



EDB Postgres Advanced Server

リリース **13**

**Database Compability for Oracle Developers Built-in
Package Guide**

2020 年 12 月 30 日

目次

第 1 章	Introduction	1
第 2 章	Packages	3
2.1	Package Components	4
2.1.1	Package Specification Syntax	4
2.1.2	Package Body Syntax	8
2.2	Creating Packages	13
2.2.1	Creating the Package Specification	13
2.2.2	Creating the Package Body	14
2.3	Referencing a Package	16
2.4	Using Packages With User Defined Types	17
2.5	Dropping a Package	21
第 3 章	Built-In Packages	23
3.1	DBMS_ALERT	24
3.1.1	登録	26
3.1.2	削除	26
3.1.3	REMOVEALL	27
3.1.4	SIGNAL	27
3.1.5	WAITANY	27
3.1.6	WAITONE	29
3.1.7	包括的な例	31
3.2	DBMS_AQ	34
3.2.1	ENQUEUE	37
3.2.2	DEQUEUE	40
3.2.3	登録	43
3.2.4	UNREGISTER	45
3.3	DBMS_AQADM	48
3.3.1	ALTER_QUEUE	51
3.3.2	ALTER_QUEUE_TABLE	52
3.3.3	CREATE_QUEUE	53
3.3.4	CREATE_QUEUE_TABLE	54
3.3.5	DROP_QUEUE	56

3.3.6	DROP_QUEUE_TABLE	57
3.3.7	PURGE_QUEUE_TABLE	58
3.3.8	START_QUEUE	59
3.3.9	STOP_QUEUE	59
3.4	DBMS_CRYPTO	61
3.4.1	DECRYPT	63
3.4.2	ENCRYPT	65
3.4.3	HASH	67
3.4.4	MAC	68
3.4.5	RANDOMBYTES	69
3.4.6	RANDOMINTEGER	69
3.4.7	RANDOMNUMBER	70
3.5	DBMS_JOB	71
3.5.1	BROKEN	73
3.5.2	CHANGE	73
3.5.3	INTERVAL	74
3.5.4	NEXT_DATE	75
3.5.5	删除	75
3.5.6	RUN	76
3.5.7	SUBMIT	76
3.5.8	WHAT	78
3.6	DBMS_LOB	79
3.6.1	APPEND	82
3.6.2	COMPARE	82
3.6.3	CONVERTTOBLOB	83
3.6.4	CONVERTTOCLOB	84
3.6.5	COPY	85
3.6.6	ERASE	86
3.6.7	GET_STORAGE_LIMIT	86
3.6.8	GETLENGTH	87
3.6.9	INSTR	87
3.6.10	READ	88
3.6.11	SUBSTR	89
3.6.12	TRIM	89
3.6.13	WRITE	90
3.6.14	WRITEAPPEND	90
3.7	DBMS_LOCK	92
3.7.1	SLEEP	92
3.8	DBMS_MVIEW	93
3.8.1	GET_MV_DEPENDENCIES	95
3.8.2	REFRESH	95

3.8.3	REFRESH_ALL_MVIEWS	97
3.8.4	REFRESH_DEPENDENT	98
3.9	DBMS_OUTPUT	101
3.9.1	CHARARR	102
3.9.2	無効にする	102
3.9.3	有効にする	102
3.9.4	GET_LINE	103
3.9.5	GET_LINES	105
3.9.6	NEW_LINE	107
3.9.7	PUT	107
3.9.8	PUT_LINE	108
3.9.9	SERVEROUTPUT	109
3.10	DBMS_PIPE	111
3.10.1	CREATE_PIPE	113
3.10.2	NEXT_ITEM_TYPE	114
3.10.3	PACK_MESSAGE	116
3.10.4	PURGE	117
3.10.5	RECEIVE_MESSAGE	118
3.10.6	REMOVE_PIPE	119
3.10.7	RESET_BUFFER	120
3.10.8	SEND_MESSAGE	121
3.10.9	UNIQUE_SESSION_NAME	122
3.10.10	UNPACK_MESSAGE	123
3.10.11	包括的な例	123
3.11	DBMS_PROFILER	127
3.11.1	FLUSH_DATA	128
3.11.2	GET_VERSION	128
3.11.3	INTERNAL_VERSION_CHECK	128
3.11.4	PAUSE_PROFILER	129
3.11.5	RESUME_PROFILER	129
3.11.6	START_PROFILER	129
3.11.7	STOP_PROFILER	130
3.11.8	Using DBMS_PROFILER	131
3.11.8.1	DBMS_PROFILER の表およびビューの問合せ	132
3.11.8.2	DBMS_PROFILER - Reference	141
3.11.8.2.1	PLSQL_PROFILER_RUNS	141
3.11.8.2.2	PLSQL_PROFILER_UNITS	142
3.11.8.2.3	PLSQL_PROFILER_DATA	142
3.11.8.2.4	PLSQL_PROFILER_RAWDATA	143
3.12	DBMS_RANDOM	155
3.12.1	INITIALIZE	156

3.12.2	NORMAL	156
3.12.3	RANDOM	157
3.12.4	SEED	157
3.12.5	SEED	158
3.12.6	STRING	158
3.12.7	TERMINATE	159
3.12.8	価値	159
3.12.9	価値	160
3.13	DBMS_REDACT	161
3.13.1	Using DBMS_REDACT Constants and Function Parameters	164
3.13.2	ADD_POLICY	179
3.13.3	ALTER_POLICY	183
3.13.4	DISABLE_POLICY	185
3.13.5	ENABLE_POLICY	186
3.13.6	DROP_POLICY	187
3.13.7	UPDATE_FULL_REDACTION_VALUES	188
3.14	DBMS_RLS	192
3.14.1	ADD_POLICY	196
3.14.2	DROP_POLICY	203
3.14.3	ENABLE_POLICY	204
3.15	DBMS_SCHEDULER	206
3.15.1	Using Calendar Syntax to Specify a Repeating Interval	208
3.15.2	CREATE_JOB	209
3.15.3	CREATE_PROGRAM	212
3.15.4	CREATE_SCHEDULE	213
3.15.5	DEFINE_PROGRAM_ARGUMENT	215
3.15.6	無効にする	216
3.15.7	DROP_JOB	217
3.15.8	DROP_PROGRAM	218
3.15.9	DROP_PROGRAM_ARGUMENT	218
3.15.10	DROP_SCHEDULE	219
3.15.11	有効にする	220
3.15.12	EVALUATE_CALENDAR_STRING	221
3.15.13	RUN_JOB	222
3.15.14	SET_JOB_ARGUMENT_VALUE	223
3.16	DBMS_SESSION	224
3.16.1	SET_ROLE	224
3.17	DBMS_SQL	225
3.17.1	BIND_VARIABLE	228
3.17.2	BIND_VARIABLE_CHAR	229
3.17.3	BIND VARIABLE RAW	230

3.17.4	CLOSE_CURSOR	230
3.17.5	COLUMN_VALUE	231
3.17.6	COLUMN_VALUE_CHAR	232
3.17.7	COLUMN_VALUE_RAW	233
3.17.8	COLUMN_VALUE_LONG	233
3.17.9	DEFINE_COLUMN	234
3.17.10	DEFINE_COLUMN_CHAR	236
3.17.11	DEFINE_COLUMN_RAW	237
3.17.12	DEFINE_COLUMN_LONG	237
3.17.13	DESCRIBE COLUMNS	238
3.17.14	EXECUTE	239
3.17.15	EXECUTE_AND_FETCH	240
3.17.16	FETCH_ROWS	242
3.17.17	IS_OPEN	243
3.17.18	LAST_ROW_COUNT	244
3.17.19	LAST_ERROR_POSITION	245
3.17.20	OPEN_CURSOR	246
3.17.21	PARSE	247
3.18	DBMS_UTILITY	249
3.18.1	LNAME_ARRAY	252
3.18.2	UNCL_ARRAY	252
3.18.3	ANALYZE_DATABASE, ANALYZE SCHEMA and ANALYZE PART_OBJECT	252
3.18.4	CANONICALIZE	254
3.18.5	COMMA_TO_TABLE	256
3.18.6	DB_VERSION	257
3.18.7	EXEC_DDL_STATEMENT	258
3.18.8	FORMAT_CALL_STACK	259
3.18.9	GET_CPU_TIME	259
3.18.10	GET_DEPENDENCY	259
3.18.11	GET_HASH_VALUE	260
3.18.12	GET_PARAMETER_VALUE	262
3.18.13	GET_TIME	263
3.18.14	NAME_TOKENIZE	263
3.18.15	TABLE_TO_COMMA	266
3.19	UTL_ENCODE	268
3.19.1	BASE64_DECODE	269
3.19.2	BASE64_ENCODE	270
3.19.3	MIMEHEADER_DECODE	271
3.19.4	MIMEHEADER_ENCODE	271
3.19.5	QUOTED_PRINTABLE_DECODE	273
3.19.6	QUOTED_PRINTABLE_ENCODE	274

3.19.7	TEXT_DECODE	275
3.19.8	TEXT_ENCODE	276
3.19.9	UUDECODE	277
3.19.10	UUENCODE	278
3.20	UTL_FILE	280
3.20.1	Setting File Permissions with utl_file.umask	283
3.20.2	FCLOSE	284
3.20.3	FCLOSE_ALL	284
3.20.4	FCOPY	285
3.20.5	FFLUSH	287
3.20.6	FOPEN	288
3.20.7	FREMOVE	288
3.20.8	FRENAME	289
3.20.9	GET_LINE	291
3.20.10	IS_OPEN	292
3.20.11	NEW_LINE	293
3.20.12	PUT	294
3.20.13	PUT_LINE	296
3.20.14	PUTF	297
3.21	UTL_HTTP	300
3.21.1	HTML_PIECES	307
3.21.2	REQ	307
3.21.3	RESP	307
3.21.4	BEGIN_REQUEST	307
3.21.5	END_REQUEST	308
3.21.6	END_RESPONSE	308
3.21.7	GET_BODY_CHARSET	309
3.21.8	GET_FOLLOW_REDIRECT	309
3.21.9	GET_HEADER	310
3.21.10	GET_HEADER_BY_NAME	311
3.21.11	GET_HEADER_COUNT	312
3.21.12	GET_RESPONSE	313
3.21.13	GET_RESPONSE_ERROR_CHECK	313
3.21.14	GET_TRANSFER_TIMEOUT	313
3.21.15	READ_LINE	314
3.21.16	READ_RAW	315
3.21.17	READ_TEXT	316
3.21.18	REQUEST	317
3.21.19	REQUEST_PIECES	318
3.21.20	SET_BODY_CHARSET	318
3.21.21	SET_FOLLOW_REDIRECT	319

3.21.22	SET_HEADER	319
3.21.23	SET_RESPONSE_ERROR_CHECK	320
3.21.24	SET_TRANSFER_TIMEOUT	320
3.21.25	WRITE_LINE	321
3.21.26	WRITE_RAW	322
3.21.27	WRITE_TEXT	323
3.21.28	END_OF_BODY	324
3.22	UTL_MAIL	325
3.22.1	SEND	326
3.22.2	SEND_ATTACH_RAW	327
3.22.3	SEND_ATTACH_VARCHAR2	328
3.23	UTL_RAW	330
3.23.1	CAST_TO_RAW	331
3.23.2	CAST_TO_VARCHAR2	331
3.23.3	CONCAT	332
3.23.4	CONVERT	333
3.23.5	LENGTH	334
3.23.6	SUBSTR	334
3.24	UTL_SMTP	336
3.24.1	CONNECTION	339
3.24.2	REPLY/REPLIES	339
3.24.3	CLOSE_DATA	339
3.24.4	COMMAND	340
3.24.5	COMMAND_REPLIES	340
3.24.6	DATA	341
3.24.7	EHLO	341
3.24.8	HELO	342
3.24.9	HELP	342
3.24.10	MAIL	343
3.24.11	NOOP	343
3.24.12	OPEN_CONNECTION	344
3.24.13	OPEN_DATA	344
3.24.14	QUIT	344
3.24.15	RCPT	345
3.24.16	RSET	345
3.24.17	VERFY	345
3.24.18	WRITE_DATA	346
3.24.19	包括的な例	346
3.25	UTL_URL	349
3.25.1	ESCAPE	349
3.25.2	UNESCAPE	351

第 4 章 Conclusion	353
索引	355

第 1 章

Introduction

Oracle のデータベース互換性とは、アプリケーションコードをほとんどまたはまったく変更せずに、アプリケーションが Oracle 環境および EDBPostgresAdvancedServer (AdvancedServer) 環境で実行されることを意味します。このガイドでは、AdvancedServer が提供するパッケージサポートに関連する機能のみに焦点を当てています。

AdvancedServer が提供する他の互換性機能の使用の詳細については、次の Web サイトで入手可能な Advanced-Server ガイドの完全なセットを参照してください。

<https://www.enterprisedb.com/edb-docs/>

第 2 章

Packages

この章では、AdvancedServer のパッケージの概念について説明します。パッケージ は、関数、プロシージャ、変数、カーソル、ユーザー定義のレコードタイプ、および共通の修飾子（パッケージ識別子）を使用して参照されるレコードの名前付きコレクションです。パッケージには次の特徴があります。

- パッケージは、関連する目的を実行する機能と手順を整理する便利な手段を提供します。パッケージの関数とプロシージャを使用する権限は、パッケージ全体に付与された 1 つの特権に依存します。すべてのパッケージプログラムは、共通名で参照する必要があります。
- パッケージ内の特定の関数、プロシージャ、変数、型などは、*public* として宣言できます。パブリックエンティティは表示されており、パッケージに対する EXECUTE 特権が付与されている他のプログラムから参照できます。パブリック関数およびパブリックプロシージャの場合、署名のみが表示されます-プログラム名、パラメータ（存在する場合）、関数の戻り値の型。これらの関数とプロシージャの SPL コードは他のユーザーがアクセスできないため、パッケージを使用するアプリケーションは、プロシージャロジック自体ではなく、署名で使用可能な情報のみに依存します。
- パッケージ内の他の関数、プロシージャ、変数、型などは、*private* として宣言できます。プライベートエンティティは、パッケージ内の関数とプロシージャで参照および使用できますが、他の外部アプリケーションでは使用できません。プライベートエンティティは、パッケージ内のプログラムでのみ使用されます。
- 関数およびプロシージャ名は、パッケージ内でオーバーロードできます。1 つ以上の関数/プロシージャを同じ名前で作成できますが、署名は異なります。これにより、同じジョブを実行するが異なるタイプの入力と同じ名前のプログラムを作成する機能が提供されます。

2.1 Package Components

パッケージは、2つの主要コンポーネントで構成されています。

- **パッケージ仕様**:これはパブリックインターフェイスです（これらはパッケージの外部で参照できる要素です）。仕様内でパッケージの一部となるすべてのデータベースオブジェクトを宣言します。
- **パッケージ本体**:これには、パッケージ仕様内で宣言されたすべてのデータベースオブジェクトの実際の実装が含まれます。

