ACCELL/SQL®

Product Enhancements

Release 8.2

目次

マルチデータベース コネクション 1

以前のバージョンでは、ACCELL/SQL のアプリケーションは 1 つのデータベースコネクションのみを使用するという制限事項がありました。USING CONNECTION 句の追加により、利用可能なデータベースコネクションにアクセスするための ACCELL/4GL 文が可能となって、その制限事項はなくなりました。利用可能なコネクションは、別の DCM(Data Connection Management) ファイル内に定義しておきます。

アプリケーション内でのデータベースコネクションの管理をサポートするために、ACCELL/4GL 文 OPEN CONNECTION と CLOSE CONNECTION が追加されました。これにより、データベースコネクションのユーザ名 / パスワードを、実行時に引き渡すことが可能となりました。アプリケーション内でログインスクリーンを表示することができます。

次のセクションに記述される変更を行ない、ACCELL/SQL Release 8.1 マニュアルを更新してください。

ACCELL/SQL: RDBMS Integration

2 章 ("Configuring Application Environments.") の最後に下記の新しい章を追加します。

異なるデータベース接続の設定

アプリケーションは、複数のデータベースにアクセスすることが可能です。このアクセスを容易にする最も簡単な方法は、データベース Connection Map(DCM) でアプリケーションで使用する全ての可能なコネクションを設定し、AMGR **-dcmfile** 引数を指定するか、$DCMFILE コンフィギュレーション変数によって DCM ファイルパスを指定します。

DCM ファイルは、論理的コネクション名を実際のデータベースコネクションにマップします。フォームスクリプトの ACCELL/4GL 文では、論理名を使用します。

DCM ファイルのフォーマットについての詳細は、6 ページ「DCM ファイルのセットアップ」を参照してください。

116 ページの「Calling an INFORMIX Stored Procedure from a Form Script」の項に下記の段落を追加します。

フォームでオープンするデータベースコネクションの数に関係なく、ストアドプロシージャは常にデフォルトのデータベースコネクションの下で実行されます。デフォルトのコネクションは、DCM ファイルで設定されます。

ACCELL/SQL: Script and Function Reference

3 章の 「ACCELL/4GL Syntax Descriptions」の以下の構文に、下記の新しい句を追加します。

ALTER SCHEMA
ALTER TABLE
CREATE SCHEMA
CREATE TABLE
CREATE VIEW
DELETE
DROP SCHEMA
DROP TABLE
DROP VIEW
GRANT
INSERT INTO
REVOKE
SELECT
SLOCK
UNLOCK
UPDATE
XLOCK

各構文の 'Syntax' セクションに最初のエレメントとして次の句を追加します。

[**USING CONNECTION** *connection*]

例 :

[**USING CONNECTION** *connection*]
ALTER SCHEMA [*schema_name*] *DDL_statements* ;

各構文の 'Argument' セクションに以下を追加します。

connection	DEFAULT キーワード、もしくはデータベースコネクション名を指定する文字列式の例 : USING CONNECTION DEFAULT あるいは USING CONNECTION 'emps' DEFAULT キーワードは、この文で使用するデフォルトのコネクションを指定します。文字列式は、データベースコネクションマップ (DCM) ファイル中のエントリと一致していなければなりません。

各構文の 'Description' セクションに以下を追加します。

データベースコネクション名を指定するための **USING CONNECTION** 句は、オプションでありこの文に対してのみ有効です。コネクションが指定されない場合、デフォルトのコネクションが使われます。デフォルトのコネクションは、DCM ファイルで設定されます。

3 章の「ACCELL/4GL Syntax Descriptions」のアルファベット順の箇所に、以下の新しい構文を追加します。

CLOSE CONNECTION

4GL 文 : close database connection

構文

CLOSE CONNECTION *connection_name*

引数

connection_name
クローズするデータベースコネクションインスタンス名を含む文字列式を指定します。

解説

CLOSE CONNECTION 構文は、オープンしたデータベースコネクションをクローズします。コネクションがクローズするとき、コネクションインスタンスと関連するアクティブなトランザクションはコミットされません。

オープンされたままの状態のコネクションは、アプリケーションが終了するとクローズされます。

関連情報

OPEN CONNECTION

OPEN CONNECTION

4GL 文: open a database connection

構文

OPEN CONNECTION *connection_name* [**DBUSER IS** *dbuser*, **DBPASSWORD IS** *dbpwd*]

引数

connection_name
オープンするコネクションインスタンスの名前を含む文字列式を指定します。

dbuser
データベースユーザの名前を含む文字列式を指定します。

dbpwd
関連するパスワードの名前を含む文字列式を指定します。

解説

OPEN CONNECTION 文は、実行時にデータベースコネクションをオープンします。データベースコネクション名は、*connection_name* 引数によって指定されます。このコネクション名は、アプリケーションで有効な DCM ファイルで定義されなければなりません。

デフォルトでは、DML、DDL 文でデータベースコネクションが必要な場合には、常にデータベースコネクションはオープンされます。OPEN CONNECTION 文により、データベースコネクション名を決定する手段として、実行時にユーザよりログイン情報を収集することが可能です。

データベースコネクションは、CLOSE CONNECTION 文によりクローズされます。

例

この例は、**s_connection** 変数により指定した名前のコネクションインスタンスを、提供されたユーザ名とパスワードを使ってオープンします。

```
OPEN CONNECTION $s_connection DBUSER IS $user, DBPASSWORD
IS $pwd;
```

関連情報

CLOSE CONNECTION

.dcm ファイルのフォーマットについての詳細は、6 ページ「DCM ファイルのセットアップ」を参照してください。

以下の新しい予約語を、付録 A の予約語のリストに追加します。

CONNECTION

CONNECTION_NAME

以下の項目を、「System Functions and Variables」の check_dirty, db_type,dbms_status, l_allow_interrupt, sp_compute, sp_return, sp_select, sql_code, sql_errmsg,sql_state, status, tx_mode, ualcdb, ualcdbp, uacllda, uacltx システム関数の解説セクションに追加します。

フォームでオープンされたデータベースコネクションの数に関係なく、これらのシステム関数は、常にデフォルトのデータベースコネクションで実行されます。デフォルトのデータベースコネクションは、DCM ファイルにおいて設定されます。

