエリーの型は関係ないので、デフォルト

([SLELCT(singleton)]) を選択して、[OK] ボタンをクリックします。

図 9-4 クエリーの選択

クエリーを選択すると、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] タブの下に表示されます。同時に、テーブル名の下にテーブルの全列が一覧表示されます。

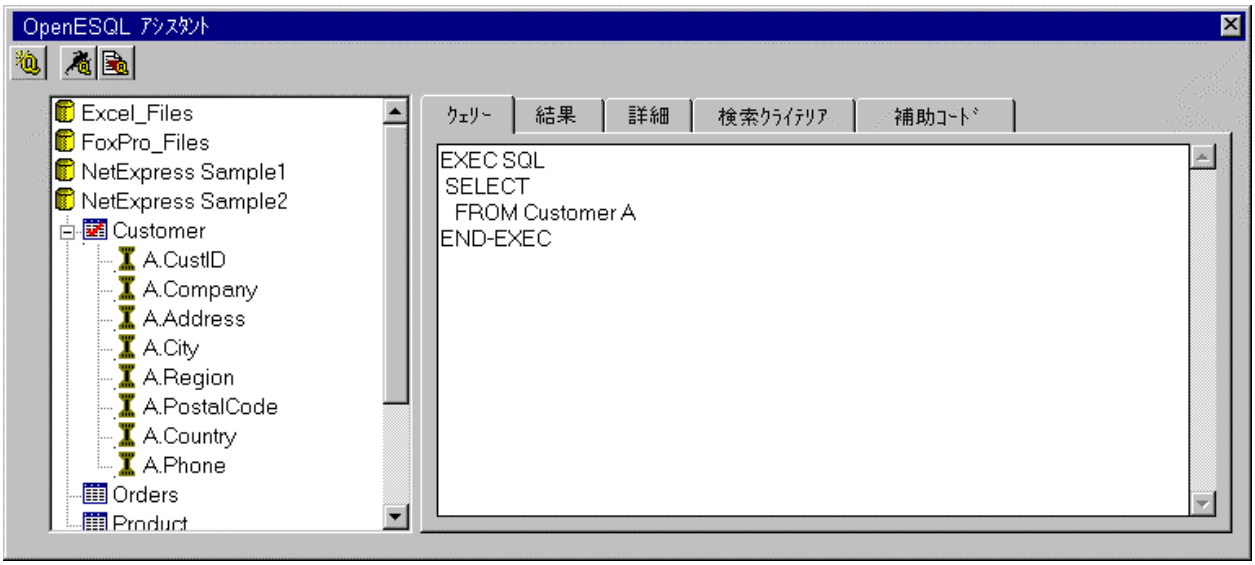

図 9-5 列の表示

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

```
SELECT FROM Customer A
```

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

また、各列名の前にも、別名がつくことに注意してください (例 A.CustID、A.Company など)。この別名により、列が属するテーブルを区別することができます。

9.3.1 列の選択

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

9.3.2 列の選択解除

選択した列を選択解除するには、その列名をダブルクリックします。列の選択と選択解除を行うたびに、COBOL コードは自動更新されます。

9.3.3 テーブルに含まれる全列の選択

テーブルの全列を選択するには、次の操作を行います。

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

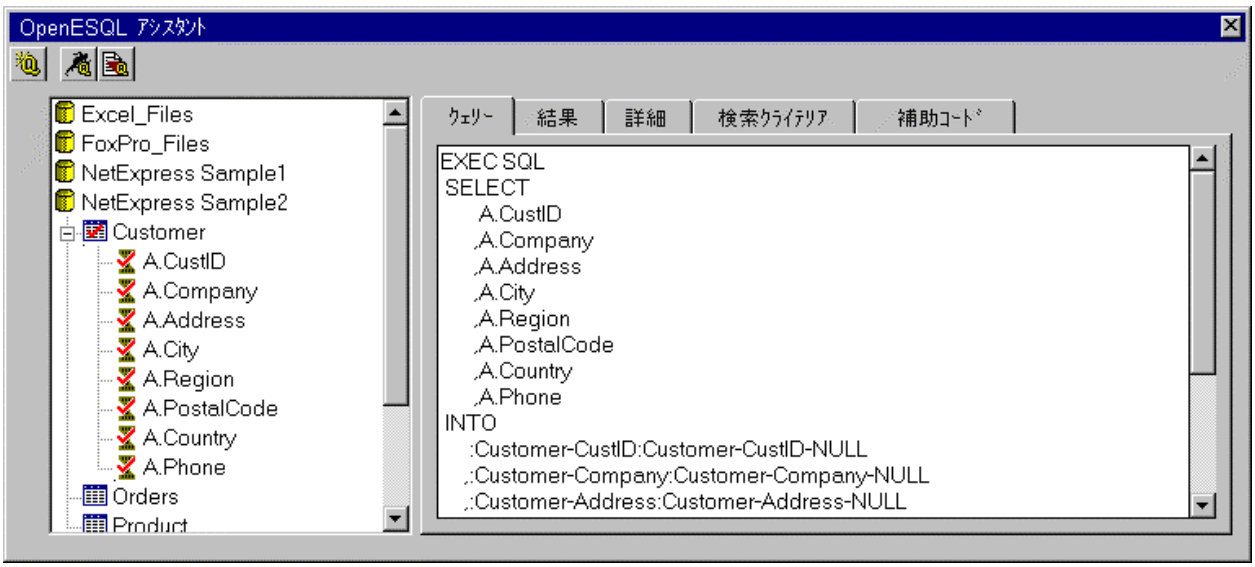

図 9-6 列の選択

列が現在選択されている場合、列アイコンがチェックされるので、列が現在選択されているかどうかを常に把握することができます。

図 9-7 列アイコン

9.4 テーブルの選択解除

選択したテーブルを選択解除するには、テーブル名またはテーブル アイコンをダブルクリックします。このテーブルから列を選択している場合は、選択解除を確認するダイアログが表示されます。[はい] ボタンをクリックすると、テーブルの選択は解除され、生成されたコードが更新されます。

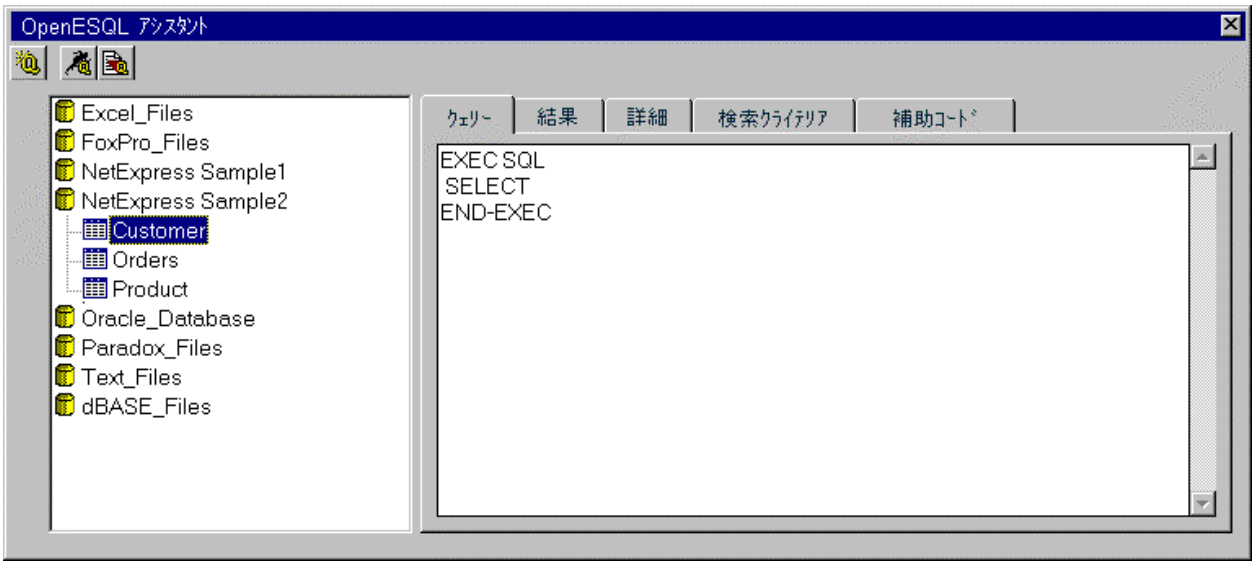

図 9-8 テーブルの選択解除

テーブルを選択解除するには、テーブル名かテーブル アイコンをクリックしてください。マイナス記号 (-) をクリックしても、テーブルの列の表示/非表示が切り替わるだけです (プラス記号 (+) をクリックするとすべての列を表示し、マイナス記号 (-) をクリックするとすべての列を非表示にします)。

例えば、Customer テーブルをダブルクリックして、再度選択します。テーブル名を右クリックし、次に [すべてのカラムを選択] をクリックして、すべての列を選択します。さらに、列を表示しないためにマイナス記号 (-) をクリックします。

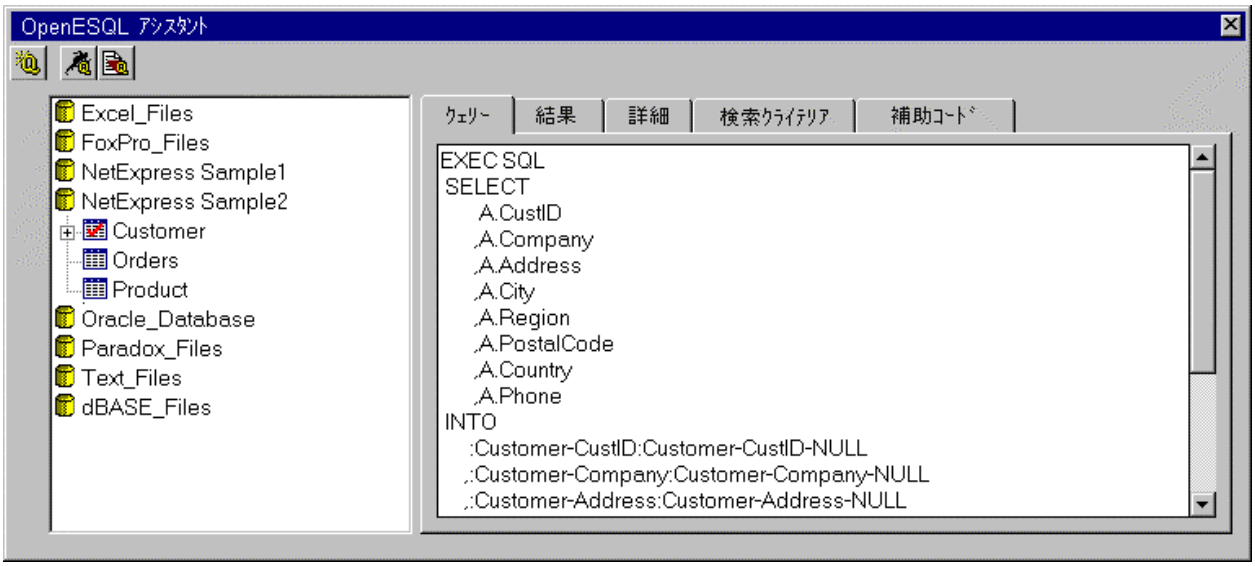

図 9-9 選択したテーブルに含まれる列の非表示

9.5 列の詳細表示

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

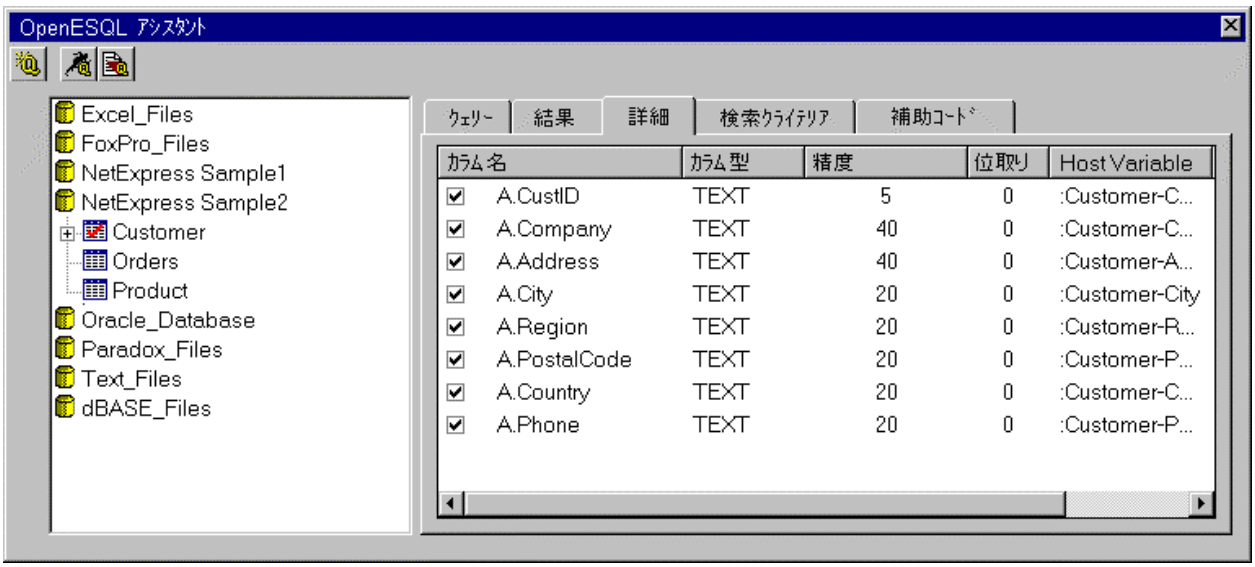

図 9-10 列の詳細表示

表示される情報は次のとおりです。

- 列名

  テーブルの各列名が表示されます。各列名の前には、テーブルの別名が追加されることに注意してください。例えば、CustID は、A.CustID のように表示されます。列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、現在その列が選択されていることを示します。

