DBMaster

ESQL/C プログラマー参照編

SYSCOM

CASEMaker Inc./Corporate Headquarters

1680 Civic Center Drive
Santa Clara, CA 95050, U.S.A.

Contact Information:
CASEMaker US Division
E-mail : info@casemaker.com
Europe Division
E-mail : casemaker.europe@casemaker.com
Asia Division
E-mail : casemaker.asia@casemaker.com(Taiwan)
E-mail : info@casemaker.co.jp(Japan)

www.casemaker.com
www.casemaker.com/support

©Copyright 1995-2015 by Syscom Computer Engineering Co.
Document No. 645049-236302/DBM54J-M09302015-ESQL
発行日:2015-09-30

ALL RIGHTS RESERVED.
本書の一部または全部を無断で、再出版、情報検索システムへ保存、その他の形式へ転作することは禁止されています。

本文には記されていない新しい機能についての説明は、CASEMakerのDBMasterをインストールしてからREADME.TXTを読んでください。

登録商標

CASEMaker、CASEMakerのロゴは、CASEMaker社の商標または登録商標です。
DBMasterは、Syscom Computer Engineering社の商標または登録商標です。
Microsoft、MS-DOS、Windows、Windows NTは、Microsoft社の商標または登録商標です。
UNIXは、The Open Groupの商標または登録商標です。
ANSIは、American National Standards Institute, Incの商標または登録商標です。

ここで使用されているその他の製品名は、その所有者の商標または登録商標で、情報として記述しているだけです。SQLは、工業用語であって、いかなる企業、企業集団、組織、組織集団の所有物でもありません。

注意事項

本書で記述されるソフトウェアは、ソフトウェアと共に提供される使用許諾書に基づきます。

保証については、ご利用の販売店にお問い合わせ下さい。販売店は、特定用途への本コンピュータ製品の商品性や適合性について、代表または保証しません。販売店は、突然の衝撃、過度の熱、冷気、湿度等の外的な要因による本コンピュータ製品へ生じたいかなる損害に対しても責任を負いません。不正な電圧や不適合なハードウェアやソフトウェアによってもたらされた損失や損害も同様です。

本書の記載情報は、その内容について十分精査していますが、その誤りについて責任を負うものではありません。本書は、事前の通知無く変更することがあります。

目次

1 はじめに 1-1
1.1 その他のマニュアル 1-3
1.2 字体の規則 1-4
2 ESQLプリプロセス 2-1
2.1 dmppccを使う 2-3
Singleton Select オプション 2-4
SQLCHECK 2-4
必須のプリコンパイル・パラメータ 2-6
3 ESQL構文 3-1
3.1 静的/動的な構文 3-2
3.2 変数 3-4
宣言セクション 3-4
ホスト変数のデータ型 3-5
ホスト変数 3-11
変数スコープ 3-12
インジケータ変数 3-14

©Copyright 1995-2015 Syscom Computer Engineering Co.

3.3 ステータス・コード 3-15
Include変数 3-15
3.4 WHENEVER文 3-19
4 データ操作 4-1
4.1 データ操作 4-2
4.2 単一行データを回収する 4-3
4.3 トランザクション処理 4-5
4.4 動的接続の構文 4-5
4.5 カーソルを使う 4-6
複数の行を回収する 4-6
カーソルを宣言する 4-7
カーソルを開く 4-8
データ回収のためにカーソルを使う 4-8
コマンド・パラメータ 4-9
カーソルでデータを削除する 4-12
カーソルでデータを更新する 4-13
カーソルを閉じる 4-13
5 BLOBデータ 5-1
5.1 PUT BLOB文 5-2
5.2 GET BLOB文 5-6
カーソルを使った複数行データ 5-6
5.3 PUT BLOBとGET BLOBの違い 5-12
6 動的ESQL 6-1
6.1 タイプ1の動的ESQL 6-2
6.2 タイプ2の動的ESQL 6-3
6.3 タイプ3の動的ESQL 6-4

©Copyright 1995-2015 Syscom Computer Engineering Co.

6.4 タイプ4の動的ESQL 6-5
SQLDAディスクリプタ 6-6
Describeコマンド 6-6
SQLDAを経由して情報を渡す 6-6
アプリケーションのステップ 6-15
6.5 動的ESQL BLOBインターフェース 6-37
ファイル・オブジェクトを保存する 6-37
Fileオブジェクトを取得する 6-39
BLOBデータを配置する...6-41
BLOBデータを取得する...6-43
7 プロジェクトとモジュール管理.........................7-1
7.1 プロジェクトとモジュール・オブジェクト............. 7-3
モジュールについて 7-3
プロジェクトについて 7-3
プロジェクトを削除する 7-5
プロジェクト/モジュールをロード/アンロードする 7-6
プロジェクトのための権限を与える/取り消す 7-7

©Copyright 1995-2015 Syscom Computer Engineering Co.

©Copyright 1995-2015 Syscom Computer Engineering Co.

1　はじめに

ESQL/Cプログラマー参照編にようこそ。DBMasterは、強力かつ柔軟なSQLデータベース管理システム(DBMS)です。会話型の構造的問合せ言語(SQL)、Microsoftのオープンデータベース結合(ODBC) 互換インタフェース、およびC 言語のための組込みSQL(ESQL/C)をサポートします。唯一の公開アーキテクチャーであるODBCインタフェースは、多種多様なプログラミングツールを使用して顧客アプリケーションを構築し、既存のODBC-適合アプリケーションを用いてデータベースに問合せることを可能にします。

DBMasterは、シングルユーザーの個人データベースから、企業全体に分散するデータベースまでに容易にスケール化することができます。どのようなデータベース構成を選択しても、重要データの安全性は、DBMasterのセキュリティ、整合性、信頼性の先進的機能によって確実に保証されます。広範なクロス-プラットフォームのサポートは、現在あるハードウェアの効力を高め、需要の増大に応じてより強力なハードウェアに拡大し、グレードアップすることを可能にします。

DBMasterは、優れたマルチメディア処理機能を提供し、あらゆるタイプのマルチメディアデータを格納、探索、検索、操作を可能にします。バイナリラージオブジェクト(BLOB)は、DBMaster の先進的セキュリティと損傷リカバリ機構を全面的に利用して、マルチメディアデータの整合性を確実にします。ファイルオブジェクト(FO)は、マルチメディアデータを管理する一方で、元のアプリケーションで各ファイルを編集する機能を維持します。

©Copyright 1995-2015 Syscom Computer Engineering Co.

本書 は、DBMasterの設計者とデータベース管理者向けに書かれており、データベース管理を通じて、ESQL\C の基本操作を体系的に学習することができます。また、リレーショナル・データベースの業務についての理解はあるものの、DBMasterには慣れていないユーザーの方にもご利用頂けますが、Windows やUNIX 環境のオペレーティング・システムについての一定の知識は必要です。経験豊富なユーザーの方は参照編として利用できます。

本書では、ESQL\Cでデータベースを管理する上での様々な命令文やプロシージャを紹介してします。本書は、Windows NT と Windows 98 環境用のDBMaster対象となっていますが、UNIX フラットフォームでも同様の関数を実行できます。より具体的な説明のために、本書を通してサンプルデータベースを用いています。

SQL は、2重モードの言語です。つまり、一般にインタラクティブSQLとして理解されているデータベースの通信とアクセスのためのインタラクティブなツールと、データベース・アクセスのためにアプリケーション・プログラムが使用するデータベースのプログラミング言語です。

1つ目のモードとして、一般的に主要なDBMSには全てSQLを使った独自のユーザー・インターフェースが備わっています。例えば、DBMasterにはdmSQL/Cがあります。ユーザーは、データベースにアクセスして保守するために、ツールを通じてSQL構文を直接入力することができます。

2番目のモードとしては、多くのメジャーなDBMSは、独自のプログラムにユーザーがSQLを使うための２つの基本技術もを備えています。それは、データベースAPIとESQL(埋め込みSQL)です。

©Copyright 1995-2015 Syscom Computer Engineering Co.

1.1 その他のマニュアル

DBMasterには、本マニュアル以外にも多くのユーザーガイドや参照編があります。特定のテーマについての詳細は、以下の書籍をご覧下さい。

- DBMasterの設計、管理、保守についての詳細は、*「データベース管理者参照編」*をご覧下さい。
- DBMasterの能力と機能性についての概要は、*「DBMaster入門編」*をご覧下さい。
- DBMasterの管理についての詳細は、*「JServer Managerユーザーガイド」*をご覧下さい。
- DBMasterの環境設定についての詳細は、*「JConfiguration Tool参照編」*をご覧下さい。
- DBMasterの機能についての詳細は、*「JDBA Toolユーザーガイド」*をご覧下さい。
- DBMasterで使用しているdmSQLのインターフェースについての詳細は、*「dmSQLユーザーガイド」*をご覧下さい。
- DBMasterで採用しているSQL言語についての詳細は、*「SQL文と関数参照編」*をご覧下さい。
- ODBCとJDBCプログラムについての詳細は、*「ODBCプログラマー参照編」*と*「JDBCプログラマー参照編」*をご覧下さい。
- エラーと警告メッセージについての詳細は、*「エラー・メッセージ参照編」*をご覧下さい。
- ネイティブDCI APIについての詳細は、*「DCI ユーザーガイド」*をご覧下さい。

1.2 字体の規則

本書は、標準の字体規則を使用しているので、簡単かつ明確に読むことができます。

斜体	斜体は、ユーザー名や表名のような特定の情報を表します。斜体の文字そのものを入力せず、実際に使用する名前をそこに置き換えてください。斜体は、新しく登場した用語や文字を強調する場合にも使用します。
太字	太字は、ファイル名、データベース名、表名、カラム名、関数名やその他同様なケースに使用します。操作の手順においてメニューのコマンドを強調する場合にも、使用します。
キーワード	文中で使用するSQL言語のキーワードは、すべて英大文字で表現します。
小さい 英大文字	小さい英大文字は、キーボードのキーを示します。 2つのキー間のプラス記号 (+) は、最初のキーを押したまま次のキーを押すことを示します。キーの間のコンマ(,)は、最初のキーを放してから次のキーを押すことを示します。
注	重要な情報を意味します。
➲ プロシージャ	一連の手順や連続的な事項を表します。ほとんどの作業は、この書式で解説されます。ユーザーが行う論理的な処理の順序です。
➲ 例	解説をよりわかりやすくするために与えられる例です。一般的に画面に表示されるテキストと共に表示されます。
コマンドライン	画面に表示されるテキストを意味します。この書式は、一般的にdmSQLコマンドやdmconfig.iniファイルの内容の入/出力を表示します。

©Copyright 1995-2015 Syscom Computer Engineering Co.

2 ESQLプリプロセス

この章では、ESQL/C プリプロセッサを使ったESQLとCの混合ソース・プログラムのコンパイルについて説明します。DBMasterのプリプロセッサ・コマンドは、*dmppcc*です。入力ESQL/Cファイルには、“.ec”が付いており、*dmppcc*出力は、C言語ファイルです。プリコンパイルの際、*dmppcc*はストアド・コマンドと各ESQL DML文を作成し、元のソースのESQL文をストアド・コマンドを呼び出す関数コールに変換します。

埋め込みSQL (ESQL)は、その他の手法を使っています。SQL文の形態を少し変更した場合、ホスト・プログラミング言語のアプリケーション・プログラムのソース・コードに直接書き込むことができます。この混合ソース・コードは、その後SQLにプリコンパイルされ、データベースに保管され、ホスト言語関数コールがストアド・コマンドを実行するために生成されます。

DBMSにアクセスするためにESQLコマンドを使うCアプリケーション・プログラムを書くことができます。DBMaster ESQL/Cプリプロセッサには、Cコンパイラのための SQLコマンドを含むアプリケーション・プログラムがあります。プリプロセッサは、データベース操作を実行するために、それからSQLコマンドをCコメントの付いたC文に変換します。

➲ ESQL/Cアプリケーション・プログラムを作成、実行する：

1. 埋め込みSQLでプログラムを設計、記述します。
2. DBMasterのESQL/Cプリプロセッサ*dmppcc*を使ってプログラムをプリプロセスします。

3. プリプロセッサで生成されたプログラムをコンパイル、リンクします。
4. プログラムを実行します。

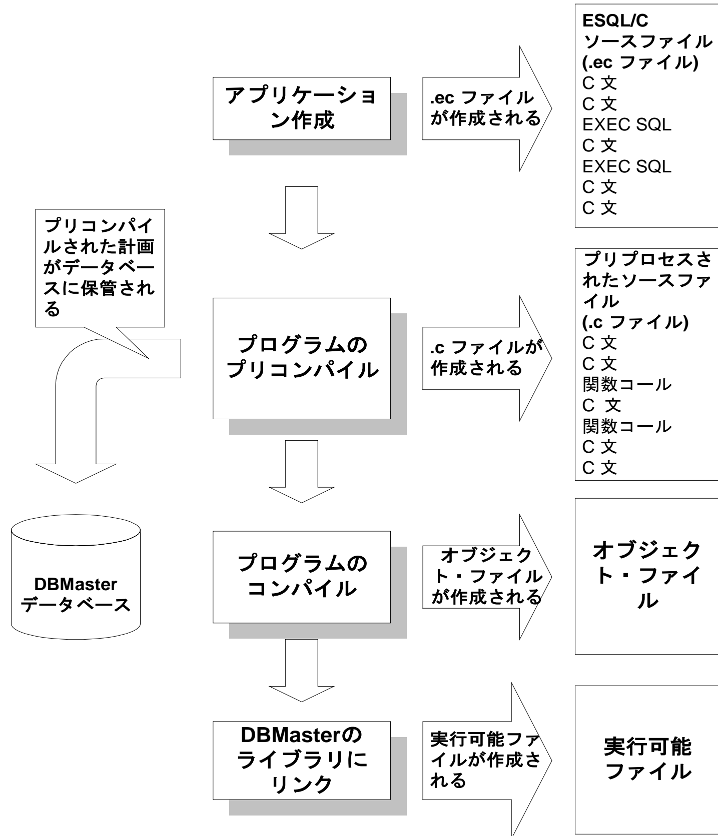

表 2-1 ESQLプログラムのフローチャート

©Copyright 1995-2015 Syscom Computer Engineering Co.

2.1 dmppccを使う

dmppcc コマンドで以下のオプションを使うことができます。

オプション	コメント
-d データベース名	必須
-u ユーザー名	必須
-p パスワード	必須
-o 出力ファイル	出力ファイル名。初期設定は、入力ファイル.c。
-l ログファイル名	Dmppccエラー・メッセージ・ログ。初期設定はstderr。
-j プロジェクト名	指定しない場合、モジュール名をプロジェクト名として利用します。
-m モジュール名	指定しない場合、入力ファイル名をモジュール名として利用します。
-s	singleton selectエラーをチェックするをonにセットします。指定しない場合の、初期設定はoffです。
-cl	sql check level = limitをセットします。初期設定はFULL。ユーザーはSQL構文で非存在表を参照することができます。
-cs	sql check level = syntaxをセットします。初期設定はFULL。dmppccは、表又はカラム情報をチェックしないで、SQL構文のみチェックします。
-n	Dmppccは、データベース・サーバーでSQLコマンドと実行計画を保管しません。
-v	バージョンを表示します。
-h	ヘルプ情報を表示します。
-sp	ストアド・プロシージャ・ソースをコンパイルします。

©Copyright 1995-2015 Syscom Computer Engineering Co.

Singleton Select オプション

➲ 例 1

Singleton Select オプションを使う：

```
dmppcc -s ex1.ec
```

➲ 例 2

Singleton SelectオプションをONにし、プリプロセスの際にランタイムのチェックを確認する：

```
EXEC SQL SELECT salary FROM emp_table WHERE emp_id = 1000 INTO :var_salary;
```

singleton selectのエラー・チェックがONになっている時に、結果タプルの数が複数の場合、エラーが戻されます。

➲ 例 3

戻されたエラー：

```
singleton SELECT can only retrieve at most one row
```

このチェックがONにされていない場合、singleton select問合せは、最初のタプル・カラムのみをホスト・パラメータに回収し、更に結果タプルをチェックしません。

SQLCHECK

SQLCHECKレベル・オプション構文：

FULL(初期設定)/LIMIT(-cl)/SYNTAX(-cs)

LIMIT (–CLにセット)

SQLCHECKオプションを、LIMIT(–clにセット)にセットしている時、ESQLプリプロセッサは、EXEC SQL 文に表が存在しない場合、エラーを戻す事無く、ESQL/C プログラムをプリプロセスし続けることが可能になります。

©Copyright 1995-2015 Syscom Computer Engineering Co.

SYNTAX (–csにセット)

SQLCHECKオプションをSYNTAX(–csにセット)にセットしている時、ESQLプリプロセッサは、ユーザーが正しいSQL 構文のEXEC SQL文を与えた場合、エラーを戻す事無く、ESQL/Cプログラムをプリプロセスし続けることが可能になります。

これは、表、カラム、セキュリティのためにSQL文で関連するセマンティック情報をチェックしません。このオプションをONにしている場合、dmppccはデータベースに接続しませんが、DBMasterのdmconfig.iniファイルのデータベース関連情報をチェックするために、ユーザーはdmppccのためにデータベース名を用意する必要があります。

(-N)

ストアド・コマンド・オプションを作成しない： (-n)

このオプションをONにしている場合、dmppccはデータベースにプリコンパイルしたプランを保管しません。但し、Dmppccは構文エラーをチェックします。このオプションは主に、マルチユーザー環境のためのものです。マルチユーザー環境では、複数の人間が同じ実行可能ファイルへのリンクがある別々のファイルをプリプロセスして、更にまだリンクされていない再プリプロセスされた関数で実行可能ファイルをテストしています。ユーザーは、「実行可能ファイルは、期限切れです。再ビルドして下さい。」というエラーを受け取るかもしれません。

このエラー・メッセージを受け取るような場合、また他の人がそのファイルをプリプロセスしていることが確かな場合、又は「ロック・タイムアウト」のエラー・メッセージを受け取る場合は、このオプションを使って下さい。

©Copyright 1995-2015 Syscom Computer Engineering Co.