パッケージ本体は、パッケージ仕様の仕様を実装します。実装の詳細と、アプリケーションからは見えないプライベートな宣言が含まれています。仕様を変更せずに、パッケージ本体をデバッグ、強化、または置換できます。同様に、実装の詳細はアプリケーションには見えないため、呼び出し側プログラムを再コンパイルせずに本文を変更できます。

2.1.1 Package Specification Syntax

パッケージ仕様は、パッケージのユーザーインターフェイス（API）を定義します。仕様には、パッケージのユーザーに表示される関数、プロシージャ、タイプ、例外、およびカーソルがリストされています。

パッケージのインターフェイスを定義するために使用される構文は次のとおりです。

```
CREATE [ OR REPLACE ] PACKAGE <package_name>
  [ <authorization_clause> ]
  { IS | AS }
  [ <declaration>; ] ...
  [ <procedure_or_function_declaration> ] ...
END [ <package_name> ] ;
```

ここで、`authorization_clause:=`

```
{ AUTHID DEFINER } | { AUTHID CURRENT_USER }
```

ここで、`procedure_or_function_declaration:=`

```
procedure_declaration | function_declaration
```

ここで、`procedure_declaration:=`

```
PROCEDURE proc_name [ argument_list ];
[ restriction_pragma; ]
```

ここで、`function_declaration:=`

```
FUNCTION func_name [ argument_list ]
    RETURN rettype [ DETERMINISTIC ];
[ restriction_pragma; ]
```

ここで、argument_list:=

```
( argument_declaration [, ...] )
```

ここで、argument_declaration:=

```
argname [ IN | IN OUT | OUT ] argtype [ DEFAULT value ]
```

ここで、restriction_pragma:=

```
PRAGMA RESTRICT_REFERENCES(name, restrictions)
```

ここで、restrictions:=

```
restriction [, ... ]
```

Parameters

package_name

package_name はパッケージに割り当てられた識別子です。各パッケージにはスキーマ内で一意の名前が必要です。

AUTHID DEFINER

AUTHID 句を省略するか AUTHID DEFINER を指定すると、パッケージ所有者の権限を使用して、データベースオブジェクトへのアクセス権限が決定されます。

AUTHID CURRENT_USER

AUTHID CURRENT_USER を指定すると、パッケージ内のプログラムを実行している現在のユーザーの特権がアクセス特権の決定に使用されます。

declaration

declaration はパブリック変数の識別子です。パブリック変数には、構文 package_name.variable を使用してパッケージの外部からアクセスできます。0、1、または複数のパブリック変数が存在する可能性があります。パブリック変数の定義は、プロシージャまたは関数の宣言の前になければなりません。

declaration は次のいずれかになります。

- 可変宣言
- 記録宣言

- コレクション宣言
- REF CURSOR とカーソル変数宣言
- TYPE のレコード、コレクション、および REF CURSORs の定義
- 例外
- オブジェクト変数宣言

proc_name

パブリックプロシージャの名前。

argname

引数の名前。引数は、関数またはプロシージャ本体内でこの名前参照されます。

IN | IN OUT | OUT

引数モード。IN は入力のみ引数を宣言します。これがデフォルトです。IN OUT は、引数が値を受け取り、値を返すことを許可します。OUT は、引数が出力専用であることを指定します。

argtype

引数のデータ型。引数の型は、基本データ型、%TYPE を使用した既存の列の型のコピー、またはネストされたテーブルやオブジェクト型などのユーザー定義型です。ベースタイプには長さを指定しないでください。たとえば、VARCHAR2 (10 ではなく VARCHAR2 を指定します)。

列のタイプは、tablename.columnname %TYPE と書くことで参照されます。これを使用すると、プロシージャをテーブルの定義の変更から独立させることができます。

DEFAULT value

DEFAULT 句は、呼び出しで入力引数が指定されていない場合、入力引数のデフォルト値を提供します。モード IN OUT または OUT の引数には DEFAULT を指定できません。

func_name

パブリック関数の名前。

rettype

戻りデータ型。

DETERMINISTIC

DETERMINISTIC は IMMUTABLE の同義語です。DETERMINISTIC 関数はデータベースを変更できず、同じ引数値が与えられたときに常に同じ結果に到達します。データベースの検索は行わず、引数リストに直接存在しない情報を使用しません。この句を含めると、すべて定数の引数を使用した関数の呼び出しを、関数の値にすぐに置き換えることができます。

restriction

以下のキーワードは互換性のために受け入れられ、無視されます。

RNDS

RNPS

TRUST

WNDS

WNPS

2.1.2 Package Body Syntax

パッケージ実装の詳細はパッケージ本体にあります。パッケージ本体には、パッケージユーザーには表示されないオブジェクトが含まれる場合があります。AdvancedServer は、パッケージ本体で次の構文をサポートしています。

```
CREATE [ OR REPLACE ] PACKAGE BODY <package_name>
{ IS | AS }
[ <private_declaration>; ] ...
[ <procedure_or_function_definition> ] ...
[ <package_initializer> ]
END [ <package_name> ] ;
```

ここで、`procedure_or_function_definition:=`

```
procedure_definition | function_definition
```

ここで、`procedure_definition:=`

```
PROCEDURE proc_name[ argument_list ]
[ options_list ]
{ IS | AS }
    procedure_body
END [ proc_name ] ;
```

ここで、`procedure_body:=`

```
[ PRAGMA AUTONOMOUS_TRANSACTION; ]
[ declaration; ] [, ...]
BEGIN
    statement; [...]
[ EXCEPTION
    { WHEN exception [OR exception] [...] THEN statement; }
    [...]
]
```

ここで、`function_definition:=`

```
FUNCTION func_name [ argument_list ]
    RETURN rettype [ DETERMINISTIC ]
[ options_list ]
{ IS | AS }
    function_body
END [ func_name ] ;
```

ここで、`function_body:=`

```
[ PRAGMA AUTONOMOUS_TRANSACTION; ]
[ declaration; ] [, ...]
BEGIN
    statement; [...]
[ EXCEPTION
    { WHEN exception [ OR exception ] [...] THEN statement; }
    [...]
]
```

ここで、`argument_list:=`

```
( argument_declaration [, ...] )
```

ここで、`argument_declaration:=`

```
argname [ IN | IN OUT | OUT ] argtype [ DEFAULT value ]
```

ここで、`options_list:=`

```
option [ ... ]
```

ここで、`option:=`

```
STRICT
LEAKPROOF
PARALLEL { UNSAFE | RESTRICTED | SAFE }
COST execution_cost
ROWS result_rows
SET config_param { TO value | = value | FROM CURRENT }
```

ここで、`package_initializer:=`

```
BEGIN
    statement; [...]
END;
```

Parameters

`package_name`

`package_name` は、これがパッケージ本体であるパッケージの名前です。この名前の既存のパッケージ仕様が存在する必要があります。

`private_declaration`

`private_declaration` は、パッケージ内の任意のプロシージャまたは関数からアクセスできるプライベート変数の識別子です。0、1、またはそれ以上のプライベート変数があります。

`private_declaration` は次のいずれかになります。

- 可変宣言
- 記録宣言
- コレクション宣言
- `REF CURSOR` とカーソル変数宣言
- `TYPE` のレコード、コレクション、および `REF CURSORs` の定義
- 例外
- オブジェクト変数宣言

`proc_name`

作成されるプロシージャの名前。

`PRAGMA AUTONOMOUS_TRANSACTION`

`PRAGMA AUTONOMOUS_TRANSACTION` は、プロシージャを自律型トランザクションとして設定するディレクティブです。

`declaration`

変数、タイプ、`REF CURSOR`、またはサブプログラム宣言。サブプログラム宣言が含まれる場合、他のすべての変数、型、および `REF CURSOR` 宣言の後に宣言する必要があります。

`statement`

SPL プログラムステートメント。`DECLARE - BEGIN - END` ブロックは、それ自体が **SPL** ステートメントと見なされることに注意してください。したがって、関数本体にはネストされたブロックが含まれる場合があります。

`exception`

`NO_DATA_FOUND`、`OTHERS` などの例外条件名

`func_name`

作成される関数の名前。

`rettype`

戻りデータ型。`argtype` にリストされている型のいずれかです。`argtype` については、`rettype` の長さを指定しないでください。

`DETERMINISTIC`

DETERMINISTIC を含めて、同じ引数値が指定されたときに関数が常に同じ結果を返すように指定します。DETERMINISTIC 関数はデータベースを変更してはなりません。

注釈: DETERMINISTIC キーワードは、PostgreSQL の IMMUTABLE オプションと同等です。

注釈: パッケージ本体のパブリック関数に DETERMINISTIC が指定されている場合、パッケージ仕様の関数宣言にも DETERMINISTIC を指定する必要があります。(プライベート関数の場合、パッケージ仕様に関数宣言はありません。)

PRAGMA AUTONOMOUS_TRANSACTION

PRAGMA AUTONOMOUS_TRANSACTION は、機能を自律型トランザクションとして設定するディレクティブです。

argname

仮引数の名前。引数は、プロシージャ本体内でこの名前で参照されます。

IN | IN OUT | OUT

引数モード。IN は入力のための引数を宣言します。これがデフォルトです。IN OUT は、引数が値を受け取り、値を返すことを許可します。OUT は、引数が出力専用であることを指定します。

argtype

引数のデータ型。引数の型は、基本データ型、%TYPE を使用した既存の列の型のコピー、またはネストされたテーブルやオブジェクト型などのユーザー定義型です。ベースタイプには長さを指定しないでください。たとえば、VARCHAR2(10) ではなく VARCHAR2 を指定します。

列のタイプは、tablename.columnname%TYPE を書き込むことで参照されます。これを使用すると、プロシージャをテーブルの定義の変更から独立させることができます。

DEFAULT value

DEFAULT 句は、プロシージャ呼び出しで指定されていない場合、入力引数のデフォルト値を提供します。モード IN OUT または OUT の引数には DEFAULT を指定できません。

注: 次のオプションは Oracle データベースと互換性がありません。これらは、AdvancedServer のみが提供する Oracle パッケージ構文の拡張機能です。

STRICT

STRICT キーワードは、NULL 引数を指定して呼び出された場合、関数が実行されないことを指定します。代わりに、関数は NULL を返します。

LEAKPROOF

LEAKPROOF キーワードは、関数が戻り値以外の引数に関する情報を明らかにしないことを指定します。

PARALLEL { UNSAFE | RESTRICTED | SAFE }

PARALLEL 句は、並列順次スキャン（並列モード）の使用を有効にします。パラレルシーケンシャルスキャンは、シリアルシーケンシャルスキャンとは対照的に、クエリ中に複数のワーカーを使用してリレーションを並行してスキャンします。

UNSAFE に設定すると、プロシージャまたは関数をパラレルモードで実行できません。このようなプロシージャまたは関数が存在すると、シリアル実行プランが強制されます。これは、PARALLEL 句が省略された場合のデフォルト設定です。

RESTRICTED に設定すると、プロシージャまたは関数を並列モードで実行できますが、実行は並列グループリーダーに制限されます。特定のリレーションの資格に並列制限のあるものがある場合、そのリレーションは並列処理のために選択されません。

SAFE に設定すると、プロシージャまたは関数を制限なしでパラレルモードで実行できます。

execution_cost

execution_cost は、cpu_operator_cost の単位で、関数の推定実行コストを示す正の数を指定します。関数がセットを返す場合、これは返される行ごとのコストです。デフォルトは 0.0025 です。

result_rows

result_rows は、クエリプランナーが関数が返すと予想される行の推定数です。デフォルトは 1000 です。

SET

SET 句を使用して、関数の継続時間のパラメーター値を指定します。

config_param はパラメータ名を指定します。

value はパラメータ値を指定します。

FROM CURRENT は、関数の終了時にパラメーター値が復元されることを保証します。

package_initializer

package_initializer のステートメントは、パッケージが最初に参照されるときにユーザーのセッションごとに 1 回実行されます。

注釈: STRICT, LEAKPROOF, PARALLEL, COST, ROWS および SET キーワードは AdvancedServer の拡張機能を提供し、Oracle ではサポートされていません。

2.2 Creating Packages

パッケージは実行可能なコードではありません。むしろ、コードのリポジトリです。パッケージを使用する場合、実際にはパッケージ内の要素を実行または参照します。

2.2.1 Creating the Package Specification

パッケージ仕様には、パッケージの外部から参照できるパッケージ内のすべての要素の定義が含まれています。これらは、パッケージのパブリック要素と呼ばれ、パッケージインターフェイスとして機能します。次のコードサンプルはパッケージ仕様です。

```
--
-- Package specification for the 'emp_admin' package.
--
CREATE OR REPLACE PACKAGE emp_admin
IS
    FUNCTION get_dept_name (
        p_deptno NUMBER DEFAULT 10
    )
    RETURN VARCHAR2;
    FUNCTION update_emp_sal (
        p_empno NUMBER,
        p_raise NUMBER
    )
    RETURN NUMBER;
    PROCEDURE hire_emp (
        p_empno          NUMBER,
        p_ename           VARCHAR2,
        p_job             VARCHAR2,
        p_sal             NUMBER,
        p_hiredate        DATE      DEFAULT sysdate,
        p_comm            NUMBER    DEFAULT 0,
        p_mgr             NUMBER,
        p_deptno          NUMBER    DEFAULT 10
    );
    PROCEDURE fire_emp (
        p_empno NUMBER
    );
END emp_admin;
```

このコードサンプルは、emp_admin パッケージ仕様を作成します。このパッケージ仕様は、2つの関数と2つのストアドプロシージャで構成されています。便宜上、OR REPLACE 句を CREATE PACKAGE ステートメントに追加することもできます。

2.2.2 Creating the Package Body

パッケージの本体には、パッケージ仕様の背後にある実際の実装が含まれています。上記の emp_admin パッケージ仕様については、仕様を実装するパッケージ本体を作成します。本体には、仕様の関数とストアードプロシージャの実装が含まれます。

```
--
-- Package body for the 'emp_admin' package.
--
CREATE OR REPLACE PACKAGE BODY emp_admin
IS
    --
    -- Function that queries the 'dept' table based on the department
    -- number and returns the corresponding department name.
    --
    FUNCTION get_dept_name (
        p_deptno          IN NUMBER DEFAULT 10
    )
    RETURN VARCHAR2
    IS
        v_dname           VARCHAR2(14);
    BEGIN
        SELECT dname INTO v_dname FROM dept WHERE deptno = p_deptno;
        RETURN v_dname;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Invalid department number ' || p_deptno);
            RETURN '';
    END;
    --
    -- Function that updates an employee's salary based on the
    -- employee number and salary increment/decrement passed
    -- as IN parameters. Upon successful completion the function
    -- returns the new updated salary.
    --
    FUNCTION update_emp_sal (
        p_empno           IN NUMBER,
        p_raise            IN NUMBER
    )
    RETURN NUMBER
    IS
        v_sal             NUMBER := 0;
    BEGIN
        SELECT sal INTO v_sal FROM emp WHERE empno = p_empno;
        v_sal := v_sal + p_raise;
        UPDATE emp SET sal = v_sal WHERE empno = p_empno;
        RETURN v_sal;
    EXCEPTION
```