- 型

  列のデータ型が表示されます。このデータ型は、接続するデータ ソースで使用します。列のデータ型は、その列の値をデータ ソースと受け渡しするホスト変数の COBOL ピクチャ句で指定するデータ型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 *テーブル名*.cpy というコピーファイルを作成することもできます。このコピーファイルでは、必要なホスト変数と正しい COBOL ピクチャ句 (テーブルの列のデータ型と一致するもの) をすべて宣言することができます。下記の*「補助コードの追加」*の項を参照してください。

- 精度

  列の総桁数が表示されます。精度は、関連する列についてだけ表示されます。列が文字列の場合、精度は列の文字列長を示します。

- 位取り

  列の数値を丸める桁数が表示されます。位取りは、関連する列についてだけ表示されます。

- ホスト変数

  OpenESQL アシスタント により、各列についてホスト変数が自動生成されます。ホスト変数名は次のような形式になります。

  :<テーブル名>-<列名>

  例えば、OpenESQL アシスタント により CustID 列に対して生成されるホスト変数名は `:Customer-CustID` となります。また、 City 列に対して生成されるホスト変数名は `:Customer-City`、Phone 列に対して生成されるホスト変数名は `:Customer-Phone` となります。

- インジケータ変数

  OpenESQL アシスタント は、ホスト変数と同様に、各列に対してインジケータ変数を生成します。インジケータ変数名は次のような形式になります。

  :<テーブル名>-<列名>-NULL

  例えば、OpenESQL アシスタント により CustID 列に対して生成されるインジケータ変数名は `:Customer-CustID-NULL` となります。また、City 列に対して生成されるインジケータ変数名は `:Customer-City-NULL`、Phone 列に対して生成されるインジケータ変数名は `:Customer-Phone-NULL` となります。

9.6 新規クエリーの作成

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

9.6.1 別のテーブルの選択

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

9.6.2 クエリーの型の変更

クエリーの型を変更するには、まず、現在選択されているテーブルを一旦選択解除し、再度同じテーブルを選択するか (同じテーブルで別の型のクエリーを作成する場合)、別のテーブルを選択する必要があります。テーブルを選択すると、クエリーの型を選択するためのダイアログが表示されます。希望する型のクエリーを選択し、[OK] ボタンをクリックします。

9.6.3 別のデータ ソースへの接続

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

9.7 SELECT 文によるクエリーの実行

NetExpress Sample2 データ ソースに接続し、Customer テーブルを選択します。次に、クエリーの型を指定するためのダイアログが表示されるので、SELECT (カーソル) を選択し、[OK] ボタンをクリックします。

図 9-11 クエリーの型 SELECT (カーソル) の選択

SELECT 文を使用した COBOL コードが自動生成され、表示されているのがわかります。

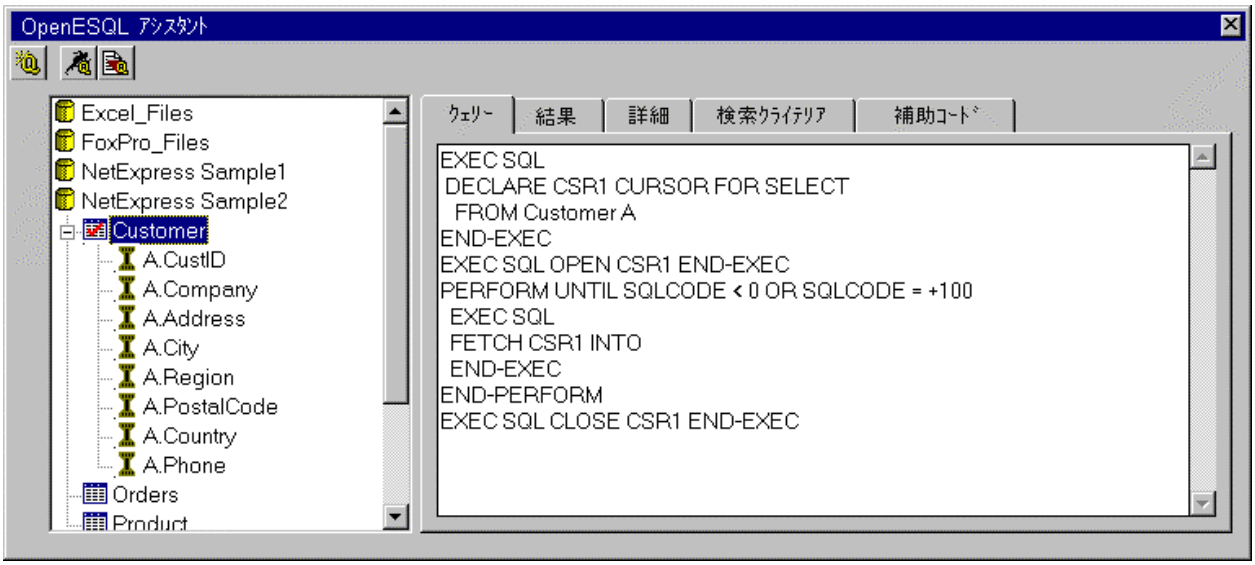

図 9-12 SELECT (カーソル) によるクエリー コード

このコードには、次の文が含まれています。

- SELECT 文で使用するカーソルを作成するための DECLARE CURSOR 文
  DECLARE CSR-1 CURSOR FOR SELECT FROM Customer A
- 対応する DECLARE CURSOR 文で指定した SELECT 文を実行する OPEN 文
  OPEN CSR-1
- カーソルの結果集合から次の行を検索する FETCH 文
  FETCH CSR-1 INTO
- カーソルをクローズする CLOSE 文
  CLOSE CSR-1

上記のとおり、このクエリーでは、検索する列と検索する内容が指定されていなため、構文エラーになります。エラー表示を確認するには、[クエリーの実行] ボタン  をクリックします。

列を選択するには、列をダブルクリックします。ここでは、A.CustID を選択します。選択した各列について、次のように自動生成コードが更新されます。

- 列が DECLARE CURSOR 文に追加されます。次に例を示します。

```
DECLARE CSR-1 CURSOR FOR SELECT

   A.CustID

FROM Customer A
```

- 列を読み込むホスト変数が FETCH 文の INTO 句に追加されます。次に例を示します。

```
FETCH CSR-1 INTO
   :Customer-CustID:Customer-CustID-NULL
```

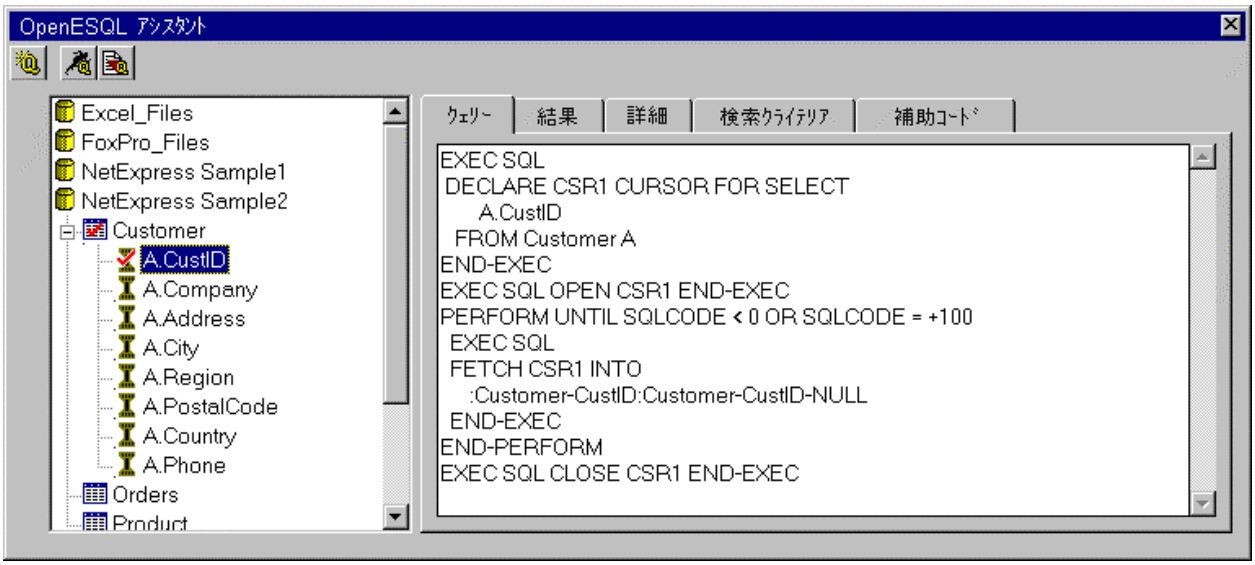

図 9-13 SELECT 文への列の追加

ここでは、A.Company と A.Phone を選択します。必要な列をすべて選択した後で、[クエリーの実行] ボタン  をクリックすると、クエリーを実行することができます。クエリー結果は、OpenESQL アシスタント により自動的に表示されます。

図 9-14 クエリー結果

9.8 検索条件の指定

検索条件 (WHERE 句) を指定すると、SELECT 文で検索する行を制限することができます。OpenESQL アシスタント には、WHERE 句で使用する検索条件を簡便に指定できる特別なダイアログがあります。

SELECT 文で検索する行数を制限するには、[検索クライテリア] タブをクリックします。

図 9-15 [検索クライテリア] タブ

検索条件を指定するには、次の手順にしたがいます。

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

2. 条件指定演算子を選択します。下向きの矢印をクリックし、有効な条件指定演算子が見つかるまでスクロールします。

3. 「受け側データ型」 を選択します。検索対象の型は、ホスト変数、リテラル定数、特殊なレジスタ、または列名です。

4. 「受け側値」を選択するか入力します。検索対象の型に列名を選択した場合、OpenESQL アシスタント により生成された有効な列名がすべて「受け側値」リスト ボックスに一覧表示されます。検索対象の型にホスト変数を選択した場合、OpenESQL アシスタント により生成された有効なホスト変数がすべて「受け側値」リスト ボックスに一覧表示されます。また、検索対象の型にリテラル定数か特殊なレジスタを選択した場合、「target value」リスト ボックスの右側にある [編集] ボタンが使用できるようになります。この場合、「受け側値」リスト ボックスに、値を直接入力することも、また、[編集] ボタンをクリックして、「定数値」エディタを表示することも可能です。

選択内容に問題がない場合は、右向きの矢印 (>) をクリックします。その結果、検索条件が右側の枠内に移動され、

同時にクエリー ウィンドウ内の COBOL コードが更新されます。

たとえば、上記のとおり作成した SELECT 文によって戻される顧客 ID を制限する場合は、次のように指定します。

1. 「カラム」リスト ボックスで A.City を選択します。
2. 条件指定演算子として "=" を選択します。
3. 検索対象の型として "定数" を選択します。
4. この段階で「受け側値」リストボックスの右側にある [編集] ボタンが使用可能になるので、このボタンをクリックし、「定数値」エディタを表示します。

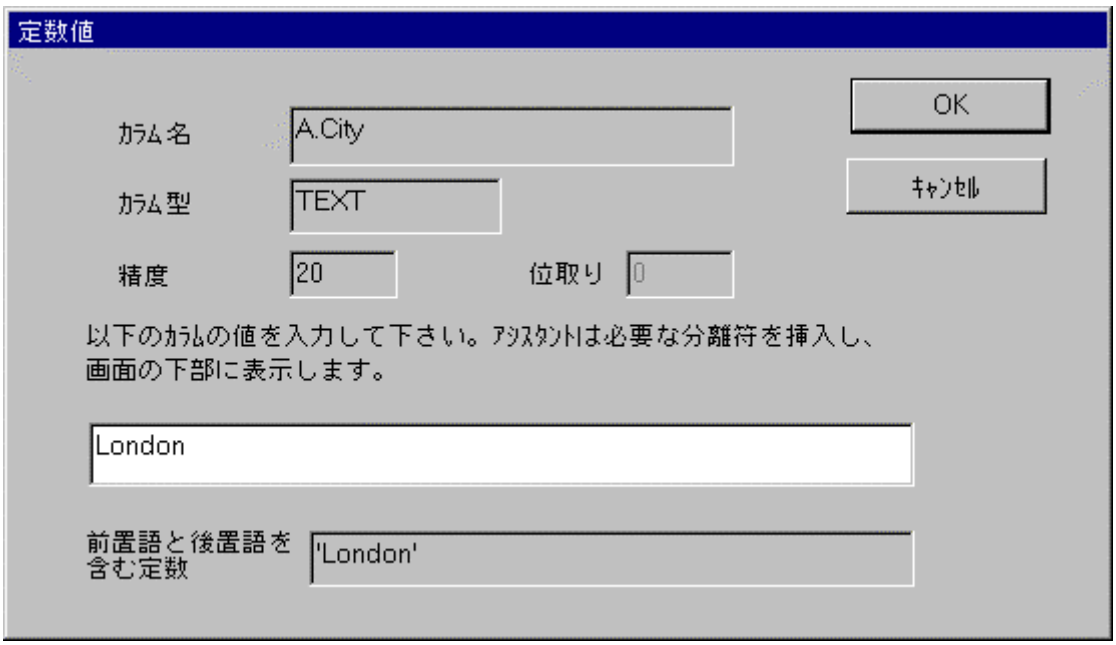

図 9-16 「定数値」エディタ

- 検索対象の値として London を選択します。OpenESQL アシスタント により正しい区切り文字が自動的に追加されていることに注意してください (この例では、単一引用符 (') が使用されています)。[OK] ボタンをクリックします。
- 今度は、右向きの矢印をクリックし、検索条件を右側の枠内に移動させます。