必須のプリコンパイル・パラメータ

➲ 例 1

ESQL/Cプログラムをプリコンパイルする際に、必ずセットする必要があるオプションは以下のとおりです。

```
dmppcc -d test_db -u db_user_id -p db_user_passwd esql_source.ec
```

test_db=はデータベース名、db_user_idはユーザー名、db_user_passwdはユーザーパスワードです。これらのパラメータは、必ずコマンド・ラインに記述しなければなりません。記述しない場合は、*dmppcc*はローカル*dmppc.ini*ファイルを探して、開こうとします。*dmppc.ini*ファイルに、パラメータを設定することもできます。

➲ 例 2

dmppcc.iniファイルの構文

```
DATABASE = database_name
USER = user_id
PASSWORD = password
SELECT_ERROR = yes/no
OUTFILE = output_filename
LOGFILE = log_filename
PROJECT = project_name
MODULE  = module_name
SQLCHECK = FULL/LIMIT/SYNTAX
```

*dmppcc*で作成された".c" ファイルは、普通のCソースコード・ファイルです。最終的な実行可能ファイルを作成するために、他の".c"や".o"ファイルと結合することができます。

➲ 例 3

プリ・ソース・プログラムex1.ecを使い、ユーザー "john"とパスワード "johnspwd"でデータベース"TESTDB"にアクセスする：

```
dmppcc -d TESTDB -u john -p johnspwd example.ec
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

➲ 例 4

出力ファイルex1.cをコンパイルし、実行可能ファイルとして、ESQLライブラリと他のDBMaster ライブラリとリンクするために、Cコンパイラを使う：

```
cc -I. -I$INCDIR -c example.c
```

ファイルexample.cが*dmppcc* コンパイラでプリプロセスされ、example.oにコンパイルされた場合、実行可能ファイルとして他のオブジェクト・ファイルとそれをリンクすることができます。

➲ 例 5

実行可能ファイルとして、–oを他のオブジェクト・ファイルにリンクする：

```
acc -o driver example.o otherap.o  -L$LIBDIR -ldmapic -lm
```

DBMasterのサンプル・ディレクトリにあるMakefileを参照することもができます。

~ DBMaster /$VERSION/samples/ESQLC。

©Copyright 1995-2015 Syscom Computer Engineering Co.

3 ESQL構文

ESQL/Cプリプロセッサは、ESQLソース・プログラムの中の、頭に“EXEC SQL”や“$”が付いた全ての文をプリプロセスします。SQL 文は、Cアプリケーション・プログラムのどこにでも置くことができます。但し、ESQLプリプロセッサは、マクロ定義にあるEXEC SQL文を扱うことができません。又、ESQL/C プリプロセッサは、ヘッダー・ファイルのEXEC SQL文をプリプロセスしません。SQL文は、必ず”EXEC SQL”か“$”の後ろにおき、これら両キーワードは、同じ行に記述し、終わりにセミコロン(;)を付けます。

➲ 例

```
if (c1 > 0) EXEC SQL COMMIT WORK;
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

3.1 静的/動的な構文

プリプロセス時にSQL文を見つけると、ESQLプログラムは静的なESQL構文を使います。削除、挿入、更新、選択操作を実行する時、そして参照される表や検索条件がわかっている場合、入力パラメータ値のみが実行時に変わるかもしれません。DBMasterは、このような構文のためにセキュリティをチェックし、実行計画にコンパイルし、データベースにその計画を保存します。

SQL文が記述とプリプロセス時にわからない場合、それは動的ESQLとなります。

注 *データベースで保管されている情報の管理方法についての詳細は、7章の"プロジェクトとモジュール管理"を参照して下さい。*

プリプロセス時に完全/部分的なSQL文がわからない場合、アプリケーションは動的ESQL構文を使います。そのSQL文は、ユーザーによって与えられ(例 dmsqlc)、SQL文全体は問合せツール(QBE)にて構成されています。DBMasterは、動的ESQL構文については、プリプロセス時にSQL構文をコンパイルすることはできません。そのため、全てのコンパイルとセキュリティ・チェックは、ランタイムに実行されます。

注 *動的ESQL構文の詳細は、6章の"動的ESQL"を参照して下さい。*

➲ **例1**

動的ESQL/Cプログラムのフォーマット

```
EXEC SQL INCLUDE DBENVCA;
EXEC SQL INCLUDE SQLCA;
main ( )
    {
     EXEC SQL BEGIN DECLARE SECTION;
     char sql_string[255];
     EXEC SQL END DECLARE SECTION;
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
    EXEC SQL CONNECT TO testdb john johnspwd;


    /* get user input for SQL statement                               */
    user_input(sql_string);
    /* Dynamic ESQL statement without host variable                   */
    EXEC SQL PREPARE statement_name FROM :sql_string;
    EXEC SQL EXECUTE statement_name;
    EXEC SQL DISCONNECT;
   }
```

➲ **例 2**

静的ESQL構文

```
EXEC SQL CONNECT TO database_name user_name password
EXEC SQL DISCONNECT [database_name]
EXEC SQL INCLUDE {DBENVCA | SQLCA | SQLDA}
EXEC SQL WHENEVER {SQLERROR | SQLWARNING | NOT FOUND}
                    {CONTINUE| STOP | GO TO label| GOTO label | DO action}
EXEC SQL BEGIN DECLARE SECTION, EXEC SQL END DECLARE SECTION
EXEC SQL [AT DATABASE_NAME] any SQL statement
EXEC SQL [AT DATABASE_NAME] DECLARE cursor_name CURSOR FOR
sql_query_statement
EXEC SQL [AT DATABASE_NAME] OPEN cursor_name [USING
host_variable_[indicator]_list]
EXEC SQL [AT  DATABASE_NAME] FETCH cursor_name [INTO
host_variable_[indicator]_list]
EXEC SQL [AT DATABASE_NAME] CLOSE cursor_name
```

➲ **例 3**

静的ESQL/Cプログラムのフォーマット：

```
EXEC SQL INCLUDE DBENVCA;
EXEC SQL INCLUDE SQLCA;
main ( )
   {
```

```
    EXEC SQL BEGIN DECLARE SECTION;
    int  emp_id;
    char emp_name[20], emp_addr[50];
    EXEC SQL END DECLARE SECTION;
    EXEC SQL CONNECT TO testdb john johnspwd;
    /* get user input for host var                                  */
    user_input(&emp_id, emp_name, emp_addr);
    EXEC SQL INSERT INTO emp_table VALUES (:emp_id, :emp_name, :emp_addr);
    EXEC SQL DISCONNECT;
   }
```

注 *ホスト変数を一般のC文で参照する時、その方法は他のC変数と同じです。ホスト変数をEXEC SQL文で参照する時、頭にコロン(:)を付けます。*

3.2 変数

CアプリケーションとDBMasterデータベース間でデータをやり取りするために、ESQL文のホスト・プログラムの*ホスト変数*と呼ばれる変数を使うことができます。ホスト変数は、常に*インジケータ変数*と呼ばれるもう1つの変数と共に使用します。ホスト変数に値が代入されている時、そのインジケータ変数は、それがNULLであるか又は切り捨てられたかどうか、その値の特性を登録します。

宣言セクション

EXEC SQL文で参照されるどのホスト変数とインジケータ変数も、宣言セクションで宣言する必要があります。宣言セクションは、EXEC SQL文のBEGIN DECLARE SECTIONとEND DECLARE SECTION内に含まれるC変数宣言で構成されています。

➲ 例

BEGIN DECLARE SECTION：

```
EXEC SQL BEGIN DECLARE SECTION;
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
varchar hoEmpNo[8];                               /* A host variable      */
int     inEmpNo;                                   /* An indicator variable
*/
EXEC SQL END DECLARE SECTION;
```

アプリケーション・プログラムにある宣言セクションの数に制限はありません。EXEC SQL 文で参照されるホスト変数とインジケータ変数がある場合、どの関数も独自の宣言セクションがあります。ホスト変数とインジケータ変数が宣言セクションで見つからない場合、DBMasterの*dmppcc*はエラーを戻します。

ホスト変数のデータ型

singleton C変数は、ホスト変数で宣言することができます。Charバッファ長を定義するために使用する1次元文字配列を除いて、C構造体とユニオンは、ホスト変数で認められていません。1次元配列を定義するためにfileobjデータ型を使うことはできません。その他のC配列は、1つのFETCH文で複数のデータ値で行セットにフェッチするために、1次元文字配列として宣言することができます。ユーザーは、CHAR又はBINARYデータ型の場合、1つのFETCH文で複数のデータ値を回収するために、2次元配列を使うことができます。

使用することができる全ESQLデータ型は、esqltype.hで定義することができます。

ESQL/Cのタイプ	タイプの定義	定義
char var_name[n]	char[n]	固定長 char 入力 (n) char 出力 (n-1) char+ NULL終端
binary var_name[n]	char[n]	固定長バイナリ 入力 (n) char 出力 (n) char
short	short	Short integer

©Copyright 1995-2015 Syscom Computer Engineering Co.

ESQL/Cのタイプ	タイプの定義	定義
int	int	integer
long	long	long integer
float	float	Float
double	double	double

下記のSQL データ型は、宣言セクションでも使用します。

ESQL/Cタイプ	タイプ定義	定義
date	typedef struct date_s { short year; unsigned short month; unsigned short day;} eq_date;	Date
time	typedef struct time_s { unsigned short hour; unsigned short minute; unsigned short second; } eq_time;	Time
timestamp	typedef struct timestamp_s { short year; unsigned short month; unsigned short day; unsigned short hour; unsigned short minute; unsigned short second; unsigned long fraction;} eq_timestamp;	Timestamp
varchar var_name[n]	typedef struct varchar_s {long len; char arr[n]; } varchar;	可変長char 入力： lenを指定しない時、Null終端文字列 出力：(n-1) char +Null終端
varcptr var_name[n]	typedef struct varcptr_s {long len; char *arr; } varcptr;	可変長char、len値の割り当てが必須 入力/出力:Varcharと同じ

©Copyright 1995-2015 Syscom Computer Engineering Co.

ESQL/Cタイプ	タイプ定義	定義
varbinary var_name[n]	typedef struct varbinary_s {long len;char arr[n];} varbinary;	可変長binary、len値の割り当てが必須 入力/出力：binaryと同じ
varbptr	typedef struct varbpr_s { long len; /* must assign a buffer length */ char *arr; /* must assign a valid buffer ptr address */ } varbptr;	可変長バイナリ 入力/出力：varbinaryと同じ

BLOBデータ型	タイプ定義	定義
Longvarchar	typedef struct longvarchar_s { long bufsize; char *buf; } longvarchar;	BLOB (テキストデータ)
Longvarbinary	typedef struct longvarbinary_s { long bufsize; char *buf; } longvarbinary;	BLOB (バイナリデータ)
fileobj	typedef struct fileobj_s { long type; char fname[MAX_FNAME_LEN]; } fileobj;	BLOB (ファイル・オブジェクト) 初期設定データ型はESQL_STORE_FILE_CONTENT。サーバ

©Copyright 1995-2015 Syscom Computer Engineering Co.

BLOBデータ型	タイプ定義	定義
		ーにファイル名だけを保存するために、ESQL_STORE_FILE_NAMEとして型をセットすることも可能

varcharデータ型は、Null終端の可変長文字列で、**varbinary**はNull終端でない可変長バイナリ文字列です。**varchar**や**varbinary**データ型のlenフィールドに実際の長さを割り当てることで、入力や出力変数の長さを調節します。**varcptr**や**varbptr**データ型では、バッファ長とバッファ・アドレスは定義されません。プログラムでそれらを使用する前に、割り当てて下さい。

*dmppcc*は、これら非標準Cデータ型をCコンパイラで認識できるC構造体に変換します。例えば、varcharはフィールド長と文字配列要素のあるC構造体に変換されます。

FILEOBJデータ型

ファイル・データ型がセットされていない場合、その初期設定のデータ型はESQL_STORE_FILE_CONTENTです。つまり、データベースにユーザーが指定したファイルの内容そのものを保存することを意味します。そのファイルをデータベースに挿入した後に削除しても問題ありません。

但し、ファイル・データ型= ESQL_STORE_FILE_NAMEとセットした場合、データベースはfnameフィールド欄で指定したファイル名のみを保存します。ユーザーは、そのファイルがDBMasterのデータベース・サーバーからアクセス可能であるようにする必要があります。そのファイルを削除した場合、DBMasterはユーザーがそれを参照するときエラーを戻します。ファイル・データ型の設定は、fileobjデータ型が入力パラメータとして使用される時のみ有効です。出力パラメータとして、データベースは常に指定したファイル名にデータをコピーしようとします。

©Copyright 1995-2015 Syscom Computer Engineering Co.

➲ 例 1

入力ホスト変数としてのFileobjデータ型：

```
EXEC SQL BEGIN DECLARE SECTION;
fileobj fname1;
EXEC SQL END DECLARE SECTION;
EXEC SQL CREATE TABLE t1 (c1 file);
strcpy(fname1.name , "u:\image_path\test1.gif");
/* This INSERT statement will store all the content of file test.gif into
database */
EXEC SQL INSERT INTO t1 VALUES (:fname1);
/* If you  want to save database storage space, and if the file is located
where DBMaster server can access network directories, you should assign the
type as ESQL_STORE_FILE_NAME. You can switch it back to store file content
by assigning the type to ESQL_STORE_FILE_CONTENT */
fname1.type = ESQL_STORE_FILE_NAME;
strcpy(fname1.name , "u:\image_path\test2.gif");
EXEC SQL INSERT INTO t1 VALUES (:fname1);
```

➲ 例 2

表t2のc1 long varcharのスキーマで出力変数とするFileobjデータ型：

```
EXEC SQL BEGIN DECLARE SECTION;
fileobj fname1;
EXEC SQL END DECLARE SECTION;
strcpy(fname1.name , "u:\image_path\test1.gif");
/* This SELECT statement will fetch the blob data from server site and put
into user's local file.   */
strcpy(fname1.name , "u:\local_path\test1.gif");
EXEC SQL select c1 from t2 into :fname1;
```

➲ 例 3

カーソルを使った出力変数としてのFileobj：

```
EXEC SQL BEGIN DECLARE SECTION;
fileobj fname1;
```

```
EXEC SQL END DECLARE SECTION;

int idx1=0;

strcpy(fname1.name , "u:\image_path\test1.gif");

EXEC SQL DECLARE myCur1 CURSOR FOR select c1 from t2 into :fname1;

While (1)

{

idx1++;

/* This FETCH statement will fetch the blob data from server site and store
it into user's local file.  If you do not change output file name, in the
next FETCH statement, the output file  will be overwritten. */

sprintf(fname1.name , "test%d.gif", idx1);

EXEC SQL FETCH myCur1;

if (SQLCODE)

  { /* Break while loop when no more data or there's error */

  if (SQLCODE != SQL_SUCCESS_WITH_INFO)

        break;

  }

}
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

ホスト変数

一旦ホスト変数が作成されると、その変数にDBMasterによってデータが格納され、アプリケーション・プログラムで使用されます。ホスト変数は、データの挿入や更新、又はWHEREやHAVING句でも使用することができます。

Cアプリケーション・プログラムでホスト変数を使う時、以下を適用します。

- C言語の標準構文とDBMasterのESQLがサポートするデータ型に従ってホスト変数を宣言します。
- *INTO*句のホスト変数は、*SELECT*文で示されたカラムの数と同じかそれ以下です。そして入力/出力ホスト変数のデータ型は、パラメータ又はプロジェクション・カラムに対応する互換データ型です。
- カラム値のタイプに互換するホスト変数を使用します。
- SQL文でホスト変数を使う場合、コロン(:)を前につけます。それ以外で使用するときは、コロンは必要ありません。

➲ 例 1

```
EXEC SQL BEGIN DECLARE SECTION;
char emp_id[20];                                /* employee ID          */
char emp_tel[20];                               /* employee telephone #  */
int indvalue = SQL_NTS;
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT emp_id, emp_tel FROM emp_tab INTO :emp_id :indvalue,
:emp_tel :indvalue;
```

➲ 例 2

```
EXEC SQL BEGIN DECLARE SECTION;
char emp_id[20];                                /* employee ID          */
char emp_tel[20];                               /* employee telephone #  */
int indvalue = SQL_NTS;  /* indicates input parameter is null terminated
*/
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
EXEC SQL END DECLARE SECTION;
strcpy(emp_id, "john Smith");
strcpy(emp_tel, "765-4321");
EXEC SQL INSERT INTO emp_tab VALUES
(:emp_id :indvalue, :emp_tel :indvalue);
```

➲ **例 3**

select条件のためのコードで初期化された出力ホスト変数hoDeptNoと入力ホスト変数hoEmpNoを使用する：

```
EXEC SQL BEGIN DECLARE SECTION;
int  hoDeptNo, hoEmpNo;
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT deptNo FROM Employee WHERE empNo = :hoEmpNo INTO :hoDeptNo;
```

変数スコープ

他のC変数同様に宣言セクションに、EXTERN又はSTATICとして変数スコープを宣言します。

➲ **例 1、外部グローバル変数：**

ESQLでサポートされているデータ型の外部変数を参照するために、ESQL変数宣言の前に“extern”を追加する：

```
EXEC SQL BEGIN DECLARE SECTION;
extern int var1;
EXEC SQL END DECLARE SECTION;
```

➲ **例 2、静的変数：**

静的変数としてローカル変数をセットするために、ESQL変数宣言の前に"static"を追加する：

```
EXEC SQL BEGIN DECLARE SECTION;
static int var1;
EXEC SQL END DECLARE SECTION;
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

➲ 例 3、関数でやり取りされる変数：

関数の入力変数から値を参照するために、又はESQLやSQL問合せ文から得た値を戻すために、変数データをコピー又はESQL変数のデータ構造体へのポインタを再割り当てます：

```
/* Method 1, copy the value to esql host variable */
func1(int input_emp_id, char *output_telno)
    {
     EXEC SQL BEGIN DECLARE SECTION;
     int  emp_id;
     char telno[15];
     EXEC SQL END DECLARE SECTION;
     emp_id = input_emp_id;                  /* copy the input value */
     EXEC SQL SELECT telno FROM emp_tab WHERE emp_id = :emp_id INTO :telno;
     strcpy(output_telno, telno);
    }
/* Method 2, reassign the buffer pointer */
func1(char *input_emp_name, char *output_telno)
    {
     EXEC SQL BEGIN DECLARE SECTION;
     varcptr p_name, p_telno;
     EXEC SQL END DECLARE SECTION;
     p_name.len  = strlen(input_emp_name);   /* input string length  */
     p_name.arr  = input_emp_name;
     p_telno.len = 15;                       /* output string length */
     p_telno.arr = output_telno;
     EXEC SQL SELECT telno FROM emp_tab WHERE emp_name = :p_name
INTO :p_telno;
    }
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

インジケータ変数

インジケータ変数は、アプリケーション・プログラムのホスト変数のNULL値と切り捨てを扱うオプション的な手法です。インジケータ変数を使用する時、SQL 文のホスト変数の後ろに付けます。宣言できるインジケータ変数のデータ型はintegerのみです。

➲ **例 1**

インジケータ変数の宣言：