(次のページに続く)

(前のページからの続き)

```

    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employee ' || p_empno || ' not found');
        RETURN -1;
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('The following is SQLERRM:');
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
        DBMS_OUTPUT.PUT_LINE('The following is SQLCODE:');
        DBMS_OUTPUT.PUT_LINE(SQLCODE);
        RETURN -1;
END;
--
-- Procedure that inserts a new employee record into the 'emp' table.
--
PROCEDURE hire_emp (
    p_empno      NUMBER,
    p_ename      VARCHAR2,
    p_job        VARCHAR2,
    p_sal        NUMBER,
    p_hiredate   DATE      DEFAULT sysdate,
    p_comm       NUMBER    DEFAULT 0,
    p_mgr        NUMBER,
    p_deptno     NUMBER    DEFAULT 10
)
AS
BEGIN
    INSERT INTO emp(empno, ename, job, sal, hiredate, comm, mgr, deptno)
        VALUES(p_empno, p_ename, p_job, p_sal,
            p_hiredate, p_comm, p_mgr, p_deptno);
END;
--
-- Procedure that deletes an employee record from the 'emp' table based
-- on the employee number.
--
PROCEDURE fire_emp (
    p_empno      NUMBER
)
AS
BEGIN
    DELETE FROM emp WHERE empno = p_empno;
END;
END;

```

2.3 Referencing a Package

パッケージ仕様内で宣言されている型、アイテム、およびサブプログラムを参照するには、ドット表記を使用します。例:

```
package_name.type_name
```

```
package_name.item_name
```

```
package_name.subprogram_name
```

emp_admin パッケージ仕様から関数を呼び出すには、次の SQL コマンドを実行します。

```
SELECT emp_admin.get_dept_name(10) FROM DUAL;
```

ここでは、パッケージ emp_admin 内で宣言された get_dept_name 関数を呼び出しています。関数への引数として部門番号を渡します。関数は部門の名前を返します。ここで返される値は ACCOUNTING で、部門番号 10 に対応します。

2.4 Using Packages With User Defined Types

次の例には、パッケージのコンテキスト内で、前の章で説明したさまざまなユーザー定義型が組み込まれています。

emp_rpt のパッケージ仕様は、2 つの関数と 2 つのプロシージャと共に、一般にアクセス可能なレコードタイプ emprec_typ と弱く型付けされた REF CURSOR、emp_refcur の宣言を示しています。関数 open_emp_by_dept は、REF CURSOR タイプ EMP_REFCUR を返します。手続き fetch_emp と close_refcur は両方とも、弱く型付けされた REF CURSOR を正式なパラメータとして宣言します。

```
CREATE OR REPLACE PACKAGE emp_rpt
IS
    TYPE emprec_typ IS RECORD (
        empno      NUMBER(4),
        ename      VARCHAR(10)
    );
    TYPE emp_refcur IS REF CURSOR;

    FUNCTION get_dept_name (
        p_deptno    IN NUMBER
    ) RETURN VARCHAR2;
    FUNCTION open_emp_by_dept (
        p_deptno    IN emp.deptno%TYPE
    ) RETURN EMP_REFCUR;
    PROCEDURE fetch_emp (
        p_refcur    IN OUT SYS_REFCURSOR
    );
    PROCEDURE close_refcur (
        p_refcur    IN OUT SYS_REFCURSOR
    );
END emp_rpt;
```

パッケージ本体には、静的カーソル、dept_cur、テーブルタイプ、depttab_typ、テーブル変数、t_dept、整数変数、t_dept_max、およびレコード変数 r_emp のプライベート変数の宣言が表示されます。

```
CREATE OR REPLACE PACKAGE BODY emp_rpt
IS
    CURSOR dept_cur IS SELECT * FROM dept;
    TYPE depttab_typ IS TABLE of dept%ROWTYPE
        INDEX BY BINARY_INTEGER;
    t_dept          DEPTTAB_TYP;
    t_dept_max      INTEGER := 1;
    r_emp           EMPREC_TYP;

    FUNCTION get_dept_name (
        p_deptno    IN NUMBER
    ) RETURN VARCHAR2
    IS
```

(次のページに続く)

(前のページからの続き)

```

BEGIN
    FOR i IN 1..t_dept_max LOOP
        IF p_deptno = t_dept(i).deptno THEN
            RETURN t_dept(i).dname;
        END IF;
    END LOOP;
    RETURN 'Unknown';
END;

FUNCTION open_emp_by_dept (
    p_deptno          IN emp.deptno%TYPE
) RETURN EMP_REFCUR
IS
    emp_by_dept EMP_REFCUR;
BEGIN
    OPEN emp_by_dept FOR SELECT empno, ename FROM emp
        WHERE deptno = p_deptno;
    RETURN emp_by_dept;
END;

PROCEDURE fetch_emp (
    p_refcur          IN OUT SYS_REFCURSOR
)
IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('EMPNO ENAME');
    DBMS_OUTPUT.PUT_LINE('-----');
    LOOP
        FETCH p_refcur INTO r_emp;
        EXIT WHEN p_refcur%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(r_emp.empno || ' ' || r_emp.ename);
    END LOOP;
END;

PROCEDURE close_refcur (
    p_refcur          IN OUT SYS_REFCURSOR
)
IS
BEGIN
    CLOSE p_refcur;
END;

BEGIN
    OPEN dept_cur;
    LOOP
        FETCH dept_cur INTO t_dept(t_dept_max);
        EXIT WHEN dept_cur%NOTFOUND;
        t_dept_max := t_dept_max + 1;
    END LOOP;

```

(次のページに続く)

(前のページからの続き)

```

CLOSE dept_cur;
t_dept_max := t_dept_max - 1;
END emp_rpt;

```

このパッケージには、プライベート静的カーソルを使用してプライベートテーブル変数 `t_dept` をロードする初期化セクションが含まれます。 `dept_cur.t_dept` は、機能 `get_dept_name` の部門名ルックアップテーブルとして機能します。

関数 `open_emp_by_dept` は、特定の部門の従業員番号と名前の結果セットに対して `REF CURSOR` 変数を返します。次に、この `REF CURSOR` 変数をプロシージャ `fetch_emp` に渡して、結果セットの個々の行を取得およびリストできます。最後に、プロシージャ `close_refcur` を使用して、この結果セットに関連付けられた `REF CURSOR` 変数を閉じることができます。

次の匿名ブロックは、パッケージ関数とパッケージプロシージャを実行します。匿名ブロックの宣言セクションで、パッケージのパブリック `REF CURSOR` タイプを使用したカーソル変数 `v_emp_cur` の宣言に注意してください。 `EMP_REFCUR`. `v_emp_cur` には、パッケージ関数とプロシージャ間で渡される結果セットへのポインタが含まれます。

```

DECLARE
    v_deptno dept.deptno%TYPE DEFAULT 30;
    v_emp_cur emp_rpt.EMP_REFCUR;
BEGIN
    v_emp_cur := emp_rpt.open_emp_by_dept(v_deptno);
    DBMS_OUTPUT.PUT_LINE('EMPLOYEES IN DEPT #' || v_deptno ||
        ': ' || emp_rpt.get_dept_name(v_deptno));
    emp_rpt.fetch_emp(v_emp_cur);
    DBMS_OUTPUT.PUT_LINE('*****');
    DBMS_OUTPUT.PUT_LINE(v_emp_cur%ROWCOUNT || ' rows were retrieved');
    emp_rpt.close_refcur(v_emp_cur);
END;

```

以下は、この匿名ブロックの結果です。

```

EMPLOYEES IN DEPT #30: SALES
EMPNO      ENAME
-----
7499       ALLEN
7521       WARD
7654       MARTIN
7698       BLAKE
7844       TURNER
7900       JAMES
*****
6 rows were retrieved

```

次の匿名ブロックは、同じ結果を達成する別の手段を示しています。パッケージプロシージャ `fetch_emp` およ

び `close_refcur` を使用する代わりに、これらのプログラムのロジックは匿名ブロックに直接コーディングされます。匿名ブロックの宣言セクションで、パッケージのパブリックレコードタイプ `EMPREC_TYP` を使用して宣言されたレコード変数 `r_emp` が追加されていることに注意してください。

```
DECLARE
    v_deptno      dept.deptno%TYPE DEFAULT 30;
    v_emp_cur      emp_rpt.EMP_REFCUR;
    r_emp          emp_rpt.EMPREC_TYP;
BEGIN
    v_emp_cur := emp_rpt.open_emp_by_dept(v_deptno);
    DBMS_OUTPUT.PUT_LINE('EMPLOYEES IN DEPT #' || v_deptno ||
        ': ' || emp_rpt.get_dept_name(v_deptno));
    DBMS_OUTPUT.PUT_LINE('EMPNO  ENAME');
    DBMS_OUTPUT.PUT_LINE('-----');
    LOOP
        FETCH v_emp_cur INTO r_emp;
        EXIT WHEN v_emp_cur%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(r_emp.empno || ' ' ||
            r_emp.ename);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('*****');
    DBMS_OUTPUT.PUT_LINE(v_emp_cur%ROWCOUNT || ' rows were retrieved');
    CLOSE v_emp_cur;
END;
```

以下は、この匿名ブロックの結果です。

```
EMPLOYEES IN DEPT #30: SALES
EMPNO      ENAME
-----
7499       ALLEN
7521       WARD
7654       MARTIN
7698       BLAKE
7844       TURNER
7900       JAMES
*****
6 rows were retrieved
```

2.5 Dropping a Package

パッケージ全体またはパッケージ本体のみを削除するための構文は次のとおりです。

```
DROP PACKAGE [ BODY ] package_name;
```

キーワード BODY を省略すると、パッケージ仕様とパッケージ本体の両方が削除されます。つまり、パッケージ全体が削除されます。キーワード BODY が指定されている場合、パッケージ本体のみがドロップされます。パッケージ仕様はそのまま残ります。package_name は、削除するパッケージの識別子です。

次のステートメントは、emp_admin のパッケージ本体のみを破棄します。

```
DROP PACKAGE BODY emp_admin;
```

次のステートメントは、emp_admin パッケージ全体をドロップします。

```
DROP PACKAGE emp_admin;
```


第 3 章

Built-In Packages

この章では、AdvancedServer で提供される組み込みパッケージについて説明します。特定のパッケージでは、パッケージの機能または手順を使用する前に、スーパーユーザー以外のユーザーにパッケージの EXECUTE 権限を明示的に付与する必要があります。ほとんどの組み込みパッケージでは、デフォルトで EXECUTE 特権が PUBLIC に付与されています。

GRANT コマンドを使用してパッケージへのアクセスを提供する方法については、次の場所にある『Oracle DevelopersSQL ガイドとのデータベース互換性*』を参照してください。

<https://www.enterprisedb.com/edb-docs>

すべてのビルトインパッケージは、ビルトインパッケージの特権を付与または取り消すときに指定する必要がある特別な sys ユーザーによって所有されます。

```
GRANT EXECUTE ON PACKAGE SYS.UTL_FILE TO john;
```

3.1 DBMS_ALERT

DBMS_ALERT パッケージは、アラートを登録、送信、および受信する機能を提供します。次の表に、サポートされる手順を示します。

機能/手順	戻りタイプ	説明
REGISTER (name)	該当なし	name という名前のアラートを受信できるように登録します
REMOVE (name)	該当なし	name という名前のアラートの登録を削除します
REMOVEALL	該当なし	すべてのアラートの登録を削除します。
SIGNAL (name, message)	該当なし	name という名前のアラートを message で通知します
WAITANY (name OUT, message OUT, status OUT, timeout)	該当なし	登録されたアラートが発生するまで待ちます。
WAITONE (name, message OUT, status OUT, timeout)	該当なし	指定されたアラート name が発生するまで待ちます。

AdvancedServer の DBMS_ALERT の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表にリストされている関数と手順のみがサポートされています。

AdvancedServer は、最大 500 の同時アラートを許可します。dbms_alert.max_alerts GUC 変数 (postgresql.conf ファイルにある) を使用して、システムで許可される同時アラートの最大数を指定できます。

dbms_alert.max_alerts 変数の値を設定するには、選択したエディターで postgresql.conf ファイル (デフォルトでは /opt/PostgresPlus/13AS/data にあります) を開き、次のように dbms_alert.max_alerts パラメーターを編集します。

```
dbms_alert.max_alerts = alert_count
```

alert_count

alert_count は、同時アラートの最大数を指定します。デフォルトでは、dbms_alert.max_alerts の値は 100 です。この機能を無効にするには、dbms_alert.max_alerts を 0 に設定します。

dbms_alert.max_alerts の GUC が正しく機能するには、custom_variable_classes パラメーターに dbms_alerts が含まれている必要があります。

```
custom_variable_classes = 'dbms_alert, ...'
```

postgresql.conf ファイルのパラメーターを編集したら、サーバーを再起動して変更を有効にする必要があります。

3.1.1 登録

REGISTER プロシージャを使用すると、指定したアラートを現在のセッションに通知できます。

```
REGISTER(<name> VARCHAR2)
```

Parameters

name

登録するアラートの名前。

Examples

次の匿名ブロックは、alert_test という名前のアラートを登録し、シグナルを待ちます。

```
DECLARE
    v_name          VARCHAR2(30) := 'alert_test';
    v_msg           VARCHAR2(80);
    v_status        INTEGER;
    v_timeout       NUMBER(3) := 120;
BEGIN
    DBMS_ALERT.REGISTER(v_name);
    DBMS_OUTPUT.PUT_LINE('Registered for alert ' || v_name);
    DBMS_OUTPUT.PUT_LINE('Waiting for signal...');
    DBMS_ALERT.WAITONE(v_name,v_msg,v_status,v_timeout);
    DBMS_OUTPUT.PUT_LINE('Alert name   : ' || v_name);
    DBMS_OUTPUT.PUT_LINE('Alert msg    : ' || v_msg);
    DBMS_OUTPUT.PUT_LINE('Alert status : ' || v_status);
    DBMS_OUTPUT.PUT_LINE('Alert timeout: ' || v_timeout || ' seconds');
    DBMS_ALERT.REMOVE(v_name);
END;

Registered for alert alert_test
Waiting for signal...
```

3.1.2 削除

REMOVE プロシージャは、指定されたアラートのセッションを登録解除します。

```
REMOVE(<name> VARCHAR2)
```

Parameters

name

登録解除するアラートの名前。

3.1.3 REMOVEALL

REMOVEALL プロシージャは、すべてのアラートのセッションを登録解除します。

```
REMOVEALL
```

3.1.4 SIGNAL

SIGNAL プロシージャは、指定されたアラートの発生を通知します。

```
SIGNAL(<name> VARCHAR2, <message> VARCHAR2)
```

Parameters

name

アラートの名前。

message

このアラートで渡す情報。

Examples

次の匿名ブロックは、alert_test のアラートを通知します。

```
DECLARE
    v_name    VARCHAR2(30) := 'alert_test';
BEGIN
    DBMS_ALERT.SIGNAL(v_name, 'This is the message from ' || v_name);
    DBMS_OUTPUT.PUT_LINE('Issued alert for ' || v_name);
END;
Issued alert for alert_test
```