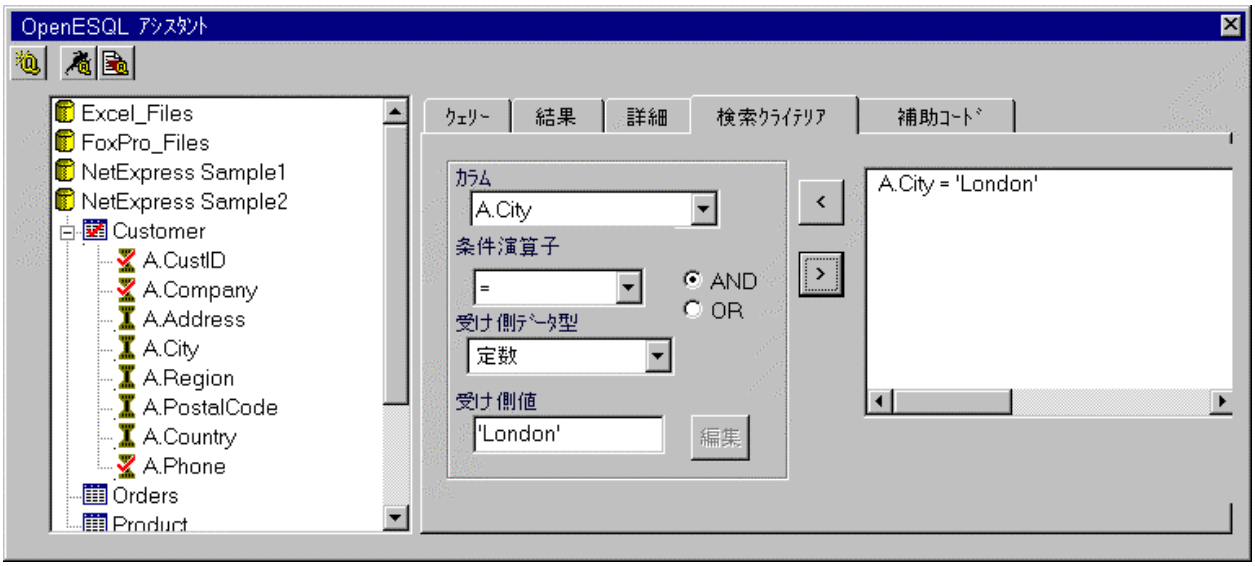

図 9-17 検索条件の指定

このクエリーを再実行すると ([クエリーを実行] ボタン  をクリックします)、City 列のエントリが London である顧客の顧客 ID だけが表示されます。

図 9-18 検索条件を指定して制限した顧客 ID

最初の検索条件を指定して、右側の枠内に移動させると、それ以降の条件は、オプション ボタンをクリックして選択する論理積 (AND) または論理和 (OR) にしたがいます。

右側の枠内に表示された検索条件を選択し、左向きの矢印 (<) をクリックすると、この選択した検索条件を右側の枠から削除し、編集できるように左側の枠内の該当するボックスに戻すことができます。検索条件を右側の枠から左側の枠へ移動させると、自動生成されたクエリー コードから対応する COBOL コードが削除されます。

9.9 データ ソースからの接続解除

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

9.10 結合テーブルの作成

複数のテーブル間に共通する列が 1 列でもあると、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することができます。

例えば、NetExpress Sample2 から、ある製品を注文した全顧客の企業名と電話番号を検索すると仮定します。一方、製品の詳細情報は Product テーブルに保存されているとします。この場合、次のように検索します。

1. OpenESQL アシスタント を起動し、NetExpress Sample2 データ ソースへ接続します。
2. Product テーブルをダブルクリックし、クエリーの型として SELECT (カーソル) を選択して、[OK] ボタンをクリックします。
3. Product テーブルを右クリックし、[すべてのカラムを選択] をクリックします。

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

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

このクエリーを生成するための COBOL コードは、次のように表示されます。

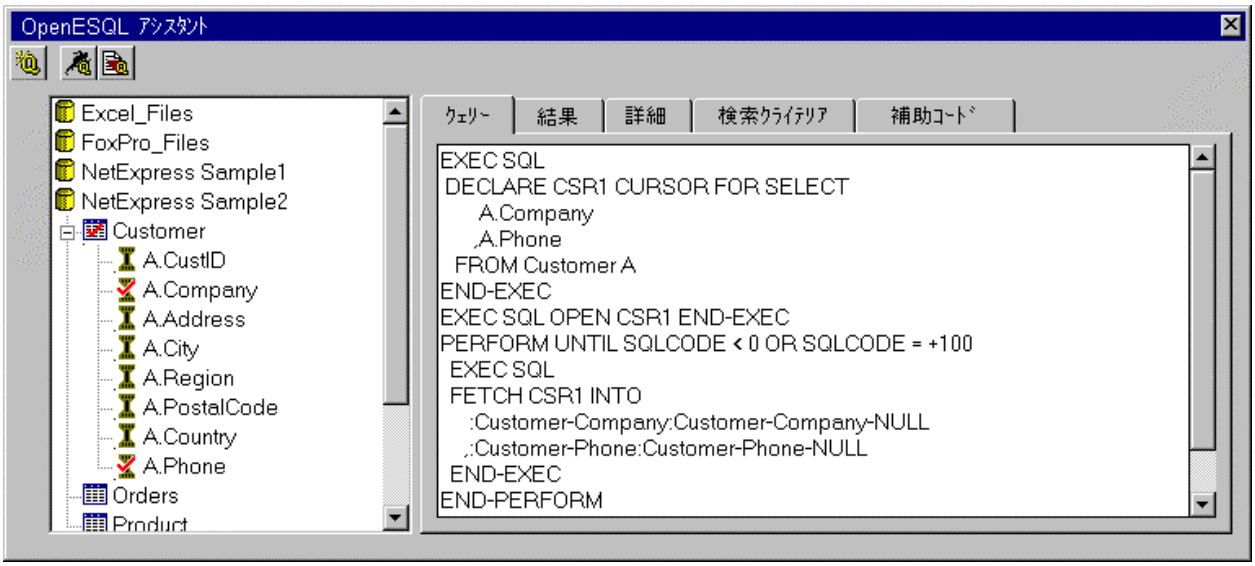

図 9-19 SELECT 文

この段階で、クエリーを実行すると、全顧客の企業名と電話番号が表示されます。

結合テーブルを作成するには、Orders テーブルをダブルクリックします。

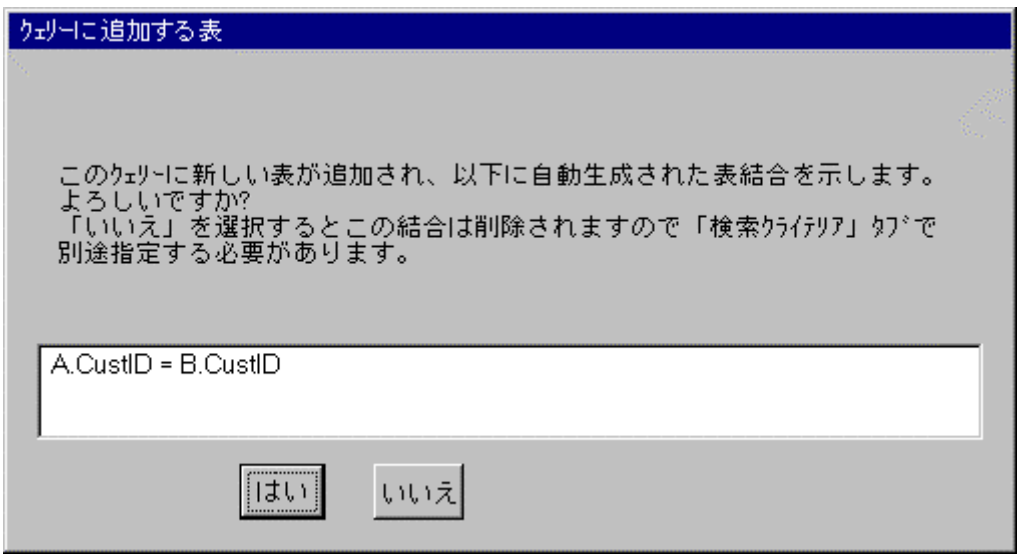

図 9-20 クエリーへのテーブルの追加

OpenESQL アシスタント は、最初に一致する列 (ここでは CustID) を使用して結合テーブルを作成します (この結合テーブル以外を使用するには、[No] ボタンをクリックして、[検索 クライテリア] タブで別の結合テーブルを指定します)。OpenESQL アシスタント により提示された結合テーブルを使用する場合は、[はい] ボタンをクリックします。その結果、COBOL コードに WHERE 句が自動的に追加されます。