```
EXEC SQL BEGIN DECLARE SECTION;
int  hoDeptNo, indDeptNo, hoEmpNo;
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT deptNo FROM Employee
                       WHERE empNo = :hoEmpNo
                       INTO :hoDeptNo :indDeptNo;
```

インジケータ変数 indDeptNoは、ホスト変数 hoDeptNoの直後に続き、頭にはコロン(:)を付けます。

インジケータ変数の値は、以下のとおりです。

インジケータ値	データベースから回収した値
SQL_NULL_DATA	NULL値が戻されます。出力ホスト変数は不確定です。
0以上	インジケータ変数は、元の出力ホスト変数のバッファ長としてセットされます。インジケータ変数がBLOBデータを取得する場合、そのインジケータ値はBLOBデータを取得するのに先立って、データベースの元の出力ホスト変数のバッファ長にセットします。

©Copyright 1995-2015 Syscom Computer Engineering Co.

➲ 例 2

データベースにNULL値を入力するためにインジケータ変数を使います。対応するホスト変数は無視されます：

```
EXEC SQL BEGIN DECLARE SECTION;
int  hoDeptNo, indDeptNo;
EXEC SQL END DECLARE SECTION;
inDeptNo = SQL_NULL_DATA;
EXEC SQL INSERT INTO Department (DeptName, DeptNo)
              VALUES ('Human Resource', :hoDeptNo :inDeptNo);
```

上記の例は、DeptNoカラムにNULL値を挿入し、hoDeptNoの値は、無視されます。

表示値	データベースで受け取る値
SQL_NULL_DATA	NULL値。
0以上	CHARとBINARYデータ型のための、入力ホスト変数の実際のバッファ長。インジケータ変数を与える時、VARCHAR/VARBINARYデータ型にセットされたホスト変数長は、無視されます。
SQL_NTS	バッファは、NULL終端のCHARとBINARYデータ型です。VARCHAR/VARBINARYデータ型にセットされたホスト変数は無視されます。

3.3 ステータス・コード

Include変数

DBMasterのプリプロセッサ、又はランタイム・アプリケーションでは、特別なデータ構造体のために３種類の宣言が必要です。

- dbenvca
- sqlca

©Copyright 1995-2015 Syscom Computer Engineering Co.

- sqlda

DBENVCA

dbenvca宣言は、DBMaster がアプリケーション・プログラムで使用する環境変数です。プログラムで必ず宣言します。

➲ 例 1

dbenvcaの構文：

```
EXEC SQL INCLUDE [EXTERN] dbenvca;
```

➲ 例 2

dbenvcaを使う：

```
file1.c
EXEC SQL INCLUDE DBENVCA;
main()
    {
     ...
     EXEC SQL .
     ...
    }
file2.c
EXEC SQL INCLUDE EXTERN DBENVCA;
func1()
    {
     ...
    }
```

sqlcaとsqldaの宣言構文は、dbenvcaと同じです。Sqlcaはステータス・コードとの連絡に使用し、sqldaは動的ESQLで使用します。

©Copyright 1995-2015 Syscom Computer Engineering Co.

SQLCA

SQLコマンドでそれぞれ実行されたステータス・コードは、SQL Communication Area (SQLCA)に戻されます。DBMasterは、Cプログラムへステータス情報を渡すために、このデータ構造体に含まれる変数を使用します。問題がある場合、情報は解析され処理されます。

SQLCAを宣言する

SQLCA構造体の変数は、ESQL/Cプログラムの中でグローバルにアクセス可能である必要があります。SQLCAとDBENVCAの双方とも、ESQL/Cソース・プログラムで宣言しなければならないグローバル変数です。SQLCAは、プリプロセッサが以下の文を見つけた時に自動的に宣言される構造体です。

➲ **例**

SQLCAを自動的に宣言する：

```
EXEC SQL INCLUDE [EXTERN] SQLCA;
```

SQLCAに戻されるステータス

データベース・サーバーからのステータス情報は、SQLCAを通じて戻されます。データを分析し、エラーや警告を取り扱うのは、アプリケーション・プログラムの役割です。

SQLCAのステータス・コードをチェックし、エラーと警告を扱うようアプリケーション・プログラムに命令する方法が２つあります。Cコードにそのコマンドを記述することができますし、Cコード・プリプロセスの際にエラー・ハンドリングを生成するSQLのWHENEVERコマンドを使うこともできます。

➲ **例**

SQLCA構文定義：

```
#define MAX_ERR_STR_LEN 256
/*------------------------------------------------------------------
* SQLCA - the SQL Communications Area (SQLCA)
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
*-----------------------------------------------------------------*/
typedef struct sqlca
    {
     unsigned char  sqlcaid[8];          /* the string "SQLCA   "      */
     long           sqlcabc;             /* length of SQLCA, in bytes */
     long           sqlcode;             /* SQLstatus code            */
     long           sqlerrml;            /* length of sqlerrmc data   */
     unsigned char  sqlerrmc[MAX_ERR_STR_LEN]; /* name of object cause
error */
     unsigned char  sqlerrp[8];          /* diagnostic information    */
     long           sqlerrd[6];          /* various count and error code
*/
     unsigned char  sqlwarn[8];         /* warning flag array          */
     unsigned char  sqlext[8];         /* extension to sqlwarn array   */
    } sqlca_t;
#define SQLCODE   sqlca.sqlcode   /* SQL status code  */
#define SQLWARN0 sqlca.sqlwarn[0] /* master warning flag */
#define SQLWARN1 sqlca.sqlwarn[1] /* string truncated */
```

SQLCODE	意味
SQL_SUCCESS or 0	成功
SQL_ERROR or (-1)	エラー
SQL_NO_DATA_FOUND or 100	検索条件を満たす行が無い。
SQL_SUCCESS_WITH_INFO or 1	警告

©Copyright 1995-2015 Syscom Computer Engineering Co.

DBMasterのエラーコードは、sqlca.sqlerrd[0]に保存されます。フェッチされる行数は、sqlca.sqlerrd[3]に保存されます。FETCH文で複数の行をフェッチするとき､それを参照することができます｡詳細については､DBMasterの“エラーとメッセージ参照偏”を参照ください。

3.4 WHENEVER文

WHENEVER文は、エラーを扱うために使うことができます。ESQL/Cプリプロセッサが WHENEVER文を見つけた時、SQL文の結果によって実行するCのエラー・ハンドリング・コードを生成します。

条件	説明
SQLERROR	SQLCODEがSQL_ERRORの時
SQLWARNING	SQLCODEがSQL_SUCCESS_WITH_INFOの時
NOT FOUND	SQLCODEがSQL_NO_DATA_FOUNDの時

STOP、CONTINUE、GOTO、DO操作を実行するようアプリケーション・プログラムに命令するために、WHENEVER文にこれらの条件を指定します。

- STOP – SQL文の戻りステータスが特定の条件を満たした時、作業をロールバックし、データベースから切断し、アプリケーション・プログラムを終了します。
- CONTINUE –以前のWHENEVER文でセットした条件を使用不可にし、エラーを引き起こした文の次の文で実行を続けます。
- GOTO label_name -アプリケーション・プログラムの中のエラー・ハンドリング・ルーチンのためのラベルに実行を命じます。
- DO c_action_statement - 戻りステータスが特定の条件を満たした時、特定のアクションを実行します。

©Copyright 1995-2015 Syscom Computer Engineering Co.

➲ 例 1

```
WHENEVER SQLERROR DO break;
WHENEVER SQLWARNING DO print_warning();
 while ()
{
EXEC SQL …..;
EXEC SQL….;
}
```

各条件の初期設定値は、CONTINUEです。WHENEVER文は、アプリケーション・プログラムのそれ以降、同じ条件の次のWHENEVERまでの全SQL文に影響します。言い換えると、WHENEVER文の最新の設定は、途中で他のWHENEVER文がそれを上書きしない限り、ファイルの終わりまで以降のEXEC SQL文の全てに影響しつづけます。

無限のループが起こるのを防ぐために、エラー・ハンドラー、又はその他の関数で、WHENEVER *check_case* CONTINUEを必ずセットして下さい。この場合の関数は、WHENEVER文でエラー・ハンドリングを要求するEXEC SQL文を含むものです。

➲ 例 2

```
int func()
    {
     ...
     EXEC SQL WHENEVER SQLERROR goto error_handle;
     EXEC SQL INSERT INTO emp_table VALUES (:emp_id, :emp_name, :emp_addr);
     ...
     ...
     return 0;
error_handle:
     EXEC SQL WHENEVER SQLERROR CONTINUE;
     printf("ERR:%s\n", sqlcode.sqlerrmc);
     return -1;
    }
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

4 データ操作

この章では、ホスト変数、インジケータ変数、NULL値の使い方を含む、アプリケーションでESQLを使ったいくつかの例を紹介します。インタラクティブSQLで利用できるものと同じSQLに加え、本書で説明するSQLも、ESQLアプリケーションの中で使用することができます。

COMMIT、ROLLBACK、CONNECT、DISCONNECT、INSERT、SELECT、UPDATE、DELETE等を含む全SQLコマンドは、ESQLアプリケーションで記述するとき、EXEC SQLを前に付けます。但し、ESQLアプリケーションでDDL SQL文を使用するのは一般的ではありません。

ESQLで使用される付加的な組み込みSQL文の例は、BEGIN (END) DECLARE SECTION、DECLARE CURSOR、INCLUDE、WHENEVERのような宣言文です。これは、CLOSE、DESCRIBE、EXECUTE (IMMEDIATE)、FETCH、OPEN、PREPAREのような付加的な実行可能文と同様です。これらの実行可能文は、埋め込みSQLでのみ使用されます。

©Copyright 1995-2015 Syscom Computer Engineering Co.

4.1 データ操作

INSERT、UPDATE、DELETEのためにアプリケーションからデータベースにデータを渡すために、ホスト変数のみが使用されます。宣言セクションでのホスト変数の宣言に加えて、ホスト変数を参照する前に、各入力ホスト変数を初期化します。

➲ 例 1

```
EXEC SQL BEGIN DECLARE SECTION;
int   hoDeptNo, inDeptNo;
varchar  hoDeptName[8];
EXEC SQL BEGIN DECLARE SECTION;
/* Use host variable hoDeptNo to input data into database              */
hoDeptNo = 1001;
EXEC SQL INSERT INTO Department (DeptName, DeptNo)
              VALUES ('Human Resource', :hoDeptNo);
```

ESQL文のインジケータ変数の構文とその値に留意して下さい。

➲ 例 2

無視されるホスト変数で、データベースにNULL値を入力する：

```
inDeptNo = SQL_NULL_DATA;
EXEC SQL INSERT INTO Department (DeptName, DeptNo)
              VALUES ('Human Resource', :hoDeptNo :inDeptNo);
```

既に述べたように、*dmppcc*はvarcharデータ型を長さと配列要素のあるC構造体に変換します。

➲ 例 3

C言語で直接にはサポートされていないvarcharのようなSQLの擬似データ型のホスト変数を使う：

```
strcpy(hoDeptName.arr, 'Human Resource');
hoDeptName.len = strlen(hoDeptName.arr);
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
hoDeptNo = 1001;
EXEC SQL INSERT INTO Department (DeptName, DeptNo)
              VALUES (:hoDeptName, :hoDeptNo);
```

➲ **例 4**

入力ホスト変数のUPDATE/DELETEを使う：

```
strcpy(hoDeptName.arr, 'Human Resource');
hoDeptName.len = strlen(hoDeptName.arr);
hoDeptNo = 1001;
EXEC SQL UPDATE Department SET DeptNo = :hoDeptNo WHERE DeptName
= :hoDeptName;
EXEC SQL DELETE FROM Department WHERE DeptNo = :hoDeptNo + 1;
```

'IS NULL'は、WHERE句でNULL値をテストするためのSQL構文のキーワードです。このキーワードは、WHERE句でNULL値を表示するためにインジケータ変数を使うことはできません。

4.2 単一行データを回収する

データベースから出力ホスト変数に1行のデータを回収することは、アプリケーションに値を渡すことです。SELECT文のINTO句で出力ホスト変数を使います。

➲ **例 1**

```
EXEC SQL BEGIN DECLARE SECTION;
int   hoDeptNo, inDeptNo, inDeptName;
varchar  hoDeptName[8];
EXEC SQL BEGIN DECLARE SECTION;
hoDeptNo = 1001;
EXEC SQL SELECT DeptName FROM Department
          WHERE DeptNo = :hoDeptNo
           INTO :hoDeptName :inDeptName;
```

この例では、hoDeptNameは出力ホスト変数で、hoDeptNoは入力ホスト変数です。問合せがNULL値を回収するか、又は値の切り捨てがあるかをチェックするために出力ホスト変数にインジケータ変数inDeptNameが追加されます。0値は、正常な結果を意味します。

SQL_NULL_DATAは、ホスト変数がNULL値を受け取ったことを示します。0以上の値は、ホスト変数の結果が切り捨てられ、インジケータ変数の値が、元の値の長さであることを意味します。

例えば、hoDeptNameの長さが8で、データベースの結果が12の長さでは、そのインジケータ変数は12になり、そのホスト変数はもとの値の最初の8文字になります。

埋め込みSELECT文の中に、すべての標準SQL句(WHERE、GROUP BY、ORDER BY、HAVING等)を使うことができます。WHERE句とHAVING句に入力ホスト変数を使うことができます。

問合せがカーソルを使う複数の行を戻した場合、どのSELECT文も正確に1行を回収します(下記参照)。何行回収されたかを見るためには、各SELECT文の後のsqlca.sqlcodeをチェックします。SQLCAのsqlcodeが、SQL_NO_DATA_FOUNDの場合のみ、データは見つかりません。SELECT文で複数行が回収された時エラーを戻すために、このオプションをDBMasterのプリプロセッサ*dmppcc*で定義する必要があります。

➲ **例 2**

```
dmppcc -d TESTDB -u john -p johnspwd -s ex1.ec
```

このオプションをセットすると、結果が複数行の場合にエラーが表示されます。

©Copyright 1995-2015 Syscom Computer Engineering Co.

4.3 トランザクション処理

トランザクションの整合性を制御するために、EXEC SQL COMMITとEXEC SQL ROLLBACKコマンドを使用します。より高度なアプリケーションのためには、全てのデータ処理において更に制御を行うために、SAVEPOINTとROLLBACK TO SAVEPOINTオプションを使います。COMMIT WORKとROLLBACK WORKは、トランザクションにセットされた全セーブポイントを消去します。COMMIT、又はROLLBACKを指示せずにプログラムを終了する場合、トランザクションの全ての変更は、ロールバックされます。

dmconfig.iniファイルにAUTOCOMMIT接続オプションをセットすることができます。AUTOCOMMIT接続オプションをONにセットすると、実行したSQL文 の全ては直ちにコミットされ、ESQLアプリケーションはロールバック文を実行することができません。アプリケーションを実行する前に必ず自動コミットオプションを0FFにして下さい。又はデータベースに接続した後に自動コミットオプションを0N/0FFにするためにESQLソース・ファイルに以下の構文を記述して下さい。

➲ **例、SET AUTOCOMMIT ON/OFFを使う：**

```
EXEC SQL SET AUTOCOMMIT {ON|OFF}
```

4.4 動的接続の構文

ESQL/C アプリケーションで複数のデータベースへのアクセスを考えるかもしれません。複数のESQLプログラムを書いて、それらを異なるデータベース名でプリプロセスし、全プログラムを実行可能ファイルとしてリンクさせるか、SQL文の前に“AT database name”を追加し、そしてSQL文を実行する前にデータベース名を定義します。

➲ **例**

```
EXEC SQL BEGIN DECLARE SECTION;
 char db1[20];
 char usr1[10];
```

```
 char pwd1[10];
 int c1;
EXEC SQL END DECLARE SECTION;
/* GetDBInfo ()is user function which will pass back database name, user,
password */
GetDBInfo(db1, usr1, pwd1);
EXEC SQL CONNECT TO :db1 :usr1 :pwd1;
EXEC SQL AT :db1 select c1 from t1 into :c1;
EXEC SQL DISCONNECT :db1;
```

4.5 カーソルを使う

複数の行を回収する

問合せが複数の行を戻す時、プログラムは問合せを別々に実行する必要があります。複数の行の問合せは、2段階で扱われます。プログラムが問合せを始めても、直ぐにはデータは得られません。その後、プログラムはカーソルを通じて一度にデータ行を要求します。

カーソルは、アプリケーション・プログラムで使用されるように、データベースから特定した行で操作することができるデータ・セレクタです。以下の操作は、DECLARE、OPEN、FETCH、CLOSE文で実行します。

➲ カーソルを使う：

1. カーソルを入れるストレージを割り当て、カーソルと関連SELECT文を宣言します。
2. 関連SELECT文の実行を開始し、カーソルを開きます。

   注 *実際には、SQL文の解剖、ホスト変数のバインド、文実行開始の３つのステップがあります。*

3. ホスト変数にデータ行をフェッチ/削除/更新し、それを処理します。

   注 *全ての行がフェッチされるまで、この手順を繰り返します。*

4. カーソルを閉じます。

©Copyright 1995-2015 Syscom Computer Engineering Co.