3.1.5 WAITANY

WAITANY プロシージャは、登録されたアラートが発生するのを待ちます。

```
WAITANY(<name> OUT VARCHAR2, <message> OUT VARCHAR2,
        <status> OUT INTEGER, <timeout> NUMBER)
```

Parameters

name

アラートの名前を受け取る変数。

message

SIGNAL プロシージャによって送信されたメッセージを受信する変数。

status

操作によって返されたステータスコード。可能な値は次のとおりです。0-アラートが発生しました。1-タイムアウトが発生しました。

timeout

アラートを待機する時間（秒）。

Examples

次の匿名ブロックは、WAITANY プロシージャを使用して、alert_test または any_alert という名前のアラートを受信します。

```
DECLARE
    v_name          VARCHAR2(30);
    v_msg           VARCHAR2(80);
    v_status        INTEGER;
    v_timeout       NUMBER(3) := 120;
BEGIN
    DBMS_ALERT.REGISTER('alert_test');
    DBMS_ALERT.REGISTER('any_alert');
    DBMS_OUTPUT.PUT_LINE('Registered for alert alert_test and any_alert');
    DBMS_OUTPUT.PUT_LINE('Waiting for signal...');
    DBMS_ALERT.WAITANY(v_name,v_msg,v_status,v_timeout);
    DBMS_OUTPUT.PUT_LINE('Alert name   : ' || v_name);
    DBMS_OUTPUT.PUT_LINE('Alert msg    : ' || v_msg);
    DBMS_OUTPUT.PUT_LINE('Alert status : ' || v_status);
    DBMS_OUTPUT.PUT_LINE('Alert timeout: ' || v_timeout || ' seconds');
    DBMS_ALERT.REMOVEALL;
END;

Registered for alert alert_test and any_alert
Waiting for signal...
```

2 番目のセッションの匿名ブロックは、any_alert のシグナルを発行します。

```
DECLARE
    v_name          VARCHAR2(30) := 'any_alert';
BEGIN
    DBMS_ALERT.SIGNAL(v_name,'This is the message from ' || v_name);
    DBMS_OUTPUT.PUT_LINE('Issued alert for ' || v_name);
END;

Issued alert for any_alert
```

制御は最初の匿名ブロックに戻り、残りのコードが実行されます。

```
Registered for alert alert_test and any_alert
Waiting for signal...
Alert name   : any_alert
Alert msg    : This is the message from any_alert
Alert status : 0
Alert timeout: 120 seconds
```

3.1.6 WAITONE

WAITONE プロシージャは、指定された登録済みアラートが発生するのを待ちます。

```
WAITONE(<name> VARCHAR2, <message> OUT VARCHAR2,
        <status> OUT INTEGER, <timeout> NUMBER)
```

Parameters

name

アラートの名前。

message

SIGNAL プロシージャによって送信されたメッセージを受信する変数。

status

操作によって返されたステータスコード。可能な値は次のとおりです。0-アラートが発生しました。1-タイムアウトが発生しました。

timeout

アラートを待機する時間（秒）。

Examples

次の匿名ブロックは、alert_test という名前のアラートを受信するために WAITONE プロシージャが使用されることを除いて、WAITANY の例で使用されるブロックに似ています。

```
DECLARE
    v_name          VARCHAR2(30) := 'alert_test';
    v_msg           VARCHAR2(80);
    v_status        INTEGER;
    v_timeout       NUMBER(3) := 120;
BEGIN
    DBMS_ALERT.REGISTER(v_name);
    DBMS_OUTPUT.PUT_   DBMS_ALERT.REGISTER(v_name);
```

(次のページに続く)

(前のページからの続き)

```

DBMS_OUTPUT.PUT_LINE('Registered for alert ' || v_name);
DBMS_OUTPUT.PUT_LINE('Waiting for signal...');
DBMS_ALERT.WAITONE(v_name,v_msg,v_status,v_timeout);
DBMS_OUTPUT.PUT_LINE('Alert name      : ' || v_name);
DBMS_OUTPUT.PUT_LINE('Alert msg       : ' || v_msg);
DBMS_OUTPUT.PUT_LINE('Alert status    : ' || v_status);
DBMS_OUTPUT.PUT_LINE('Alert timeout:  ' || v_timeout || ' seconds');
DBMS_ALERT.REMOVE(v_name);LINE('Registered for alert ' || v_name);
DBMS_OUTPUT.PUT_LINE('Waiting for signal...');
DBMS_ALERT.WAITONE(v_name,v_msg,v_status,v_timeout);
DBMS_OUTPUT.PUT_LINE('Alert name      : ' || v_name);
DBMS_OUTPUT.PUT_LINE('Alert msg       : ' || v_msg);
DBMS_OUTPUT.PUT_LINE('Alert status    : ' || v_status);
DBMS_OUTPUT.PUT_LINE('Alert timeout:  ' || v_timeout || ' seconds');
DBMS_ALERT.REMOVE(v_name);
END;

Registered for alert alert_test
Waiting for signal...

```

2 番目のセッションで匿名ブロックによって alert_test に送信された信号:

```

DECLARE
    v_name  VARCHAR2(30) := 'alert_test';
BEGIN
    DBMS_ALERT.SIGNAL(v_name,'This is the message from ' || v_name);
    DBMS_OUTPUT.PUT_LINE('Issued alert for ' || v_name);
END;

Issued alert for alert_test

```

最初のセッションが警告され、制御が匿名ブロックに戻り、残りのコードが実行されます。

```

Registered for alert alert_test
Waiting for signal...
Alert name      : alert_test
Alert msg       : This is the message from alert_test
Alert status    : 0
Alert timeout:  120 seconds

```


3.1.7 包括的な例

次の例では、2つのトリガーを使用して、dept テーブルまたは emp テーブルが変更されたときにアラートを送信します。匿名ブロックはこれらのアラートをリッスンし、アラートを受信するとメッセージを表示します。

以下は、dept および emp テーブルのトリガーです。

```
CREATE OR REPLACE TRIGGER dept_alert_trig
    AFTER INSERT OR UPDATE OR DELETE ON dept
DECLARE
    v_action          VARCHAR2(25);
BEGIN
    IF INSERTING THEN
        v_action := ' added department(s) ';
    ELSIF UPDATING THEN
        v_action := ' updated department(s) ';
    ELSIF DELETING THEN
        v_action := ' deleted department(s) ';
    END IF;
    DBMS_ALERT.SIGNAL('dept_alert',USER || v_action || 'on ' ||
        SYSDATE);
END;

CREATE OR REPLACE TRIGGER emp_alert_trig
    AFTER INSERT OR UPDATE OR DELETE ON emp
DECLARE
    v_action          VARCHAR2(25);
BEGIN
    IF INSERTING THEN
        v_action := ' added employee(s) ';
    ELSIF UPDATING THEN
        v_action := ' updated employee(s) ';
    ELSIF DELETING THEN
        v_action := ' deleted employee(s) ';
    END IF;
    DBMS_ALERT.SIGNAL('emp_alert',USER || v_action || 'on ' ||
        SYSDATE);
END;
```

次の匿名ブロックはセッションで実行されますが、dept および emp テーブルの更新は他のセッションで発生します。

```
DECLARE
    v_dept_alert      VARCHAR2(30) := 'dept_alert';
    v_emp_alert       VARCHAR2(30) := 'emp_alert';
    v_name            VARCHAR2(30);
    v_msg             VARCHAR2(80);
    v_status          INTEGER;
```

(次のページに続く)

(前のページからの続き)

```

v_timeout          NUMBER(3) := 60;
BEGIN
  DBMS_ALERT.REGISTER(v_dept_alert);
  DBMS_ALERT.REGISTER(v_emp_alert);
  DBMS_OUTPUT.PUT_LINE('Registered for alerts dept_alert and emp_alert');
  DBMS_OUTPUT.PUT_LINE('Waiting for signal...');
  LOOP
    DBMS_ALERT.WAITANY(v_name,v_msg,v_status,v_timeout);
    EXIT WHEN v_status != 0;
    DBMS_OUTPUT.PUT_LINE('Alert name   : ' || v_name);
    DBMS_OUTPUT.PUT_LINE('Alert msg    : ' || v_msg);
    DBMS_OUTPUT.PUT_LINE('Alert status : ' || v_status);
    DBMS_OUTPUT.PUT_LINE('-----' ||
      '-----');
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('Alert status : ' || v_status);
  DBMS_ALERT.REMOVEALL;
END;

Registered for alerts dept_alert and emp_alert
Waiting for signal...

```

次の変更は、ユーザー **mary** によって行われます。

```

INSERT INTO dept VALUES (50,'FINANCE','CHICAGO');
INSERT INTO emp (empno,ename,deptno) VALUES (9001,'JONES',50);
INSERT INTO emp (empno,ename,deptno) VALUES (9002,'ALICE',50);

```

次の変更は、ユーザー **john** によって行われます。

```

INSERT INTO dept VALUES (60,'HR','LOS ANGELES');

```

以下は、トリガーから信号を受信する匿名ブロックによって表示される出力です。

```

Registered for alerts dept_alert and emp_alert
Waiting for signal...
Alert name   : dept_alert
Alert msg    : mary added department(s) on 25-OCT-07 16:41:01
Alert status : 0
-----
Alert name   : emp_alert
Alert msg    : mary added employee(s) on 25-OCT-07 16:41:02
Alert status : 0
-----
Alert name   : dept_alert
Alert msg    : john added department(s) on 25-OCT-07 16:41:22
Alert status : 0

```

(次のページに続く)

(前のページからの続き)

```
-----  
Alert status : 1
```

3.2 DBMS_AQ

EDBPostgresAdvancedServerAdvancedQueuing は、AdvancedServer データベースのメッセージキューとメッセージ処理を提供します。ユーザー定義のメッセージはキューに保存されます。キューのコレクションはキューテーブルに格納されます。DBMS_AQADM パッケージのプロシージャは、メッセージキューとキューテーブルを作成および管理します。DBMS_AQ パッケージを使用して、メッセージをキューに追加したり、キューからメッセージを削除したり、PL/SQL コールバック・プロシージャを登録または登録解除します。

AdvancedServer は、SQL コマンドを使用した DBMS_AQ パッケージの拡張（非互換）機能も提供します。次の SQL コマンドの詳細については、*Oracle Developers* のデータベース互換性 SQL ガイド を参照してください。

- ALTER QUEUE
- ALTER QUEUE TABLE
- CREATE QUEUE
- CREATE QUEUE TABLE
- DROP QUEUE
- DROP QUEUE TABLE

DBMS_AQ パッケージは、メッセージのエンキュー、メッセージのデキュー、コールバックプロシージャの管理を可能にするプロシージャを提供します。サポートされる手順は次のとおりです。

機能/手順	戻りタイプ	説明
ENQUEUE	該当なし	メッセージをキューに投稿します。
DEQUEUE	該当なし	メッセージが利用可能な場合または利用可能な場合に、キューからメッセージを取得します。
REGISTER	該当なし	コールバックプロシージャを登録します。
UNREGISTER	該当なし	コールバックプロシージャの登録を解除します。

AdvancedServer の DBMS_AQ の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表にリストされている手順のみがサポートされています。

AdvancedServer は、以下にリストされている定数の使用をサポートしています。

定数	説明	パラメータ用
DBMS_AQ.BROWSE (0)	ロックせずにメッセージを読みます。	dequeue_options_t. dequeue_mode

次のページに続く

表 1 – 前のページからの続き

DBMS_AQ.LOCKED (1)	この定数は定義されていますが、使用されるとエラーを返します。	dequeue_options_t. dequeue_mode
DBMS_AQ.REMOVE (2)	読んだ後にメッセージを削除します。デフォルト。	dequeue_options_t. dequeue_mode
DBMS_AQ.REMOVE_NODATA (3)	この定数は定義されていますが、使用されるとエラーを返します。	dequeue_options_t. dequeue_mode
DBMS_AQ.FIRST_MESSAGE (0)	検索語に一致する最初の利用可能なメッセージを返します。	dequeue_options_t. navigation
DBMS_AQ.NEXT_MESSAGE (1)	検索語に一致する次の利用可能なメッセージを返します。	dequeue_options_t. navigation
DBMS_AQ.NEXT_TRANSACTION (2)	この定数は定義されていますが、使用されるとエラーを返します。	dequeue_options_t. navigation
DBMS_AQ.FOREVER (-1)	検索語に一致するメッセージが見つからない場合、デフォルトで永遠に待機します。	dequeue_options_t.wait
DBMS_AQ.NO_WAIT (0)	検索語に一致するメッセージが見つからない場合、待機しないでください。	dequeue_options_t.wait
DBMS_AQ.ON_COMMIT (0)	デキューは、現在のトランザクションの一部です。	enqueue_options_t. visibility, dequeue_options_t. visibility
DBMS_AQ.IMMEDIATE (1)	この定数は定義されていますが、使用されるとエラーを返します。	enqueue_options_t. visibility, dequeue_options_t. visibility
DBMS_AQ.PERSISTENT (0)	メッセージはテーブルに保存する必要があります。	enqueue_options_t. delivery_mode
DBMS_AQ.BUFFERED (1)	この定数は定義されていますが、使用されるとエラーを返します。	enqueue_options_t. delivery_mode

次のページに続く

表 1 – 前のページからの続き

DBMS_AQ.READY (0)	メッセージを処理する準備ができていることを指定します。	message_properties_t.state
DBMS_AQ.WAITING (1)	メッセージが処理されるのを待っていることを指定します。	message_properties_t.state
DBMS_AQ.PROCESSED (2)	メッセージが処理されたことを指定します。	message_properties_t.state
DBMS_AQ.EXPIRED (3)	メッセージが例外キューにあることを指定します。	message_properties_t.state
DBMS_AQ.NO_DELAY (0)	この定数は定義されていますが、使用するとエラーを返します	message_properties_t.delay
DBMS_AQ.NEVER (NULL)	この定数は定義されていますが、使用するとエラーを返します	message_properties_t.expiration
DBMS_AQ.NAMESPACE_AQ (0)	DBMS_AQ キューからの通知を受け入れます。	sys.aq\$_reg_info.namespace
DBMS_AQ.NAMESPACE_ANONYMOUS (1)	この定数は定義されていますが、使用するとエラーを返します	sys.aq\$_reg_info.namespace

次の表にリストされている DBMS_AQ 構成パラメータは、postgresql.conf ファイルで定義できます。構成パラメータを定義した後、DBMS_AQ パッケージを起動して、キューおよびキュー表に保持されているメッセージを使用および管理できます。