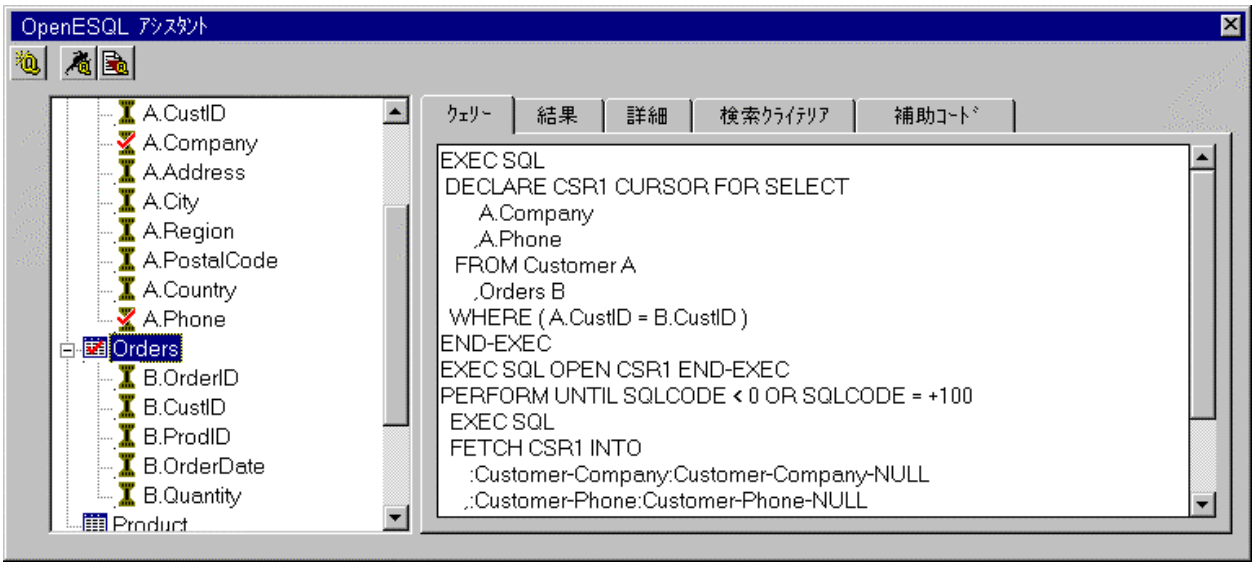

図 9-21 COBOL コードへの結合テーブルの追加

クエリーを完成させるには、次の操作を行います。

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

論理積 (AND) の選択と挿入は、自動的に行われることに注意してください。

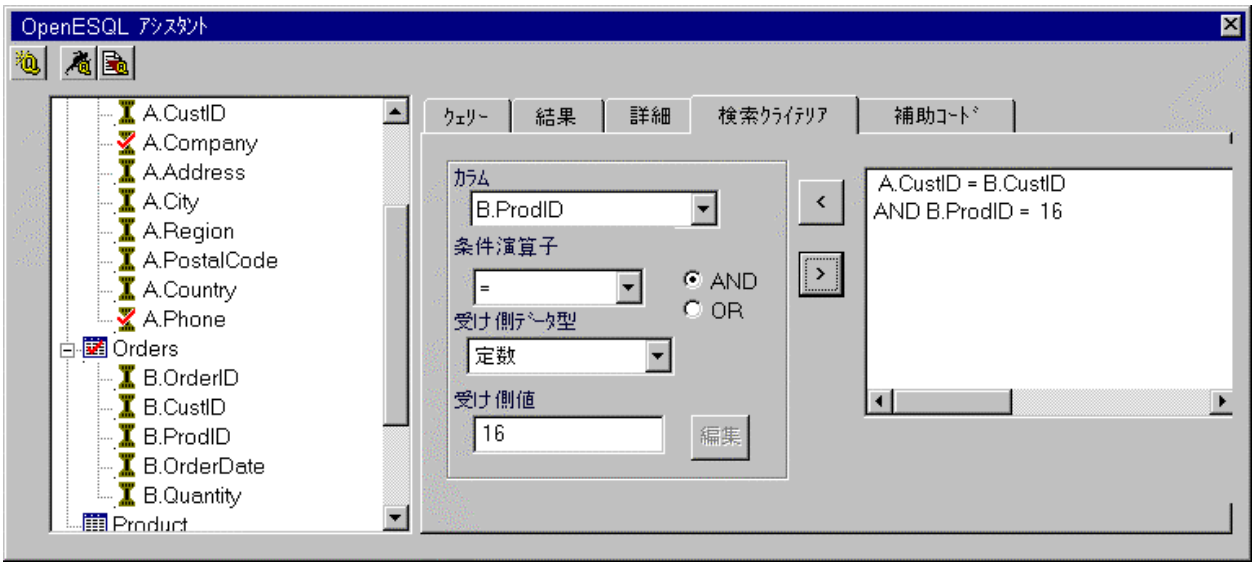

図 9-22 検索条件の指定

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

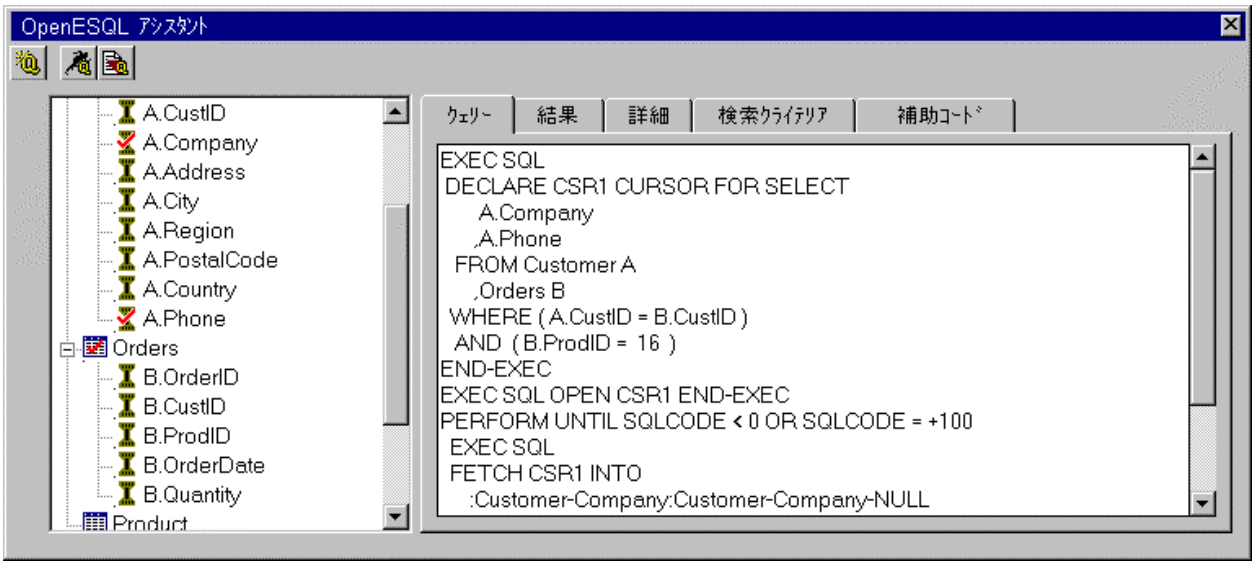

図 9-23 コードに追加された検索条件

[クエリーの実行]ボタン  をクリックして、クエリーを再実行すると、製品 ID が 16 である製品 (パブロワ) を注文した顧客の企業名と電話番号だけが表示されます。

図 9-24 コードの実行

9.11 プログラムへの埋め込み SQL の追加

SQL クエリーを正しく作成したら、今度は、[現在のプログラムにクエリーを挿入する] ボタン  をクリックして、このクエリーをプログラムに追加します。このとき、NetExpress で適切なプロジェクトが開いていること、また、コードを追加するプログラムも開いており編集できる状態であることを確認してください。このチュートリアルでは、次の操作を行います。

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

これで、プログラムの現在の挿入箇所に SQL クエリーが追加されます。

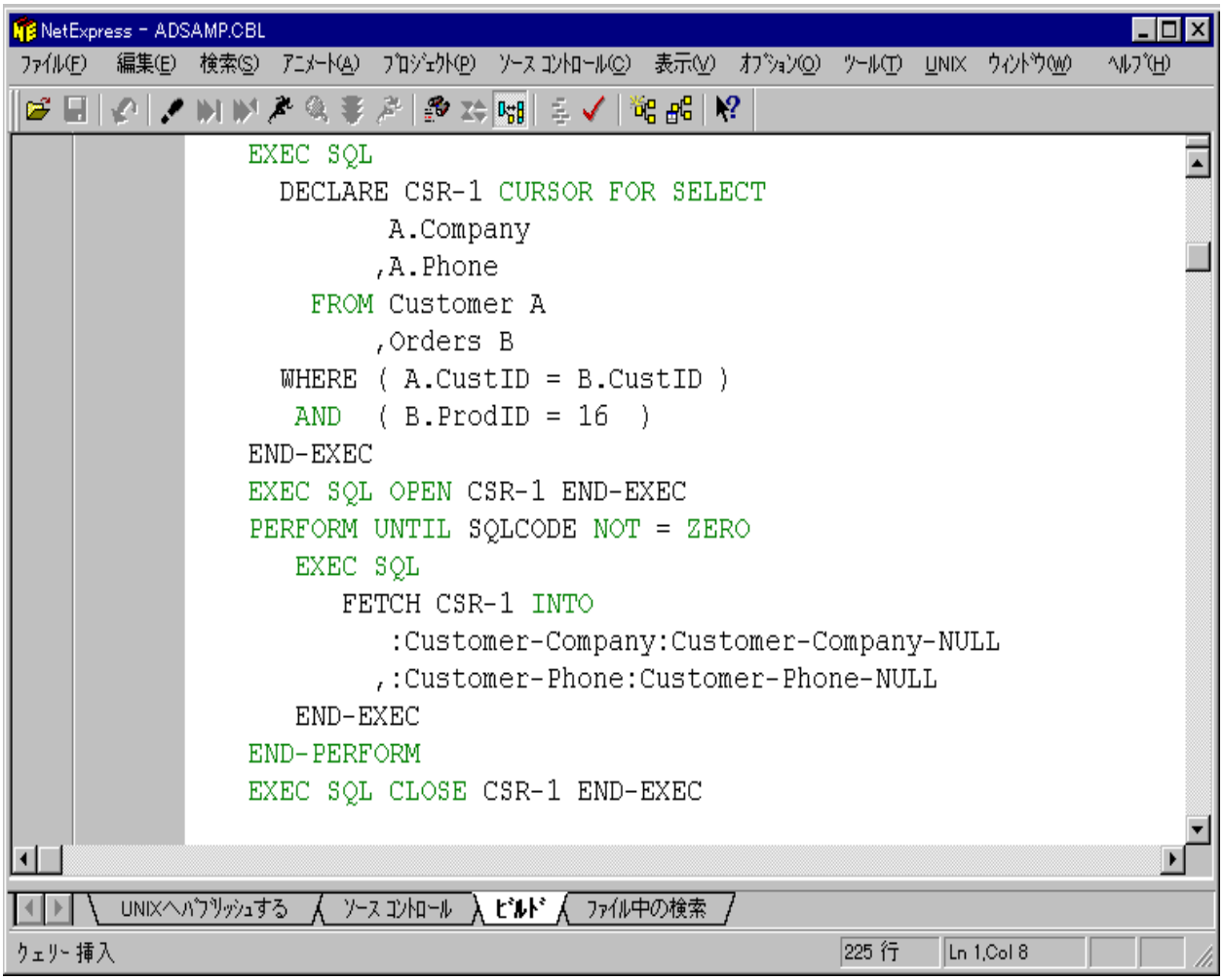

図 9-25 プログラムへの埋め込み SQL の追加

9.12 補助コードの追加

プログラムに埋め込み SQL を追加した後で、正常にプログラムを実行するために必要なコード修正を行います。例えば、データベースへアクセスする SQL を正常に実行するには、データ ソースへ接続するための CONNECT 文を挿入する必要があります。

OpenESQL アシスタント では、必要な補助コードを自動生成することができます。この機能を使用するには、[補助コード] タブをクリックします。その結果、次のような画面が表示されます。

図 9-26 [補助コード] タブ

OpenESQL アシスタント を使用すると、次の文や宣言を挿入することができます。

- CONNECT 文

  OpenESQL アシスタント により、現在のデータ ソース接続に関する情報をもとに CONNECT 文を自動生成することができます。

- SQLCA 宣言

  OpenESQL アシスタント により、INCLUDE 文を自動生成し、プログラムに SQL 通信領域 (SQLCA) を追加することができます。

- ホスト変数宣言

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

- DISCONNECT 文

  OpenESQL アシスタント により、DISCONNECT 文を自動生成し、現在のデータ ソースとの接続を解除することができます。

OpenESQL アシスタント では、自動生成した補助コードをプログラム中の現在の挿入箇所に追加します。そのため、補助コードを挿入する前に、プログラム中のカーソル位置が正しいことを確認する必要があります。例えば、CONNECT 文は、データ ソースへアクセスするためのコードの前に挿入する必要があります。また、SQLCA、またはホスト変数を含むコピーファイルを追加するための INCLUDE 文は、プログラムのデータ部に挿入する必要があります。

OpenESQL アシスタント を使用して挿入するコードと、手動で挿入するコードは、選択することができます。

OpenESQL アシスタント を使用して、補助コードを生成し挿入するには、次の操作を行います。

1. CONNECT 文 などの、生成するコードの型の横にあるオプション ボタンをクリックします。
2. プログラム中のカーソル位置が正しいか確認します。
3. [現在のプログラムにクエリーを挿入する] ボタン  をクリックします。

これで、プログラム中の現在の挿入箇所に補助コードが挿入されます。

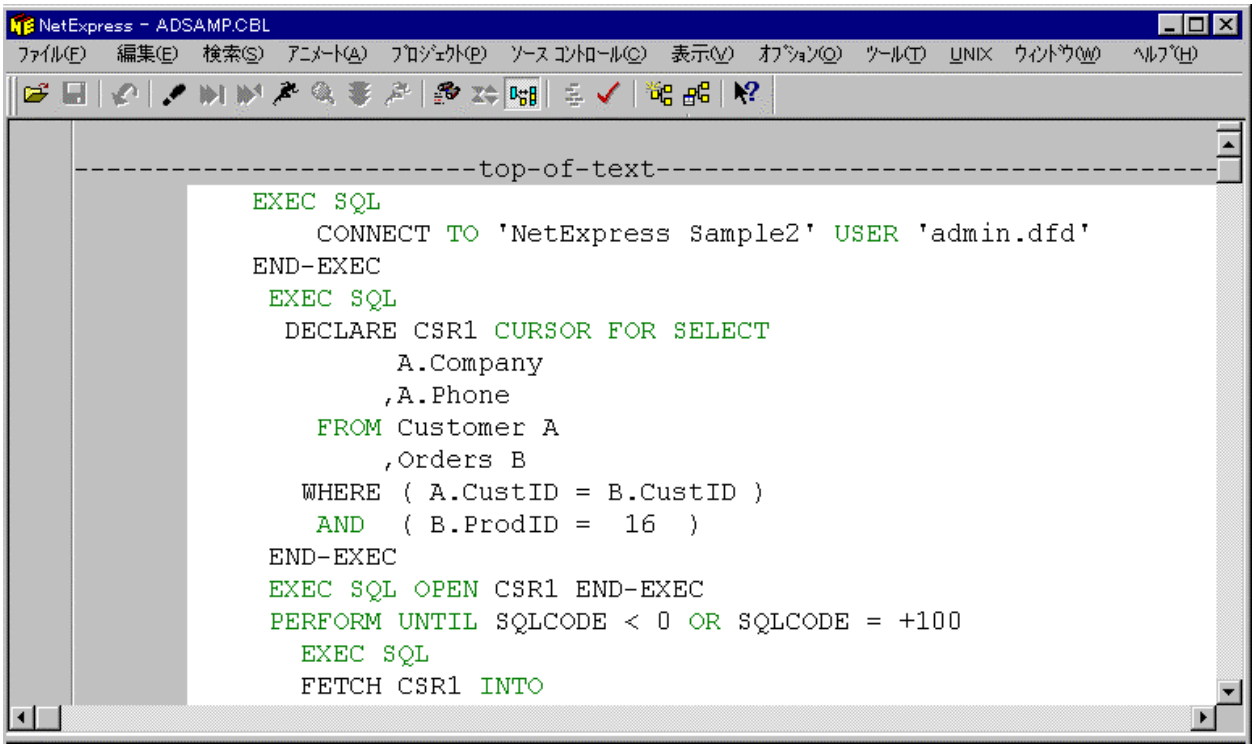

図 9-27 補助コードの挿入

9.13 OpenESQL アシスタント の終了

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

第10章 DB2

本章では、NetExpress を使用してコンパイルおよびリンクされた、埋め込み SQL 文を持つ 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 エクスターナル コンパイラ モジュール (ECM) は、NetExpress が提供する新しいタイプの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デザインされています。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出しに変換されます。