カーソルを宣言する

単一行を回収する場合を除いて、アプリケーション・プログラムの中に各SELECTコマンドのための新規カーソルを宣言します。

➲ **例 1**

DECLARE構文：

```
EXEC SQL DECLARE cursor_name [SCROLL] CURSOR FOR select_statement

        [INTO :output_host_var :indicator_var
[, :output_host_var :indicator_var]]
```

INTO host_variable句は、FETCH文で定義することも可能です。データをフェッチするためのあらゆる方法(例、FETCH NEXT、PREVIOUS、LAST、FIRST、ABSOLUTE、RELATIVE)を利用したい場合、カーソルを宣言する際に、SCROLLを指定します。

➲ **例 2**

単一FETCH文で複数の行をフェッチするために、SCROLLキーワードを指定する：

```
EXEC SQL DECLARE vendCursor SCROLL CURSOR FOR

          SELECT vendorName FROM vendors

           WHERE vendorNumber = :inputNo;
```

➲ **例 3**

替わりにDECLARE文でINTO句を使う：

```
EXEC SQL DECLARE vendCursor CURSOR FOR

          SELECT vendorName FROM vendors

           WHERE vendorNumber = :inputNo

            INTO :vendName;
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

カーソルを開く

カーソルは、特定の行の内容を操作するために、開いた状態である必要があります。DECLAREコマンドの指定したカーソル名の前にOPENコマンドを付けます。

➲ 例 1

OPEN構文：

```
EXEC SQL OPEN cursor_name
       [USING :input_host_var [:indicator_var]
           [, :input_host_var [:indicator_var]]]
```

➲ 例 2

カーソルを開くためのSQLコマンド：

```
EXEC SQL OPEN vendCursor;
```

OPENコマンドを実行した時、カーソルが見つけ、検索条件を満たす結果セットの最初の行にポイントします。カーソルに入力ホスト変数がある場合、OPENカーソル文の前か中に、全入力ホスト変数のための値を指定する必要があります。上記のケースでは、入力ホスト変数がDECLARE文で定義されているので、OPENカーソル文で再び入力ホスト変数を指定する必要はありません。

データ回収のためにカーソルを使う

カーソルは、FETCHコマンドを使ってデータを回収します。ループでは、FETCHはカーソルを結果セットにし、INTO句で指名されたホスト変数にSELECTリストで指定したカラム値をコピーして、現在の行を回収します。

➲ 例

FETCH構文：

```
EXEC SQL FETCH [NEXT | PREVIOUS | FIRST | LAST | ABSOLUTE nth_position |
RELATIVE nth_position] [num_rows ROWS]

         cursor_name
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
        [INTO :input_host_var [:indicator_var]
             [, :input_host_var [:indicator_var], ...]]
```

INTO句は、FETCH文で無視されます。nth_positionとnum_rowsは、ホスト変数又は定数integerです。

注 *PREVIOUS、FIRST、LAST、ABSOLUTE nth_position、RELATIVE nth_position、num_rows ROWSは、SCROLL オプションで定義されたカーソルとでのみ利用できます。*

コマンド・パラメータ

NEXT

結果セットの中の次の行を戻します。NEXTは、初期設定のカーソル・フェッチです。

➲ **例**

NEXTコマンド：

```
EXEC SQL FETCH cur1;              /* default is fetch next */
EXEC SQL FETCH NEXT cur1 INTO :c1, :c2;
```

PREVIOUS

結果セットの中の前の行を戻します。

➲ **例**

PREVIOUSコマンド：

```
EXEC SQL FETCH PREVIOUS cur1 INTO :c1, :c2;
```

FIRST

結果セットの中のカーソルを最初の行に移動し、最初の行を戻します。

©Copyright 1995-2015 Syscom Computer Engineering Co.

➲ **例**

FIRSTコマンド：

```
EXEC SQL FETCH FIRST cur1 INTO :c1, :c2;
```

LAST

結果セットの中の最後の行にカーソルを移動し、最後の行を戻します。

➲ **例**

LASTコマンド：

```
EXEC SQL FETCH LAST cur1 INTO :c1, :c2;
```

ABSOLUTE N

結果セットの中の*n*番目の行を戻します。*n* が負数の場合、戻された行は結果セットの最後の行から数えて*n*番目です。

➲ **例**

ABSOLUTE nコマンド：

```
n = 10;
EXEC SQL FETCH ABSOLUTE :n cur1 INTO :c1, :c2;
```

RELATIVE N

最近フェッチした行の後ろの*n*番目の行を戻します。*n*が負数の場合、戻された行はカーソルの関連位置から後ろ向きに数えて*n*番目です。

➲ **例**

RELATIVE nコマンド：

```
n = 10;
EXEC SQL FETCH RELATIVE :n cur1 INTO :c1, :c2;
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

FETCH

➲ 例

カーソルから結果をフェッチするためのSQLコマンド：

```
EXEC SQL FETCH vendCursor INTO :vendName;
```

各行を回収するためにFETCHコマンドを使います。結果セットの全ての行が回収された後、DBMasterはこれ以上行が見つからないことを表示するためにSQLCAのSQLCODEフィールドを値SQL_NO_DATA_FOUNDにセットします。インジケータ変数は、NULL値を検出するためにホスト変数と共に宣言します。

FETCH ROWS

num_rows ROW構文をFETCH文、若しくはホスト変数で配列されたINTOで指定した時、FETCH文で複数のデータ値を回収することができます。

➲ 例

FETCH ROWSコマンド：

```
EXEC SQL BEGIN DECLARE SETION;

int nrows;

int host1[50];

int host2[50];

char host3[50][100];  /* 50: means the maximum available fetched data
values,100: means the maximum data buffer length of each element. */

EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE myCur SCROLL CURSOR FOR SELECT c1,c2,c3 FROM table1
INTO :host1, :host2, :host3;

EXEC SQL OPEN myCur;

nrows = 50;

/* loop until no data found */

while (SQLCODE != SQL_NO_DATA_FOUND)

{

     EXEC SQL FETCH :nrows ROWS myCur; /* This will fetch 50 rows into host1,
host2, host3, because the maximum fetched  */
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
     for (j = 0; j < sqlca.sqlerrd[3]; j++) /* print out according to the
number of returned rows */

      printout(host1, host2, host3);

}

EXEC SQL CLOSE myCur;
```

カーソルでデータを削除する

カーソルで、一度に表から1行を選択、削除することができます。さらに行を削除する場合、別個にフェッチする必要があります。

➲ 行を削除する：

1. SELECTコマンドでカーソルを宣言します。
2. カーソルを開き、削除する行にそれを配置させるために、OPENとFETCHコマンドを使います。
3. DELETEコマンドを実行します。
4. その他の行を削除する場合、別のFETCHコマンドでカーソルを再配置します。

➲ 例

```
EXEC SQL DELETE FROM supplier WHERE CURRENT OF vendCursor;
```

連続する削除の間に、COMMIT WORKコマンドを使わないで下さい。このコマンドを実行すると、カーソルを閉じ、削除プロセスを終了します。AUTOCOMMITモードをONにして運用すると、同じ結果になります。DELETE又はUPDATE WHERE CURRENT OFカーソル文を使う前に、AUTOCOMMITモードをOFFにして下さい。

©Copyright 1995-2015 Syscom Computer Engineering Co.

カーソルでデータを更新する

カーソルで、一度に１行を更新することができます。さらに行を更新する場合、カーソルを再配置します。

➲ **行を更新する：**

1. SELECTコマンドでカーソルを宣言します。
2. カーソルを開き、更新する行にそれを配置するために、OPENとFETCHコマンドを使います。
3. UPDATEコマンドを実行します。
4. その他の行を更新する場合、別のFETCHコマンドでカーソルを再配置します。

➲ **例**

```
EXEC SQL UPDATE supplier SET price = price + 10
          WHERE CURRENT OF vendCursor;
```

連続する更新の間に、COMMIT WORKコマンドを使わないで下さい。又、AUTOCOMMITモードをOFFにしておいて下さい。これらのいずれも、カーソルを閉じ、更新プロセスを終了させます。

カーソルを閉じる

COMMIT WORKとROLLBACK WORKコマンドも、実際にはカーソルを閉じますが、CLOSE CURSORコマンドで明確にカーソルを閉じることができます。カーソルが必要無くなった時、割り当てたリソースを解放するためにカーソルを閉じます。

➲ **例 1**

CLOSE CURSOR構文：

```
EXEC SQL CLOSE cursor_name
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

➲ 例 2

カーソルを閉じるためのSQLコマンド：

```
EXEC SQL CLOSE vendCursor;
```

カーソルは削除され、再利用したり、開いたり、フェッチしたりすることができません。

©Copyright 1995-2015 Syscom Computer Engineering Co.

5 BLOBデータ

BLOBデータのサイズが準備時にわからない時、またはバッファのサイズが充分でない時、データ全体を回収するまで、部分的なBLOBデータをその都度PUT、若しくはGETしたいと考えるかもしれません。

プログラムに充分なバッファサイズを割り当てるか、BLOB全体をGETできるかどうかが問題ではない場合、BLOBカラムを回収する場合、本来のESQL構文を使います。

©Copyright 1995-2015 Syscom Computer Engineering Co.

5.1 PUT BLOB文

➲ PUT BLOB文を使う：

1. このBLOBホスト変数が後でバインドされることを表すために、ESQL文を準備し、疑問符でBLOBホスト変数を宣言します。

例、PREPARE構文：

```
EXEC SQL PREPARE stmt_name FROM “SQL SYNTAX”
| :sql_string_host_variable];
```

dmppccは、“SQL構文”を静的ESQL構文として扱います。その構文は解析され、実行計画はプリプロセスの際に保存されます。

dmppccは、PREPAREの中にsql_string_host_variableが含まれている時、それを動的ESQL構文として扱います。その構文は、ランタイムまで解析されません。詳細については、“動的ESQL”の章を参照して下さい。

GET BLOBやPUT BLOB構文でBLOBを扱いたい場合、“?”を使ってBLOBホスト変数として定義します。

例：emp_tableにemp_picを挿入する：

```
EXEC SQL PREPARE stmt1 FROM

         "INSERT INTO emp_table (emp_id, emp_pic) VALUES
(:emp_id, ?)";
```

2. このESQL文を実行する：

```
EXEC SQL EXECUTE stmt_name;
```

例：

```
EXEC SQL EXECUTE stmt1;
```

3. PUT BLOBを開始するタイミングを宣言する：

```
EXEC SQL BEGIN PUT BLOB FOR stmt_name;
```

例：

```
EXEC SQL BEGIN PUT BLOB FOR stmt1;
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

4. BLOBを配置し、bufsizeとbufptrの情報をファイルするために、どのBLOB変数を使用するかを定義します。PUT BLOBの順序は、はじめから最後にアンバインドされたBLOBのカラムの順にします。

```
EXEC SQL PUT BLOB FOR stmt_name USING :host_var [:indicator_var]

/*host_var's data type must be longvarchar or longvarbinary)       */
```

例：

```
strcpy(buf, "This is a test");

b1.buf     = buf;

b1.bufsize = strlen(buf);

EXEC SQL PUT BLOB FOR stmt1 USING :b1;
```

5. このカラムでこれ以上PUTするBLOBデータが無くなるまで、ステップ４を行って下さい。

6. PUTするBLOBカラムが複数ある場合、次のBLOBカラムを置き始めるタイミングを宣言して、ステップ４に戻ります。

```
EXEC SQL PUT NEXT BLOB FOR stmt_name;
```

例：

```
EXEC SQL PUT NEXT BLOB FOR stmt1;
```

7. BLOBカラム全体がデータベースにPUTされたら、PUT BLOB 操作が終了したことを宣言します。

```
EXEC SQL END PUT BLOB FOR stmt_name;
```

例：

```
EXEC SQL END PUT BLOB FOR stmt1;
```

➲ 例 1

完全なPUT BLOB文：

```
 * Prepare an insert statement; the input BLOB columns should use a

 * question mark to denote it.
*****************************************************************/
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
EXEC SQL PREPARE stmt1 FROM "insert into emp_table \
              (emp_id, emp_pic, emp_memo) values (:id, ?, ?)";
id = 1000+j;
/*****************************************************************
* Execute this statement.
*****************************************************************/
EXEC SQL execute stmt1;

/*****************************************************************
* If there are 300 characters to put into emp_pic, and we want to
* put it 100 characters at a time.
*****************************************************************/
pic_buffer.bufsize  = 100;         /* max buffer size                */
/* you must allocate enough buffer as indicated in field bufsize, or
*point to a valid buffer pointer */
pic_buffer.buf  = user_buf;
/*****************************************************************
* Begin PUT BLOB.
*****************************************************************/
EXEC SQL BEGIN PUT BLOB FOR stmt1;
/*****************************************************************
* Loop 3 times to put data.
*****************************************************************/
for (i=0; i < 3; i++)
    {
     sprintf(pic_buffer.buf, "user's picture %dth's data........ ", i);
     EXEC SQL PUT BLOB FOR stmt1 USING :pic_buffer;
    }

/*****************************************************************
* Now start to put the next BLOB column's data.
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
*****************************************************************/
EXEC SQL put next blob FOR stmt1;
/*****************************************************************
* If there are 200 characters to put into emp_memo, and we want to
* put it 100 characters at a time.
*****************************************************************/
memo_buffer.bufsize  = 100;         /* max buffer size               */
/* you must allocate enough buffer as indicated in field bufsize, or
*point to a valid buffer pointer */
memo_buffer.buf  = user_buf;
/*****************************************************************
* loop 2 times to put data
*****************************************************************/
for (i=0; i < 2; i++)
    {
     sprintf(memo_buffer.buf, "user's memo %dth's data...........", i);
     EXEC SQL PUT BLOB FOR stmt1 USING :memo_buffer;
    }
/*****************************************************************
* end PUT BLOB
*****************************************************************/
EXEC SQL END PUT BLOB FOR stmt1;
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

5.2 GET BLOB文

カーソルを使った複数行データ

➲ GET BLOB文を使う：

1. ESQL文を準備し、疑問符でBLOBホスト変数を宣言します(このBLOBホスト変数がまだバインドされていないことを意味します)。

```
EXEC SQL PREPARE stmt_name FROM "SQL SYNTAX";
```

動的ESQL構文と異なり、プリプロセス時にSQL構文がわかり、バインドしたホスト変数名を含みます。それがBLOBで、GET/PUT BLOBを使う場合は、'?'を指定します。

emp_tableからemp_picをフェッチすると想定します。

例：

```
EXEC SQL PREPARE stmt1 FROM "select emp_pic from emp_table into ?";
```

2. 準備した文のためにカーソルを宣言します。

```
EXEC SQL DECLARE cursor_name CURSOR FOR stmt_name;
```

例：

```
EXEC SQL DECLARE myCur CURSOR FOR stmt1;
```

3. カーソルを開きます。

例：

```
EXEC SQL OPEN myCur;
```

4. カーソルをフェッチします。

例：

```
EXEC SQL FETCH myCur;
```

5. GET BLOBを開始するタイミングを宣言する必要はありません。但し、, BLOBホスト変数のカラム数、利用できるバッファ・サイズ、有効なバッファ・ポインタを定義する必要があります。

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
EXEC SQL GET BLOB COLUMN :blobcol_num  FOR stmt_name USING :host_var
[:indicator_var]
```

例 1：

```
EXEC SQL BEGIN DECLARE SECTION;

int nCol;

longvarchar b1;

EXEC SQL END DECLARE SECTION;

nCol = 1;        /* assign nCol as the column order in projection */

b1.bufsize = 50;  /* if you want to get 50 bytes at a time        */

b1.buf  = buf;    /* assign a valid buffer pointer to b1          */

EXEC SQL GET BLOB COLUMN :nCol FOR stmt2 USING :b1
```

注　*BLOBカラムを取得する前に、残りのバッファサイズを取得するためにインジケータ変数を指定することができます。*

例 2：

```
EXEC SQL PREPARE stmt1 FROM "select c6 from d1 into ?";

EXEC SQL EXECUTE stmt1;

/* fetch BLOB with size = 0 first, to know total size with indicator
*/

nCol    = 1;

b1.bufsize = 0;

b1.buf     = wkbuf;

EXEC SQL GET BLOB COLUMN :nCol FOR stmt1 USING :b1 :i_b1;

if (SQLCODE == 0)

     printf("left size is %d\n", i_b1);
```

```
/* loop fetch BLOB with size = 2, and print out   "last" remain size
*/
for (i = 0; SQLCODE != SQL_NO_DATA_FOUND; i++)
    {
     b1.bufsize = 2;        /* get 1 char plus null ptr at a time */
     EXEC SQL GET BLOB COLUMN :nCol FOR stmt1 USING :b1 :i_b1;
     if (SQLCODE != SQL_NO_DATA_FOUND)
          printf("b1 = %s, last remain size is %d\n", b1.buf, i_b1);
     chkErr();
    }
```

注 *BLOBを回収するためにDBMasterにメモリを割り当てさせる時、bufsize = DB_ALLOCATE_MEMORYにセットすることも可能です。DBMasterは、次のFETCH又はCLOSE CURSOR文を呼び出す時に、BLOBに関連する割り当てメモリを解放します。このオプションの使用には注意して下さい。なぜなら、BLOBのために充分なメモリを割り当てることで、システムに"メモリ不足です"のエラーが発生するかもしれません。加えて、次のFETCH又はCLOSE文の後にデータベースによってメモリが解放されるので、BLOB変数のポインタは、無効なアドレスを参照します。これにより、使用しているプログラムにコア・ダンプやエラー実行が起こるかもしれません。*

6. さらに取得するBLOBデータがある場合は、前述の手順を繰り返します。

例：

```
EXEC SQL PREPARE stmt1 FROM "select c6 from d1 into ?";
EXEC SQL EXECUTE stmt1;
/* fetch BLOB with size = 0 first, to know total size with indicator
*/
nCol    = 1;
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
b1.bufsize = 0;
b1.buf     = wkbuf;
EXEC SQL GET BLOB COLUMN :nCol FOR stmt1 USING :b1 :i_b1;
if (SQLCODE == 0)
     printf("left size is %d\n", i_b1);
/* loop fetch BLOB with size = 2, and print out   "last" remain size
*/
for (i = 0; SQLCODE != SQL_NO_DATA_FOUND; i++)
    {
     b1.bufsize = 2;        /* get 1 char plus null ptr at a time */
     EXEC SQL GET BLOB COLUMN :nCol FOR stmt1 USING :b1 :i_b1;
     if (SQLCODE != SQL_NO_DATA_FOUND)
          printf("b1 = %s, last remain size is %d\n", b1.buf, i_b1);
     chkErr();
    }
```

7. 全データが回収されたか、残りのBLOBデータを取得する必要がない場合、これ以上行が見つからなくなるまで、カーソルのための残りの結果バッファをフェッチし続けます。