パラメータ	説明
dbms_aq.max_workers	実行するワーカーの最大数。
dbms_aq.max_idle_time	ワーカーが終了するまで待機する必要があるアイドル時間。
dbms_aq.min_work_time	ワーカーが終了する前に実行できる最小時間。
dbms_aq.launch_delay	ワーカーを作成する間の最小時間。
dbms_aq.batch_size	単一のトランザクションで処理するメッセージの最大数。デフォルトのバッチサイズは 10 です。
dbms_aq.max_databases	データベースの DBMS_AQ のハッシュテーブルのサイズ。デフォルト値は 1024 です。
dbms_aq.max_pending_retries	DBMS_AQ の保留中の再試行のハッシュテーブルのサイズ。デフォルト値は 1024 です。

3.2.1 ENQUEUE

ENQUEUE プロシージャは、キューにエントリを追加します。署名は次のとおりです。

```
ENQUEUE (
  <queue_name> IN VARCHAR2,
  <enqueue_options> IN DBMS_AQ.ENQUEUE_OPTIONS_T,
  <message_properties> IN DBMS_AQ.MESSAGE_PROPERTIES_T,
  <payload> IN <type_name>,
  <msgid> OUT RAW)
```

Parameters

queue_name

既存のキューの名前（オプションでスキーマ修飾）。スキーマ名を省略すると、サーバーは SEARCH_PATH で指定されたスキーマを使用します。Oracle とは異なり、引用符で囲まれていない識別子は保存する前に小文字に変換されることに注意してください。特殊文字を含めるか、大文字と小文字を区別する名前を使用するには、名前を二重引用符で囲みます。

キューの作成の詳細については、DBMS_AQADM.CREATE_QUEUE を参照してください。

enqueue_options

enqueue_options はタイプの値、enqueue_options_t:

```
DBMS_AQ.ENQUEUE_OPTIONS_T IS RECORD (
  visibility BINARY_INTEGER DEFAULT ON_COMMIT,
  relative_msgid RAW(16) DEFAULT NULL,
  sequence_deviation BINARY_INTEGER DEFAULT NULL,
  transformation VARCHAR2(61) DEFAULT NULL,
  delivery_mode PLS_INTEGER NOT NULL DEFAULT PERSISTENT);
```

現在、enqueue_options_t でサポートされている唯一のパラメーター値は次のとおりです。

visibility	ON_COMMIT。
delivery_mode	PERSISTENT
sequence_deviation	NULL
transformation	NULL
relative_msgid	NULL

message_properties

message_properties はタイプの値、message_properties_t:

```
message_properties_t IS RECORD(  
    priority INTEGER,  
    delay INTEGER,  
    expiration INTEGER,  
    correlation CHARACTER VARYING(128) COLLATE pg_catalog."C" ,  
    attempts INTEGER,  
    recipient_list "AQ$RECIPIENT_LIST_T" ,  
    exception_queue CHARACTER VARYING(61) COLLATE pg_catalog."C" ,  
    enqueue_time TIMESTAMP WITHOUT TIME ZONE,  
    state INTEGER,  
    original_msgid BYTEA,  
    transaction_group CHARACTER VARYING(30) COLLATE pg_catalog."C" ,  
    delivery_mode INTEGER  
DBMS_AQ.PERSISTENT);
```

message_properties_t でサポートされている値は次のとおりです。

priority	キューテーブル定義に priority を参照する sort_list が含まれている場合、このパラメーターはメッセージがデキューされる順序に影響します。値が小さいほど、デキューの優先度が高くなります。
delay	メッセージをデキューまたは NO_DELAY に使用できるようになるまでの秒数を指定します。
expiration	expiration パラメーターを使用して、メッセージの有効期限が切れるまでの秒数を指定します。
correlation	相関を使用して、エントリに関連付けられるメッセージを指定します。デフォルトは NULL です。
attempts	これは、メッセージをデキューする試行回数を指定するシステム管理値です。
recipient_list	このパラメーターはサポートされていません。
exception_queue	exception_queue パラメータを使用して、メッセージの有効期限が切れたり、ロールバックが多すぎるトランザクションによってデキューされた場合に、メッセージの移動先となる例外キューの名前を指定します。
enqueue_time	enqueue_time は、レコードがキューに追加された時間です。この値はシステムによって提供されます。
state	このパラメーターは DBMS_AQ によって維持されます。状態は次のとおりです。 DBMS_AQ.WAITING –遅延に達していません。 DBMS_AQ.READY –キューエントリは処理の準備ができています。 DBMS_AQ.PROCESSED –キューエントリが処理されました。 DBMS_AQ.EXPIRED –キューエントリは例外キューに移動されました。
original_msgid	このパラメーターは互換性のために受け入れられ、無視されます。
transaction_group	このパラメーターは互換性のために受け入れられ、無視されます。
delivery_mode	このパラメーターはサポートされていません。DBMS_AQ.PERSISTENT の値を指定します。

payload

payload パラメータを使用して、キューエントリに関連付けられるデータを提供します。ペイロードタイプは、対応するキューテーブルの作成時に指定されたタイプと一致する必要があります (DBMS_AQADM.CREATE_QUEUE_TABLE を参照)。

msgid

msgid パラメーターを使用して、一意の (システム生成) メッセージ識別子を取得します。

Example

次の匿名ブロックは DBMS_AQ.ENQUEUE を呼び出し、work_order という名前のキューにメッセージを追加します。

```
DECLARE

    enqueue_options    DBMS_AQ.ENQUEUE_OPTIONS_T;
    message_properties DBMS_AQ.MESSAGE_PROPERTIES_T;
    message_handle      raw(16);
    payload              work_order;

BEGIN

    payload := work_order('Smith', 'system upgrade');

DBMS_AQ.ENQUEUE (
    queue_name          => 'work_order',
    enqueue_options     => enqueue_options,
    message_properties  => message_properties,
    payload              => payload,
    msgid               => message_handle
);
END;
```

3.2.2 DEQUEUE

DEQUEUE プロシージャはメッセージをデキューします。署名は次のとおりです。

```
DEQUEUE (
    <queue_name> IN VARCHAR2,
    <dequeue_options> IN DBMS_AQ.DEQUEUE_OPTIONS_T,
    <message_properties> OUT DBMS_AQ.MESSAGE_PROPERTIES_T,
    <payload> OUT <type_name>,
    <msgid> OUT RAW)
```

Parameters

queue_name

既存のキューの名前（オプションでスキーマ修飾）。スキーマ名を省略すると、サーバーは SEARCH_PATH で指定されたスキーマを使用します。Oracle とは異なり、引用符で囲まれていない識別子は保存する前に小文字に変換されることに注意してください。特殊文字を含めるか、大文字と小文字を区別する名前を使用するには、名前を二重引用符で囲みます。

キューの作成の詳細については、DBMS_AQADM.CREATE_QUEUE を参照してください。

dequeue_options

dequeue_options はタイプの値、dequeue_options_t:

```

DEQUEUE_OPTIONS_T IS RECORD (
    consumer_name CHARACTER VARYING(30),
    dequeue_mode INTEGER,
    navigation INTEGER,
    visibility INTEGER,
    wait INTEGER,
    msgid BYTEA,
    correlation CHARACTER VARYING(128),
    deq_condition CHARACTER VARYING(4000),
    transformation CHARACTER VARYING(61),
    delivery_mode INTEGER);

```

現在、dequeue_options_t でサポートされているパラメーター値は次のとおりです。

consumer_name	NULL でなければなりません。
dequeue_mode	デキュー操作のロック動作。次のいずれかでなければなりません: DBMS_AQ.BROWSE –ロックを取得せずにメッセージを読み取ります。 DBMS_AQ.LOCKED –ロックを取得した後にメッセージを読み取ります。 DBMS_AQ.REMOVE –メッセージを削除する前にメッセージを読み取ります。 DBMS_AQ.REMOVE_NODATA –メッセージを読み取りますが、メッセージは削除しません。
navigation	取得されるメッセージを識別します。次のいずれかでなければなりません: FIRST_MESSAGE –検索用語に一致するキュー内の最初のメッセージ。 NEXT_MESSAGE –最初の実語に一致する次の利用可能なメッセージ。
visibility	ON_COMMIT でなければなりません-現在のトランザクションをロールバックすると、デキューされたアイテムはキューに残ります。
wait	0 より大きい数値でなければなりません。または: DBMS_AQ.FOREVER –無期限に待機します。 DBMS_AQ.NO_WAIT –待たないでください。
msgid	デキューされるメッセージのメッセージ ID。
correlation	互換性のために受け入れられ、無視されます。
deq_condition	メッセージをデキューする必要があるかどうかを示す BOOLEAN 値に評価される VARCHAR2 式
transformation	互換性のために受け入れられ、無視されます。
delivery_mode	PERSISTENT でなければなりません。現時点では、バッファリングされたメッセージはサポートされていません。

message_properties

message_properties はタイプの値、message_properties_t:

```

message_properties_t IS RECORD(
    priority INTEGER,
    delay INTEGER,
    expiration INTEGER,
    correlation CHARACTER VARYING(128) COLLATE pg_catalog."C",
    attempts INTEGER,
    recipient_list "AQ$_RECIPIENT_LIST_T",
    exception_queue CHARACTER VARYING(61) COLLATE pg_catalog."C",
    enqueue_time TIMESTAMP WITHOUT TIME ZONE,
    state INTEGER,
    original_msgid BYTEA,
    transaction_group CHARACTER VARYING(30) COLLATE pg_catalog."C",
    delivery_mode INTEGER
    DBMS_AQ.PERSISTENT);

```

message_properties_t でサポートされている値は次のとおりです。

priority	キューテーブル定義に優先度を参照する sort_list が含まれている場合、このパラメーターはメッセージがデキューされる順序に影響します。値が小さいほど、デキューの優先度が高くなります。
delay	メッセージをデキューまたは NO_DELAY に使用できるようになるまでの秒数を指定します。
expiration	expiration パラメーターを使用して、メッセージの有効期限が切れるまでの秒数を指定します。
correlation	相関を使用して、エントリに関連付けられるメッセージを指定します。デフォルトは NULL です。
attempts	これは、メッセージをデキューする試行回数を指定するシステム管理値です。
recipient_list	このパラメーターはサポートされていません。
exception_queue	exception_queue パラメータを使用して、メッセージの有効期限が切れたり、ロールバックが多すぎるトランザクションによってデキューされた場合に、メッセージの移動先となる例外キューの名前を指定します。
enqueue_time	enqueue_time は、レコードがキューに追加された時間です。この値はシステムによって提供されます。
state	このパラメーターは DBMS_AQ によって維持されます。状態は次のとおりです。 DBMS_AQ.WAITING –遅延に達していません。 DBMS_AQ.READY –キューエントリは処理の準備ができています。 DBMS_AQ.PROCESSED –キューエントリが処理されました。 DBMS_AQ.EXPIRED –キューエントリは例外キューに移動されました。
original_msgid	このパラメーターは互換性のために受け入れられ、無視されます。
transaction_group	このパラメーターは互換性のために受け入れられ、無視されます。
delivery_mode	このパラメーターはサポートされていません。 DBMS_AQ.PERSISTENT の値を指定します。

payload

payload パラメータを使用して、デキュー操作でメッセージのペイロードを取得します。ペイロードタイプは、キューテーブルの作成時に指定されたタイプと一致する必要があります。

msgid

msgid パラメーターを使用して、一意のメッセージ識別子を取得します。

Example

次の匿名ブロックは DBMS_AQ.DEQUEUE を呼び出し、キューおよびペイロードからメッセージを取得します。

```
DECLARE

    dequeue_options    DBMS_AQ.DEQUEUE_OPTIONS_T;
    message_properties DBMS_AQ.MESSAGE_PROPERTIES_T;
    message_handle      raw(16);
    payload             work_order;

BEGIN
    dequeue_options.dequeue_mode := DBMS_AQ.BROWSE;

    DBMS_AQ.DEQUEUE (
        queue_name        => 'work_queue',
        dequeue_options    => dequeue_options,
        message_properties => message_properties,
        payload           => payload,
        msgid             => message_handle
    );

    DBMS_OUTPUT.PUT_LINE (
        'The next work order is [' || payload.subject || '].'
    );
END;
```

ペイロードは DBMS_OUTPUT.PUT_LINE で表示されます。

3.2.3 登録

REGISTER プロシージャを使用して、アイテムがエンキューまたはデキューされるときに通知される電子メールアドレス、プロシージャ、または URL を登録します。署名は次のとおりです。

```
REGISTER (
    <reg_list> IN SYS.AQ$_REG_INFO_LIST,
    <count> IN NUMBER)
```

Parameters

reg_list

reg_list は AQ\$_REG_INFO_LIST タイプのリストです。登録する各サブスクリプションに関する情報を提供します。リスト内の各エントリは AQ\$_REG_INFO タイプであり、以下を含むことができます。

属性	種類	説明
name	VARCHAR2 (128)	サブスクリプションの（オプションでスキーマ修飾された）名前。
namespace	数値	サポートされている値は DBMS_AQ.NAMESPACE_AQ (0) のみです
callback	VARCHAR2 (4000)	通知時に実行されるアクションを説明します。現在、PL/SQL プロシージャのコールのみがサポートされています。呼び出しの形式は次のとおりです。 plsql://schema.procedure どこで: schema は、プロシージャが存在するスキーマを指定します。 procedure は、通知されるプロシージャの名前を指定します。
context	RAW (16)	コールバックプロシージャに必要なユーザー定義の値。

count

count は reg_list のエントリ数です。

Example

次の匿名ブロックは DBMS_AQ.REGISTER を呼び出し、アイテムがキューに追加またはキューから削除されたときに通知されるプロシージャを登録します。DECLARE セクションで識別される各サブスクリプションに対して、属性セット（sys.aq\$_reg_info タイプ）が提供されます。

```
DECLARE
    subscription1 sys.aq$_reg_info;
    subscription2 sys.aq$_reg_info;
    subscription3 sys.aq$_reg_info;
    subscriptionlist sys.aq$_reg_info_list;
BEGIN
    subscription1 := sys.aq$_reg_info('q', DBMS_AQ.NAMESPACE_AQ,
    'plsql://assign_worker?PR=0',HEXTORAW('FFFF'));
    subscription2 := sys.aq$_reg_info('q', DBMS_AQ.NAMESPACE_AQ,
    'plsql://add_to_history?PR=1',HEXTORAW('FFFF'));
    subscription3 := sys.aq$_reg_info('q', DBMS_AQ.NAMESPACE_AQ,
    'plsql://reserve_parts?PR=2',HEXTORAW('FFFF'));
```

(次のページに続く)

(前のページからの続き)

```
subscriptionlist := sys.aq$_reg_info_list(subscription1,  
subscription2, subscription3);  
dbms_aq.register(subscriptionlist, 3);  
commit;  
  
END;  
/
```

subscriptionlist は sys.aq\$_reg_info_list タイプで、前述の sys.aq\$_reg_info オブジェクトが含まれています。リスト名とオブジェクト数は dbms_aq.register に渡されます。

3.2.4 UNREGISTER

UNREGISTER プロシージャを使用して、エンキューおよびデキューに関連する通知をオフにします。署名は次のとおりです。

```
UNREGISTER(  
  <reg_list> IN SYS.AQ$_REG_INFO_LIST,  
  <count> IN NUMBER)
```