10.1 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 は次のデータ型をサポートします。

10.1.1 10 進数

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 として宣言することができます。

10.1.2 追加データ型

追加データ型は、次の形の 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--)--+
```

level_number
: 1 ～ 48 の範囲内

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

size
: BLOB 型、CLOB 型および DBCLOB 型の場合には必須で指定。K (キロバイトを意味します)、M (メガバイト) または G (ギガバイト)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グループ項目になります。グループ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、同等の 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既存のグループ項目は、COBOL では区別をつけることができません)。 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 だけです。TIMESTAMP はどちらの形式を使用しても宣言できます。

SQL 構文	同等の 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(26).

10.2 複合 SQL

現在 DB2 V5.2 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 に検知され、エラーが発生します。ただし、DB2 はこの状態からいつも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。

10.3 ユーザー定義関数

ユーザー定義関数 (UDF) への参照を含むプログラムは、他のモジュールを呼び出します。これは、そのモジュールに、適切な値を戻す、ユーザー提供のコードが含まれているからです。UDF コード自身は SQL を含んでいません。

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 でモジュールを書くことができますが、現在 DB2 で指定できる言語は C だけです。次の例は、これらがどのように行われるかを示します。ユーザー定義関数とパラメータ記述の詳細については、DB2 のマニュアルに提供されています。

COBOL で記述されたユーザー定義関数は、現在 UNIX ではサポートされていません。

備考 : クライアント/サーバーの構成では、UDF モジュールはサーバー上で呼び出され、これらの制限はサーバーだけに適用されます。サーバーに問題がなければ、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 セグメントでは、指数を計算する単純 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 をコンパイルすることができます。

```
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 が提供するものです。これは 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 呼び出し規約に準じています。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っては .dll または .dlw となる) 、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
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) を生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) 探し出せる場所に置く必要があります。

備考 : 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。

10.4 埋め込み SQL サポートの拡張

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

10.4.1 INCLUDE 文

次のような形式の文は、

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

次の文と全く同じように許可および処理されます。

```
copy filename
```

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

UNIX
ファイル名の指定に関わらず、AIX では sqlca の特殊な場合としてファイル名は小文字に変換されます。

10.4.2 DECLARE TABLE 文

次のような形の文は許可され、コメントとして扱われます。

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

10.4.3 整数ホスト変数

埋め込み SQL サポートでは、整数の形式を USAGE COMP-5 にする必要があります。ユーザー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ための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。

10.4.4 修飾付きホスト変数

ホスト変数は、DB2 for MVS 互換の構文を使用して修飾することができます。

たとえば、あるホスト変数を次のように定義したとします。

```
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
```

10.4.5 ホスト変数グループ (ホスト構造) とインジケータ配列 (標識配列)

グループ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順に参照するべき SQL 文が、代わりにグループ名を参照することによって、短縮されてしまうことがあります。インジケータ変数 (標識変数) 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持つインジケータ変数のテーブルを定義して、このテーブルへの参照を定義してください (OCCURS 句を持つ項目です。その項目を含むグループ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たとします。

```
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.
```

このような例では、次のような手続き文は、

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

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、同等の SQL 文を示すためだけに再定義されます (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。

10.4.6 NOT 演算子 (NOT キーワード) (¬)

DB2 では演算子 ¬=, ¬> および ¬< を使用することができます。これらは <>、 <= および >= にマップされます。NOT 演算子 (NOT キーワード) 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 演算子を定義することができます。

10.4.7 連結演算子 (ストリングの連結) (||)

国によっては、連結演算子 (ストリングの連結) 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することができます。

10.4.8 SQL 通信領域

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

```
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 ファイルがユーザーのソース コードに含まれます。このソース ファイルは、DB2 ECM を通ることにより、SQLCA の COBOL 定義を含んでいます。

この文を含めないと、DB2 ECM は自動的に領域を割当てますが、プログラム内でこの領域のアドレスを指定することはできません。ただし、SQLCODE および SQLSTATE のどちらか一方、または両方を宣言すれば、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ーが定義したフィールドにコピーします。SQLCA 全体を定義することをお勧めします (この機能は ANSI 互換のために用意されています)。

MFSQLMESSAGETEXT を定義してある場合には、SQLCODE の非ゼロ条件の後ろで、DB2 ECM は MFSQLMESSAGETEXT データ項目の内容を更新し、例外条件を記述します。この場合は、文字データ項目として宣言してください (PIC X(n) です。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。

10.4.9 オブジェクト指向 COBOL 構文のサポート

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

- DB2 コンパイラ指令の INIT オプションは、OO プログラム内で使用されると無効になります。つまり、DB2(INIT=PROT) コンパイラ指令の機能を使用したい場合には、非オブジェクト指向モジュールを使用し、そのモジュールをDB2 コンパイラ指令でコンパイルする必要があります。
- METHOD 内で EXEC SQL WHENEVER 文を使用すると、SQL 文を持つ同一 CLASS 内でコード化された追加 METHOD は、先行する定義済み WHENEVER 文内で参照されるセクションを持つ必要があります。これを行わないと、セクション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することができます。

10.4.10 入れ子の COBOL プログラムのサポート

DB2 ECM では、入れ子の COBOL プログラムを用いて作業することができます。

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。これを避けるには、DB2 指令 IGNORE-NESTED を使用してください。IGNORE-NESTED 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 コードを生成したいプログラム中に PROGRAM-ID を指定してください。指定しない場合は、エラーが発生します。

10.5 DB2 INIT 指令

DB2 の初期のバージョンでは、実行時に SQL 構文でデータベースへ接続する方法を提供していませんでした。そのため、呼出しを適切な 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、SQLINIT モジュールまたは SQLINI2 モジュールを提供していましたが、これらのルーチンは現在は提供されていません。代わりに、DB2 ECM が DB2 コンパイラ指令の INIT オプションの設定に応じて、適切な CONNECT 文を生成します。

INIT 指令には、アプリケーションが異常終了した場合でも、データベース接続を正しく終了させる追加オプションがあります。これにより、データベースの破損を防ぎます。アプリケーションが異常終了した場合には、最後に行った COMMIT 以降の変更はすべてロールバックされます。このデータベース保護は、DB2 コンパイラ指令で INIT=PROT オプションを指定することで選択できます。

INIT 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。その代わりに、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 オプションを持つ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と、プログラムが異常終了する可能性があります。

10.6 コンパイル

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 つのステップと論理的に同等です - 事前コンパイルによって SQL 行をホスト言語文に変更、次に結果ソースをコンパイルする。これらのステップ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 とともに行っています。

ユーザーが 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。次の*「DB2 コンパイラ指令」* を参照してください。

DB2 コンパイラ指令が必要となる場合を除いて、通常、埋め込み SQL を含むプログラムは SQL 以外のプログラムと同じ方法でコンパイルされます。追加モジュールをリンクする必要があるときに実行可能 (バイナリ) ファイルを作成する場合にだけ、特別な動作が必要です。SQL コードを含むプログラムはそのほかのプログラムと同じよう

にアニメートすることができます。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。

10.6.1 DB2 コンパイラ指令

DB2 コンパイラ指令のオプションを指定するには、プログラム中に $SET 文を使用します。

例：