ACCELL/SQL: Managing an Application

1 章「Configuring Application Environments」の最後に、下記の新しい章を追加します。

DCM ファイルのセットアップ

DCM ファイルは、論理コネクション名と実際のデータベースコネクションとの関連付けを行なっています。論理名は、フォームスクリプト中の DML, DDL や各 4GL 文内で USING CONNECTION 句を使って指定します。実際のデータベースコネクションは、コンフィグレーション変数の設定の集まりとして定義しておきます。

DCM ファイルは、1 つの ASCII ファイルで標準のテキストエディタで作成、管理できます。コネクションの定義には、コネクション名を与え必要なデータベースの外部プリファレンスを以下の構文でセットします。

[*connection* **]**
variable = value [; comment]
[variable = value [; comment]] . . .
[default] [; comment]

CONNECTION = *connection*
[[*connection*] [; comment]]
CONNECTION = *connection*

引数の定義 :

[*connection*]
データベースコネクションのヘッダで括弧 ([と]) が必要です。connection は、データベースコネクション名を表す文字列です。

データベースコネクションは、DCM ファイル内ではユニークにする必要があります。コネクション名は、大文字小文字の区別はしませんので、大文字小文字を使ってコネクション名を区別することはできません。

[default]
デフォルトとなるコネクションセクションのヘッダです。デフォルトコネクションは、*connection* セクションで指定しておきます。コネクションは、DCM ファイル内で前もって定義しておく必要があります。

variable
DCM 変数名です。有効な変数名と値については、Summary を参照してください。

value
変数への有効な値です。コンフィグレーション変数の値を参照する場合には、＄DBPASSWORD のようにプレフィックスにドル符号 ($) をつけて指定します。

comment
オプションのコメントです。

CONNECTION
コネクションを別名で指定する場合のキーワードです。CONNECTION 変数は、[default] フォーマットで使用しなければなりません。また、CONNECTION は [connection] フォーマットでも使い、いくつかのコネクション名の間でデータベースコネクション情報を共有するときに使います。セクションに CONNECTION 変数を設定したときは、そのセクションには他の変数をセットできません。

データベースコネクション名は任意の順序に設定できますが、別名に参照されるセクションは前もって指定します。CONNECTION 変数を含むセクションは DCM ファイルの最後に入れておき、コネクションセクションを指定します。それぞれの変数 = 値の指定は新しい行にします。変数名は大文字小文字を区別しません。

値は大文字小文字を区別します。値は 1 行で指定し、複数行に記述できません。値の終わりは、スペースにより指定します。値の中にスペースを入れるには、ダブルクォテーション (") あるいはシングルクォテーション (') で囲む必要があります。値にスペースを含まない場合、クォテーションマークはオプションです。

コメントはセミコロン (;) を記述し、行のその後はコメントになります。コメントは変数のセットと同じ行に記述できますが、コメントを変数から区別するために変数との間にはスペースを入れておく必要があります。(例えば、ディレクトリパスにセミコロンを入れる場合もあります) 読みやすくするために、DCM ファイルの中に空白行を入れることもできます。

アプリケーションの実行時、アプリケーションで使うためにコネクション定義を行なっている DCM ファイルを指定できます。

以下のテーブルリストは、DCM ファイルに指定できるコンフィグレーション変数です。それぞれの DBMS には、固有のコンフィグレーション変数があります。

要約　　DCM 変数とサポートしている DBMS

DBTYPE 値	コンフィグレーション変数	備考
INFORMIX	DBNAME	INFORMIX SE では、DBPATH と SQLEXEC は OS のコマンドレベルでセットしておきます。
ORACLE	DBPASSWORD DBSERVER DBUSER	
SYBASE	DBNAME DBPASSWORD DSQUERY DBUSER	
U2000	DBNAME DBPASSWORD DBPATH DBSCHEMA DBHOST DBUSER	ユーザ名とパスワードは別々に指定し、user_name/password のようには指定できません。パスワードは暗号化して DCM ファイルに入れます。

DCM ファイルフォーマットの例 :

```
[VIDEO1]
DBTYPE=ORACLE
DBSERVER=tns:c_support
DBUSER=class1
DBPASSWORD=class1

[pits]
DBTYPE=U2000
DBPATH=/dbs1/pitsdb
DBNAME=file.db
DBHOST=quarry
DBUSER=bxf/ZxWPPKxd0
;SCHEMA=cts
SCHEMA=pits

[test]
DBTYPE=U2000
DBPATH=$DON_DBPATH
DBNAME=$DON_DBNAME
DBHOST=$DON_DBHOST
DBUSER=$DON_DBUSER

[default]
CONNECTION=pits
```

DCM ファイルには、3 つのデータベースコネクションの定義、**video1**、**pits**、**test** があります。デフォルトコネクション名は **pits** です。

ACCELL/SQL: Configuration Variable and Utility Reference

以下の新しい項目を、53 ページの「DBTYPE configuration variable description」に追加します。

マルチデータベースをアクセスしている場合、以下の場合でのみ DBTYPE は DBMS のタイプを示しています。

- デフォルトのデータベースコネクションが、コンフィグレーション変数 (例えば DSQUERY、DBNAME など) で指定されている。
- DBTYPE が、DCM ファイルのデータベースコネクションセクションに指定されていない。

以下の新しいコンフィグレーション変数は、1 章 「Configuration Variable Reference」に追加します。

DCMFILE

DCMFILE コンフィグレーション変数は、データベースコネクションマップ (DCM) ファイル名前を指定します。DCM ファイルには、アプリケーションで使用するデータベースコネクションのリストを含みます。DCM ファイルは、3 文字の拡張子 .dcm を持ちます。

要約 *DCMFILE コンフィグレーション変数*

有効値	有効なパス名
依存関係	使用している OS にとって正しいディレクトリパスのフォーマットを指定
デフォルト値	$DCMFILE
例	B シェル : DCMFILE= $HOME/apps/connect.dcm

DCM ファイルについての詳細は、6 ページの「DCM ファイルのセットアップ」を参照してください。

HDATYPE

HDATYPE コンフィグレーション変数は、カスタムマネージャにリンクしているデータベースコネクションタイプのリストを指定します。異なるデータベースアクセスを行う HAD (Heterogeneous Database Access) 機能を使用する場合、amgr.ld を使ってカスタムマネージャを作成する前に、このプリファレンスを設定しておきます。

HDATYPE が設定されていない場合には、DBTYPE が使われます。

データベースタイプは、カンマあるいはスペースで区切ることができます。INFORMIX と ORACLE の両方を指定する場合、リストの中で INFORMIX を ORACLE の前にします。

要約 *HDATYPE コンフィグレーション変数*

有効値	“INFORMIX” は INFORMIX DBMS “ORACLE” は ORACLE DBMS “SYBASE” は SYBASE Adaptive Server DBMS ”U2000” は Unify DataServer DBMS
依存関係	使用している OS で正しいディレクトリパスのフォーマットを指定
デフォルト値	なし
例	HDATYPE= “U2000 SYBASE”; export HDATYPE
追加ヘルプ	amgr.ld.