Parameter

reg_list

reg_list は AQ\$_REG_INFO_LIST タイプのリストです。登録する各サブスクリプションに関する情報を提供します。リスト内の各エントリは AQ\$_REG_INFO タイプであり、以下を含むことができます。

属性	種類	説明
name	VARCHAR2 (128)	サブスクリプションの（オプションでスキーマ修飾された）名前。
namespace	数値	サポートされている値は DBMS_AQ.NAMESPACE_AQ (0) のみです
callback	VARCHAR2 (4000)	通知時に実行されるアクションを説明します。現在、PL/SQL プロシージャのコールのみがサポートされています。呼び出しの形式は次のとおりです。 plsql://schema.procedure どこで: schema は、プロシージャが存在するスキーマを指定します。 procedure は、通知されるプロシージャの名前を指定します。
context	RAW (16)	プロシージャに必要なユーザー定義の値。

count

count は reg_list のエントリ数です。

Example

次の匿名ブロックは DBMS_AQ.UNREGISTER を呼び出し、DBMS_AQ.REGISTER の例で指定された通知を無効にします。

```

DECLARE
    subscription1 sys.aq$_reg_info;
    subscription2 sys.aq$_reg_info;
    subscription3 sys.aq$_reg_info;
    subscriptionlist sys.aq$_reg_info_list;
BEGIN
    subscription1 := sys.aq$_reg_info('q', DBMS_AQ.NAMESPACE_AQ,
    'plsql://assign_worker?PR=0',HEXTORAW('FFFF'));
    subscription2 := sys.aq$_reg_info('q', DBMS_AQ.NAMESPACE_AQ,
    'plsql://add_to_history?PR=1',HEXTORAW('FFFF'));
    subscription3 := sys.aq$_reg_info('q', DBMS_AQ.NAMESPACE_AQ,
    'plsql://reserve_parts?PR=2',HEXTORAW('FFFF'));

    subscriptionlist := sys.aq$_reg_info_list(subscription1,
    subscription2, subscription3);
    dbms_aq.unregister(subscriptionlist, 3);
    commit;
END;
```

(次のページに続く)

(前のページからの続き)

/

subscriptionlist は `sys.aq$_reg_info_list` タイプで、前述の `sys.aq$_reg_info` オブジェクトが含まれています。リスト名とオブジェクト数は `dbms_aq.unregister` に渡されます。

3.3 DBMS_AQADM

EDBPostgresAdvancedServerAdvancedQueuing は、AdvancedServer データベースのメッセージキューとメッセージ処理を提供します。ユーザー定義のメッセージはキューに保存されます。キューのコレクションはキューテーブルに格納されます。DBMS_AQADM パッケージのプロシージャは、メッセージキューとキューテーブルを作成および管理します。DBMS_AQ パッケージを使用して、メッセージをキューに追加したり、キューからメッセージを削除したり、PL/SQL コールバック・プロシージャを登録または登録解除します。

AdvancedServer は、SQL コマンドを使用した DBMS_AQ パッケージの拡張（非互換）機能も提供します。次の SQL コマンドの詳細については、*OracleDevelopers* のデータベース互換性 *SQL* ガイド を参照してください。

- ALTER QUEUE
- ALTER QUEUE TABLE
- CREATE QUEUE
- CREATE QUEUE TABLE
- DROP QUEUE
- DROP QUEUE TABLE

DBMS_AQADM パッケージは、キューおよびキュー表を作成および管理できるプロシージャを提供します。

機能/手順	戻りタイプ	説明
ALTER_QUEUE	該当なし	既存のキューを変更します。
ALTER_QUEUE_TABLE	該当なし	既存のキューテーブルを変更します。
CREATE_QUEUE	該当なし	キューを作成します。
CREATE_QUEUE_TABLE	該当なし	キューテーブルを作成します。
DROP_QUEUE	該当なし	既存のキューを削除します。
DROP_QUEUE_TABLE	該当なし	既存のキューテーブルを削除します。
PURGE_QUEUE_TABLE	該当なし	キューテーブルから 1 つ以上のメッセージを削除します。
START_QUEUE	該当なし	キューをプロシージャのエンキューおよびデキューに使用できるようにします。
STOP_QUEUE	該当なし	エンキューおよびデキュー手順でキューを使用不可にする

AdvancedServer の DBMS_AQADM の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表にリストされている関数と手順のみがサポートされています。

AdvancedServer は、以下にリストされている引数の使用をサポートしています。

定数	説明	パラメータ用
DBMS_AQADM.TRANSACTIONAL (1)	この定数は定義されていますが、使用されるとエラーを返します。	message_grouping
DBMS_AQADM.NONE (0)	キュー表のメッセージのグループ化を指定するために使用します。	message_grouping
DBMS_AQADM.NORMAL_QUEUE (0)	create_queue ～ specify queue_type で使用します。	queue_type
DBMS_AQADM.EXCEPTION_QUEUE (1)	create_queue とともに使用して queue_type を指定します。	queue_type
DBMS_AQADM.INFINITE (-1)	create_queue とともに使用して retention_time を指定します。	retention_time
DBMS_AQADM.PERSISTENT (0)	メッセージはテーブルに保存する必要があります。	enqueue_options_t. delivery_mode
DBMS_AQADM.BUFFERED (1)	この定数は定義されていますが、使用されるとエラーを返します。	enqueue_options_t. delivery_mode
DBMS_AQADM.PERSISTENT_OR_BUFFERED (2)	この定数は定義されていますが、使用されるとエラーを返します。	enqueue_options_t. delivery_mode

3.3.1 ALTER_QUEUE

ALTER_QUEUE 手順を使用して、既存のキューを変更します。署名は次のとおりです。

```
ALTER_QUEUE (
  <max_retries> IN NUMBER DEFAULT NULL,
  <retry_delay> IN NUMBER DEFAULT 0
  <retention_time> IN NUMBER DEFAULT 0,
  <auto_commit> IN BOOLEAN DEFAULT TRUE)
  <comment> IN VARCHAR2 DEFAULT NULL,
```

Parameters

queue_name

新しいキューの名前。

max_retries

max_retries は、デキューステートメントでメッセージを削除する最大試行回数を指定します。max_retries の値は、ROLLBACK ステートメントごとに増分されます。失敗した試行回数が max_retries で指定された値に達すると、メッセージは例外キューに移動します。0 を指定して、再試行が許可されないことを示します。

retry_delay

retry_delay は、ROLLBACK の後にメッセージの再処理がスケジュールされるまでの秒数を指定します。0 を指定して、メッセージをすぐに再試行することを示します（デフォルト）。

retention_time

retention_time は、メッセージがデキューされてから保存される時間の長さ（秒単位）を指定します。また、0（デフォルト）を指定して、デキュー後にメッセージを保持しないことを示すことも、INFINITE を指定してメッセージを永久に保持することもできます。

auto_commit

このパラメーターは互換性のために受け入れられ、無視されます。

comment

comment は、キューに関連付けられたコメントを指定します。

Example

次のコマンドは、work_order という名前のキューを変更し、retry_delay パラメーターを 5 秒に設定します。

```
EXEC DBMS_AQADM.ALTER_QUEUE(queue_name => 'work_order', retry_delay => 5);
```

3.3.2 ALTER_QUEUE_TABLE

ALTER_QUEUE_TABLE 手順を使用して、既存のキューテーブルを変更します。署名は次のとおりです。

```
ALTER_QUEUE_TABLE (  
  <queue_table> IN VARCHAR2,  
  <comment> IN VARCHAR2 DEFAULT NULL,  
  <primary_instance> IN BINARY_INTEGER DEFAULT 0,  
  <secondary_instance> IN BINARY_INTEGER DEFAULT 0,
```

Parameters

queue_table

キューテーブルの（オプションでスキーマ修飾された）名前。

comment

comment パラメータを使用して、キューテーブルに関するコメントを提供します。

primary_instance

primary_instance は互換性のために受け入れられて保存されますが、無視されます。

secondary_instance

secondary_instance は互換性のために受け入れられますが、無視されます。

Example

次のコマンドは、work_order_table という名前のキューテーブルを変更します。

```
EXEC DBMS_AQADM.ALTER_QUEUE_TABLE  
  (queue_table => 'work_order_table', comment => 'This queue table  
contains work orders for the shipping department.');
```

キューテーブルの名前は work_order_table です。このコマンドは、キューテーブルの定義にコメントを追加します。

3.3.3 CREATE_QUEUE

CREATE_QUEUE 手順を使用して、既存のキューテーブルにキューを作成します。署名は次のとおりです。

```
CREATE_QUEUE (
  <queue_name> IN VARCHAR2
  <queue_table> IN VARCHAR2,
  <queue_type> IN BINARY_INTEGER DEFAULT NORMAL_QUEUE,
  <max_retries> IN NUMBER DEFAULT 5,
  <retry_delay> IN NUMBER DEFAULT 0
  <retention_time> IN NUMBER DEFAULT 0,
  <dependency_tracking> IN BOOLEAN DEFAULT FALSE,
  <comment> IN VARCHAR2 DEFAULT NULL,
  <auto_commit> IN BOOLEAN DEFAULT TRUE)
```

Parameters

queue_name

新しいキューの名前。

queue_table

新しいキューが存在するテーブルの名前。

queue_type

新しいキューのタイプ。 queue_type の有効な値は次のとおりです。

DBMS_AQADM.NORMAL_QUEUE –この値は通常のキューを指定します（デフォルト）。

DBMS_AQADM.EXCEPTION_QUEUE –この値は、新しいキューが例外キューであることを指定します。例外キューは、デキュー操作のみをサポートします。

max_retries

max_retries は、デキューステートメントでメッセージを削除する最大試行回数を指定します。max_retries の値は、ROLLBACK ステートメントごとに増分されます。失敗した試行回数が max_retries で指定された値に達すると、メッセージは例外キューに移動します。システムテーブルのデフォルト値は 0 です。ユーザーが作成したテーブルのデフォルト値は 5 です。

retry_delay

retry_delay は、ROLLBACK の後にメッセージの再処理がスケジュールされるまでの秒数を指定します。0 を指定して、メッセージをすぐに再試行することを示します（デフォルト）。

retention_time

retention_time は、メッセージがデキューされてから保存される時間の長さ（秒単位）を指定します。また、0（デフォルト）を指定して、デキュー後にメッセージを保持しないことを示すことも、INFINITE

を指定してメッセージを永久に保持することもできます。

dependency_tracking

このパラメーターは互換性のために受け入れられ、無視されます。

comment

comment は、キューに関連付けられたコメントを指定します。

auto_commit

このパラメーターは互換性のために受け入れられ、無視されます。

Example

次の匿名ブロックは、work_order_table テーブルに work_order という名前のキューを作成します。

```
BEGIN
DBMS_AQADM.CREATE_QUEUE ( queue_name => 'work_order', queue_table =>
'work_order_table', comment => 'This queue contains pending work orders.');
```

```
END;
```

3.3.4 CREATE_QUEUE_TABLE

CREATE_QUEUE_TABLE 手順を使用して、キューテーブルを作成します。署名は次のとおりです。

```
CREATE_QUEUE_TABLE (
  <queue_table> IN VARCHAR2,
  <queue_payload_type> IN VARCHAR2,
  <storage_clause> IN VARCHAR2 DEFAULT NULL,
  <sort_list> IN VARCHAR2 DEFAULT NULL,
  <multiple_consumers> IN BOOLEAN DEFAULT FALSE,
  <message_grouping> IN BINARY_INTEGER DEFAULT NONE,
  <comment> IN VARCHAR2 DEFAULT NULL,
  <auto_commit> IN BOOLEAN DEFAULT TRUE,
  <primary_instance> IN BINARY_INTEGER DEFAULT 0,
  <secondary_instance> IN BINARY_INTEGER DEFAULT 0,
  <compatible> IN VARCHAR2 DEFAULT NULL,
  <secure> IN BOOLEAN DEFAULT FALSE)
```

Parameters

queue_table

キューテーブルの（オプションでスキーマ修飾された）名前。

queue_payload_type

キューテーブルに格納されるデータのユーザー定義タイプ。RAW データ型を指定するには、RAW 型を識別するユーザー定義型を作成する必要があることに注意してください。

storage_clause

storage_clause パラメータを使用して、キューテーブルの属性を指定します。TABLESPACE オプションのみが適用されることに注意してください。他のすべては互換性のために受け入れられ、無視されます。TABLESPACE 句を使用して、テーブルが作成されるテーブルスペースの名前を指定します。

storage_clause は次のうちの 1 つ以上です。

```
TABLESPACE tablespace_name, PCTFREE integer, PCTUSED integer,
INITRANS integer, MAXTRANS integer or STORAGE storage_option.
```

storage_option は次のうちの 1 つ以上です。

```
MINEXTENTS integer, MAXEXTENTS integer, PCTINCREASE integer, INITIAL
size_clause, NEXT, FREELISTS integer, OPTIMAL size_clause, BUFFER_
POOL {KEEP|RECYCLE|DEFAULT}.
```

sort_list

sort_list はキューのデキュー順序を制御します。キューのソートに使用される列の名前を（昇順で）指定します。現在受け入れられている値は、enq_time と priority の次の組み合わせです。

enq_time, priority

priority, enq_time

priority

enq_time

multiple_consumers

multiple_consumers キューテーブルはサポートされていません。

message_grouping

指定する場合、message_grouping は NONE でなければなりません。

comment

comment パラメータを使用して、キューテーブルに関するコメントを提供します。

auto_commit

auto_commit は互換性のために受け入れられますが、無視されます。

primary_instance

`primary_instance` は互換性のために受け入れられて保存されますが、無視されます。

`secondary_instance`

`secondary_instance` は互換性のために受け入れられますが、無視されます。

`compatible`

`compatible` は互換性のために受け入れられますが、無視されます。

`secure`

`secure` は互換性のために受け入れられますが、無視されます。

Example

次の匿名ブロックは、名前（ `VARCHAR2` ）とプロジェクトの説明（ `TEXT` ）を保持する属性を持つタイプ（ `work_order` ）を最初に作成します。次に、ブロックはそのタイプを使用してキューテーブルを作成します。

```
BEGIN

CREATE TYPE work_order AS (name VARCHAR2, project TEXT, completed BOOLEAN);

EXEC DBMS_AQADM.CREATE_QUEUE_TABLE
    (queue_table => 'work_order_table',
     queue_payload_type => 'work_order',
     comment => 'Work order message queue table');

END;
```

キューテーブルの名前は `work_order_table` で、タイプ `work_order` のペイロードが含まれています。コメントは、これが `Work order message queue table` であることを示しています。

3.3.5 DROP_QUEUE

キューを削除するには、`DROP_QUEUE` 手順を使用します。署名は次のとおりです。

```
DROP_QUEUE (
    <queue_name>   IN VARCHAR2,
    <auto_commit>  IN BOOLEAN DEFAULT TRUE)
```

Parameters

`queue_name`

削除するキューの名前。

`auto_commit`

`auto_commit` は互換性のために受け入れられますが、無視されます。

Example

次の匿名ブロックは、`work_order` という名前のキューを削除します。

```
BEGIN
DBMS_AQADM.DROP_QUEUE(queue_name => 'work_order');
END;
```