```
$SET DB2(INIT=PROT BIND COLLECTION=MYSCHEMA)
```

10.6.1.1 デフォルト値

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 経由で DRDA サーバーに接続されているかどうかに依存します。このため、これらのオプションのデフォルトのコンパイラ設定は "not set" です。この場合は、DB2 には値が何も渡されず、デフォルト値は (適用可能な場合) DB2 自身によって判断されます。これらの値については、IBM DB2 の解説書を参照してください。

次の表は、DB2 コンパイラ指令のオプションを示します。デフォルト値は、太字に下線が引かれています。

オプション	説明
ACCESS=*package name*,ACCESS, NOACCESS	パッケージの名前を作成しデータベースに格納することを指定します。パラメータなしで ACCESS を指定すると、パッケージ名のデフォルトはプログラム名になります (.CBL 拡張子はつきません)。 同義語は PACKAGE です。
BIND=*bindfile*, BIND, NOBIND	作成するバインド ファイルの名前を指定します。パラメータなしで BIND を指定すると、バインド ファイルのデフォルトは、ファイル名の拡張子が .BND に置き換わったプログラム名になります。 同義語は BINDFILE です。

オプション	説明	
BLOCK={UNAMBIG | ALL | NO}	パッケージ作成の際に、レコード ブロック化モードを使用することを指定します。行ブロック化についての情報は、*『IBM DB2 管理の手引き』*、または *『アプリケーション・プログラミングの手引き』* を参照してください。	
	ALL	読込み専用カーソル、または FOR UPDATE OF として指定されていないカーソルのブロック化を指定します。不定カーソルは読込み専用として扱います。
	NO	カーソルをブロック化しないことを指定します。不定カーソルは更新可能として扱います。
	UNAMBIG	読込み専用カーソル、または FOR UPDATE OF として指定されていないカーソルのブロック化を指定します。不定カーソルは更新可能として扱います。
	同義語は BLOCKING です。	
COLLECTION=schema name, NOCOLLECTION	パッケージに COLLECTION (集合) 識別子 (8 文字長) を指定します。指定しない場合は、パッケージを処理しているユーザーの権限識別子 (許可 ID) が使用されます。	
COMMIT={1 | 2 | 3 | 4}	COMMIT 文が暗黙に生成される場所を指定します。	
	1	COMMIT 文は暗黙に生成されません。
	2	STOP RUN 文とプログラムの最後に COMMIT 文が暗黙に生成されます。
	3	STOP RUN 文、EXIT PROGRAM 文、およびプログラムの最後に COMMIT 文が暗黙に生成されます。
	4	すべての SQL 文の後に COMMIT が暗黙に生成されます。
CONCAT=(ascii character code | 33)	CONCAT 記号 (|) に使用する ASCII 文字コードを指定します。	
CONNECT={1 | 2}, NOCONNECT	CONNECT 文をタイプ 1 CONNECT、またはタイプ 2 CONNECT として処理することを指定します。	

オプション	説明	
CTRACE, NOCTRACE	サポート担当へトレース ファイルの提出が必要な場合に、トレース ファイルを作成します。作成されるファイルのファイル名は sqltrace.txt です。	
DB=*database name*, DB	プログラムがアクセスするデータベースの名前を指定します。パラメータなしで DB を指定すると、環境変数 DB2DBDFT で指定されたデータベースが使用されます。	
DEFERRED_PREPARE={NO | YES | ALL}, NODEFERRED_PREPARE	DB2 共通サーバー データベース、または DRDA データベースにアクセスする時のパフォーマンスを拡張します。このオプションは、SQL PREPARE 文のフローと、それに関連する OPEN 文、DESCRIBE 文、または EXECUTE 文のフローを結び付け、プロセス間、またはネットワークのフローを最小化します。	
	NO	PREPARE 文は、発行と同時に実行されます。
	YES	PREPARE 文の実行は、対応する OPEN 文、DESCRIBE 文、または EXECUTE 文が発行されるまで遅延されます。PREPARE 文に INTO 句を使用した場合には、遅延されません。INTO 句は SQLDA がすぐに戻される必要があるからです。ただし、パラメータ マーカーを使用しないカーソルに対して PREPARE INTO 句が発行される場合には、PREPARE の実行時に、事前に OPEN されているカーソルによって処理が最適化されます。
	ALL	パラメータ マーカーを含む PREPARE INTO 句が遅延されることを除いて、YES と同様です。PREPARE INTO 文がパラメータ マーカーを含まない場合でも、カーソルの事前 OPEN が行われます。PREPARE 文が SQLDA を戻す INTO 句を使用している場合には、OPEN 文、DESCRIBE 文、または EXECUTE 文が発行されて戻されるまで、アプリケーションはこの SQLDA の内容を参照することはできません。

オプション	説明	
DEGREE={1 | degree-of-parallelism | ANY}, NODEGREE	I/O 並列処理を使用してクエリーを実行するかどうかを指定します。	
	1	並列 I/O 操作を禁止します。
	degree-of-I/O-parallelism	並列 I/O 操作の次数を指定します。値は 2 ～ 32767 (包括的) です。
	ANY	並列 I/O 操作を許可します。
DISCONNECT={EXPLICIT | CONDITIONAL | AUTOMATIC}, NODISCONNECT	AUTOMATIC	コミット時にすべてのデータベース接続を切断することを指定します。
	CONDITIONAL	データベース接続が RELEASE とマークされている場合、またはオープンしている WITH HOLD カーソルを持たない場合には、コミット時に切断することを指定します。
	EXPLICIT	RELEASE 文によって明示的にリリースをマークされているデータベース接続だけを、コミット時に切断することを指定します。
EXPLAIN={NO | YES | ALL}, NOEXPLAIN	パッケージの各 SQL 文に対して選択されたアクセス プランに関する情報を Explain テーブルに格納します。DRDA はこのオプションの ALL 値をサポートしていません。	
	NO	Explain 情報は取り込まれません。
	YES	Explain テーブルを生成して、選択されたアクセス プランの情報を格納します。
	ALL	適格な静的 SQL 文それぞれに対する Explain 情報が Explain テーブルに置かれます。さらに、CURRENT EXPLAIN SNAPSHOT レジスタが NO に設定されている場合でも、適格な動的 SQL 文の Explain 情報が実行時に集められます。特殊レジスタの詳細については、*『IBM DB2 SQL 解説書』* を参照してください。

オプション	説明	
EXPLSNAP={NO | YES | ALL}, NOEXPLSNAP	Explain Snapshot 情報を Explain テーブルに格納します。この DB2 の事前コンパイル/バインド オプションは DRDA ではサポートされていません。	
	NO	Explain Snapshot は取り込まれません。
	YES	適格な静的 SQL 文に対する Explain Snapshot が Explain テーブルに置かれます。
	ALL	適格な静的 SQL 文それぞれに対する Explain Snapshot が Explain テーブルに置かれます。さらに、CURRENT EXPLAIN SNAPSHOT レジスタが NO に設定されている場合でも、適格な動的 SQL 文の Explain Snapshot 情報が実行時に集められます。特殊レジスタの詳細については、*『IBM DB2 SQL 解説書』* を参照してください。
FORMAT={DEF | USA | EUR | ISO | JIS | LOC}	日付/時刻フィールドがホスト変数の列表現に割り当てられたときの日付時刻形式を指定します。DEFはデータベースの国番号に関連付けられている日付時刻形式です。	
	EUR	欧州日付時刻形式の IBM 標準。
	ISO	国際標準化機構の日付時刻形式。
	JIS	日本工業規格の日付時刻形式。
	LOC	データベースの国番号に関連付けられているローカルの日付時刻形式。
	USA	米国日付時刻形式の IBM 標準。
	同義語は DATETIME です。	

<table>
<tr><th>オプション</th><th colspan="2">説明</th></tr>
<tr><td>FUNCPATH=schema-name , NOFUNCPATH</td><td colspan="2">静的 SQL 内でユーザー定義の互いに異なる型と関数を解決するために使用される関数パスを指定します。このオプションを指定しない場合、デフォルトの関数パスは次のとおりです。

<code>"SYSIBM","SYS FUN",USER</code>

USER は USER 特殊レジスタの値です。この DB2 事前コンパイル/バインド オプションは DRDA ではサポートされていません。</td></tr>
<tr><td></td><td>schema-name</td><td>短い SQL 識別子 (SQL 短識別子)。通常識別子または区切り識別子であり、アプリケーション サーバーに存在するスキーマを識別します。事前コンパイル時やバインド時に、スキーマが存在するかどうかの妥当性検査は行いません。同じスキーマが関数パスに複数回現れることはできません。指定できるスキーマの数は結果関数パスの長さによって制限され、最大 254 バイト以下です。 SYSIBM スキーマを明示的に指定する必要はありません。SYSIBM スキーマは、関数パスに含まれていなくても、最初のスキーマであることが暗黙の前提とされます。詳細は、『IBM DB2 SQL 解説書』 を参照してください。</td></tr>
<tr><td>IGNORE-NESTED=program-id, IGNORE-NESTED, NOIGNORE-NESTED</td><td colspan="2">入れ子のプログラム中で、DB2 インターフェイス コードの生成を開始する個所の program-id を指定します。program-id よりも前に現れる入れ子のプログラムは無視され、DB2 インターフェイス コードは生成されません。program-id は COBOL ソース コードで指定してください。そうしないと、コンパイル エラーになります。パラメータなしで IGNORE-NESTED を指定すると、program-id のデフォルトは、拡張子が .CBL に置き換わったプログラム名になります。</td></tr>
<tr><td>INIT={PROT | s | x }, NOINIT</td><td colspan="2">プログラムで SQL を初期化します。このオプションは、OO プログラム内で使用された場合には、無効になります。</td></tr>
<tr><td></td><td>PROT</td><td>STOP RUN 時にデータベースを保護する必要があるが、初期化したくない場合に、SQL プログラムに使用します。</td></tr>
<tr><td></td><td>S</td><td>共有モードでデータベースを使用します。</td></tr>
<tr><td></td><td>X</td><td>排他モードでデータベースを使用します。</td></tr>
</table>

オプション	説明	
INSERT={DEF | BUF}, NOINSERT	プログラムを事前コンパイル、またはバインドして、DB2 クライアントから DB2 サーバーへ、パフォーマンス向上のためにデータ挿入のバッファを要求することを許可します。	
	BUF	アプリケーションからの挿入をバッファします。
	DEF	アプリケーションからの挿入をバッファしません。
ISOLATION={CS | RR | UR | RS | NC}	このパッケージにバインドされたプログラムが、実行中の他のプログラムの影響から分離される程度を判断します。分離レベルについての詳細は、*『IBM DB2 SQL 解説書』* を参照してください。.	
	CS	カーソル固定を分離レベルとして指定します。
	NC	(No Commit) コミットメント コントロールを使用しないことを指定します。この分離レベルは DB2 ではサポートされていません。
	RR	繰返し可能読取り (反復可能読取り) を分離レベルとして指定します。
	RS	読取り固定を分離レベルとして指定します。読取り固定を使用すると、パッケージ内の SQL 文は、他のアプリケーションが読み込んで変更した行に対するプロセスから分離されて実行されます。
	UR	非コミット読取りを分離レベルとして指定します。

オプション	説明	
LANGLEVEL={SAA1 | NONE | MIA}	このオプションについての詳細は、*『IBM DB2 アプリケーション・プログラミングの手引き』* を参照してください。	
	MIA	FOR UPDATE 句は位置指定更新のオプションです。C の NULL で終わる文字列は、空白文字で詰められ、常に NULL 終端文字を含みます。このオプションは DDCS ではサポートされていません。
	SAA1	位置指定更新で更新されたすべての列に FOR UPDATE 句が必要です。C の NULL で終わる文字列は空白文字で詰められ、切捨てが起こった場合には NULL 終端文字を含みません。
	NONE	SAA1 と同義語です。
	同義語は STDLVL です。	
MSGAREA={data-item-name | MFSQLMESSAGETEXT },NOMSGAREA)	英数字データ項目の名前を指定します。この項目がプログラム ソースに存在する場合は、自動的に DB2 エラー条件がこの項目に記述されます (SQLCODE がゼロではないとき)。	
NOT={ascii character code | 170 }	NOT 文字 (¬) に使用する ASCII 文字コードを指定します。	
PASS={password |userid.password}, NOPASS	データベースに接続する userid と password を指定します。	
QUALFIX, NOQUALFIX	ホスト変数を DB2 に宣言するときに、DB2 ECM がこれらのホスト変数の名前に 3 文字を追加します。これにより、修飾によって起こる問題 (修飾されないと 2 つ以上のホスト変数が同じ名前になる場合) を避けることができますが、次の副作用があります。	
	(1)	ホスト変数名の最大長は 27 文字。
	(2)	DB2 エラー メッセージで、ホスト変数の名前に 3 つの追加文字が付加されて表示されることがある。

オプション	説明	
QUERYOPT= optimization-level, NOQUERYOPT	パッケージに含まれるすべての静的 SQL 文に対して望ましい最適化レベルを示します。デフォルト値は 5 です。使用可能な最適化レベルの詳細は、*「SQL 解説書」* の SET CURRENT QUERY OPTIMIZATION 文を参照してください。この DB2 事前コンパイル/バインド オプションは DRDA ではサポートされていません。	
SQLFLAG={MVSDB2V23 | MVSDB2V31 | MVSDB2V41}-SYNTAX, NOSQLFLAG	指定した SQL 言語構文から、偏差を識別し報告します。 sqlflag オプションに加えて、bindfile か package オプションが指定された場合にだけ、バインド ファイルまたはパッケージが作成されます。 次のいづれかのオプションが指定された場合にだけ、ローカル構文検査が行われます。bindfile、package、sqlerror check、syntax。 sqlflag を指定しない場合には、フラッガ機能は呼び出されず、バインド ファイルまたはパッケージは影響されません。	
	MVSDB2V23-SYNTAX	MVS DB2 バージョン 2.3 SQL 言語構文に対して SQL 文が検査されます。構文からの偏差はすべてプリコンパイラ リストに報告されます。
	MVSDB2V31-SYNTAX	MVS DB2 バージョン 3.1 SQL 言語構文に対して SQL 文が検査されます。構文からの偏差はすべてプリコンパイラ リストに報告されます。
	MVSDB2V41-SYNTAX	MVS DB2 バージョン 4.1 SQL 言語構文に対して SQL 文が検査されます。構文からの偏差はすべてプリコンパイラ リストに報告されます。
	同義語は FLAG です。	

オプション	説明	
SQLRULES={DB2 | STD}, NOSQLRULES	タイプ 2 CONNECT を DB2 の規則に準じて処理するか、ISO/ANS SQL92 に基づく標準 (STD) 規則に準じて処理するかを指定します。	
	DB2	現行の接続から他の確立済み (休眠中 (休止状態)) 接続へ切り替える SQL CONNECT 文の使用を許可します。
	STD	新規接続を確立するだけの SQL CONNECT 文の使用を許可します。休眠中の接続へ切り替えるには、SQL SET CONNECTION 文を使用してください。
	同義語は RULES です。	
SQLWARN={YES | NO}, NOSQLWARN	動的 SQL 文のコンパイルから (PREPARE または EXECUTABLE IMMEDIATE 経由で) 警告が戻されるか、または記述処理から (PREPARE...INTO または DESCRIBE 経由で) 警告が戻されるかを示します。この DB2 事前コンパイル/バインド オプションは DRDA ではサポートされていません。	
	NO	SQL コンパイラから警告は戻されません。
	YES	SQL コンパイラから警告が戻されます。
	備考 : SQLCODE +238 は例外です。この警告は sqlwarn オプションの値に関わらず戻されます。 同義語は WARN です。	
SYNCPOINT={ONEPHASE | TWOPHASE | NONE}, NOSYNCPOINT	複数のデータベース接続間でどのようにコミットまたはロールバックを調整するかを指定します。	
	NONE	トランザクション マネージャ (TM) を使用して 2 フェーズ コミットを行わないことを指定します。単一の更新操作、複数の読取り操作を強制しません。COMMIT は各参加データベースに送られます。コミットがどれか失敗した場合には、アプリケーションに回復責任があります。

オプション	説明	
	ONEPHASE	TM を使用して 2 フェーズ コミットを行わないことを指定します。複数データ トランザクション内の各データベースによって行われた作業をコミットするには、1 フェーズ コミットが使用されます。
	TWOPHASE	TM を使用して、このプロトコルをサポートするデータベース間の 2 フェーズ コミットを調節することを指定します。

10.6.1.2 DRDA DB2 **指令**

DRDA サーバーへ接続している場合にだけ、次のオプションが有効です。

オプション	説明	
ACTION={ADD | REPLACE }, NOACTION	ACTION はパッケージが追加できるか、または置きかえられるかどうかを示します。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。	
	ADD	名前付きパッケージが存在せず、新規パッケージが作成されることを示します。パッケージが既に存在する場合には、実行は停止し、診断エラー メッセージが戻されます。
	REPLACE	古いパッケージが、同じ位置、集まり、およびパッケージ名を持つ新規パッケージ置きかえられることを示します。
CCSIDG=double-ccsid , NOCCSIDG	コード化文字セット識別子 (CCSID) を、CREATE 文 および ALTER TABLE SQL 文の (特定の CCSID 句なしの) 文字列定義の 2 バイト文字に使用できることを指定する整数。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。このオプションが指定されない場合には、DRDA サーバーはシステム定義のデフォルト値を使用します。	
CCSIDM=mixed-ccsid , NOCCSIDM	コード化された文字セット識別子 (CCSID) を、CREATE 文 および ALTER TABLE SQL 文の (特定の CCSID 句なしの) 文字列定義の混合バイト文字に使用できることを指定する整数。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。このオプションが指定されない場合には、DRDA サーバーはシステム定義のデフォルト値を使用します。	

オプション	説明	
CCSIDS=sbcs-ccsid, NOCCSIDS	コード化された文字セット識別子 (CCSID) を、CREATE 文および ALTER TABLE SQL 文の (特定の CCSID 句なしの) 文字列定義の 1 バイト文字に使用できることを指定する整数。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。このオプションが指定されない場合には、DRDA サーバーはシステム定義のデフォルト値を使用します。	
CHARSUB={DEFAULT | BIT | SBCS | MIXED}, NOCHARSUB	CREATE 文および ALTER TABLE SQL 文の列定義に使用されるデフォルトの文字サブタイプを指定します。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。	
	BIT	明示的にサブタイプが指定されていない新規文字列のすべてに FOR BIT DATA SQL 文字サブタイプを使用します。
	DEFAULT	明示的にサブタイプが指定されていない新規文字列のすべてに、対象システムが定義したデフォルトを使用します。
	MIXED	明示的にサブタイプが指定されていない新規文字列のすべてに FOR MIXED DATA SQL 文字サブタイプを使用します。
	SBCS	明示的にサブタイプが指定されていない新規文字列のすべてに FOR SBCS DATA SQL 文字サブタイプを使用します。
DEC={31 | 15} , NODEC	10 進数算術演算に使用する最大精度を指定します。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。このオプションが指定されない場合には、DRDA サーバーはシステム定義のデフォルト値を使用します。 15 を指定すると、10 進数算術演算に 15 桁の精度を使用します。 31 を指定すると、10 進数算術演算に 31 桁の精度を使用します。	

オプション	説明	
DECDEL={PERIOD | COMMA}, NODECDEL	小数点定数および浮動小数点定数の小数点識別子として、ピリオド (.) またはコンマ (,) のどちらを使用するかを指定します。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。このオプションが指定されない場合には、DRDA サーバーはシステム定義のデフォルト値を使用します。	
	COMMA	コンマ (,) を小数点識別子として使用します。
	PERIOD	ピリオド (.) を小数点識別子として使用します。
DYNAMICRULES={BIND | RUN}, NODYNAMICRULES	パッケージの動的 SQL が実行されたときに、どの権限識別子 (許可 ID) を使用するかを指定します。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。	
	BIND	動的 SQL の実行に使用される権限識別子 (許可 ID) はパッケージの所有者であることを示します。
	RUN	動的 SQL の実行に使用される権限識別子 (許可 ID) はパッケージ実行者の authid であることを示します。
LEVEL=consistency-token, NOLEVEL	一貫性トークンを使用しているモジュールのレベルを定義します。consistency token は、長さ 8 文字までの英数字の値です。RDB パッケージの一貫性トークンは、リクエスタのアプリケーションとリレーショナル データベース パッケージが同期していることを検証します。この DRDA の事前コンパイル オプションは DB2 ではサポートされていません。 備考 : 通常の場合は、このオプションを使用しないことをお薦めします。	
OWNER=authorization-id, NOOWNER	パッケージの所有者に権限識別子 (許可 ID) (8 文字長) を指定します。所有者は、パッケージで SQL 文を実行するのに必要な権限を持つ必要があります。このオプションが明示的に指定されていない場合には、デフォルトは事前コンパイル/バインド プロセスの一次権限識別子 (1次許可 ID) です。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。 同義語は SCHEMA です。	

オプション	説明	
QUALIFIER=qualifier-name, NOQUALIFIER	パッケージに含まれる修飾なしのテーブル名、ビュー、索引およびエイリアス (別名) に 18 文字の暗黙の修飾子を与えます。所有者が明示的に指定されているかどうかに関わらず、デフォルトは所有者の権限 ID です。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。 同義語は CATALOG です。	
RELEASE={COMMIT | DEALLOCATE}, NORELEASE	各 COMMIT ポイントで、またはアプリケーションの終了時にリソースが開放されるかどうかを示します。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。	
	COMMIT	各 COMMIT ポイントでリソースを解放します。動的 SQL 文に使用されます。
	DEALLOCATE	アプリケーションの終了時にだけリソースを解放します。
REPLVER=version-id, NOREPLVER	パッケージの特定のバージョンを置き換えます。バージョン識別子は、どのバージョンのパッケージを置き換えるかを指定します。最大長は 254 文字です。	
RETAIN={YES | NO} , NORETAIN	RETAIN はパッケージが置き換えられるときに、EXECUTE 権限が保持されるかどうかを示します。パッケージの所有者が変わった場合は、新しい所有者が以前の所有者に BIND 権限と EXECUTE 権限を付与します。	
	NO	パッケージを置き換えるときに EXECUTE 権限を保持しません。
	YES	パッケージを置き換えるときに EXECUTE 権限を保持します。

オプション	説明	
SQLERROR={NOPACKAGE | CHECK | CONTINUE}, NOSQLERROR	エラーが発生した場合に、パッケージを作成するか、またはファイルをバインドするかどうかを示します。	
	CHECK	対象システムが、バインドされている SQL 文の構文およびセマンティックスの検査をすべて行うことを指定します。パッケージはこのプロセスの部分としては作成されません。パッケージの作成中に、同じ名前とバージョンを持つ既存のパッケージがある場合には、既存のパッケージは削除されるか、または ACTION REPLACE が指定されていれば、置き換えられます。
	CONTINUE	SQL エラーが発生した場合でも、パッケージやバインド ファイルは作成されます。このオプションは DB2 ではサポートされていません。
	NOPACKAGE	エラーが発生した場合には、パッケージやバインド ファイルは作成されません。
	構文が package オプションとともに使用されると、package は無視されます。 同義語は ERROR です。	
STRDEL={APOSTROPHE | QUOTE}, NOSTRDEL	SQL 文内の文字区切りとしてアポストロフィー (') または引用符 (") のどちらを使用するか指定します。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。このオプションが指定されない場合には、DRDA サーバーはシステム定義のデフォルト値を使用します。 APOSTROPHE を指定すると、アポストロフィー (') を文字区切りとして使用します。 QUOTE を指定すると、引用符 (") を文字区切りとして使用します。	
SYNTAX	SQLERROR=CHECK 指令の同義語です。	
TEXT=label, NOTEXT	パッケージの記述。最大長は 255 文字です。デフォルト値は空白です。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。	

オプション	説明	
VALIDATE={RUN | BIND}, NOVALIDATE	データベース管理者が権限エラーとオブジェクトが存在しないエラーを検査する時を決めます。妥当性検査にはパッケージ所有者の権限 ID が使用されます。この DRDA 事前コンパイル/バインド オプションは DB2 ではサポートされていません。	
	BIND	妥当性検査は事前コンパイル/バインド時に行われます。オブジェクトがすべて存在しない場合や、権限がすべて所持されていない場合には、エラー メッセージが生成されます。SQLERROR CONTINUE を指定している場合には、エラーメッセージの代わりにパッケージまたはバインド ファイルが生成されます。ただし、エラー中の文は実行可能ではありません。
	RUN	妥当性検査はバインド時に行われます。オブジェクトがすべて存在し、権限がすべて所持されている場合には、実行時に再検査は行われません。 事前コンパイル/バインド時にオブジェクトがすべて存在しない場合、または権限がすべて所持されていない場合には、警告メッセージが生成され、SQLERROR CONTINUE オプションの設定に関わらず、パッケージは問題なくバインドされます。ただし、事前コンパイル/バインド時に失敗した SQL 文の権限検査と存在検査は、実行時に再度行われる可能性があります。
VERSION=version-id, NOVERSION	パッケージのバージョン識別子を定義します。バージョン識別子は、英数字、$、#、@、_、-、または . で、長さは 254 文字以内です。この DRDA 事前コンパイル オプションは DB2 ではサポートされていません。	

10.7 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戻されます。これらのメッセージの詳細については、ご使用のデータベース システムのマニュアルに説明されています。ホスト変数を参照しているメッセージでは、ホスト変数の名前がわずかに変わります。ハイフンが下線 (_) になり、3 文字以下の文字が名前の終わりに追加されますが、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。

実行時のエラー条件は、SQLCODE の非ゼロ値によって示されます。MFSQLMESSAGETEXT の定義をすれば、MFSQLMESSAGETEXT データ項目に説明文が置かれます。このデータ項目についての詳細は、上述の *「SQL 通信領域 (連絡域)」* セクションを参照してください。

例