例：

```
#define MAX_BUF_SIZE 101
/* Prepare a SELECT statement, the output BLOB column should use    */
/* a question mark to denote it.                                    */
EXEC SQL PREPARE stmt1 FROM "select emp_id, emp_pic from emp_table
                               into :id, ?";
/* Declare a cursor to associate it with this statement.           */
```

```
EXEC SQL DECLARE myCur CURSOR FOR stmt1;

/* Open this cursor.                                      */

EXEC SQL OPEN myCur;

/* To get one MAX_BUF_SIZE character at a time, we must fill the

* following field first.                                  */

nCol = 2;                    /* second output column in projection.*/

pic_buffer.bufsize  = MAX_BUF_SIZE;   /* max buffer size           */

/* You must allocate enough buffer as indicated in field bufsize, or

   point to a valid buffer pointer.                        */

pic_buffer.buf  = user_buf;

/* loop fetch result.                                      */

while (1)

    {

     EXEC SQL FETCH myCur INTO :id, ?; /* fetch cursor            */

     if (SQLCODE)

         {

          if (SQLWARN0 != 'W')

              break;

         }

     printf("emp_id = %d\n", id);

     printf("emp_pic = ");

     /* loop get BLOB data until no data is found.               */

     for (i = 0; SQLCODE != SQL_NO_DATA_FOUND; i++)

         {
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
          EXEC SQL GET BLOB COLUMN :nCol FOR stmt1
                             USING :pic_buffer :pic_ind;
          if (SQLCODE != SQL_NO_DATA_FOUND)
              {
               if (pic_ind == SQL_NULL_DATA)
                   printf("(null)");
               else
                   {
                    for(j = 0; j < MAX_BUF_SIZE-1; j++)
                    printf("%c", pic_buffer.buf[j]);
                   }
              }
         }
     printf("\n");
    }

/* Close the cursor.                                                  */
EXEC SQL CLOSE myCur;
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

5.3 PUT BLOBとGET BLOBの違い

順序よくBLOBデータを取得する必要はないので、取得するカラムを割り当てることができます。GET BLOB文を使う場合、データは左のカラムから右のカラム(小さいカラム番号から大きいカラム番号)に回収されるとは限りません。BLOBデータ全体を必要としない場合は、それ全体を取得する必要もありません。

PUT BLOBは、必ず最初のBLOBカラムから開始し、最後まで回収し、開始と終了のタイミングを表示します。各BLOBカラムを順にPUTしないか、或いはPUT BLOBを終了するタイミングを表示しない場合、BLOBカラムはデータベースに挿入されません。

それゆえ、ユーザーが切断又はプログラムを終了した時に、操作が終わらないので、その操作は中止されます。データベースから切断する時、最後の文が取り消されたことを示すエラーが表示されます。

©Copyright 1995-2015 Syscom Computer Engineering Co.

6　動的ESQL

ESQL文を記述するには２つの方法があります。より簡単で一般的な方法は、静的な埋め込みによるものです。これは、プリコンパイルの前に、ソース・プログラム・テキストの一部としてSQL文を記述することを意味します。ここまで、もっぱら静的ESQLについて説明してきました。

静的ESQLが非常に有益であっても、プログラムを記述する際にSQL文のための正確な構文を知ることが必要です。アプリケーションによっては、ランライム時にユーザーの入力に従ってSQL文を作成する機能が必要とされます。動的E SQLでそれを実現することができます。つまり、プログラムが文字の文字列としてSQL文を構成し、ランライム時にデータベースにそれを渡します。動的ESQL文の全部又は一部は、プリコンパイル時にはわかりません。その完全な文は、ランタイム時にメモリで構築されます。

動的ESQL文は、それがSELECT文かどうか、又はホスト変数がわかっているかどうかによって、4通りに分類されます。それぞれの動的ESQLは、異なる方法でESQL文のプログラムを作ります。

©Copyright 1995-2015 Syscom Computer Engineering Co.

タイプ	特徴	方法
1	問合せ無し、入力ホスト変数無し	直ちに実行
2	問合せ無し、入力ホスト変数の数がわかっている	準備/実行
3	問合せ、入力/出力ホスト変数の数がわかっている	カーソル
4	入力か出力ホスト変数の数がわからない	SQLDA

6.1 タイプ1の動的ESQL

タイプ1の動的ESQLは、入力ホスト変数の無い非SELECT文です。このサンプルケースです、動的ESQLを処理するためにEXECUTE IMMEDIATEを使うことができます。

➲ 例

タイプ1の動的ESQL：

```
EXEC SQL BEGIN DECLARE SECTION;
varchar upd_str[100];
EXEC SQL END DECLARE SECTION;
sprintf(upd_str.arr, "UPDATE part SET qty = qty -1 WHERE ");
gets (update_condition);                      /* get dynamic upd condition
*/
strcat (upd_str.arr, update_condition);    /* construct dynamic SQL    */
upd_str.len = strlen(upd_str.arr);

EXEC SQL EXECUTE IMMEDIATE FROM :upd_str;   /* execute it              */
EXEC SQL COMMIT WORK;
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

6.2 タイプ2の動的ESQL

タイプ2の動的ESQLは、少し複雑です。これは入力ホスト変数の数がわかっている非SELECT文です。プリコンパイル時に入力ホスト変数の数が決まっていない場合は、タイプ4の動的ESQLを使います(この章の後部の節を参照して下さい)。

➲ タイプ2の動的ESQLアプリケーションを構築する：

1. 宣言セクションで、全ての入力ホスト変数を宣言します。
2. 以下の文を準備します。

```
EXEC SQL PREPARE statement_name FROM :statement_string;
```

3. 全ての入力ホスト変数とインジケータ変数の値をセットします。
4. 以下の文を実行します。

```
EXEC SQL EXECUTE statement_name USING :input_var1, :input_var2;
```

➲ 例

タイプ2の動的ESQLアプリケーションを構築します。

```
EXEC SQL BEGIN DECLARE SECTION;
varchar del_str[80];
int  ord_number;
EXEC SQL END DECLARE SECTION;
char del_condition[80];
/* there is an input variable :iVord, it is a place holder             */
sprintf(del_str.arr, "DELETE FROM order WHERE ordid = :iVord AND ");
gets (del_condition);                /* get the DYNAMIC delete condition
*/
strcat (del_str.arr, del_condition); /* construct dynamic SQL          */
del_str.len = strlen(del_str.arr);
EXEC SQL PREPARE del_stmt FROM :del_str;
/* please note the relationship between the input host                 */
/* variable ord_number and placeholder iVord.                          */
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
gets (ord_number);                    /* set host variable value          */
EXEC SQL EXECUTE del_stmt USING :ord_number;
EXEC SQL COMMIT WORK;
```

6.3 タイプ3の動的ESQL

タイプ3の動的ESQLは、入力と出力ホスト変数の数がわかっているSELECT文です。プリコンパイル時に入力と出力ホスト変数の数を決めていない場合、タイプ4の動的ESQLを使います。

➲ タイプ3の動的ESQLを処理する：

1. ホスト変数の内部に文の文字列､各入力変数を'?'で置き換えたSQL文を含む文の文字列を準備します。
2. 文名と文の文字列を指定するESQL PREPARE文を実行します。

```
EXEC SQL PREPARE statement_name FROM :statement_string;
```

3. カーソル名と文名を指定するESQL DECLARE CURSOR文を実行します。

```
EXEC SQL DECLARE cursor_name CURSOR FOR statement_name;
```

4. 各入力変数のための値を指定します。
5. 入力変数のリストでカーソルを開きます。
6. ループで、結果を出力変数のリストにフェッチします。
7. カーソルを閉じます。

➲ 例

タイプ3の動的ESQLを処理する：

```
EXEC SQL BEGIN DECLARE SECTION;
varchar sel_str[100];
int  ord_num, ord_date, custor_num;
EXEC SQL END DECLARE SECTION;
/* 1. Put a build statement string inside a host variable, including one
*/
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
/*    placeholder for each input variable.                                    */
gets(condition);
sprintf(sel_str.arr, "SELECT Ordid, Orddate FROM order WHERE CusId = :c \
                         AND %s", condition);
sel_str.len = strlen(sel_str.arr);
/* 2. Prepare the statement.                                                  */
EXEC SQL PREPARE sel_stmt FROM :sel_str;
/* 3. Declare the cursor.                                                     */
EXEC SQL DECLARE emp_cursor CURSOR FOR sel_stmt;
/* 4. Specify a value for each input variable.                                */
gets (custor_num);
/* 5. Open the cursor with a list of input variables.                         */
EXEC SQL OPEN emp_cursor USING :custor_num;
/* 6. In a loop, fetch the result to a list of output variables.              */
do
    {
     EXEC SQL FETCH emp_cursor INTO :ord_num, :ord_date;
     printf("ord_num = %d ord_date = %d\n", ord_num, ord_date);
    } while (sqlca.sqlcode == SQL_SUCCESS ||
            sqlca.sqlcode == SQL_SUCCESS_WITH_INFO)
/* 7. Close the cursor.                                                       */
EXEC SQL CLOSE emp_cursor;
```

6.4 タイプ4の動的ESQL

タイプ4の動的ESQLは、プリコンパイル時に定義されていない入力/出力ホスト変数があるSQL文です。このような動的ESQLでは、SQLDAディスクリプタ、ホスト変数を使用します。タイプ4には多くのステップがあります。但し、そのステップの状況によっては、省略することができます。例えば、出力ホスト変数の数がわからないけど、入力ホスト変数の数がわかっている、又はその逆も同様です。

©Copyright 1995-2015 Syscom Computer Engineering Co.

SQLDAディスクリプタ

SQLDAディスクリプタは、アプリケーションとDBMasterが、数、値、長さ、データ型、各ホスト変数の名前を保存する場所です。また、動的ESQL文の変数値を表示します。

SQLDAには、ホスト変数の数とホスト変数の内容についての情報があります。ホスト変数の数は、現在のSQL文で使用されている入力又は出力ホスト変数の数と同じです。内容の情報は、２つの情報の集まりです。１つは、カラム情報から成ります。もう1つは、ホスト変数情報で構成されています。

Describeコマンド

タイプ4の動的ESQLは、ランタイム時にユーザーが文を入力するまで、SQL文で使用されているカラムの数がわかりません。それゆえ、アプリケーションは、データベースの内と外で情報をやり取りするために必要な入力/出力ホスト変数がいくつであるかわかりません。

これが、SQLDAでホスト変数の情報を保管するために、ディスクリプタ・エリアを使う理由です。動的ESQLが準備された後、アプリケーション・プログラムは、入力(DESCRIBE BIND VARIABLE)のための入力カラムがいくつあるのか、出力(DESCRIBE SELECT LIST)のための出力カラムがいくつあるのか、それらが何であるのかをDBMasterに確認するためにDescribeコマンドを使います。

Describeコマンドは、カラム数(例、ホスト変数)や、これらのカラムのデータをディスクリプタに置きます。ループでは、アプリケーションがどのような種類の入力/出力カラムがあるのかをチェックし、ホスト変数のためのスペースを割り当てます。

SQLDAを経由して情報を渡す

DBMasterには、SQLDAの割り当て/解放を行う２つの関数があります：

```
allocate_descriptor_storage()
free_descriptor_storage()
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

エラーが発生した場合、情報もSQLCAで保管されます。

エラーのプロトタイプは、以下のとおり：

```
int allocate_descriptor_storage(long maxNumber, char **descriptor_name);
```

➲ 例

SQLDAディスクリプタ'desc1'に、最大10を割り当てる：

```
char  *desc1;
long   maxNumber = 10;
/* SQLCODE is macro of sqlca.code                                       */
/* support error_handle() is a function for error handling              */
allocate_descriptor_storage(maxNumber, &desc1);
if (SQLCODE == SQL_ERROR) error_handle();
```

➲ 例

SQLDAディスクリプタ'desc1'を、解放する：

```
free_descriptor_storage(desc1);
if (SQLCODE == SQL_ERROR) error_handle();
```

SQLDAは、ホスト変数とカラム情報で構成されています。カラム情報は、説明時にDBMasterによってセットされます。ホスト変数情報は、アプリケーションによってセットされます。インジケータ変数情報は、DBMasterかアプリケーションによってセットされます。アプリケーションは、ホスト変数情報をセットするために関数SetSQLDA()を使い、カラム情報を取得するために関数GetSQLDA()を使うことができます。エラーが発生した場合、情報もSQLCAに保管されます。

SetSQLDAプロトタイプ・コマンドは以下のとおり：

```
int SetSQLDA(char *descriptor_name, short host_variable_number,
            short option, long option_value);
```

GetSQLDAプロトタイプ・コマンドは以下のとおり：

```
int GetSQLDA(char *descriptor_name, short projection_column_number,
            short option, void *option_value);
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

関数のための引数：

オプション	説明
descriptor_name	allocate_descriptor_storage()で割り当てられたSQLDAディスクリプタ
host_variable_number	ホスト変数の数
projection_column_number	プロジェクト・カラム一覧の数
option	オプションのシンボル
option_value	有効なオプション値

注　*オプションがSQLDA_NUM_OF_HV、又はSQLDA_MAX_FETCH_ROWSの時、host_variable_number又はprojection_column_numberは使用されません。*

オプションのシンボルは、SET又はGETする情報の種類を指定するために使用します。

オプション	説明
SQLDA_DATABUF	option_valueがデータ・バッファ・ポインタであることを指定します。
SQLDA_DATABUF_LEN	option_valueがデータ・バッファの最大長であることを指定します。
SQLDA_DATABUF_TYPE	option_valueがデータ・バッファのデータ型であることを指定します。
SQLDA_BLOB_FLAG	ホスト変数は、BLOB方法で取り扱われていることを指定します。
SQLDA_PUT_DATA_LEN	option_valueがデータを置いたホスト変数のサイズであることを指定します。
SQLDA_INDICATOR	option_valueがインジケータ値であることを指定します。

©Copyright 1995-2015 Syscom Computer Engineering Co.

オプション	説明
SQLDA_MAX_FETCH_ROWS	option_valueがフェッチした行の最大数であることを指定します。
SQLDA_STORE_FILE_TYPE	option_valueが保管ファイルのタイプであることを指定します。(ESQL_STORE_FILE_CONTENT又はESQL_STORE_FILE_NAME)
SQLDA_COLTYPE	option_valueがホスト変数のカラム型であることを指定します。
SQLDA_COLLEN	option_valueがホスト変数のカラム長であることを指定します。
SQLDA_COLPREC	option_valueがホスト変数のカラム精度であることを指定します。
SQLDA_COLSCALE	option_valueがカラム変数のカラム・スケールであることを指定します。
SQLDA_COLNULLABLE	option_valueがNULL値を含むことができるカラム・ホスト変数であること指定します。
SQLDA_COLNAME	option_valueがホスト変数のカラム名であることを指定します。
SQLDA_COLNAME_LEN	option_valueがホスト変数のカラム名長であることを指定します。
SQLDA_NUM_OF_HV	option_valueがこの現在のSQL分で使用しているホスト変数の総数であることを指定します。

これらのオプション値のデータ型と設定は、以下の表のとおりです。

オプション	オプション値
SQLDA_NUM_OF_HV	整数 0 ~ 252
SQLDA_MAX_FETCH_ROWS	正の整数

©Copyright 1995-2015 Syscom Computer Engineering Co.

オプション	オプション値
SQLDA_DATABUF	有効なポインタ
SQLDA_DATABUF_LEN	正の整数
SQLDA_DATABUF_TYPE	下記の「データ・バッファのデータ型」の表を参照して下さい。
SQLDA_STORE_FILE_TYPE	ESQL_STORE_FILE_CONTENT ESQL_STORE_FILE_NAME
SQLDA_COLTYPE	下記の「カラムのデータ型」の表を参照して下さい。
SQLDA_COLLEN	SQLDA_COLTYPEの値による正の整数
SQLDA_COLPREC	SQLDA_COLTYPEの値による正の整数
SQLDA_COLSCALE	SQLDA_COLTYPEの値による正の整数
SQLDA_COLNULLABLE	SQL_NO_NULLS–カラムはNull不可 SQL_NULLABLE – カラムはNull可
SQLDA_COLNAME	文字の文字列
SQLDA_COLNAME_LEN	整数1 ~ 18
SQLDA_INDICATOR	SQL_NULL_DATA – NULLデータを入力 SQL_DEFAULT_PARAM – DEFAULT値を入力 SQL_NTS – NULL終端までデータを入力 正の整数 – 入力データの実際の長さ
SQLDA_BLOB_FLAG	SQLDA_BLOB_ON SQLDA_BLOB_OFF
SQLDA_PUT_DATA_LEN	正の整数

©Copyright 1995-2015 Syscom Computer Engineering Co.

データ・バッファのデータ型は、以下の表で説明します。
SQLDA_DATABUF_TYPEの値は、SQLDA_DATABUFで指定されるデータ・バッファにあるデータ型をDBMasterに伝えます。

SQLDA_DATABUF_TYPE	SQLDA_DATABUFのC型
SQL_C_CHAR	文字ポインタ(印刷可能な文字の文字列)
SQL_C_LONG	Long
SQL_C_SHORT	Short
SQL_C_FLOAT	Float
SQL_C_DOUBLE	Double
SQL_C_BINARY	文字ポインタ(印刷不可の文字の文字列)
SQL_C_DATE	ESQL定義タイプ‘eq_date’(esqltype.hを参照)
SQL_C_TIME	ESQL定義タイプ‘eq_time’(esqltype.hを参照)
SQL_C_TIMESTAMP	ESQL定義タイプ‘eq_timestamp’(esqltyep.hを参照)
SQL_C_FILE	文字の文字列(SQLDA_DATABUFの値は、ファイル名です)

カラムのデータ型は、以下の表で説明します。

SQLDA_COLTYPE	カラムの対応データ型
SQL_CHAR	char
SQL_VARCHAR	varchar
SQL_LONGVARCHAR	Long varchar
SQL_BINARY	binary
SQL_LONGVARBINARY	Long varbinary
SQL_INTEGER	int
SQL_SMALLINT	smallint

©Copyright 1995-2015 Syscom Computer Engineering Co.