以下の新しいパラメータは、133 ページの「AMGR utility」の構文の最後に追加します。

[- **dcmfile** *DCM_path_and_file_name*]

以下の引数の記述は、110-111 ページの引数のリストに追加します。

dcmfile *DCM_path_and_file_name*

アプリケーションに必要なデータベースコネクション情報を得るために、DCM ファイルのパスと名前を指定します。

動的 SQL 2

ACCELL/SQL は、動的 SQL をサポートするようになりました。動的 SQL を使用して、実行時に SQL 文を組み立てて実行できます。

以下のセクションに記載する変更を行ない、ACCELL/SQL リリース 8.1 のマニュアルを更新してください。

ACCELL/SQL: Writing Form Scripts

「ACCELL/SQL: Writing Form Scripts」の 8 章「Using Script Statements」の最後に、以下の新しいセクションを追加します。

動的 SQL 文

動的 SQL を使うことで、フォームスクリプト内の SQL 文のすべてあるいは一部を実行時に指定できます。動的 ACCELL/4GL SQL 文を使っていない場合は、アプリケーションの実行以前に SQL 文をスクリプトに書いておきます。テーブル、列、行の名前を前もって知っておく必要があります。

動的 SQL では、列、行、テーブルの名前をフォームスクリプト内の変数にしておくことができます。変数の値はユーザが入力したり、フォームスクリプトで編集したりできます。アプリケーション実行時、SQL 文は実際の変数の値で組み立てられ実行に移されます。

動的 SQL により、実行時 SQL 文はユーザ入力あるいはランタイムデータを基にフォームスクリプトで つくることができます。

動的 SQL 文の実行

動的 SQL 文の実行には、EXECUTE_SQL コマンドを使います。実行する文は、文字列式を用います。SQL 文に入力パラメータを使う場合、パラメータの値を指定するため USING 句を使います。SELECT 文のように文からの戻りがあるとき、EXECUTE_SQL コマンドは SET コマンドの内側に入れておきます。

動的 SQL を使うとき、ユーザ入力を受け取る変数をスクリプトに準備する必要があります。例えば、下記の SQL 文を動的に実行したいと仮定します。

```
UPDATE order_tbl SET price = 5;
```

次の例では、ユーザが入力した値で価格 (price) が更新されます。

```
FIELD field1
  ON FIELD
   INPUT
   EXECUTE_SQL 'UPDATE order_tbl SET price = ?' using
$field1;
```

アプリケーションの中で動的 SQL 文を 2 回以上使う場合には、次のセクションで記述するように動的な文を前もって準備しておく必要があります。

動的 SQL 文の準備

動的 SQL では SQL 文を定義してセーブしておくことで、後でスクリプト内で実行したり関数に渡して実行することができます。このことを SQL 文の準備と呼び、後程利用するために文を準備しておきます。

文を準備することの利点は、SQL 文の構文解析や最適化などのオーバーヘッドなしで複数回 SQL 文を実行できることです。

SQL 文を準備するため*ハンドル*を指定します。SQL ハンドルは、1 つの SQL 文を識別するものです。SQL ハンドルは、PREPARE_SQL 文を使ってつくります。ハンドルがつくられると、ハンドルを含む変数が有効範囲から外れたり、他の PREPARE_SQL 文で再利用したり、FREE_SQL 文を実行するまでそのままで残っています。

ハンドルがつくられると、ハンドルが示している SQL 文は、EXECUTE_SQL 文を使って実行します。また、参照するための SQL 文についての情報を、いくつかのハンドル属性に持っています。

注記 – 動的 SQL 文を 1 回だけ実行する場合、EXECUTE_SQL 文を使います。文を準備したり解放する必要はありません。準備した SQL 文を 2 回以上使うときには、PREPARE_SQL と EXECUTE_SQL 文を組み合わせて使用すべきでしょう。

次の例では、PREPARE_SQL 文を使って UPDATE 文を準備し、UPDATE 文は **update_hndl** という SQL ハンドルにストアします。スクリプト内の後で、EXECUTE_SQL 文が update_hndl に含まれている文を実行します。

```
BEFORE FORM
  PREPARE_SQL 'UPDATE order_tbl SET price = ?' into
   $update_hndl;
. . .

AFTER FIND
  EXECUTE_SQL $update_hndl USING $field1;
```

AFTER FIND セクションを実行するときにはいつでも *update_hndl* が実行されます。

SQL ハンドルの使用を終えたときには、**FREE_SQL** 文を使ってメモリリソースを解放します。以下の例では、**update_hndl** SQL ハンドルが利用していたリソースを解放しています。

```
FREE_SQL $update_hndl;
```

SQL 文属性

ACCELL/SQL には、SQL クエリとそれに関連した入出力パラメータに関するハンドル属性が用意されています。これらの属性は、DBMS に依存するもので常に利用できるものではありません。

準備した文の入力パラメータに関する情報を得るためには、以下の属性を利用できます。

INPUT_ACCELL_TYPE[ ]

INPUT_COUNT

INPUT_DB_COLUMN_TYPE[ ]

INPUT_DB_COLUMN_TYPE_CODE[ ]

INPUT_PRECISION[ ]

INPUT_SCALE[ ]

同様に DBMS が動的 SQL 文を受け取り、結果を返します。この結果を、出力パラメータと呼びます。ACCELL/SQL は、出力パラメータの表示のため、以下の属性を用意しています。

OUTPUT_ACCELL_TYPE[ ]

OUTPUT_COUNT

OUTPUT_DB_COLUMN_TYPE[ ]

OUTPUT_DB_COLUMN_TYPE_CODE[ ]

OUTPUT_NAME[ ]

OUTPUT_PRECISION[ ]

OUTPUT_SCALE[ ]