```
801-S

**    External Compiler Module message

**    SQ0100 SQL1032N  No start database manager command was issued.

**    SQLSTATE=57019
```

10.8 リンク

アプリケーションにリンクするには、次の手順を行ってください。

1. NetExpress 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 ファイルまたは .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
   ```

10.9 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、ご使用の SQL システムのマニュアルを参照してください。

10.10 DB2 アプリケーションを UNIX でパブリッシュする

DB2 アプリケーションを UNIX でパブリッシュしたい場合には、次の手順を行ってください。

1. 次のように、ログイン環境を構成して、COBOPT 環境変数をデータベース パラメータ ファイルの名前に設定します。

   ```
   COBOPT=/usr/lpp/db2_vv_rr_mmmm/lib/db2mkrts.args

   export COBOPT
   ```

   vv、*rr* および *mmmm* の値は、マシンにインストールされている DB2 システムのバージョン、リリースお

よびモディフィケーション レベルを表します。

2. ファイルの先頭行から -vdd オプションを削除して、COBOPT 環境変数で名付けられたファイルを編集してください。

3. $COBDIR/src/sql ディレクトリから次のように入力してください。

```
mkrts sqlinit.o

cp rts32 $COBDIR
```

第11章 COBSQL

COBSQLは、Micro Focus COBOLといくつかのサード・パーティ製データベース・ プリコンパイラの間の統合化インタフェースを提供します。以下のプリコンパイラを対象とします:

- Oracle Pro*COBOL Version 1.8

COBSQLは、すでに上記のプリコンパイラを Micro Focus COBOL製品とともに使用していて アプリケーションをNetExpressに移行したい場合にのみ使用するようお勧めします。 または、UNIXで実行するアプリケーションをクロス開発している場合にも使用できます。

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 お勧めします。

備考 Oracleのプリコンパイラは入れ子のプログラムをサポートしていません。 また、COBSQLはオブジェクト指向COBOL構文をサポートしていません。 OO COBOLを使用する場合には、OpenESQLを使用して下さい。

11.1 概要

COBOL プログラムに埋め込み SQL ステートメントを記述すれば、Oracle や Sybase 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。埋め込み SQL は次の形式で記述します。

```
EXEC SQL
```

SQL ***ステートメント***

```
END-EXEC
```

埋め込み SQL ステートメント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 （または Sybase）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 ステートメント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。

この方法では、各データベース ルーチンの呼び出し形式に留意する必要がありません。ただし、プログラムのアニメーションではプリコンパイラの出力コードが表示され、埋め込み SQL ステートメントを含むオリジナルのソースコードを見ることができません。この問題は COBSQL を使用すれば回避できます。

COBSQL は サードパーティ のスタンドアロン型プリコンパイラと Micro Focus COBOL の環境を統合するインタフェースとして機能します。EXEC SQL ステートメントを含むプログラムを COBSQL で処理すれば、アニメーションでプリプロセッサの出力コードの代わりに変換前のソース コードを表示することが可能になります。

この章では、Oracle や Sybase のプリコンパイラと COBSQL を 組み合わせ、プログラムのコンパイルとアニメーションを実行する方法について説明します。

11.2 使用方法

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 ステートメントでプログラム内に記述できます。NetExpress の「ビルド設定」ダイアログボックスで設定することも可能です。

COBSQL への指令送信を終了するには、COBOL の ENDP 指令を使用する必要があります。プロジェクト設定（または NetExpress の「ビルド設定」ダイアログボックス）に次の行を追加します。

```
Preprocess(Cobsql) csqltype=oracle end-c comp5=yes endp;
```

NetExpress 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、システムによって標準の COBOL 指令が追加されるため、かなら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「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 を介し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。したがって、ENDP指令を記述しないと、COBSQL への指令送信が継続され、COBOL コンパイラには指令が渡されません。

11.2.1 指令の指定

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、かならず PREPROCESS"COBSQL" を記述する必要があります。

COBSQL 指令は、NetExpress 標準の指令 ファイル（cobol.dir）に記述することも可能です。

---

cobol.dir ファイルに関する注意点

- 各行には 1 つ以上のコンパイラ 指令を指定できます。
- 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。
- P(COBSQL) または PREPROCESS(COBSQL) が検出されると、同じ行の残りの部分（または ENDP までの部分）はプリプロセッサに渡されます。

- プリプロセス ステートメント（P(COBSQL) または PREPROCESS(COBSQL)）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 指令として一括処理されます。したがって、プリプロセス ステートメントと一連のオプションの組み合わせは、かならず 1 つの行に記述する必要があります。

COBSQL 指令とプリコンパイラ 指令は、専用のファイル（cobsql.dir）に記述することも可能です。このファイルは、カレント ディレクトリか、または $COBDIR で指定されたディレクトリに保存します。COBSQL はカレント ディレクトリ、$COBDIR で指定されたディレクトリの順序で cobsql.dir ファイルを検索します。この検索はファイルが見つかった時点で終了するため、cobsql.dir ファイルをカレント ディレクトリに保存した場合には、$COBDIR で指定されたディレクトリは検索されません。

---

cobsql.dir ファイルに関する注意点

- 各行には 1 つ以上のOBSQL 指令を指定できます。
- 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。
- 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。
- END-C、END、または END-COBSQL が検出されると、同じ行の残りの部分はデータベース プリコンパイラに渡されます。
- データベース プリコンパイラに渡す一連のオプションは、すべて同じ行に記述してください。

COBSQL は、まず cobsql.dir ファイルを処理し、つづいて「ビルド設定」ダイアログボックスで設定された各指令を処理します。

多くの指令は、直前に NO を記述すれば無効化できます。たとえば、DISPLAY を無効化するには NO DISPLAY と記述します。以下の指令一覧では、NO で無効化できる指令にはアスタリスク（*）が付けられています。なお、標準ではすべての指令が無効（NO 付きで設定した状態）です。

一部の指令は短縮名で指定することが可能です。以下の指令一覧では、該当する指令の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 に渡すことが可能な指令もあります。（以下の「*COBOL 指令*」参照。）この方法を使えば、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) csqlt=ora makesyn end-c comp5=yes
```