3.3.6 DROP_QUEUE_TABLE

キューテーブルを削除するには、`DROP_QUEUE_TABLE` 手順を使用します。署名は次のとおりです。

```
DROP_QUEUE_TABLE (
  <queue_table> IN VARCHAR2,
  <force> IN BOOLEAN default FALSE,
  <auto_commit> IN BOOLEAN default TRUE)
```

Parameters

`queue_table`

キューテーブルの（オプションでスキーマ修飾された）名前。

`force`

`force` キーワードは、エントリを含むテーブルを削除するときの `DROP_QUEUE_TABLE` コマンドの動作を決定します。

ターゲットテーブルにエントリが含まれ、`force` が `FALSE` の場合、コマンドは失敗し、サーバーはエラーを発行します。

ターゲットテーブルにエントリが含まれ、`force` が `TRUE` の場合、コマンドはテーブルと依存オブジェクトをドロップします。

`auto_commit`

`auto_commit` は互換性のために受け入れられますが、無視されます。

Example

次の匿名ブロックは、`work_order_table` という名前のテーブルを削除します。

```
BEGIN
  DBMS_AQADM.DROP_QUEUE_TABLE ('work_order_table', force => TRUE);
END;
```

3.3.7 PURGE_QUEUE_TABLE

キューテーブルからメッセージを削除するには、PURGE_QUEUE_TABLE 手順を使用します。署名は次のとおりです。

```
PURGE_QUEUE_TABLE (
    <queue_table> IN VARCHAR2,
    <purge_condition> IN VARCHAR2,
    <purge_options> IN aq$_purge_options_t)
```

Parameters

queue_table

queue_table は、メッセージを削除するキューテーブルの名前を指定します。

purge_condition

purge_condition を使用して、パージするメッセージを決定するときにサーバーが評価する条件（SQL の WHERE 句）を指定します。

purge_options

purge_options は aq\$_purge_options_t タイプのオブジェクトです。aq\$_purge_options_t オブジェクトには以下が含まれます。

属性	種類	説明
Block	ブール	テーブル内のすべてのキューで排他ロックを保持する必要がある場合は、TRUE を指定します。デフォルトは FALSE です。
delivery_mode	整数	delivery_mode は、パージされるメッセージのタイプを指定します。受け入れられる値は DBMS_AQ.PERSISTENT のみです。

Example

次の匿名ブロックは、YES の completed 列に値を持つ work_order_table からすべてのメッセージを削除します。

```
DECLARE
    purge_options dbms_aqadm.aq$_purge_options_t;
BEGIN
    dbms_aqadm.purge_queue_table('work_order_table', 'completed = YES',
    purge_options);
END;
```

3.3.8 START_QUEUE

START_QUEUE プロシージャを使用して、キューをエンキューおよびデキューできるようにします。署名は次のとおりです。

```
START_QUEUE (
    <queue_name> IN VARCHAR2,
    <enqueue> IN BOOLEAN DEFAULT TRUE,
    <dequeue> IN BOOLEAN DEFAULT TRUE)
```

Parameters

queue_name

queue_name は、開始するキューの名前を指定します。

enqueue

エンキューを有効にするには TRUE を指定し（デフォルト）、現在の設定を変更しない場合は FALSE を指定します。

dequeue

TRUE を指定するとデキューが有効になり（デフォルト）、FALSE を指定すると現在の設定が変更されません。

Example

次の匿名ブロックは、work_order という名前のキューをエンキューに使用できるようにします。

```
BEGIN
DBMS_AQADM.START_QUEUE
(queue_name => 'work_order');
END;
```

3.3.9 STOP_QUEUE

STOP_QUEUE プロシージャを使用して、指定されたキューでのエンキューまたはデキューを無効にします。署名は次のとおりです。

```
STOP_QUEUE (
    <queue_name> IN VARCHAR2,
    <enqueue> IN BOOLEAN DEFAULT TRUE,
    <dequeue> IN BOOLEAN DEFAULT TRUE,
    <wait> IN BOOLEAN DEFAULT TRUE)
```

Parameters

queue_name

queue_name は、停止するキューの名前を指定します。

enqueue

エンキューを無効にするには TRUE を指定し（デフォルト）、現在の設定を変更しない場合は FALSE を指定します。

dequeue

TRUE を指定するとデキューが無効になり（デフォルト）、FALSE を指定すると現在の設定が変更されません。

wait

TRUE を指定すると、指定された変更を適用する前に未完了のトランザクションが完了するのを待つようサーバーに指示します。キューの停止を待機している間、指定されたキューからのエンキューまたはデキューが許可されているトランザクションはありません。キューをすぐに停止するには、FALSE を指定します。

Example

次の匿名ブロックは、work_order という名前のキューからのエンキューおよびデキューを無効にします。

```
BEGIN
DBMS_AQADM.STOP_QUEUE(queue_name =>'work_order', enqueue=>TRUE,
dequeue=>TRUE, wait=>TRUE);
END;
```

未処理のトランザクションが完了すると、エンキューとデキューが停止します。

3.4 DBMS_CRYPTO

DBMS_CRYPTO パッケージには、RAW、BLOB または CLOB データの暗号化または復号化を可能にする機能と手順が含まれています。DBMS_CRYPTO 関数を使用して、暗号的に強力なランダム値を生成することもできます。

次の表に、DBMS_CRYPTO 関数と手順を示します。

機能/手順	戻りタイプ	説明
DECRYPT(src, typ, key, iv)	RAW	RAW データを解読します。
DECRYPT(dst INOUT, src, typ, key, iv)	なし	BLOB データを解読します。
DECRYPT(dst INOUT, src, typ, key, iv)	なし	CLOB データを解読します。
ENCRYPT(src, typ, key, iv)	RAW	RAW データを暗号化します。
ENCRYPT(dst INOUT, src, typ, key, iv)	なし	BLOB データを暗号化します。
ENCRYPT(dst INOUT, src, typ, key, iv)	なし	CLOB データを暗号化します。
HASH(src, typ)	RAW	RAW データにハッシュアルゴリズムを適用します。
HASH(src)	RAW	CLOB データにハッシュアルゴリズムを適用します。
MAC(src, typ, key)	RAW	指定されたハッシュアルゴリズムとキーを使用して、指定された RAW データのハッシュ MAC 値を返します。
MAC(src, typ, key)	RAW	指定されたハッシュアルゴリズムとキーを使用して、指定された CLOB データのハッシュ MAC 値を返します。
RANDOMBYTES(number_bytes)	RAW	指定された数の暗号的に強力なランダムバイトを返します。
RANDOMINTEGER()	INTEGER	ランダムな INTEGER を返します。
RANDOMNUMBER()	NUMBER	ランダムな NUMBER. を返します

DBMS_CRYPTO 関数およびプロシージャは、次のエラーメッセージをサポートします。

ORA-28239 - DBMS_CRYPTO.KeyNull

ORA-28829 - DBMS_CRYPTO.CipherSuiteNull

ORA-28827 - DBMS_CRYPTO.CipherSuiteInvalid

Oracle とは異なり、AdvancedServer は、以前に暗号化された情報を再暗号化してもエラー ORA-28233 を返しま

せん。

RAW と BLOB は PostgreSQL の BYTEA データ型の同義語であり、CLOB は TEXT の同義語であることに注意してください。

3.4.1 DECRYPT

DECRYPT 関数またはプロシージャは、ユーザー指定の暗号アルゴリズム、キー、およびオプションの初期化ベクトルを使用してデータを復号化します。DECRYPT 関数のシグネチャは次のとおりです。

```
DECRYPT
(<src> IN RAW, <typ> IN INTEGER, <key> IN RAW, <iv> IN RAW
DEFAULT NULL) RETURN RAW
```

DECRYPT プロシージャのシグネチャは次のとおりです。

```
DECRYPT
(<dst> INOUT BLOB, <src> IN BLOB, <typ> IN INTEGER, <key> IN RAW,
<iv> IN RAW DEFAULT NULL)
```

または

```
DECRYPT
(<dst> INOUT CLOB, <src> IN CLOB, <typ> IN INTEGER, <key> IN RAW,
<iv> IN RAW DEFAULT NULL)
```

プロシージャとして呼び出されると、DECRYPT は BLOB または CLOB のデータをユーザー指定の BLOB に返します。

Parameters

`dst`

`dst` は、DECRYPT プロシージャの出力が書き込まれる BLOB の名前を指定します。DECRYPT 手順は、`dst` に現在ある既存のデータを上書きします。

`src`

`src` は、復号化されるソースデータを指定します。関数として DECRYPT を呼び出す場合は、RAW データを指定します。手順として DECRYPT を呼び出す場合は、BLOB または CLOB のデータを指定します。

`typ`

`typ` は、ブロック暗号タイプと修飾子を指定します。これは、`src` が暗号化されたときに指定されたタイプと一致する必要があります。AdvancedServer は、次のブロック暗号アルゴリズム、修飾子、および暗号スイートをサポートしています。

Block Cipher Algorithms	
ENCRYPT_DES	CONSTANT INTEGER:= 1;
ENCRYPT_3DES	CONSTANT INTEGER:= 3;
ENCRYPT_AES	CONSTANT INTEGER:= 4;
ENCRYPT_AES128	CONSTANT INTEGER:= 6;
ENCRYPT_AES192	CONSTANT INTEGER:= 192;
ENCRYPT_AES256	CONSTANT INTEGER:= 256;
Block Cipher Modifiers	
CHAIN_CBC	CONSTANT INTEGER:= 256;
CHAIN_ECB	CONSTANT INTEGER:= 768;
Block Cipher Padding Modifiers	
PAD_PKCS5	CONSTANT INTEGER:= 4096;
PAD_NONE	CONSTANT INTEGER:= 8192;
Block Cipher Suites	
DES_CBC_PKCS5	CONSTANT INTEGER:= ENCRYPT_DES + CHAIN_CBC + PAD_PKCS5;
DES3_CBC_PKCS5	CONSTANT INTEGER:= ENCRYPT_3DES + CHAIN_CBC + PAD_PKCS5;
AES_CBC_PKCS5	CONSTANT INTEGER:= ENCRYPT_AES + CHAIN_CBC + PAD_PKCS5;

key

key は、ユーザー定義の復号化キーを指定します。これは、src が暗号化されたときに指定されたキーと一致する必要があります。

iv

iv（オプション）は初期化ベクトルを指定します。src が暗号化されたときに初期化ベクトルが指定された場合、src を復号化するときに初期化ベクトルを指定する必要があります。デフォルトは NULL です。

Examples

次の例では、DBMS_CRYPTO.DECRYPT 関数を使用して、passwords テーブルから取得した暗号化パスワードを復号化します。

```
CREATE TABLE passwords
(
  principal VARCHAR2(90) PRIMARY KEY, -- username
  ciphertext RAW(9) -- encrypted password
);

CREATE FUNCTION get_password(username VARCHAR2) RETURN RAW AS
  typ      INTEGER := DBMS_CRYPTO.DES_CBC_PKCS5;
  key      RAW(128) := 'my secret key';
  iv       RAW(100) := 'my initialization vector';
```

(次のページに続く)

(前のページからの続き)

```
password RAW(2048);
BEGIN

    SELECT ciphertext INTO password FROM passwords WHERE principal = username;

    RETURN dbms_crypto.decrypt(password, typ, key, iv);
END;
```

DECRYPT を呼び出すときは、ENCRYPTING がターゲットのときに使用されたものと同じ暗号タイプ、キー値、および初期化ベクトルを渡す必要があることに注意してください。

3.4.2 ENCRYPT

ENCRYPT 関数または手順は、ユーザー指定のアルゴリズム、キー、およびオプションの初期化ベクトルを使用して、RAW、BLOB または CLOB データを暗号化します。ENCRYPT 関数のシグネチャは次のとおりです。

```
ENCRYPT
(<src> IN RAW, <typ> IN INTEGER, <key> IN RAW,
 <iv> IN RAW DEFAULT NULL) RETURN RAW
```

ENCRYPT プロシージャのシグネチャは次のとおりです。

```
ENCRYPT
(<dst> INOUT BLOB, <src> IN BLOB, <typ> IN INTEGER, <key> IN RAW,
 <iv> IN RAW DEFAULT NULL)
```

または

```
ENCRYPT
(<dst> INOUT BLOB, <src> IN CLOB, <typ> IN INTEGER, <key> IN RAW,
 <iv> IN RAW DEFAULT NULL)
```

プロシージャとして呼び出されると、ENCRYPT は BLOB または CLOB のデータをユーザー指定の BLOB に返します。

Parameters

dst

dst は、ENCRYPT プロシージャの出力が書き込まれる BLOB の名前を指定します。ENCRYPT 手順は、dst に現在ある既存のデータを上書きします。

src

`src` は、暗号化されるソースデータを指定します。関数として `ENCRYPT` を呼び出す場合は、`RAW` データを指定します。手順として `ENCRYPT` を呼び出す場合は、`BLOB` または `CLOB` のデータを指定します。

`typ`

`typ` は、`ENCRYPT` で使用されるブロック暗号タイプと任意の修飾子を指定します。`AdvancedServer` は、以下にリストされているブロック暗号アルゴリズム、修飾子、および暗号スイートをサポートしています。

Block Cipher Algorithms	
<code>ENCRYPT_DES</code>	<code>CONSTANT INTEGER:= 1;</code>
<code>ENCRYPT_3DES</code>	<code>CONSTANT INTEGER:= 3;</code>
<code>ENCRYPT_AES</code>	<code>CONSTANT INTEGER:= 4;</code>
<code>ENCRYPT_AES128</code>	<code>CONSTANT INTEGER:= 6;</code>
<code>ENCRYPT_AES192</code>	<code>CONSTANT INTEGER:= 192;</code>
<code>ENCRYPT_AES256</code>	<code>CONSTANT INTEGER:= 256;</code>
Block Cipher Modifiers	
<code>CHAIN_CBC</code>	<code>CONSTANT INTEGER:= 256;</code>
<code>CHAIN_ECB</code>	<code>CONSTANT INTEGER:= 768;</code>
Block Cipher Padding Modifiers	
<code>PAD_PKCS5</code>	<code>CONSTANT INTEGER:= 4096;</code>
<code>PAD_NONE</code>	<code>CONSTANT INTEGER:= 8192;</code>
Block Cipher Suites	
<code>DES_CBC_PKCS5</code>	<code>CONSTANT INTEGER:= ENCRYPT_DES + CHAIN_CBC + PAD_PKCS5;</code>
<code>DES3_CBC_PKCS5</code>	<code>CONSTANT INTEGER:= ENCRYPT_3DES + CHAIN_CBC + PAD_PKCS5;</code>
<code>AES_CBC_PKCS5</code>	<code>CONSTANT INTEGER:= ENCRYPT_AES + CHAIN_CBC + PAD_PKCS5;</code>