SYBASE Adaptive Server では、文の実行直後のみ、これらの属性値は有効となります。

別の属性を使ってパラメータから情報を取得できます。

- ACCELL タイプ

  入出力パラメータのデータタイプを表します。入力データタイプは、ACCELL/SQL データタイプで STRING あるいは NUMERIC などです。

- データベースの列タイプ

  DBMS の列タイプを表します。属性のデフォルト値は、DATE_TIME 以外 ACCELL_TYPE 属性の値と同じです。

- データベースの列タイプコード

  データベースコードを表示することで、DBMS の列タイプを表します。DBMS コードは、データベースにより異なります。データベースの列タイプコードの用途のひとつは、AMOUNT と FLOAT のデータタイプを識別することです。また、データベースの列が DATE_TIME データタイプか判断できます。データベースの列コードでは、OUTPUT_ACCELL_TYPE を DATE あるいは TIME に設定することができます。

- 精度

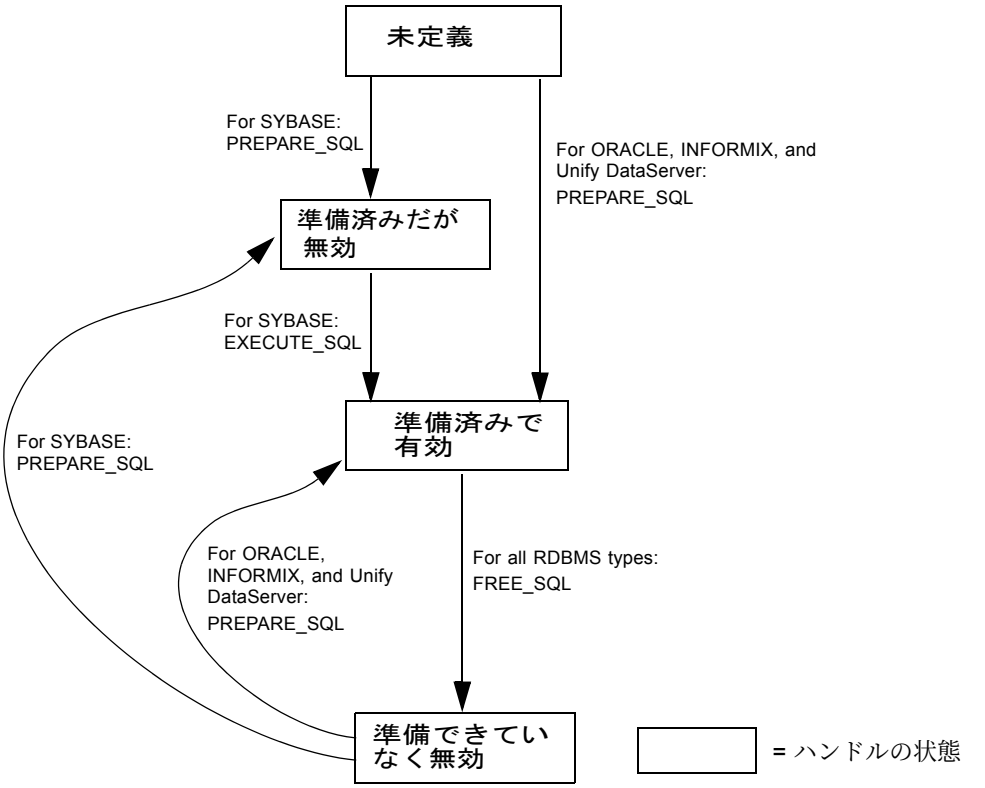
  AMOUNT または NUMERIC データベース列の有効桁数を指定します。STRING フィールドでは、データベース列の文字数です。FLOAT フィールドでは、ビット数の精度になります。

- 位取り

  NUMERIC フィールドでの小数点以下の桁数を指定します。

ハンドルの状態

ハンドルは、準備されている、あるいは有効な状態でのみ属性へのアクセスができます。ORACLE、INFORMIX、Unify DataServer では SQL ハンドルに文をストアし、PREPARE_SQL を使って準備を行なえば有効 (Valid) になります。SYBASE では、実行するまで有効となりません。下のイラストは、いくつかの 4GL 文がハンドルを準備済み (Prepared) と有効にする様子を表しています。



ハンドルの属性にアクセスする前に、IS_VALID と IS_PREPARED 属性を使ってハンドルの状態をテストすべきでしょう。この属性は、以下の組み合わせで値を返します。

要約 *ハンドルの有効性と準備のテスト*

IS_VALID	IS_PREPARED	State of SQL Handle
TRUE	TRUE	文は準備済みで有効になっていて実行することができます。属性には、文を準備したときに基づき有効な値が入っています。
FALSE	FALSE	無効。属性には有効な値が入っていません。
FALSE	TRUE	(SYBASE のみ) 準備済みで実行していない状態。属性には有効な値が入っていません。

ハンドル属性の使用例

以下の例では、多くの動的 SQL 文とハンドル属性を使用しています。スクリプトは、ユーザが **field1** フィールドに SQL **SELECT** 文を入力することを想定しています。ユーザが開発者定義コマンド **execute_dsql** のボタンをクリックすると、スクリプトは以下のステップを実行します。

- SELECT 文から返る行をストアする answer という配列を宣言します。
- **user_hndl** という SQL ハンドルに **field1** フィールドに入れた SQL 文を準備します。
- **field1** が有効な SQL 文かチェックします。**field1** の準備でエラーが起きたときには、ユーザは SQL 文の再入力か変更を求められます。
- ユーザに SQL 文を実行するか確認します。
- 列が DATE_TIME データタイプのときには、ユーザはデータベースからの値をセーブしておく適切な ACCELL/SQL データタイプを選択します。
- user_hndl を実行し SELECT 文の結果を answer 汎用配列変数にストアします。

```
#include "sscodes.h"

FORM form0
```

```
PUBLIC MATRIX answer[1 to UNKNOWN ESTIMATING 10, 1 TO
UNKNOWN ESTIMATING 10]

COMMAND execute_dsql
BEGIN
  PREPARE_SQL $field1 INTO $user_hndl;
  IF (status$() != SS_NORM) THEN
    DISPLAY 'Wrong syntax. Modify your statement again.'
FOR(SET $j to 1; $j <= $user_hndl:OUTPUT_COUNT;
   SET $j to $j + 1)
    BEGIN
    IF $user_hndl:OUTPUT_DB_COLUMN_TYPE[j] = 'DATE_TIME'
    THEN
    BEGIN
    DISPLAY NOTICE 'Would you like the output of' +
    $user_handl:OUTPUT_NAME[j] +
    'in DATE or TIME data type?'
    LABELS 'Date format'
    DEFAULT, 'Time format' RESULT INTO format_answer
    IF $format_answer = 0 THEN
      SET $user_hndl:OUTPUT_ACCELL_TYPE[j]
      TO 'DATE'
    ELSE
      SET $user_hndl:OUTPUT_ACCELL_TYPE[j] TO
      'TIME'
  END
END
```