SQLDA_COLTYPE	カラムの対応データ型
SQL_REAL	float
SQL_DOUBLE	double
SQL_DECIMAL	decimal
SQL_DATE	date
SQL_TIME	time
SQL_TIMESTAMP	timestamp
SQL_FILE	file

GetSQLDA()を使って、参照される全てのオプションの値を取得することができます。但し、オプションによっては、SetSQLDA()でセットすることができません。

以下のオプションはセットできません。

- SQLDA_COLTYPE
- SQLDA_COLLEN
- SQLDA_COLPREC
- SQLDA_COLSCALE
- SQLDA_COLNULLABLE
- SQLDA_COLNAME
- SQLDA_COLNAME_LEN
- SQLDA_NUM_OF_HV

以下の表は、何がSQLDAオプションをセットするか、またいつセットされるかについての詳細情報を提供します。アプリケーションはDBMasterからカラム情報を要求するために、まずDESCRIBEを使うことを理解しておいてください。DBMaster はカラム情報をセットします。それからアプリケーシ

©Copyright 1995-2015 Syscom Computer Engineering Co.

ョンはホスト変数情報をセットし、DBMasterにその文を実行するように命じます。

入力ホスト変数：

オプションのシンボル	セットするアプリケーション	セットするタイミング
SQLDA_NUM_OF_HV	DBMaster	DESCRIBE BIND VARIABLESの間
SQLDA_DATABUF	アプリケーション	OPEN/EXECUTEの前
SQLDA_DATABUF_LEN	アプリケーション	OPEN/EXECUTEの前
SQLDA_DATABUF_TYPE	アプリケーション	OPEN/EXECUTEの前
SQLDA_STORE_FILE_TYPE	アプリケーション	OPEN/EXECUTEの前
SQLDA_COLTYPE	DBMaster	DESCRIBE BIND VARIABLESの間
SQLDA_COLLEN	DBMaster	DESCRIBE BIND VARIABLESの間
SQLDA_COLPREC	DBMaster	DESCRIBE BIND VARIABLESの間
SQLDA_COLSCALE	DBMaster	DESCRIBE BIND VARIABLESの間
SQLDA_COLNULLABLE	DBMaster	DESCRIBE BIND VARIABLESの間
SQLDA_COLNAME	未使用	--
SQLDA_COLNAME_LEN	未使用	--
SQLDA_INDICATOR	アプリケーション	OPEN/EXECUTEの前

©Copyright 1995-2015 Syscom Computer Engineering Co.

オプションのシンボル	セットするアプリケーション	セットするタイミング
SQLDA_MAX_FETCH_ROWS	アプリケーション	OPENの前
SQLDA_BLOB_FLAG	アプリケーション	OPEN/EXECUTEの前
SQLDA_PUT_DATA_LEN	アプリケーション	PUT BLOBの前

出力 ホスト変数：

ディスクリプタ・フィールド	セットするアプリケーション	セットするタイミング
SQLDA_NUM_OF_HV	DBMaster	DESCRIBE SELECT LISTの間
SQLDA_DATABUF	アプリケーション	FETCHの前
SQLDA_DATABUF_LEN	アプリケーション	FETCHの前
SQLDA_DATABUF_TYPE	アプリケーション	FETCHの前
SQLDA_STORE_FILE_TYPE	未使用	DESCRIBE SELECT LISTの間
SQLDA_COLTYPE	DBMaster	DESCRIBE SELECTLISTの間
SQLDA_COLLEN	DBMaster	DESCRIBE SELECT LISTの間
SQLDA_COLPREC	DBMaster	DESCRIBE SELECT LISTの間
SQLDA_COLSCALE	DBMaster	DESCRIBE SELECT LISTの間
SQLDA_COLNULLABLE	DBMaster	DESCRIBE SELECT LISTの間

©Copyright 1995-2015 Syscom Computer Engineering Co.

ディスクリプタ・フィールド	セットするアプリケーション	セットするタイミング
SQLDA_COLNAME	DBMaster	DESCRIBE SELECT LISTの間
SQLDA_COLNAME_LEN	DBMaster	DESCRIBE SELECT LISTの間
SQLDA_INDICATOR	DBMaster	FETCHの間
SQLDA_BLOB_FLAG	アプリケーション	FETCHの前
SQLDA_PUT_DATA_LEN	未使用	--

アプリケーションのステップ

タイプ4の動的ESQLアプリケーションの構築には多くのステップがあります。入力/出力ホスト変数がないことがわかっている場合、ステップによっては省略することができます。

➲ **タイプ4の動的ESQLアプリケーションを構築する：**

1. ディスクリプタ変数を宣言します。
2. 最大数で動的入力/出力ホスト変数のためにSQLDAを割り当てます。
3. 文名と文文字列を指定するSQL PREPARE文を実行します。

```
EXEC SQL PREPARE statement_name FROM :statement_string;
```

4. ステップ3で準備した文のためのカーソルを宣言します。

```
EXEC SQL DECLARE cursor_name CURSOR FOR statement_name; // step 4 and
5 for input host variable.
```

注 *入力ホスト変数がある時のみステップ５を行います。*

5. ステップ３で準備した文で入力ホスト変数を説明し、バインドしたディスクリプタに情報を置きます。

```
EXEC SQL DESCRIBE BIND VARIABLES FOR statement_name INTO
descriptor_name;
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

a) 入力ホスト変数の長さをセットします。

b) 入力ホスト変数のデータ型をセットします。

c) 入力ホスト変数の値のためのストレージを割り当てます。

d) 入力ホスト変数の値をセットします。

e) 入力インジケータ変数の値をセットします。

6. ステップ４で宣言したカーソルを開き、カーソルが使用するディスクリプタ変数を指定します。

```
EXEC SQL OPEN cursor_name USING DESCRIPTOR descriptor_name; // step 7
and 8 for output host variable.
```

7. ステップ３で準備した文で出力カラム・プロジェクションを説明し、バインドしたディスクリプタにこれらの説明情報を置きます。

```
EXEC SQL DESCRIBE SELECT LIST FOR statement_name INTO descriptor_name;
```

8. 出力ホスト変数の長さをセットし、出力ホスト変数おデータ型をセットし、出力ホスト変数の値のためのストレージを割り当てます。

9. ステップ４で宣言したカーソルでデータをフェッチし、バインドしたディスクリプタのデータ・バッファにフェッチしたデータを置きます。

```
EXEC SQL FETCH cursor_name USING descriptor_name;
```

10. カーソルを閉じます。

```
EXEC SQL CLOSE cursor_name;
```

11. SQLDA (SQLDAのpデータ・フィールド)のためにユーザー割り当てメモリ・スペースを解放します。

12. ディスクリプタを割り当てから解放します。

➲ 例 1

タイプ4の動的ESQLアプリケーション：

```
#define maxNumber 10
#define STRING_LEN 128
EXEC SQL BEGIN DECLARE SECTION;
varchar stmt_str[128];
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
EXEC SQL END DECLARE SECTION;
/* 0. declare descriptor variables                                          */
char *input_descriptor, *select_descriptor;
long  nHv=0, nCol=0;
char *pColName, *pData;
long  colType, colScale, colNullable;
long   colLen, colNameLen, dataType, colPrec;
/* connect to database                                                      */
EXEC SQL CONNECT TO :dbname :user :password;
/* 1. allocate SQLDA for dynamic in/out host variables by maxNumber     */
allocate_descriptor_storage(maxNumber, &input_descriptor);
if (SQLCODE == SQL_ERROR) error_handle();
allocate_descriptor_storage(maxNumber, &select_descriptor);
if (SQLCODE == SQL_ERROR) error_handle();
/* 2. EXEC SQL PREPARE statement_name FROM :statement_string            */
gets(stmt_str.arr);
stmt_str.len = strlen(stmt_str.arr);
EXEC SQL PREPARE demo_stmt FROM :stmt_str;
/* 3. EXEC SQL DECLARE cursor_name CURSOR FOR statement_name            */
EXEC SQL DECLARE demo_cursor CURSOR FOR demo_stmt;
/* 4. EXEC SQL DESCRIBLE BIND VARIABLES FOR statement_name INTO         */
/*    input_descriptor                                                  */
EXEC SQL DESCRIBE BIND VARIABLES FOR demo_stmt INTO input_descriptor;
GetSQLDA(input_descriptor, 0, SQLDA_NUM_OF_HV, &nHv);
if (SQLCODE == SQL_ERROR) error_handle();
printf("There are %d returned input host variables: \n\n", nHv);
/* 5. set length of input host variables, set dataType of input host    */
/*   variables,allocate storage for value of input host variables, set */
/*   value of input host variables, set value of input indicates, set  */
/*    type, len, allocate buffer, value                                 */
for (i = 1; i <= nHv; i++)
```

```
    {
     pData = malloc(STRING_LEN);
     SetSQLDA(input_descriptor, i, SQLDA_DATABUF, pData);
     if (SQLCODE == SQL_ERROR) error_handle();
     SetSQLDA(input_descriptor, i, SQLDA_DATABUF_TYPE, SQL_C_CHAR);
     if (SQLCODE == SQL_ERROR) error_handle();
     strcpy(pData, 'dynamic ESQL/C example');
     datalen = strlen(pData);
     SetSQLDA(input_descriptor, i, SQLDA_INDICATOR, datalen);
     if (SQLCODE == SQL_ERROR) error_handle();
    }
/* 6. EXEC SQL OPEN cursor_name USING DESCRIPTOR input_descriptor       */
EXEC SQL OPEN demo_cursor USING DESCRIPTOR input_descriptor;
/* 7. EXEC SQL DESCRIBLE SELECT LIST FOR statement_name INTO            */
/*    select_descriptor                                                 */
EXEC SQL DESCRIBE SELECT LIST FOR demo_stmt INTO select_descriptor;
GetSQLDA(select_descriptor, 0, SQLDA_NUM_OF_HV, &nCol);
if (SQLCODE == SQL_ERROR) goto error_label;
printf("There are %d returned columns: \n\n", nCol);
for (i = 1; i <= nCol; i++)
    {
     printf("column %d : \n");
     GetSQLDA(select_descriptor, i, SQLDA_COLNAME_LEN, &colNameLen);
     if (SQLCODE == SQL_ERROR) error_handle();
     GetSQLDA(select_descriptor, i, SQLDA_COLNAME, &pColName);
     if (SQLCODE == SQL_ERROR) error_handle();
     GetSQLDA(select_descriptor, i, SQLDA_COLTYPE, &colType);
     if (SQLCODE == SQL_ERROR) error_handle();
     GetSQLDA(select_descriptor, i, SQLDA_COLLEN, &colLen);
     if (SQLCODE == SQL_ERROR) error_handle();
     GetSQLDA(select_descriptor, i, SQLDA_COLPREC, &colPrec);
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
  if  (SQLCODE == SQL_ERROR) error_handing(); )
     GetSQLDA(select_descriptor, i, SQLDA_COLSCALE, &colScale);
     if (SQLCODE == SQL_ERROR) error_handle();
     GetSQLDA(select_descriptor, i, SQLDA_COLNULLABLE, &colNullable);
     if (SQLCODE == SQL_ERROR) error_handle();
     printf(" column name length = %ld \n", colNameLen);
     printf(" column name = %s \n", pColName);
     printf(" column type = %ld \n", colType);
     printf(" column length = %ld \n", colLen);
     printf(" column precision = %ld \n", colPrec);
     printf(" column scale = %ld \n", colScale);
     printf(" column nullable = %ld \n", colNullable);
    }
/* 8. set length of output host variables, set dataType of output host   */
/* variables, allocate storage for value of output host variable         */
for (i = 1; i <= nCol; i++)
    {
     GetSQLDA(select_descriptor, i, SQLDA_COLTYPE, &colType);
     if (SQLCODE == SQL_ERROR) error_handle();
     GetSQLDA(select_descriptor, i, SQLDA_COLLEN, &colLen);
     if (SQLCODE == SQL_ERROR) error_handle();
     switch (colType)
         {
          case SQL_CHAR:
               pData = malloc(colLen+1);
               SetSQLDA(select_descriptor, i, SQLDA_DATABUF, pData);
               if (SQLCODE == SQL_ERROR) error_handle();
               SetSQLDA(select_descriptor, i, SQLDA_DATABUF_LEN,
colLen+1);
               /* '+1' for null terminate                                */
               if (SQLCODE == SQL_ERROR) error_handle();
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
SetSQLDA(select_descriptor,i,SQLDA_DATABUF_TYPE,SQL_C_CHAR);
               if (SQLCODE == SQL_ERROR) error_handle();
               break;
          case SQL_INTEGER:
               pData = malloc(4);
               SetSQLDA(select_descriptor, i, SQLDA_DATABUF, pData);
               if (SQLCODE == SQL_ERROR) error_handle();
               SetSQLDA(select_descriptor, i, SQLDA_DATABUF_LEN, 4);
               if (SQLCODE == SQL_ERROR) error_handle();
               SetSQLDA(select_descriptor, i, SQLDA_DATABUF_TYPE,
SQL_C_LONG);
               if (SQLCODE == SQL_ERROR) error_handle();
               break;
         }
     }
/* 9. EXEC SQL FETCH cursor_name USING select_descriptor               */
while (1)
    {
     EXEC SQL FETCH demo_cursor USING select_descriptor;
if (SQLCODE != SQL_SUCCESS && SQLCODE != SQL_SUCCESS_WITH_INFO)
        break;
     for (i = 1; i <= nCol; i++)
         {
          GetSQLDA(select_descriptor, i, SQLDA_DATABUF, &pData);
          if (SQLCODE == SQL_ERROR) error_handle();
          GetSQLDA(select_descriptor, i, SQLDA_DATABUF_TYPE, &dateType);
          if (SQLCODE == SQL_ERROR) error_handle();
          switch (dataType)
              {
               case SQL_C_CHAR:
                    printf(" %s ", pData);
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
                    break;
                case SQL_C_LONG:
                    printf(" %ld ",*(long *)pData);
                    break;
              }
         }
    }   /* end of while loop */

/* 10. EXEC SQL CLOSE cursor_name                                        */
EXEC SQL CLOSE demo_cursor;

/* 11. free user buffer                                                  */
for (i = 1; i <= nHv; i++)
    {
     GetSQLDA(input_descriptor, i, SQLDA_DATABUF, &pData);
     if (SQLCODE == SQL_ERROR) error_handle();
     free(pData);
    }
for (i = 1; i <= nCol; i++)
    {
     GetSQLDA(select_descriptor, i, SQLDA_DATABUF, &pData);
     if (SQLCODE == SQL_ERROR) error_handle();
     free(pData);
    }
/* 12. De_allocate descriptor                                            */
free_descriptor_storage(input_descriptor);
if (SQLCODE == SQL_ERROR) error_handle();
free_descriptor_storage(select_descriptor);
if (SQLCODE == SQL_ERROR) error_handle();
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