```
mode=ansi endp omf(gnt) list();
```

- このコマンドラインでは、`end-c` までに 2 つの COBSQL 指令（`csqlt=ora` および `makesyn`）が指定されています。
- さらに、 `endp` までに 2 つのプリコンパイラ 指令（`comp5=yes` および `mode=ansi`）が指定されています。（この場合、プリコンパイラは Pro*COBOL です。）
- コンパイラ 指令は `omf(gnt)` と `list()` です。

11.2.2 COBSQL 指令の一覧

COBSQLTYPE CSQLT	使用するプリコンパイラ（ORACLE または SYBASE）を指定します。（指定例：COBSQLTYPE=ORACLE）
CSTART* CST	COBSQL が実行時にデータベース支援モジュールをロードするようにします。
CSTOP* CSP	COBSQL が、アプリケーションが異常終了したときロールバックを実行できる ストップ ラン モジュールをロードするようにします。
DEBUGFILE* DEB	プリプロセッサ デバッグ ファイル（.deb ファイル）を作成します。
DISPLAY* DIS	プリプロセッサ統計情報を表示します。 この指令は最初に COBSQL が正しくプリプロセッサを呼び出しているかを確認するときにだけ使用します。
END-COBSQL END-C END	COBSQL 指令の終わりを示します。残りの指令は SQL プリコンパイラに渡されます。
KEEPCBL	プリコンパイルされたソース ファイル（.cbl ファイル）を保存します。
MAKESYN	すべての COMP ホスト変数を COMP-5 ホスト変数に変換します。COBSQL の MAKESYN 指令 セットがない場合、デフォルト動作では、ホスト変数だけでなくすべての変数が COMP から COMP-5 に変換されます。
NOMAKESYN	COBSQL による COMP ホスト変数から COMP-5 ホスト変数への変換を停止します。
SQLDEBUG	Micro Focus 社が COBSQL をデバッグするためのいくつかのファイルを作成します。これらのファイルには、プリコンパイラからの出力（通常は .cbl 拡張子）、プリコンパイラが作成したリスト ファイル（通常は .lis 拡張子）、そして .sdb 拡張子を持つ COBSQL デバッグ ファイルがあります。さらに KEEPCBL と TRACE をオンにします。

TRACE*	トレース ファイル（.trc ファイル）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、ステータス更新を示します。 この指令は最初に COBSQL が正しくプリプロセッサを呼び出しているかを確認するときにだけ使用します。

11.2.3 COBOL 指令の一覧

BELL*	エラーが発生したときにベルを鳴らすかどうかを制御します。
BRIEF*	SQL エラー番号と共にエラー テキストを表示するかどうかを制御します。
CONFIRM*	受け付けられた指令、および拒否された指令を表示します。
LIST*	プリコンパイラ リスト ファイル（.lis ファイル）を保存します。
WARNING*	報告するSQLエラーの最低重大度を指定します。

11.3 COBSQL アプリケーションの作成

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支援モジュールは、このドキュメントでは一覧していません。以下の説明は、必要なすべての支援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、しかも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前提としています。

COBSQL アプリケーションでは、インポート ライブラリ（csqlsupp.lib）をリンクすることによって、COBSQL が追加した呼び出しを COBSQL の初期化および実行停止モジュールに変換します。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同モジュールを同梱する必要があります。この汎用支援モジュールは、Oracle と Sybase のどちらのデータベースを使用する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。このライブラリは NetExpress¥base¥lib ディレクトリに格納されています。csqlsupp.dll モジュールは NetExpress¥base¥bin ディレクトリにあります。

アプリケーションをメイン 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に csqlsupp.lib をリンクします。

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、本来のサイズよりやや大きくなります。なお、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。

メイン プログラムだけを CSTART と CSTOP でコンパイルした場合には、csqlsupp.lib のリンクも メイン プログラムのみに実行します。つまり、CSTART （または CSTOP） COBSQL 指令でコンパイルしたプログラムには、か

ならず csqlsupp.lib をリンクする必要があります。

11.4 Copybook Preprocessor によるコピーブックの展開

コピーブックを操作するために 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。いくつかのプリコンパイラはインクルードされたコピーブックを開いたり、展開できますが、コピーや置き換えを実行したり、panvalet コマンドを理解できるプリコンパイラはありません。この問題は、Micro Focus Copybook 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（例、COBSQL）にコピーブックを取り扱うためのメカニズム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 と同じルールでコピーブックを取り扱いますので、すべての COPY ステートメント関連の指令は自動的に取り入れられ、COBCPY 環境変数を探索します。また、CP は以下の SQL ステートメントも展開します。

```
EXEC SQL

   INCLUDE ...

END-EXEC
```

CP および CP で使用できる指令の詳細については、NetExpress のオンライン マニュアル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。）

コピーブックの拡張子は、Oracle では .pco と .cob、Sybase では .pco と .cbl です。CP によるコピーファイルと INCLUDE ステートメントの変換を正しく実行するには、次の COBOL コンパイラ 指令を使用します。

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

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変換されるため、データベース プリコンパイラでは実質的に、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 には、アニメーションの実行中にコピーファイルにアクセスできるというメリットもあります。

CP は `INCLUDE SQLCA` ステートメントを検出すると、次の処理を実行します。

- カレント 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、標準では .cbl または .cpy です。）
- COBCPY で指定されたディレクトリ内で sqlca.*ext* を検索します。
- たとえば、NetExpress のインストール ディレクトリの source サブディレクトリには、サンプルの SQLCA ファイル（sqlca.cpy）がインストールされており、使用するデータベースの SQLCA ファイルが COBCPY で

指定されたディレクトリに存在しない場合に使用されます。ただし、このサンプル ファイルをそのまま使用すると、作成したプログラムが正しく実行できない可能性があります。

CP の SY 指令を使用すれば、SQLCA インクルード ファイルの展開を無効化できます。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ます。したがって、同プリコンパイラ用のコードを処理する場合には、SY 指令で CP による SQLCA 展開を無効化してください。

Oracle のプリコンパイラは 2 種類の変数（COMP および COMP-5）のいずれかを使用してコードを生成するため、それぞれの変数に対応する 2 組のコピーファイルを持っています。標準のコピーファイル セット（sqlca.cob、oraca.cob、sqlda.cob）には COMP データ項目、もう 1 つのコピーファイル セット（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ければ、コピーファイルがカレント 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れば、コピーファイルの正しいレポートを生成できます。COBSQL のみを使用すると行番号が正しくカウントされず、検出されたエラーは表示されないか、または不正な行に表示されてしまいます。

11.5 各国語対応（NLS）

COBSQL のエラー メッセージ表示に使用する言語は LANG 環境変数で指定します。各国語対応の詳細については、ヘルプ ファイルの索引で「NLS」を選択してください。LANG 環境変数の設定については「LANG」を選択します。

ほとんどのデータベース クライアントは、ある程度までローカライズすることが可能ですが、LANG 環境変数の設定条件は COBOL システムの場合とは異なります。したがって、通常は LANG 変数の代わりに COBLANG 環境変数を使用してください。

COBLANG 変数の設定は COBOL システムのみに反映されるため、データベース クライアント側では LANG 変数を使用できます。なお、COBLANG 変数の設定を反映するためには、NetExpress の ¥bin サブディレクトリに mflang*nn*.lbr ファイル（*nn* は COBLANG の設定値）を作成しておく必要があります。たとえば、COBLANG を 05（イギリス英語）に設定する場合には、NetExpress¥Base¥Bin ディレクトリ（NetExpress は NetExpress のインストール ディレクトリ）に mflang05.lbr を作成します。

COBLANG の設定は COBSQL のエラー メッセージのみに反映されます。データベース プリコンパイラのエラー メッセージは COBSQL で処理されないため、COBLANG の設定も反映されません。

11.6 例

ここでは、COBSQL を使用してプログラムをコンパイルする際に NetExpress のコマンドラインに入力するコマンドの例を、Oracle のプリコンパイラを使用する場合と Sybase のプリコンパイラを使用する場合に分けて紹介しています。

11.6.1 Oracle プリコンパイラを使用する場合

```
cobol sample.pco anim nognt preprocess(cobsql)

   cstart cstop CSQLT=ORA end-c comp5=yes endp;
```

UNIX でのコマンド例

```
cob -a -v -k sample.pco -C "p(cobsql) cstop cobsqltype==ORACLE"
```

11.6.2 Sybase のプリコンパイラを使用する場合

```
cobol example1.pco confirm preprocess(cobsql)

   cstop csp cobsqltype=sybase preprocess(cp) sy endp;
```

UNIX でのコマンド例

```
cob -a -v -P -k example1.pco -C"p(cobsql) csp CSQLT==syb"
```

11.6.3 Informix のプリコンパイラを使用する場合

```
cob -a -k demo1.eco -C "p(cobsql) cobsqltype==informix-new"
```

11.7 トラブルシューティング

COBSQL の使用時に問題が発生した場合は、Micro Focus の技術サポートにご連絡ください。技術サポートによる原因究明を円滑化するため、COBSQL の SQLDEBUG 指令を使用してコンパイルを実行し、オリジナルのソース ファイル、データベースのプリコンパイラが生成した拡張ソース ファイル、トレース ファイル（.trc ファイル）、データベース リスト ファイル（.lis ファイル）、コマンドライン デバッグ ファイル（.sdb ファイル）、およびプロジェクト ファイル（プロジェクトと同じ名前で拡張子.app のファイル）を COBSQL 指令の設定データと共に .zip ファイルに圧縮して技術サポートに送付してください。

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。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。ほとんどの SQL 製品には、SQL ネットワークの設定を確認するための ping ユーティリティが付属しています。

- SQL による対話テスト

  SQL ネットワークに問題がなければ、SQL ステートメントによる対話テストを行います。SQL 製品には通常、SQL ステートメントをキー入力して結果を確認するためのシンプルなユーティリティと、SQL 対話テストに使用できるサンプル データベースが付属しています。

- スタンドアロン プリコンパイラ

  スタンドアロン プリコンパイラの動作をチェックします。アイコンまたはコマンドライン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。通常、SQL 製品には、プリコンパイラ付きのサンプル アプリケーションが付属しています。

- 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。NetExpress でテスト用プロジェクトを作成し、サンプル プログラムと同じディレクトリに SQLCA コピーファイルを格納した後、プリコンパイラを実行して動作をチェックします。

11.7.1 主なチェック ポイント

上記の各項目をチェックしても原因を特定できない場合には、さらに以下のポイントをチェックしてください。

- 製品のバージョン

  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 の混在

ベンダー提供のドキュメントとサンプル アプリケーションをチェックします。

- 環境設定

  PATH およびその他の環境変数の設定と設定ファイルの内容をチェックします。

- 指令

  通常、COBSQL はデータベース プリコンパイラに渡すコマンドラインを表示しません。コマンドラインを表示するには、SQLDEBUG 指令を設定します。コマンドラインの表示は、プリコンパイラでコマンドライン エラーが検出されたときに必要になります。コマンドライン 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 コマンドライン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と、COBSQL は次のエラー メッセージを表示します。

  `* 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と、COBSQL は次のエラー メッセージを表示します。プリコンパイラで重大なエラーが発生したため、出力ファイルを生成できなかった可能性があります。

  `* CSQL-E-024: ファイル `*`filename`*` の I/O が検出されました`

  `* CSQL-E-023: ファイル ステータス  3 / 5`

  filename は、データベース プリコンパイラが生成すべき出力ファイルの名前を示します。

- 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るにも関わらず、COBSQL が展開後のソース ファイルのデータ不足を知らせるエラー メッセージ（展開されたソースの早期終了）を表示する場合には、データベース プリコンパイラが生成したソース ファイルとオリジナルのソース 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。

  上記のメッセージは、プログラムに SQL ステートメントが含まれていない場合にも表示されます。ほとんどのデータベース プリコンパイラは、SQL ステートメントが見つからないと出力ファイルの生成を中断するため、オリジナルのソース ファイルを部分的に反映した出力ファイルが生成されるからです。

11.7.2 Oracle プリコンパイラを使用する際の留意点

- 次に挙げる Oracle プリコンパイラのオプションは、作成する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。したがって、これらの指令の設定を変更する場合には、事前に Oracle のドキュメントに十分に目を通してください。

  DBMS

  HOLD_CURSOR

  MAXOPENCURSORS

  MODE

  RELEASE_CURSOR

- Oracle データベースにアクセスするプログラムでは、ALTER SESSION コマンドによって OPTIMIZER_GOAL の値を変更することが可能です。ただし、OPTIMIZER_GOAL の値を変更する際には、特定 SQL ステートメントの処理効率よりもアプリケーションの全般的なパフォーマンスを優先してください。前者の最適化には HINTS も使用できます。ALTER SESSION、OPTIMIZER_GOAL、および HINTS の詳細については、Oracleのドキュメントをご覧ください。

- Oracle では、埋め込み SQL ステートメント内に配列を使用してネットワークのアクセス負荷を抑制することが可能です。この方法を使用すれば、アプリケーションはバッチ SQL コマンド（単一の SQL ステートメントによる複数行のデータ処理）を実行できます。詳細については、*ホスト変数*の章の「*ホスト配列*」をご覧ください。

  アプリケーションは通常、1 つの SQL ステートメントで 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 sample3.pco というサンプル プログラムが付属しています。（ファイル名が異なる場合もあります。） 配列の詳細については、「*ORACLE プリコンパイラ・ガイド Pro*Cobol サプリメント*」の説明をご覧ください。

- Pro*Cobol による BINARY ホスト変数の扱いは、COMP の PICTURE で定義された変数と同様です。したがって、comp5=yes 指令を使用すれば、BINARY 変数を COMP-5 に変換できます。

- Pro*Cobol では、SQLCA コピーファイルがカレント ディレクトリに存在する場合を除き、かならず INCLUDE 指令で同ファイルをインクルードする必要があります。SQLCA ファイルがカレント ディレクトリに存在せず、しかも INCLUDE 指令も指定されていない場合には、*filename*.pco ファイルにエラー メッセージが出力されます。（*filename* は半角 8 文字の文字列です。）

- Pro*Cobol の実行時には、さまざまな有益な情報が生成されます。COBSQL VERBOSE 指令を使用すると、これらの情報の一部を COBSQL の実行時に表示できます。

Pro*Cobol が生成する情報をできるだけ多く取り込むには、xref=yes 指令を使用します。この指令は Pro*Cobol の設定ファイル（pcccob.cfg）に記述できます。

- COBSQL DISPLAY 指令を使用すれば、処理中の Pro*Cobol 指令を画面に表示できます。（指令名に続いて Pro*Cobol で生成された統計情報が表示されます。）
- COBOL LIST 指令を使用すると、COBSQL が収集した Pro*Cobol の情報はすべて COBOL チェッカーに渡され、COBOL リストの最下部に表示されます。

索引