EXECUTE_SQL、**FREE_SQL**、**PREPARE_SQL** 文についての詳細は、19 ページの「ACCELL/SQL: Script and Function Reference」を参照してください。

「Calling the Function」の 12 章 311 ページの最後に追加します。

- 未定義 (UNDEFINED) の汎用変数が参照パラメータとして関数に渡され、関数が SQL 文を変数に準備した場合、変数は SQL ハンドルになります。変数は、関数と呼び元のスクリプト内で SQL ハンドルになります。

ACCELL/SQL: Script and Function Reference

「ACCELL/4GL Syntax Descriptions」の 3 章アルファベット順に、以下の新しい文を追加します。

EXECUTE_SQL

スクリプト文 : 動的 SQL 文の実行

構文

[**USING CONNECTION** *connection*]
EXECUTE_SQL { *SQL_handle* | *SQL_clause* }
[**USING** *expression_list*]
{ *EXECUTING_clause* | **;** }

引数

connection

キーワード DEFAULT またはデータベースコネクションの名前を示す文字列式です。例えば、

```
USING CONNECTION DEFAULT
```

または

```
USING CONNECTION 'emps'
```

キーワード DEFAULT を指定すると、アプリケーションに設定しているデフォルトのコネクションになります。文字列式は、DCM(Database Connection Map) ファイルのエントリの 1 つと一致する必要があります。

SQL_clause

文字列式 (String, Text) は、実行する新しい SQL 文を指定します。文中に疑問符 (?) を使用すると、expression_list に置き換えられます。SQL 文が SELECT 文の場合、結果を受け取るために前に SET 文を置いてください。

SQL_handle

PREPARE_SQL 文の実行により作成された SQL 文のハンドルです。前もって SQL ハンドルにコネクションを指定している場合、USING CONNECTION 句を重ねて指定しないでください。connection を指定すると、status$( ) システム関数は SS_HNDLNTAL を返します。

expression_list

動的 SQL 文の入力パラメータとして使われる式のリストを指定します。式の数は、入力パラメータとして必要な数以上にします。必要な数より少ない場合、その式は実行されず status$ システム関数からエラーが返ります。

EXECUTING_clause

ACCELL/4GL の EXECUTING 句です。EXECUTING_clause は、SQL_handle が SELECT 文を含んでいる場合のみ有効です。この句は、選択したデータベース行ごとに実行されます。マトリックス変数 (2 次元配列) のための EXECUTE_SQL 文には、EXECUTING 句を使用できません。

解説

EXECUTE_SQL 文は、動的 SQL 文を実行します。式を使って新しく文を記述することも、準備してある文を SQL ハンドル変数により参照することもできます。

UNSING 句はオプションで、この文だけに有効なデータベースコネクション名を指定します。コネクションの指定がないと、デフォルトコネクションが使われます。

SQL 文が SELECT 文のとき、オプションの EXECUTING ブロックはそれぞれの選択行に対し実行します。

実行した文に出力パラメータがない場合、以下の属性は初期値のままです。

OUTPUT_NAME
OUTPUT_ACCELL_TYPE
OUTPUT_DB_COLUMN_TYPE
OUTPUT_PRECISION
OUTPUT_SCALE
OUTPUT_DB_COLUMN_TYPE_CODE

準備した文に入力パラメータがない場合、以下の属性は初期値のままです。

INPUT_NAME
INPUT_ACCELL_TYPE
INPUT_DB_COLUMN_TYPE
INPUT_PRECISION
INPUT_SCALE
INPUT_DB_COLUMN_TYPE_CODE

INPUT_COUNT と OUTPUT_COUNT 属性を使って、文の実行後に入出力パラメータの数を確認できます。

以下の文は、EXECUTE_SQL 文における EXECUTING ブロック内では有効ではありません。

DELETE SELECTED ROW

UPDATE SELECTED ROW

例

次の例では、フォーム作成時に **music_collection** テーブル内の owner_f_name がすべて Jim に設定されます。

```
BEFORE FIND
EXECUTE_SQL 'UPDATE music_collection SET owner_f_name =
\'Jim\'';
```

SELECT 文を実行する場合には、**SELECT** 文からの出力をストアするために **SET** 文を併せて使用する必要があります。このマニュアルの **SET** 文の例を参照してください。

次は、上記の **music_collection** フォームで使っているデータベースの値を変更する **form1** のフォームスクリプトです。**UPDATE** 文は rating の値を 5 に変更しています。

```
FORM form1
BEFORE FORM
USING CONNECTION 'u2kdb' EXECUTE_SQL 'UPDATE
music_collection SET \
rating TO 5';
```

参照

PREPARE_SQL
FREE_SQL

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

FREE_SQL

スクリプト文 : SQL ハンドル用リソース解放

構文

FREE_SQL *SQL_handle_var*

引数

SQL_handle_var

PREPARE_SQL 文の実行によりつくられた SQL 文のハンドルです。前もって SQL ハンドルにコネクションを指定している場合、USING CONNECTION 句を重ねて指定しないでください。connection を指定すると、status$( ) システム関数は SS_HNDLNTAL を返します。

解説

FREE_SQL 文は、指定した *SQL_handle_var* が参照する準備済みの文が、使用しているリソースをすべて解放します。SQL ハンドルを明示的に解放する必要はありません。ACCELL マネージャは、変数が有効範囲から外れたり、PREPARE_SQL 文で再利用された場合、SQL ハンドルを解放します。

IS_VALID 属性を使って、SQL ハンドルを解放する必要があるか確認できます。SQL ハンドルを明示的に解放する必要はありません。変数が有効範囲から外れたり、PREPARE_SQL 文で再利用された場合、SQL ハンドルは自動的に解放されます。

status$() システム関数を使って、SQL ハンドルがハンドルであるか、SQL ハンドルが有効であるか確認できます。SQL ハンドルが有効でない場合、status$() システム関数は SS_INVHNDL を返します。SQL ハンドルがハンドルでない場合、status$( ) は SS_NOTHNDL を返します。