➲ 例 2

単一FETCH文で複数の行を回収するためにSQLDAを使う：

```
#include "testdb.h"
/****************************************************************/
 *   The example will show how to write a dynamic ESQL program using SQLDA
 *   and to query with an unknown number of input and output host variables
 *
 *  Table customer in database STORE is used in the following demo example.
 *    Schema of table customer is (cid int, lname(32), fname(32)).
 ****************************************************************/
#define MAX_ENTRY   10
#define STRING_LEN  128
#define CONDITION   103
#define MAX_FETCH_ROWS   10
#define NUM_OF_FETCH_ROWS 5
/****************************************************************
*  include header
****************************************************************/
   EXEC SQL INCLUDE SQLCA;
   EXEC SQL INCLUDE SQLDA;
   EXEC SQL INCLUDE DBENVCA;

main()
{

   /****************************************************************
        *  error handling
*******************************************************/
   EXEC SQL WHENEVER SQLERROR GOTO error_label;

   /****************************************************************
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
      *  declare SQL host variables
*******************************************************/
   EXEC SQL BEGIN DECLARE SECTION;
   varchar cuser[8], passwd[8];
   varchar demoquery[64];
   varchar democursor[8];
   char dbname[18];                  /* char type is fix length string */
   int      nFetchRows = NUM_OF_FETCH_ROWS;
   EXEC SQL END DECLARE SECTION;

   /*****************************************************************
      *  declare variables
      *****************************************************/
   int        i, rc = 0;
   char       fgConn = 0;
   long       datalen, colLen;
   long       nHv=0, nCol=0;
   char      *input_descriptor;
   char      *select_descriptor;
   char      *pData, *pColName;
   int       count;
     /****************************************************************
      *  connect to database
*******************************************************/
   strcpy(dbname, TEST_DBNAME);       /* get db,user,password info */
   strcpy(cuser.arr, "SYSADM");
   cuser.len = strlen(cuser.arr);
   strcpy(passwd.arr, "");
   passwd.len = strlen(passwd.arr);
     EXEC SQL CONNECT TO :dbname :cuser :passwd;
   fgConn = 1;
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
   EXEC SQL SET AUTOCOMMIT OFF;
      /*****************************************************************
       * step1. allocate SQLDA storage for input and output host variables
*******************************************************/
   rc = allocate_descriptor_storage(MAX_ENTRY, &input_descriptor);
   if (rc < 0) goto error_label;
   rc = allocate_descriptor_storage(MAX_ENTRY, &select_descriptor);
   if (rc < 0) goto error_label;
      /*****************************************************************
       *  clear all tuples of table customer
*******************************************************/
   EXEC SQL DELETE FROM customer;
   /*****************************************************************
       *  insert data for test
*******************************************************/
   EXEC SQL INSERT INTO customer VALUES(1000, 'aaa', 'test1');
   EXEC SQL INSERT INTO customer VALUES(2000, 'bbb', 'test2');
   EXEC SQL INSERT INTO customer VALUES(3000, 'ccc', 'test3');
   EXEC SQL INSERT INTO customer VALUES(4000, 'ddd', 'test4');
   EXEC SQL INSERT INTO customer VALUES(5000, 'eee', 'test5');
   EXEC SQL INSERT INTO customer VALUES(6000, 'fff', 'test6');
   EXEC SQL INSERT INTO customer VALUES(7000, 'ggg', 'test7');
   EXEC SQL INSERT INTO customer VALUES(8000, 'hhh', 'test8');
   EXEC SQL INSERT INTO customer VALUES(9000, 'iii', 'test9');
   EXEC SQL INSERT INTO customer VALUES(10000, 'jjj', 'test10');
   EXEC SQL INSERT INTO customer VALUES(11000);
   exec sql commit work;

   /*****************************************************************
       * specify the query demoquery with host variable, you can also ask
       *  user to input the query string from the terminal at run time
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
****************************************************/
   sprintf(demoquery.arr, "%s %s",
                  "SELECT cid, lname, memo, memo2 FROM customer",
                  "WHERE cid > ?");
   demoquery.len = strlen(demoquery.arr);
      /****************************************************************
       *  step2. prepare the query
****************************************************/
   EXEC SQL PREPARE demo_stmt FROM :demoquery;
      /****************************************************************
       *  step3. declare cousor for the query
****************************************************/
   EXEC SQL DECLARE democursor SCROLL CURSOR FOR demo_stmt;
      /****************************************************************
       *  step4. describle input host variables information (including
number,
       *      type, length, ...) and put them into SQLDA input_descriptor
       *         for reference
****************************************************/
   EXEC SQL DESCRIBE BIND VARIABLES FOR demo_stmt INTO input_descriptor;

   rc = GetSQLDA(input_descriptor, 0, SQLDA_NUM_OF_HV, &nHv);
   if (rc < 0) goto error_label;
   printf("There are %d input host variables in the query \n\n", nHv);
      /****************************************************************
       *  step5. set data type and buffer length of input buffer , and
       *         allocate it. Then, set these values.
       *       (note: assume the input data is character string less than

       *                128 bytes.)
****************************************************/
   for (i = 1; i <= nHv; i++)
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
        {
        pData = MALLOC(STRING_LEN);
        rc = SetSQLDA(input_descriptor, i, SQLDA_DATABUF, pData);
        if (rc < 0) goto error_label;
        rc = SetSQLDA(input_descriptor, i, SQLDA_DATABUF_TYPE,
SQL_C_CHAR);
        if (rc < 0) goto error_label;
        sprintf(pData, "%d", CONDITION);
        datalen = strlen(pData);
        rc = SetSQLDA(input_descriptor, i, SQLDA_INDICATOR, datalen);
        if (rc < 0) goto error_label;
        }
   /****************************************************************
       *  step6. open cursor using data and information of
       *         SQLDA input_descriptor
*******************************************************/
   EXEC SQL OPEN democursor USING DESCRIPTOR input_descriptor;
   /****************************************************************
       *  step7. describle output host variables information
       *         (including projection number, column name, column type,
       *     column length, ....) and put them into SQLDA select_descriptor
       *         for reference
*******************************************************/
   EXEC SQL DESCRIBE SELECT LIST FOR demo_stmt INTO select_descriptor;

   rc = GetSQLDA(select_descriptor, 0, SQLDA_NUM_OF_HV, &nCol);
   if (rc < 0) goto error_label;
   printf("There are %d columns returned \n\n", nCol);
   printColInfo(select_descriptor);
   /*********************************************************
       * step8. bind output host variables buffer information (including
buffer
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
       *       type, buffer length) and allocate buffers for each output
host
       *         variables.
*******************************************************/
   bindBufInfo(select_descriptor);
   /**************************************************************
       * step9. fetch data by cursor and print out the results, including:
       *         column name and data
*******************************************************/
   for (i = 1; i <= nCol; i++)
         {
         rc = GetSQLDA(select_descriptor, i, SQLDA_COLNAME, &pColName);
         if (rc < 0) goto error_label;
         printf("   %s    ", pColName);
         }
   printf("\n");
      for (i = 1; i <= nCol; i++)
         printf("========= ");
   printf("\n");

       do
               {
               EXEC SQL FETCH :nFetchRows ROWS democursor USING
select_descriptor;
               if (sqlca.sqlcode == SQL_NO_DATA_FOUND)
                       break;
#if 0
               EXEC SQL GET BLOB column 2 FOR demo_stmt;
               EXEC SQL GET BLOB column 3 FOR demo_stmt;
#endif
               /* print out result by buffer data type */
               printResult(select_descriptor);
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
                printf("\n");
                } while(sqlca.sqlcode == SQL_SUCCESS ||
                            sqlca.sqlcode == SQL_SUCCESS_WITH_INFO);
        printf("\n");
   /****************************************************************
        *  step10. close cursor
*******************************************************/
        EXEC SQL CLOSE democursor;
error_label:
   /****************************************************************
        *  print out error information
*******************************************************/
        if (sqlca.sqlcode)
           {
           printf("SQLSTATE: %ld \n", sqlca.sqlcode);
           printf("error code: %ld \n", sqlca.sqlerrd[0]);
           printf("error message: %s \n", sqlca.sqlerrmc);
           }
  /*********************************************************
        *  step11. deallocate SQLDA storage
*******************************************************/
        for (i = 1; i <= nHv; i++)
           {
           rc = GetSQLDA(input_descriptor, i, SQLDA_DATABUF, &pData);
           FREE(pData);
           }
        for (i = 1; i <= nCol; i++)
           {
           rc = GetSQLDA(select_descriptor, i, SQLDA_DATABUF, &pData);
           FREE(pData);
           }
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
        if (input_descriptor)
           {
           rc = free_descriptor_storage(input_descriptor);
           if (rc < 0)
                  {
                  printf("SQLSTATE: %ld \n", sqlca.sqlcode);
                  printf("error code: %ld \n", sqlca.sqlerrd[0]);
                  printf("error message: %s \n", sqlca.sqlerrmc);
                  }
           }
        if (select_descriptor)
           {
           rc = free_descriptor_storage(select_descriptor);
           if (rc < 0)
                  {
                  printf("SQLSTATE: %ld \n", sqlca.sqlcode);
                  printf("error code: %ld \n", sqlca.sqlerrd[0]);
                  printf("error message: %s \n", sqlca.sqlerrmc);
                  }
           }
/*************************************************************
    *  disconnect from database
    *************************************************************/
        EXEC SQL WHENEVER SQLERROR CONTINUE;
        if (fgConn)
           EXEC SQL DISCONNECT;
}
/*****************************************************************
 *  printColInfo() --
 *    print out column information from SQLDA by describle SELECT LIST
 ****************************************************************/
void printColInfo(char *desc)
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
{
   int i, rc=0;
   long nCol=0;
   char *pColName;
   long   colType, colPrec, colScale, colNullable;
   long   colLen, colNameLen;
   rc = GetSQLDA(desc, 0, SQLDA_NUM_OF_HV, &nCol);

   for (i = 1; i <= nCol; i++)
          {
          printf("column %ld information :\n", i);
          rc = GetSQLDA(desc, i, SQLDA_COLNAME_LEN, &colNameLen);
          rc = GetSQLDA(desc, i, SQLDA_COLNAME, &pColName);
          if (pColName != NULL)
                 {
                 printf("column name = %s \n", pColName);
                 printf("column name length = %d \n", colNameLen);
                 }

          rc = GetSQLDA(desc, i, SQLDA_COLTYPE, &colType);
          switch (colType)
                 {
                 case SQL_CHAR:
                    printf("column type = char \n");
                        break;
                 case SQL_DECIMAL:
                        printf("column type = decimal \n");
                        break;
                 case SQL_INTEGER:
                        printf("column type = integer \n");
                        break;
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
            case SQL_SMALLINT:
                   printf("column type = smallint \n");
                   break;
            case SQL_FLOAT:
            case SQL_REAL:
                   printf("column type = float \n");
                   break;
            case SQL_DOUBLE:
                   printf("column type = double \n");
                   break;
            case SQL_VARCHAR:
                   printf("column type = varchar \n");
                   break;
            case SQL_DATE:
                   printf("column type = date \n");
                   break;
            case SQL_TIME:
                   printf("column type = time \n");
                   break;
            case SQL_TIMESTAMP:
                   printf("column type = timestamp \n");
                   break;
            case SQL_LONGVARCHAR:
                   printf("column type = longvarchar \n");
                   break;
            case SQL_BINARY:
                   printf("column type = binary \n");
                   break;
            case SQL_LONGVARBINARY:
                   printf("column type = longvarbinary \n");
                   break;
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
                case SQL_FILE:
                       printf("column type = file \n");
                       break;
                default:
                       break;
                }

         rc = GetSQLDA(desc, i, SQLDA_COLLEN, &colLen);
         printf("column length = %ld \n", colLen);
         rc = GetSQLDA(desc, i, SQLDA_COLPREC, &colPrec);
         printf("column precision = %ld \n", colPrec);
         rc = GetSQLDA(desc, i, SQLDA_COLSCALE, &colScale);
         printf("column scale = %d \n", colScale);
         rc = GetSQLDA(desc, i, SQLDA_COLNULLABLE, &colNullable);
         if (colNullable == 0)
                printf("column is not nullable \n");
         else
                printf("column is nullable \n");
         printf("\n");
         }
}
/*****************************************************************
 *  bindBufInfo() --
 *   set output host variables buffer information (including buffer type,
 *   buffer length) and allocate buffers for each output host variables.
 *****************************************************************/
void bindBufInfo(char *desc)
{
   int  i, rc=0;
   char *pData;
   long   nCol=0, colType, dataType;
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
long   dataLen;
int  nFetch = MAX_FETCH_ROWS;
rc = SetSQLDA(desc, 0, SQLDA_MAX_FETCH_ROWS, nFetch);
rc = GetSQLDA(desc, 0, SQLDA_NUM_OF_HV, &nCol);
for (i = 1; i <= nCol; i++)
       {
       rc = GetSQLDA(desc, i, SQLDA_COLTYPE, &colType);
       switch (colType)
              {
              case SQL_CHAR:
              case SQL_VARCHAR:
              case SQL_LONGVARCHAR:
              case SQL_BINARY:
              case SQL_LONGVARBINARY:
              case SQL_FILE:
              case SQL_DECIMAL:
              case SQL_DATE:
              case SQL_TIME:
              case SQL_TIMESTAMP:
                 pData = MALLOC(STRING_LEN*nFetch);
                     dataLen = STRING_LEN-1; /* for null terminate */
                     dataType = SQL_C_CHAR;
                     break;
              case SQL_INTEGER:
                     pData = MALLOC(4*nFetch);
                     dataLen = 4;
                     dataType = SQL_C_LONG;
                     break;
              case SQL_SMALLINT:
                     pData = MALLOC(2*nFetch);
                     dataLen = 2;
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
                        dataType = SQL_C_SHORT;
                        break;
                case SQL_FLOAT:
                case SQL_REAL:
                        pData = MALLOC(4*nFetch);
                        dataLen = 4;
                        dataType = SQL_C_FLOAT;
                        break;
                case SQL_DOUBLE:
                        pData = MALLOC(8*nFetch);
                        dataLen = 8;
                        dataType = SQL_C_DOUBLE;
                        break;
                default:
                        break;
                }
          rc = SetSQLDA(desc, i, SQLDA_DATABUF, pData);
          rc = SetSQLDA(desc, i, SQLDA_DATABUF_LEN, dataLen);
          rc = SetSQLDA(desc, i, SQLDA_DATABUF_TYPE, dataType);
          }
}
/*****************************************************************
 *  printResult() --
 *    print out column data by their data type
*********************************************************/
void printResult(char *desc)
{
   int   i,j, rc=0;
   long   nCol=0, dataType[MAX_ENTRY+1];
   long   *ind[MAX_ENTRY+1];
   long   *pind;
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
    char *pData[MAX_ENTRY+1];
    int   retRows = pSQLCA->sqlerrd[3];
    int   dataLen[MAX_ENTRY+1];
    int   maxFetch;
    rc = GetSQLDA(desc, 0, SQLDA_NUM_OF_HV, &nCol);
    rc = GetSQLDA(desc, 0, SQLDA_MAX_FETCH_ROWS, &maxFetch);
    for (i = 1; i <= nCol; i++)
           {
       rc = GetSQLDA(desc, i, SQLDA_INDICATOR, &ind[i]);
           rc = GetSQLDA(desc, i, SQLDA_DATABUF_TYPE, &dataType[i]);
           rc = GetSQLDA(desc, i, SQLDA_DATABUF_LEN, &dataLen[i]);
           rc = GetSQLDA(desc, i, SQLDA_DATABUF, &pData[i]);
       }
    for (j = 0; j < retRows; j++)
       {
       for (i = 1; i <= nCol; i++)
          {
          if (*ind[i] == SQL_NULL_DATA)
             printf("NULL  ");
          else
             switch (dataType[i])
                {
                case SQL_C_CHAR:
                     printf(" %s ", pData[i]);
                     break;
                case SQL_C_LONG:
                     printf(" %l5d ", *(long *)pData[i]);
                     break;
                case SQL_C_SHORT:
                     printf(" %5d ", *(short *)pData[i]);
                     break;
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
                case SQL_C_FLOAT:
                     printf(" %f ", *(float *)pData[i]);
                     break;
                case SQL_C_DOUBLE:
                     printf(" %lf ", *(double *)pData[i]);
                     break;
                default:
                     break;
                }
          ind[i]++;
          pData[i] += dataLen[i];
          }
       printf("\n");
       }
}
```

SQLDAに入力ホスト変数の値を渡すために、以下のオプションをセットします。

- ホスト変数のために有効なデータ・バッファを割り当て、そのバッファのデータ型に応じてデータ・バッファに値を割り当てます。
- データバッファのポインタ、タイプ、長さ、インジケータをセットします。

注 *インジケータ値をセットしない場合、DBMasterは実際の入力データ長をデータ・バッファの長さに使用します。*

SQLDAから出力ホスト変数の値を取得するために、以下のオプションをセットします。

- ホスト変数のために有効なデータ・バッファを割り当てます。
- データ・バッファのポインタ、タイプ、長さをセットします。

©Copyright 1995-2015 Syscom Computer Engineering Co.

6.5 動的ESQL BLOBインターフェース

動的ESQLでは、PUT BLOBやGET BLOBメカニズムを使います。加えて、タイプ4の動的ESQLは、BLOBを*put/get*するために、静的ESQLと同じBLOBメカニズムを使うことができます。

ESQLには、メモリ・ストレージとファイル・ストレージの２つのスタイルのBLOBがあります。メモリに保存されているBLOBは、BLOBデータとして参照されます。ファイルに保存されているBLOBデータは、ファイル・オブジェクトです。BLOBデータとファイル・オブジェクトのPUT/GETの方法の詳細については、このセクションにあるタイプ4の動的ESQL (SQLDA)を参照して下さい。タイプ4の動的ESQLの本来のステップに加えて、BLOBインターフェースのために、さらに追加のステップが必要です。

ファイル・オブジェクトを保存する

ファイル・オブジェクトは、要素又はオブジェクト名で保存することができます。データベースにファイルの要素をデータとして保存すると、それを保存した後、外部ファイルはデータベースと関係が無くなります。データベースにファイル名をデータとして保存すると、ファイル・オブジェクトの要素は、外部ファイルに保存されたままです。

SetSQLDA() 関数のSQLDA_STORE_FILE_TYPEを使って、保管したファイル・オブジェクトのタイプを指定します。option_valueをESQL_STORE_FILE_CONTENTにセットすると、SQLDA_DATABUFで指定したファイルの内容が、データベースに保存されることを意味します。option_valueをESQL_STORE_FILE_NAMEにセットするとSQLDA_DATABUFで指定したファイル名が、データベースに保存されることを意味します。DBMasterでは、SQLDA_STORE_FILE_TYPEの初期設定値は、ESQL_STORE_FILE_CONTENTです。

ファイル・オブジェクトを保存するために、SQLDAに以下のオプションをセットします。

©Copyright 1995-2015 Syscom Computer Engineering Co.

- ファイル名のための文字データ・バッファを割り当てます。ファイル名の最大長はMAX_FNAME_LEN = 79です。データ・バッファにファイル名をセットします。
- SQLDAで、データ・バッファSQLDA_DATABUFのポインタをセットします。
- SQLDAで、データ・バッファ・タイプ SQLDA_DATABUF_TYPEをSQL_C_FILEにセットする。
- SQLDAで、ファイル・タイプ SQLDA_STORE_FILE_TYPEを、ESQL_STORE_FILE_CONTENT、又はESQL_STORE_FILE_NAMEにセットします。
- SQLDAで、インジケータSQLDA_INDICATORを実際のファイル名長にセットします。

➲ 例

customer table cid、cname、memoを使って、ファイル・オブジェクトmary_memo.foから“memo”カラムにデータをPUTする：

```
#define maxNumber 10
EXEC SQL BEGIN DECLARE SECTION;
varchar stmt_str[128];
EXEC SQL END DECLARE SECTION;
char *input_descriptor;
char *pData;
long  datalen;
allocate_descriptor_storage(maxNumber, &input_descriptor);
if (SQLCODE == SQL_ERROR) error_handle();
strcpy(stmt_str.arr, "INSERT INTO customer VALUES(1, 'mary', ?)");
stmt_str.len = strlen(stmt_str.arr);
EXEC SQL PREPARE demo_stmt FROM :stmt_str;
EXEC SQL DESCRIBE BIND VARIABLES FOR demo_stmt INTO input_descriptor;
pData = malloc(MAX_FNAME_LEN);
SetSQLDA(input_descriptor, 1, SQLDA_DATABUF, pData);
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
if (SQLCODE == SQL_ERROR) error_handle();
SetSQLDA(input_descriptor, 1, SQLDA_DATABUF_TYPE, SQL_C_FILE);
if (SQLCODE == SQL_ERROR) error_handle();
/* Suppose we will put 'file name' into database */
SetSQLDA(input_descriptor. 1, SQLDA_STORE_FILE_TYPE,
ESQL_STORE_FILE_NAME);
If (SQLCODE == SQL_SRROR) error_handing();)
/* Suppose mary's memo data is stored in file 'mary_memo.fo'          */
strcpy(pData, "mary_memo.fo");
datalen = strlen(pData);
SetSQLDA(input_descriptor, 1, SQLDA_INDICATOR, datalen);
if (SQLCODE == SQL_ERROR) error_handle();


EXEC SQL EXECUTE demo_stmt USING DESCRIPTOR input_descriptor;
/* suppose FreeSQLDA() can free SQLDA and all buffers allocated by
application                           */
FreeSQLDA(input_descriptor);
```

Fileオブジェクトを取得する

ファイル・オブジェクトを取得する時は、SQLDAで以下のオプションをセットして下さい。

- ファイル名 (ファイル名の最大長は、MAX_FNAME_LEN = 79)のために文字データ・バッファを割り当てます。データ・バッファ(ファイルは、カラム“memo”のデータを保存します)にファイル名をセットします。
- コマンドSQLDA_DATABUFを使って、SQLDAにデータ・バッファのポインタをセットします。
- SQLDAで、データ・バッファ・タイプSQLDA_DATABUF_TYPEをSQL_C_FILEにセットします。
- SQLDAで、データ・バッファ最大長SQLDA_DATABUF_LENをセットします。

©Copyright 1995-2015 Syscom Computer Engineering Co.