FREE_SQL 文が成功すると、IS_VALID 属性は FALSE にセットされます。

例

次の例では、**chng_owner_hndl** SQL ハンドルの使用していたリソースを解放しています。

```
BEFORE FORM
PREPARE_SQL 'UPDATE music_collection SET owner_f_name =
\'Jim\''
INTO chng_owner_hndl;
EXECUTE_SQL $chng_owner_hndl;
```

```
FREE_SQL $chng_owner_hndl;
```

参照

EXECUTE_SQL
PREPARE_SQL

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

PREPARE_SQL

スクリプト文 : 動的 SQL 文準備 (DBMS 依存)

構文

[**USING** [**CONNECTION**] *connection*]
PREPARE_SQL *SQL_expression* **INTO** *SQL_handle_var* **;**

引数

connection
キーワード DEFAULT またはデータベースコネクションの名前を示す文字列式です。

```
USING CONNECTION DEFAULT
```

または

```
USING CONNECTION 'emps'
```

キーワード DEFAULT を指定すると、アプリケーションに設定しているデフォルトのコネクションになります。文字列式は、DCM(Database Connection Map) ファイルのエントリの 1 つと一致する必要があります。

SQL_expression
文字列式 (String, Text) は、実行する新しい SQL 文を指定します。SQL 文は、DBMS によりサポートされていなければなりません。

SQL_handle_var
SQL_expression で指定した文と指定したコネクションをストアする変数です。SQL ハンドルがすでに文と関連付けられている場合、その文は解放されます。

解説

PREPARE_SQL 文は DBMS に対し、文のテキストのチェックと実行の準備を命令します。

UNSING 句は、オプションでこの文だけに有効なデータベースコネクション名を指定します。コネクションの指定がないと、デフォルトコネクションが使われます。

文が準備されると、DBMS がサポートしている SQL ハンドルの属性すべてが参照可能になります。準備オペレーションが成功すると、IS_PREPARED 属性は TRUE にセットされます。

SQL ハンドルを使って次のタスクを実行できます。

- 属性参照
  例 :

```
SET temp TO titles_hndl:OUTPUT_ACCELL_TYPE
```

- 関数へ渡す参照パラメータとしての SQL ハンドル
  例 :

```
calculate_pay(titles_hndl)
```

- REPARE_SQL、EXECUTE_SQL、FREE_SQL など ACCELL/4GL 文の実行
  例 :

```
BEFORE FORM
  PREPARE_SQL 'UPDATE order_tbl SET price = ?'
      into $update_hndl;
. . .
  EXECUTE_SQL $update_hndl USING $field1;
  FREE_SQL $update_hndl;
```

実行した文に出力パラメータがない場合、以下の属性は初期値のままです。

OUTPUT_NAME[ ]

OUTPUT_ACCELL_TYPE[ ]

OUTPUT_DB_COLUMN_TYPE[ ]

OUTPUT_PRECISION[ ]

OUTPUT_SCALE[ ]

OUTPUT_DB_COLUMN_TYPE_CODE[ ]

準備した文に入力パラメータがない場合、以下の属性は初期値のままです。

INPUT_NAME[ ]

INPUT_ACCELL_TYPE[ ]

INPUT_DB_COLUMN_TYPE[ ]

INPUT_PRECISION[ ]

INPUT_SCALE[ ]

INPUT_DB_COLUMN_TYPE_CODE[ ]

SQL ハンドルを明示的に解放する必要はありません。変数が有効範囲から外れたり、PREPARE_SQL 文で再利用された場合、SQL ハンドルは自動的に解放されます。SQL ハンドルの明示的な解放については、FREE_SQL 文を参照してください。

SYBASE Adaptive Server では、EXECUTE_SQL 文実行まで SQL ハンドル属性の値は無効です。

例

次の例では、フォームが作成されると PREPARE_SQL 文 を使って、music_collection テーブルの owner_f_name のすべてを Jim に更新する UPDATE 文を準備しています。UPDATE 文は、chng_owner_hndl SQL ハンドルにストアされます。EXECUTE_SQL 文 は、chng_owner_hndl SQL ハンドルにストアされている UPDATE SQL 文を実行します。

```
BEFORE FORM
  PREPARE_SQL 'UPDATE music_collection SET owner_f_name =
  \'Jim\''
  INTO chng_owner_hndl;
  EXECUTE_SQL $chng_owner_hndl;
```

参照

EXECUTE_SQL
FREE_SQL

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

「Attributes:」の 1 章に、以下の新しい属性を追加します。

INPUT_ACCELL_TYPE[ ]

Unify DataServer のみ

INPUT_ACCELL_TYPE[ ] 属性は、動的 SQL 文における入力パラメータの ACCELL/SQL データタイプを示しています。IS_PREPARED 属性が FALSE または NULL の場合、INPUT_ACCELL_TYPE[ ] 属性は有効な値ではありません。

要約 *INPUT_ACCELL_TYPE[]*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	STRING
有効値	AMOUNT BINARY BOOL DATE FLOAT NUMERIC STRING TEXT TIME DATETIME

関連情報

属性

IS_PREPARED
OUTPUT_ACCELL_TYPE[ ]

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

INPUT_COUNT

Unify DataServer のみ

INPUT_COUNT 属性は、準備済みの動的 SQL 文が必要な入力列の数を示します。

IS_PREPARED 属性が FALSE または NULL の場合、INPUT_COUNT 属性は有効な値ではありません。

要約 *INPUT_COUNT*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	NUMERIC
有効値	NULL 数値 .

関連情報

属性 IS_PREPARED
OUTPUT_COUNT

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

INPUT_DB_COLUMN_TYPE[ ]

INPUT_DB_COLUMN_TYPE[ ] 属性は、入力パラメータにある列のデータタイプを示します。列のタイプは、DBMS に依存します。

IS_PREPARED 属性が FALSE または NULL の場合、INPUT_DB_COLUMN_TYPE [ ] 属性は有効な値ではありません。

要約 *INPUT_DB_COLUMN_TYPE[]*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	STRING
有効値	NULL AMOUNT BINARY BOOL DATE FLOAT NUMERIC STRING TEXT TIME DATETIME デフォルト値 : NULL

関連情報

属性

IS_PREPARED
OUTPUT_DB_COLUMN_TYPE[ ]

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

INPUT_DB_COLUMN_TYPE_CODE[ ]

INPUT_DB_COLUMN_TYPE_CODE[ ] 属性は、DBMS のタイプコードを使って入力パラメータのデータタイプを示します。列のタイプは、DBMS に依存します。

IS_PREPARED 属性が FALSE または NULL の場合、INPUT_DB_COLUMN_TYPE_CODE [ ] 属性は有効な値ではありません。

要約 *INPUT_DB_COLUMN_TYPE_CODE[]*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	NUMERIC
有効値	NULL 有効な数値 デフォルト値 : NULL

関連情報

属性

IS_PREPARED
OUTPUT_DB_COLUMN_TYPE_CODE[ ]

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

DBMS の列タイプコードについては、使用する DBMS のマニュアルを参照してください。

INPUT_PRECISION[ ]

Unify DataServer のみ

INPUT_PRECISION[ ] 属性は、AMOUNT または NUMERIC データベース列の有効桁数を示します。STRING フィールドでは、データベース列の文字数です。FLOAT フィールドでは、ビット数の精度になります。

IS_PREPARED 属性が FALSE または NULL の場合、INPUT_PRECISION [ ] 属性は有効な値ではありません。

要約 *INPUT_PRECISION[]*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	NUMERIC
有効値	NULL 数値 デフォルト値 : NULL

関連情報

IS_PREPARED
OUTPUT_PRECISION

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

INPUT_SCALE[ ]

Unify DataServer のみ

INPUT_SCALE[ ] 属性は、NUMERIC と AMOUNT データタイプの小数点以下の桁数を入力パラメータを示します。IS_PREPARED 属性が FALSE または NULL の場合、INPUT_SCALE [ ] 属性は有効な値ではありません。

要約 *INPUT_SCALE[]*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	NUMERIC
有効値	NULL 数値 デフォルト値 : NULL

関連情報

属性

IS_PREPARED
OUTPUT_SCALE

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

IS_PREPARED

IS_PREPARED 属性は、SQL ハンドルの状態を示します。IS_PREPARED は、TRUE または FALSE という値のどちらかを持ちます。

準備済みの動的 SQL 文にとって SQL ハンドルが有効な情報であるとき、IS_PREPARED 属性は TRUE です。SQL ハンドルが準備できていないとき、IS_PREPARED 属性は FALSE です。

DBMS が準備処理をサポートしていない場合 、文が実行されるまで IS_PREPARED 属性は TRUE にはセットされません。

要約 *IS_PREPARED[]*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	BOOL
有効値	TRUE FALSE

関連情報

属性

IS_VALID

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

IS_VALID

IS_VALID 属性は、SQL ハンドルが動的 SQL 文にとって有効な情報を持っているかどうか示します。IS_VALID と IS_PREPARED 属性の組み合わせで、SQL ハンドルの属性が参照可能かを示しています。

IS_VALID が TRUE で IS_PREPARED も TRUE の場合、以下の属性は SQL ハンドルについて正しい情報を持っています。

- STATEMENT_TEXT
- CONNECTION_NAME

IS_VALID が FALSE のとき、以下の SQL ハンドル属性はゼロ・レングスの値になります。

- INPUT_ACCELL_TYPE[ ]
- INPUT_COUNT
- INPUT_DB_COLUMN_TYPE[ ]
- INPUT_DB_COLUMN_TYPE_CODE[ ]
- INPUT_PRECISION[ ]
- INPUT_SCALE[ ]
- OUTPUT_ACCELL_TYPE[ ]
- OUTPUT_COUNT
- OUTPUT_DB_COLUMN_TYPE[ ]
- OUTPUT_DB_COLUMN_TYPE_CODE[ ]
- OUTPUT_PRECISION[ ]
- OUTPUT_SCALE[ ]

FREE_SQL 文を使って解放している場合、SQL ハンドルは無効です。

要約 *IS_VALID*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	Boolean
有効値	TRUE FALSE

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

OUTPUT_ACCELL_TYPE[ ]

DBMS 依存

OUTPUT_ACCELL_TYPE[ ] 属性は、出力パラメータの ACCELL/SQL データタイプを示します。SYBASE Adaptive Server では、文の実行直後のみこれらの属性値は有効となります。デフォルトで ACCELL/SQL は、DBMS データタイプを ACCELL/SQL データタイプに変換します。デフォルトのデータタイプとは別のデータタイプを指定する場合に使用してください。

IS_PREPARED 属性が FALSE または NULL の場合、OUTPUT_ACCELL_TYPE [ ] 属性リストはゼロ・レングスです。

要約 *OUTPUT_ACCELL_TYPE[]*

ACCELL/Generator	設定不可
ACCELL/4GL	設定可能
データタイプ	STRING
有効値	NULL AMOUNT BINARY BOOL DATE FLOAT NUMERIC STRING TEXT TIME DATETIME デフォルト値 : NULL

例

例えば SYBASE Adaptive Server の DATETIME データタイプを使っている場合、ACCELL/SQL は DATE データタイプを選択します。この属性を使えば、TIME データタイプと指定することで TIME 情報が取得できます。

```
IF ($my_handle:OUTPUT_DB_COLUMN_TYPE[i] = 'DATE_TIME')
THEN SET $my_handle:OUTPUT_ACCELL_TYPE[i] TO 'TIME'
```

関連情報

属性

INPUT_ACCELL_TYPE
IS_PREPARED

OUTPUT_COUNT

DBMS 依存

OUTPUT_COUNT 属性は、動的 SELECT 文が返す出力列の数を示します。SYBASE Adaptive Server では、文の実行直後のみこれらの属性値は有効となります。

IS_PREPARED 属性が FALSE または NULL の場合、OUTPUT_COUNT 属性リストはゼロ・レングスです。

要約 *OUTPUT_COUNT*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	Numeric
有効値	NULL 数値

関連情報

属性

INPUT_COUNT
IS_PREPARED

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

OUTPUT_DB_COLUMN_TYPE[ ]

DBMS 依存

OUTPUT_DB_COLUMN_TYPE[ ] 属性は、出力パラメータにある列のデータタイプを示します。列のタイプは、DBMS に依存します。SYBASE Adaptive Server では、文の実行直後のみこれらの属性値は有効となります。