この例では、データはカラム“memo”から回収され、ファイル・オブジェクト(mary_memo.fo)に配置されます。

➲ 例

```
#define maxNumber 10
EXEC SQL BEGIN DECLARE SECTION;
varchar stmt_str[128];
EXEC SQL END DECLARE SECTION;
char *select_descriptor;
char *pData;
long  datalen;
allocate_descriptor_storage(maxNumber, &select_descriptor);
if (SQLCODE == SQL_ERROR) error_handle();
strcpy(stmt_str.arr, "SELECT memo FROM customer WHERE cname = 'mary'");
stmt_str.len = strlen(stmt_str.arr);
EXEC SQL PREPARE demo_stmt FROM :stmt_str;
EXEC SQL DECLARE demo_cursor CURSOR FOR demo_stmt;
EXEC SQL OPEN demo_cursor;
EXEC SQL DESCRIBE SELECT LIST FOR demo_stmt INTO select_descriptor;
pData = malloc(MAX_FNAME_LEN);
SetSQLDA(select_descriptor, 1, SQLDA_DATABUF, pData);
if (SQLCODE == SQL_ERROR) error_handle();
SetSQLDA(select_descriptor, 1, SQLDA_DATABUF_TYPE, SQL_C_FILE);
if (SQLCODE == SQL_ERROR) error_handle();
SetSQLDA(select_descriptor, 1, SQLDA_DATABUF_LEN, MAX_FNAME_LEN);
if (SQLCODE == SQL_ERROR) error_handle();
strcpy(pData, "mary_memo.fo");
EXEC SQL FETCH demo_cursor USING select_descriptor;
/* support PrintFile() can print out content of file                    */
printf("mary memo content - " \n);
PrintFile("mary_memo.fo");
EXEC SQL CLOSE demo_cursor;
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
/* support FreeSQLDA() can free SQLDA and all buffers allocated by
application                                              */
FreeSQLDA(select_descriptor);
```

BLOBデータを配置する

タイプ4の動的ESQLを使ってBLOBデータを配置する時、SQLDAに以下のオプションをセットします。

EXECUTEの前に、

- 入力データのためのデータ・バッファを割り当てます。
- SQLDA_DATABUFコマンドを使って、SQLDAにデータ・バッファのポインタをセットします。
- コマンドSQLDA_DATABUF_TYPEを使って、SQLDAにデータ・バッファ・タイプをセットします。
- ホスト変数がデータベースに配置することを指定するために、SQLDAに、BLOBフラグ、SQLDA_BLOB_FLAGをセットします。
- “BEGIN PUT BLOB”の前に、データ・バッファに入力データを入れます。
- SQLDAに、このデータの長さ SQLDA_PUT_DATA_LENを指定します。

この例では、データはデータbufferDataからカラム“memo”に配置されます。

➲ 例

```
#define maxbufsize 256
#define maxNumber 10
EXEC SQL BEGIN DECLARE SECTION;
varchar stmt_str[128];
EXEC SQL END DECLARE SECTION;
char *input_descriptor;
char *pData;
long  datalen;
boolean fgEnd = FALSE;
```

```
allocate_descriptor_storage(maxNumber, &input_descriptor);
if (SQLCODE == SQL_ERROR) error_handle();
strcpy(stmt_str.arr, "INSERT INTO customer VALUES(1, 'mary', ?)");
stmt_str.len = strlen(stmt_str.arr);
EXEC SQL PREPARE demo_stmt FROM :stmt_str;
EXEC SQL DESCRIBE BIND VARIABLES FOR demo_stmt INTO input_descriptor;
pData = malloc(maxbufsize);
SetSQLDA(input_descriptor, 1, SQLDA_DATABUF, pData);
if (SQLCODE == SQL_ERROR) error_handle();
SetSQLDA(input_descriptor, 1, SQLDA_DATABUF_TYPE, SQL_C_CHAR);
if (SQLCODE == SQL_ERROR) error_handle();
SetSQLDA(input_descriptor, 1, SQLDA_BLOB_FLAG, SQLDA_BLOB_ON);
if (SQLCODE == SQL_ERROR) error_handle();
EXEC SQL EXECUTE demo_stmt USING DESCRIPTOR input_descriptor;
EXEC SQL BEGIN PUT BLOB FOR demo_stmt;
/* support for mary's memo data is retrieved through function getInData(),
*/
/* if no more input data will be retrieved, fgEnd will be assigned to TRUE
*/
while(!fgEnd)
    {
     getInData(pData, maxbufsize, fgEnd);
     datalen = strlen(pData);
     SetSQLDA(input_descriptor, 1, SQLDA_PUT_DATA_LEN, datalen);
     if (SQLCODE == SQL_ERROR) error_handle();
     EXEC SQL PUT BLOB FOR demo_stmt;
    }


EXEC SQL END PUT BLOB FOR demo_stmt;
/* support FreeSQLDA() can free SQLDA and all buffers allocated by
application */
FreeSQLDA(input_descriptor);
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

BLOBデータを取得する

タイプ4の動的ESQLを使ってBLOBデータを回収する時、SQLDAに以下のオプションをセットします。

Get BLOBを使う前に、

- フェッチしたデータを保存するために、データ・バッファを割り当てます。
- SQLDAにデータ・バッファSQLDA_DATABUFのポインタをセットします。
- SQLDAにデータ・バッファ・タイプSQLDA_DATABUF_TYPEをセットします。
- SQLDAで、SQLDA_DATABUF_LENでデータ・バッファ最大長をセットします。

FETCHの前に、

- フェッチしたデータを保存するためにデータ・バッファを割り当てます。
- SQLDAにデータ・バッファのポインタをセットします。
- SQLDAにデータ・バッファ・タイプをセットします。
- SQLDAで、カラムがデータを取得することを指定するために、BLOBフラグをセットします。
- SQLDAにデータ・バッファの最大長をセットします。
- GET BLOBの前に、SQLDAでデータを取得するために、GET DATA長を指定します。

➲ **例**

データ・バッファ“pData”で、“memo”カラムからデータを取得する：

```
#define maxbufsize 256
#define maxNumber 10
```

©Copyright 1995-2015 Syscom Computer Engineering Co.

```
EXEC SQL BEGIN DECLARE SECTION;
varchar stmt_str[128];
EXEC SQL END DECLARE SECTION;
char *select_descriptor;
char *pData;
long  datalen;
boolean fgEnd = FALSE;
allocate_descriptor_storage(maxNumber, &select_descriptor);
if (SQLCODE == SQL_ERROR) error_handle();
strcpy(stmt_str.arr, "SELECT memo FROM customer WHERE cname = 'mary');
stmt_str.len = strlen(stmt_str.arr);
EXEC SQL PREPARE demo_stmt FROM :stmt_str;
EXEC SQL DECLARE demo_cursor CURSOR FOR demo_stmt;
EXEC SQL OPEN demo_cursor;
EXEC SQL DESCRIBE SELECT LIST FOR demo_stmt INTO select_descriptor;
SetSQLDA(select_descriptor, 1, SQLDA_BLOB_FLAG, SQLDA_BLOB_ON);
if (SQLCODE == SQL_ERROR) error_handle();
EXEC SQL FETCH demo_cursor USING select_descriptor;
pData = malloc(maxbufsize);
SetSQLDA(select_descriptor, 1, SQLDA_DATABUF, pData);
if (SQLCODE == SQL_ERROR) error_handle();
SetSQLDA(select_descriptor, 1, SQLDA_DATABUF_TYPE, SQL_C_CHAR);
if (SQLCODE == SQL_ERROR) error_handle();
SetSQLDA(select_descriptor, 1, SQLDA_DATABUF_LEN, maxbufsize);
if (SQLCODE == SQL_ERROR) error_handle();
printf("mary memo content - " \n);
do
        {
        EXEC SQL GET BLOB COLUMN 1 FOR demo_stmt;
If (SQLCODE ! = SQL_SUCCESS && SQLCODE != SQL_SUCCESS_WITH_INFO)
        Break;
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

```
        Printf(" %s ", pData);
        }
)
while (SQLCODE == SQL_SUCCESS || SQLCODE == SQL_SUCCESS_WITH_INFO)
    {
     EXEC SQL GET BLOB COLUMN 1 FOR demo_stmt;
     printf(" %s ", pData);
    }
EXEC SQL CLOSE demo_cursor;
/* support FreeSQLDA() can free SQLDA and all buffers allocated by
application */
FreeSQLDA(select_descriptor);
```

カラムからデータを取得するGET BLOBを使う前に、カラムのための総データ長を取得します。

- SQLDA_DATABUF_LENを0にセットします。
- カラムからBLOBをGETします。
- SQLDA_INDICATORの値(カラムの総データ長)を取得します。

©Copyright 1995-2015 Syscom Computer Engineering Co.

©Copyright 1995-2015 Syscom Computer Engineering Co.

7 プロジェクトとモジュール管理

ESQL/C ソースファイルをプリプロセスするために、DBMaster のESQL/C プリプロセッサ*dmppcc*を使う時、データベース名、ユーザー名、パスワードを与えます。ESQL/C ソースファイルのプリプロセスのために使用するユーザー名は、データベースに接続して、そのファイルをプリプロセスするための接続権限とリソース権限を有している必要があります。

➲ 例 1

構文：

```
dmppcc -d test_db -u db_user_id -p db_user_passwd esql_source.ec
```

ESQL/Cソースファイルを記述するとき、アプリケーション・プログラムをデータベースに接続できるようにするために、ソースファイルにCONNECT文を追加する必要があります。

➲ 例 2

CONNECT文：

```
EXEC SQL CONNECT TO dbname dbuser dbusr_passwd;
```

パラメータ	構文
Dbname	[identifier | :host_variable_identifier]

©Copyright 1995-2015 Syscom Computer Engineering Co.

パラメータ	構文
Dbuser	[identifier | :host_variable_identifier]
dbusr_passwd	[| identifier | :host_variable_identifier]

Dbname、dbuser、dbusr_passwdが、ESQL/Cアプリケーションの宣言セクションで宣言されている場合、識別子やcharデータ型のホスト変数になる可能性があります。dbusr_passwdは、ユーザーにパスワードが無いときは無視されます。ESQL/Cプログラムをプリプロセスし、ESQL/Cプログラムの実行時のためにデータベースと同じユーザー名、dbuserを使う必要はありません。例えば、"bank"プログラムを開発するために、ユーザー"acc_dba"を使ったバンキングのデータベースのように。

シェルコマンドで、以下のように入力して下さい。

➲ 例 3

データベースbankのユーザーacc_dba：

```
dmppcc -d bank -u acc_dba -p acc_dba connect.ec
```

➲ 例 4

ファイルconnect.ecの内容：

```
EXEC SQL INCLUDE DBENVCA;
EXEC SQL INCLUDE SQLCA;
connect()
    {
     /* use the clerk account to connect to the bank database           */
     EXEC SQL CONNECT TO bank clerk pd_clerk;
     if (SQLCODE) return SQLCODE;
     ...
     do_clerk_operation();
     ....
    }
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

7.1 プロジェクトとモジュール・オブジェクト

モジュールについて

ESQL/Cソースファイルをプリプロセスする時、DBMaster は実行計画を作成し、モジュールと呼ばれるコンポーネントにデータベースに関連する全情報を保存します。モジュール名を指定しない場合、初期設定のモジュール名は、ESQL/Cソースファイル名になります。

ユーザーが実行計画の古いバージョンにアクセスしようとしたとき、エラーが起こります。他のESQL開発者が同じESQLプログラムを再度プリプロセスする時、dmppccは以前保存した計画を削除し、新たに保存する計画を作成します。ESQLプログラムを実行する時にエラーメッセージ"the executable may be out of date, please rebuild it"を頻繁に受け取る場合、関連する.cファイルをコンパイルし、実行可能ファイルに再リンクします。但し、これは実行可能ファイルのバージョン・エラーですが、それを無視したいと考えるかもしれません。同じESQLアプリケーションで異なるESQLモジュールを開発している多くの開発者がいるような場所では、開発段階の時にこのエラーメッセージを無視するために"-n"を使うことができます。パフォーマンスを最適化、ESQLプロジェクト管理の問題を削減するするためには、アプリケーションのコーディングが終了した後に"-n"オプションを削除します。

プロジェクトについて

どの開発者のアプリケーション・システムにも、複数のESQL/Cモジュールを含むことができます。開発者が各モジュールを個々に管理(権限を供与/取り消し)すると、その負担は少なくありません。プロジェクトの目的は、開発者グループのESQL/Cモジュールを全て一緒にし、より簡単にアプリケーション・システムを組織することです。ESQL/C ソースファイルをプリプロセスした後、*dmppcc* はモジュール名に加えてプロジェクト名を保存します。プロジェクト名を指定しないと、初期設定のプロジェクト名はモジュール名と同じになります。

ESQLソースファイルをプリプロセスする時に、データベースにプロジェクトが存在しない場合、DBMasterは自動的にモジュールを保存するためにプロジェクトを作成します。プロジェクトが既に存在する場合、DBMasterは自動的に新規モジュールをプロジェクトに関連付けます。各モジュールは1つのプロジェクトとしか関連付けることができません。

ESQLのプロジェクトとモジュールについての情報を見る場合、SQL文でシステム表SYSPROJECTを参照することができます。

➲ 例

```
select * from SYSPROJECT;
```

カラム	意味
PROJECT_NAME	ESQLプロジェクト名
PROJECT_OWNER	ESQLプロジェクト所有者
MODULE_NAME	ESQLモジュール名
MODULE_OWNER	ESQLモジュール所有者
MODULE_SOURCE	モジュールのソースファイル名
REF_CMD	参照されるコマンド番号(dmppccが内部で使います)

ESQL実行計画の情報は、SYSCMDINFOシステム表に保存されます。このシステム表は、ESQL 実行計画を保存するだけでなく、ストアド・コマンドやストアド・プロシージャのためのその他の実行計画も保存します。各分野についての詳細情報は、ストアド・コマンドのマニュアルを参照して下さい。

カラム	意味
MODULENAME	ESQLモジュール名

©Copyright 1995-2015 Syscom Computer Engineering Co.

カラム	意味
CMDNAME	ESQLプリプロセッサが生成したコマンド名
CMDOWNER	ESQLプリプロセッサが生成したコマンド所有者
STATEMENT	元のSQL文
DATA	実行計画データ
DESCPARM	説明パラメータ
STATUS	有効、又は無効

注　*dmppccに–cs又は–nのオプションをセットした場合、DBMasterは実行計画、モジュール、プロジェクト、関連する表の所有者名を保存しません。異なるESQL/Cプリプロセッサのユーザーとプログラムの実行のユーザーによってエラーが引き起こされるのを防ぐために、オプション–cs又は–nをセットするときに、SQL文に各表のための所有者名を付けることをお勧めします。*

プロジェクトを削除する

プロジェクトがESQLモジュールとの関連を維持するために使用されるので、プロジェクトが必要無くなった時は、DROP PROJECT文を使って、そのプロジェクトの全ての関連する実行計画と情報を削除することができます。

➲ 例

DROP PROJECT構文：

```
DROP PROJECT project_name;
```

プロジェクトの所有者かデータベース管理者のみが、プロジェクトを削除、又は他のユーザーに実行権限を与える/取り消すことができます。

プロジェクトからモジュールを削除する

モジュールが必要無くなったとき、プロジェクトからモジュールを、またデータベースからこのモジュールに関連する全ストアド・コマンドを削除

©Copyright 1995-2015 Syscom Computer Engineering Co.

することができます。プロジェクトには1つのモジュールしかないので、モジュールが削除された場合、そのプロジェクトもデータベースから削除されます。

プロジェクトの所有者かデータベース管理者のみが、プロジェクトからモジュールを削除することができます。

プロジェクト/モジュールをロード/アンロードする

関連するプロジェクト又は特定のモジュールをアンロード/ロードするために、dm SQLでPROJECT/MODULEのLOAD/UNLOAD関数を使うことができます。

関連する構文は以下のリストのとおりです。"dmSQLユーザーガイド"でUNLOAD/LOAD構文の詳細を見ることができます。

➲ **例 1**

UNLOAD構文；

```
UNLOAD PROJECT FROM [ower_pattern.]project_pattern TO script_name

UNLOAD MODULE [ower_pattern.]module_pattern FROM PROJECT
[owner_name.]project_name TO script_name.
```

➲ **例 2**

UNLOADを使う：

```
dmSQL> UNLOAD PROJECT FROM project1 TO project.scr;

dmSQL> UNLOAD MODULE module1 FROM PROJECT project1 TO module.scr;
```

➲ **例 3**

LOAD構文：

```
LOAD PROJECT FROM script_name.

LOAD MODULE FROM script_name.
```


©Copyright 1995-2015 Syscom Computer Engineering Co.

➲ 例 4

LOADコマンドを使う：

```
dmSQL> LOAD PROJECT FROM project.scr;
dmSQL> LOAD MODULE FROM module.scr;
```

注　*UNLOAD/LOAD関数は、dmSQLでのみ使用することができます。*

プロジェクトのための権限を与える/取り消す

他のアプリケーション・システムのユーザーに、プロジェクトの実行権限を与える/取り消すことができます。

GRANT/REVOKEのSQL構文：

```
GRANT EXECUTE ON PROJECT project_name TO auth_user_list;
REVOKE EXECUTE ON PROJECT project_name FROM auth_user_list;
```

ユーザーが、その実行権限がないプロジェクトを実行しようとする時、タランタイム時にエラーが戻ります。そのプロジェクトは開発者がデータベースでモジュールをグループ化/管理するためのものなので、アプリケーション・システムには異なるプロジェクトからモジュールへの多くのリンクがある可能性があります。この場合、アプリケーションのユーザーは、実行権限がある部分についてのみ実行することができます。

ESQLに関連する権限情報は、SYSAUTHEXEシステム表に保存されています。権限のあるユーザーをチェックするために、表を調べることができます。

カラム	意味
OBJNAME	プロジェクト名
OWNER	プロジェクト所有者
OBJTYPE	PROJECT、又はSTORE COMMAND、或いはSTORE PROCEDURE
GRANTEE	権限を持つユーザー

©Copyright 1995-2015 Syscom Computer Engineering Co.

©Copyright 1995-2015 Syscom Computer Engineering Co.