IS_PREPARED 属性が FALSE または NULL の場合、OUTPUT_DB_COLUMN_TYPE [ ] 属性リストはゼロ・レングスです。

要約 *OUTPUT_DB_COLUMN_TYPE[]*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	STRING
有効値	NULL AMOUNT BINARY BOOL DATE FLOAT NUMERIC STRING TEXT TIME DATETIME デフォルト値 : NULL

関連情報

属性

INPUT_DB_COLUMN_TYPE[ ]

IS_PREPARED

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

OUTPUT_DB_COLUMN_TYPE_CODE[ ]

DBMS 依存

OUTPUT_DB_COLUMN_TYPE_CODE[ ] 属性は、DBMS で使われているタイプコードを示します。多くの DBMS は、利用可能な値のリストを提供しています。SYBASE Adaptive Server では、文の実行直後のみこれらの属性値は有効となります。

IS_PREPARED 属性が FALSE または NULL の場合、OUTPUT_DB_COLUMN_TYPE_CODE [ ] 属性リストはゼロ・レングスです。

要約 *OUTPUT_DB_COLUMN_TYPE_CODE[]*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	NUMERIC
有効値	NULL 有効な数値 デフォルト値 : NULL

関連情報

属性

INPUT_DB_COLUMN_TYPE_CODE
IS_PREPARED

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

OUTPUT_NAME[ ]

DBMS 依存

OUTPUT_NAME[ ] 属性は、SELECT 文からの出力パラメータの名前を示します。DBMS から列名が得られない場合、OUTPUT_NAME[ ] 属性は NULL になります。SYBASE Adaptive Server では、文の実行直後のみこれらの属性値は有効となります。

IS_PREPARED 属性が FALSE または NULL の場合、OUTPUT_NAME [ ] 属性リストはゼロ・レングスです。

要約 *OUTPUT_NAME[]*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	STRING
有効値	NULL 有効な文字列 デフォルト値 : NULL

関連情報

属性

IS_PREPARED

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

OUTPUT_PRECISION[ ]

DBMS 依存

INPUT_PRECISION[ ] は、AMOUNT または NUMERIC データベース列の有効桁数を示します。STRING フィールドでは、データベース列の文字数です。FLOAT フィールドでは、ビット数の精度になります。精度に使うデータタイプは、データベースに依存します。SYBASE Adaptive Server では、文の実行直後のみこれらの属性値は有効となります。

IS_PREPARED 属性が FALSE または NULL の場合、OUTPUT_PRECISION [ ] 属性リストはゼロ・レングスです。

要約 *OUTPUT_PRECISION[]*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	NUMERIC
有効値	NULL 数値 デフォルト値 : NULL

関連情報

属性

INPUT_PRECISION[ ]

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

OUTPUT_SCALE[ ]

DBMS 依存

OUTPUT_SCALE[ ] 属性は、NUMERIC と AMOUNT データタイプの小数点以下の桁数の入力パラメータを示します。SYBASE Adaptive Server では、文の実行直後のみこれらの属性値は有効となります。

IS_PREPARED 属性が FALSE または NULL の場合、OUTPUT_SCALE [ ] 属性リストはゼロ・レングスです。

要約 *OUTPUT_SCALE[]*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	NUMERIC
有効値	NULL 数値

関連情報

属性

INPUT_SCALE[ ]

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

STATEMENT_TEXT

STATEMENT_TEXT 属性は、動的 SQL 文に指定したテキストを表しています。STATEMENT_TEXT 属性は、SQL ハンドルが有効な場合には常に文のテキストを持っています。SQL ハンドルが、アクティブからアクティブでない状態になっても属性値はそのままです。

IS_VALID 属性が FALSE のとき、STATEMENT_TEXT 属性は NULL です。

要約 *STATEMENT_TEXT*

ACCELL/Generator	設定不可
ACCELL/4GL	読み取り専用
データタイプ	String
有効値	NULL 有効な文字列

例

次の例では、動的 SQL 文のテキストが画面上に表示されます。

```
BEFORE FORM
  PREPARE_SQL 'UPDATE music_collection SET \
   owner_f_name = "Jim"' INTO $chng_owner_hndl;
  IF (status$() = SS_NORM) THEN
   BEGIN
       EXECUTE_SQL $chng_owner_hndl;
       DISPLAY 'The SQL statement was ' +
        $chng_owner_hndl:STATEMENT_TEXT for
        fyi_message WAIT;
   END
```

動的 SQL 文の準備と実行に関する構文についての情報は、PREPARE_SQL と EXECUTE_SQL を参照してください。

動的 SQL についての詳細は、12 ページの「動的 SQL 文」を参照してください。

281 ページの SET statement description に以下を追加します。

動的 SQL における SET の使用

次の例では、ユーザからの入力を受け入れています。ユーザがアーティスト (artist) の名前を、field1 フィールドにタイプします。SELECT 文は一致したタイトル (title) を取り出し、そのタイトルを変数 title_names にストアします。DISPLAY 文はタイトル名を FYI_MESSAGE に表示します。複数の行が見つかった場合は、DISPLAY 文はそれぞれの行を表示します。

```
FIELD field1
  ON FIELD
     INPUT
     IF $field1 IS NOT NULL THEN
     BEGIN
     SET $title_names TO EXECUTE_SQL ‘ SELECT title FROM
       music_collection  WHERE artist = ? ‘ USING $field1 ;
     EXECUTING
     BEGIN
     DISPLAY ‘ The titles are ‘ + $title_names for
     FYI_MESSAGE WAIT;
     END
     SET $field1 to NULL
  END
```

4 章の「System Functions and Variables」のアルファベット順に、以下の新しいシステム関数を追加します。

encrypt_db_password$( )

システム関数 : 文字列暗号化

構文

encrypt_db_password$(*password* **)**

引数

password　文字列は、暗号化するユーザパスワードを表しています。

戻り値

(STRING)　暗号化したパスワードです。関数実行時にエラーが起きた場合、NULL 文字列が返ります。

解説

encrypt_db_password$( ) 暗号化したパスワードを返します。このシステム関数は、Unify DataServer でサイト固有のログイン画面を作成するとき利用できます。

例

次は username と password という 2 つのテキストフィールドを持つフォームで、**encrypt_db_password$( )** 関数を使った例です。

```
SET $username TO $username_field;
SET $password TO encrypt_db_password$( $password_field );
OPEN CONNECTION 'connection1'
    DBUSER IS $username, DBPASSWORD IS $password
```