`key`

`key` は暗号化キーを指定します。

`iv`

`iv`（オプション）は初期化ベクトルを指定します。デフォルトでは、`iv` は `NULL` です。

Examples

次の例では、`DBMS_CRYPTO.DES_CBC_PKCS5` ブロック暗号スイート（事前定義されたアルゴリズムと修飾子のセット）を使用して、`passwords` テーブルから取得した値を暗号化します。

```
CREATE TABLE passwords
(
  principal  VARCHAR2(90) PRIMARY KEY, -- username
  ciphertext RAW(9) -- encrypted password
);
```

(次のページに続く)

(前のページからの続き)

```

CREATE PROCEDURE set_password(username VARCHAR2, cleartext RAW) AS
  typ          INTEGER := DBMS_CRYPTO.DES_CBC_PKCS5;
  key          RAW(128) := 'my secret key';
  iv           RAW(100) := 'my initialization vector';
  encrypted    RAW(2048);
BEGIN
  encrypted := dbms_crypto.encrypt(cleartext, typ, key, iv);
  UPDATE passwords SET ciphertext = encrypted WHERE principal = username;
END;

```

ENCRYPT は password を暗号化するとき my secret key のキー値と my initialization vector の初期化ベクトルを使用します。password を復号化するとき同じキーと初期化ベクトルを指定します。

3.4.3 HASH

HASH 関数は、ユーザー指定のアルゴリズムを使用して、RAW または CLOB 値のハッシュ値を返します。HASH 機能は、次の 3 つの形式で使用できます。

```

HASH
  (<src> IN RAW, <typ> IN INTEGER) RETURN RAW

HASH
  (<src> IN CLOB, <typ> IN INTEGER) RETURN RAW

```

Parameters

src

src は、ハッシュ値が生成される値を指定します。RAW、BLOB、または CLOB の値を指定できます。

typ

typ は HASH 機能タイプを指定します。AdvancedServer は、以下にリストされている HASH 機能タイプをサポートしています。

HASH Functions	
HASH_MD4	CONSTANT INTEGER:= 1;
HASH_MD5	CONSTANT INTEGER:= 2;
HASH_SH1	CONSTANT INTEGER:= 3;

Examples

次の例では、DBMS_CRYPTO.HASH を使用して、文字列 cleartext source の md5 ハッシュ値を検索します。

```

DECLARE
    typ INTEGER := DBMS_CRYPTO.HASH_MD5;
    hash_value RAW(100);
BEGIN

    hash_value := DBMS_CRYPTO.HASH('cleartext source', typ);

END;

```

3.4.4 MAC

MAC 関数は、ユーザー指定の MAC 関数を使用して、RAW または CLOB 値のハッシュされた MAC 値を返します。MAC 機能は、次の 3 つの形式で使用できます。

```

MAC
    (<src> IN RAW, <typ> IN INTEGER, <key> IN RAW) RETURN RAW

MAC
    (<src> IN CLOB, <typ> IN INTEGER, <key> IN RAW) RETURN RAW

```

Parameters

src

src は、MAC 値が生成される値を指定します。RAW、BLOB、または CLOB の値を指定します。

typ

typ は使用する MAC 機能を指定します。AdvancedServer は、以下の MAC 機能をサポートしています。

MAC Functions	
HMAC_MD5	CONSTANT INTEGER:= 1;
HMAC_SH1	CONSTANT INTEGER:= 2;

key

key は、ハッシュされた MAC 値の計算に使用されるキーを指定します。

Examples

次の例では、文字列 cleartext source のハッシュ値 MAC を検索します。

```

DECLARE
    typ INTEGER := DBMS_CRYPTO.HMAC_MD5;
    key RAW(100) := 'my secret key';

```

(次のページに続く)

(前のページからの続き)

```

    mac_value RAW(100);
BEGIN

    mac_value := DBMS_CRYPTO.MAC('cleartext source', typ, key);

END;
```

DBMS_CRYPTO.MAC は、cleartext source の MAC 値を計算するときに my_secret キーのキー値を使用します。

3.4.5 RANDOMBYTES

RANDOMBYTES 関数は、暗号的にランダムなバイトを含む、指定された長さの RAW 値を返します。署名は次のとおりです。

```

RANDOMBYTES
(<number_bytes> IN INTEGER) RETURNS RAW
```

Parameter

number_bytes

number_bytes は、返されるランダムなバイト数を指定します

Examples

次の例では、RANDOMBYTES を使用して 1024 バイト長の値を返します。

```

DECLARE
    result RAW(1024);
BEGIN
    result := DBMS_CRYPTO.RANDOMBYTES(1024);
END;
```

3.4.6 RANDOMINTEGER

RANDOMINTEGER() 関数は、0 と 268,435,455 の間のランダムな INTEGER を返します。署名は次のとおりです。

```

RANDOMINTEGER() RETURNS INTEGER
```

Examples

次の例では、RANDOMINTEGER 関数を使用して、暗号的に強力なランダムな INTEGER 値を返します。

```
DECLARE
    result INTEGER;
BEGIN
    result := DBMS_CRYPTO.RANDOMINTEGER();
    DBMS_OUTPUT.PUT_LINE(result);
END;
```

3.4.7 RANDOMNUMBER

RANDOMNUMBER() 関数は、0 と 268,435,455 の間のランダムな NUMBER を返します。署名は次のとおりです。

```
RANDOMNUMBER() RETURNS NUMBER
```

Examples

次の例では、RANDOMNUMBER 関数を使用して、暗号的に強力な乱数を返します。

```
DECLARE
    result NUMBER;
BEGIN
    result := DBMS_CRYPTO.RANDOMNUMBER();
    DBMS_OUTPUT.PUT_LINE(result);
END;
```


3.5 DBMS_JOB

DBMS_JOB パッケージは、ジョブの作成、スケジューリング、および管理を提供します。ジョブは、以前にデータベースに保存されたストアードプロシージャを実行します。SUBMIT プロシージャは、ジョブ定義を作成および保存するために使用されます。ジョブ識別子は、関連付けられたストアードプロシージャと、ジョブを実行するタイミングと頻度を説明する属性とともに、ジョブに割り当てられます。

このパッケージは pgAgent スケジューラに依存しています。デフォルトでは、AdvancedServer インストーラーは pgAgent をインストールしますが、DBMS_JOB を使用する前に pgAgent サービスを手動で開始する必要があります。pgAgent, DBMS_JOB をアンインストールした後にこのパッケージを使用してジョブをスケジュールしようとする、エラーがスローされます。DBMS_JOB は pgAgent がインストールされていることを確認しますが、サービスが実行されていることは確認しません。

次の表に、サポートされている DBMS_JOB 手順を示します。

機能/手順	戻りタイプ	説明
BROKEN(job, broken [, next_date])	該当なし	特定のジョブが破損しているか、破損していないことを指定します。
CHANGE(job, what, next_date, interval, instance, force>)	該当なし	ジョブのパラメーターを変更します。
INTERVAL(job, interval)	該当なし	ジョブが実行されるたびに再計算される日付関数を使用して、実行頻度を設定します。この値は、実行の次の日付/時刻になります。
NEXT_DATE(job, next_date)	該当なし	ジョブを実行する次の日付/時刻を設定します。
REMOVE(job)	該当なし	データベースからジョブ定義を削除します。
RUN(job)	該当なし	破損とマークされている場合でも、ジョブを強制的に実行します。
SUBMIT(job OUT, what [, next_date [, interval [, no_parse]]])	該当なし	ジョブを作成し、その定義をデータベースに保存します。
WHAT(job, what)	該当なし	ジョブによって実行されるストアードプロシージャを変更します。

AdvancedServer の DBMS_JOB の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表にリストされている関数と手順のみがサポートされています。

DBMS_JOB を使用する前に、データベースのスーパーユーザーは pgAgent 拡張子を作成する必要があります。psql クライアントを使用してデータベースに接続し、コマンドを呼び出します:

```
CREATE EXTENSION pgagent;
```

ジョブが実行されるタイミングと頻度は、相互作用する 2 つのパラメーター `next_date` と `interval` に依存します。 `next_date` パラメータは、ジョブが実行される次の日付/時刻を指定する日付/時刻値です。 `interval` パラメータは、日付/時刻値に評価される日付関数を含む文字列です。

ジョブを実行する直前に、 `interval` パラメーターの式が評価されます。結果の値は、ジョブとともに保存されている `next_date` 値を置き換えます。その後、ジョブが実行されます。このようにして、 `interval` の式は各ジョブの実行前に繰り返し再評価され、次の実行に `next_date` の日付/時刻が提供されます。

注釈: データベースユーザーは、 `pgAgent` サーバーを起動してジョブを実行するジョブとスケジュールを作成したユーザーと同じである必要があります。

次の例では、次のストアードプロシージャ `job_proc` を使用しています。これは、単一の `VARCHAR2` 列を含むテーブル `jobrun` にタイムスタンプを挿入するだけです。

```
CREATE TABLE jobrun (  
    runtime VARCHAR2(40)  
);  
  
CREATE OR REPLACE PROCEDURE job_proc  
IS  
BEGIN  
    INSERT INTO jobrun VALUES ('job_proc run at ' || TO_CHAR(SYSDATE,  
        'yyyy-mm-dd hh24:mi:ss'));  
END;
```

3.5.1 BROKEN

BROKEN プロシージャは、ジョブの状態を破損または未破損のいずれかに設定します。破損したジョブは、RUN 手順を使用しない限り実行できません。

```
BROKEN(<job> BINARY_INTEGER, <broken> BOOLEAN [, <next_date> DATE ])
```

Parameters

job

壊れているまたは壊れていないとして設定されるジョブの識別子。

broken

TRUE に設定すると、ジョブの状態は破損に設定されます。FALSE に設定すると、ジョブの状態は破損しないように設定されます。壊れたジョブは、RUN 手順を使用しない限り実行できません。

next_date

ジョブを実行する日付/時刻。デフォルトは SYSDATE です。

Examples

ジョブ ID104 のジョブの状態を破損に設定します。

```
BEGIN
    DBMS_JOB.BROKEN(104,true);
END;
```

状態を破損していない状態に戻します。

```
BEGIN
    DBMS_JOB.BROKEN(104,false);
END;
```

3.5.2 CHANGE

CHANGE プロシージャは、実行されるストアードプロシージャ、ジョブが次に実行される日付/時刻、および実行される頻度を含む特定のジョブ属性を変更します。

```
CHANGE(<job> BINARY_INTEGER <what> VARCHAR2, <next_date> DATE,
      <interval> VARCHAR2, <instance> BINARY_INTEGER, <force> BOOLEAN)
```

Parameters

job

変更するジョブの識別子。

what

ストアードプロシージャ名。既存の値を変更しない場合は、このパラメーターを `null` に設定します。

next_date

ジョブが次に実行される日付/時刻。既存の値を変更しない場合は、このパラメーターを `null` に設定します。

interval

評価されると、ジョブが実行される次の日付/時刻を提供する日付関数。既存の値を変更しない場合は、このパラメーターを `null` に設定します。

instance

この引数は無視されますが、互換性のために含まれています。

force

この引数は無視されますが、互換性のために含まれています。

Examples

次に実行するジョブを 2007 年 12 月 13 日に変更します。他のパラメーターは変更しないでください。

```
BEGIN
    DBMS_JOB.CHANGE(104,NULL,TO_DATE('13-DEC-07','DD-MON-YY'),NULL, NULL,
    NULL);
END;
```

3.5.3 INTERVAL

INTERVAL プロシージャは、ジョブを実行する頻度を設定します。

```
INTERVAL(<job> BINARY_INTEGER, <interval> VARCHAR2)
```

Parameters

job

変更するジョブの識別子。

interval

評価されると、ジョブが実行される次の日付/時刻を提供する日付関数。 `interval` が `NULL` で、ジョブが完了すると、ジョブはキューから削除されます。

Examples

週に 1 回実行するようにジョブを変更します。

```
BEGIN
    DBMS_JOB.INTERVAL(104, 'SYSDATE + 7');
END;
```

3.5.4 NEXT_DATE

NEXT_DATE プロシージャは、ジョブが次に実行される日付/時刻を設定します。

```
NEXT_DATE(<job> BINARY_INTEGER, <next_date> DATE)
```

Parameters

job

次の実行日が設定されるジョブの識別子。

next_date

ジョブが次に実行される日付/時刻。

Examples

2007 年 12 月 14 日に次に実行するジョブを変更します。

```
BEGIN
    DBMS_JOB.NEXT_DATE(104, TO_DATE('14-DEC-07', 'DD-MON-YY'));
END;
```

3.5.5 削除

REMOVE プロシージャは、指定されたジョブをデータベースから削除します。ジョブを再実行するには、SUBMIT 手順を使用してジョブを再送信する必要があります。ジョブに関連付けられたストアードプロシージャは削除されないことに注意してください。

```
REMOVE(<job> BINARY_INTEGER)
```

Parameter

job

データベースから削除されるジョブの識別子。

Examples

データベースからジョブを削除します。

```
BEGIN
    DBMS_JOB.REMOVE(104);
END;
```

3.5.6 RUN

RUN プロシージャは、ジョブの状態が壊れていても、ジョブを強制的に実行します。

```
RUN(<job> BINARY_INTEGER)
```

Parameter

job

実行されるジョブの識別子。

Examples

ジョブを強制的に実行します。

```
BEGIN
    DBMS_JOB.RUN(104);
END;
```

3.5.7 SUBMIT

SUBMIT プロシージャは、ジョブ定義を作成し、データベースに保存します。ジョブは、ジョブ識別子、実行するストアードプロシージャ、ジョブを最初に実行するタイミング、およびジョブを次に実行する日付/時刻を計算する日付関数で構成されます。

```
SUBMIT(<job> OUT BINARY_INTEGER, <what> VARCHAR2
[, <next_date> DATE [, <interval> VARCHAR2 [, <no_parse> BOOLEAN ]]])
```

Parameters

job

ジョブに割り当てられた識別子。

what

ジョブによって実行されるストアードプロシージャの名前。

next_date

ジョブが次に実行される日付/時刻。デフォルトは SYSDATE です。

interval

評価されると、ジョブが実行される次の日付/時刻を提供する日付関数。interval が null に設定されている場合、ジョブは 1 回だけ実行されます。Null がデフォルトです。

no_parse

TRUE に設定した場合、ジョブの作成時にストアードプロシージャの構文チェックを行わず、ジョブが最初に実行されたときにのみチェックします。FALSE に設定した場合は、ジョブ作成時に手順を確認してください。デフォルトは FALSE です。

注釈: no_parse オプションは、SUBMIT() のこの実装ではサポートされていません。互換性のためにのみ含まれています。

Examples

次の例では、ストアードプロシージャ job_proc を使用してジョブを作成します。ジョブはすぐに実行され、interval パラメーター SYSDATE + 1 で設定されたように、その後 1 日に 1 回実行されます。

```
DECLARE
    jobid INTEGER;
BEGIN
    DBMS_JOB.SUBMIT(jobid, 'job_proc;', SYSDATE,
        'SYSDATE + 1');
    DBMS_OUTPUT.PUT_LINE('jobid: ' || jobid);
END;

jobid: 104
```

ジョブはすぐにプロシージャ job_proc を実行し、テーブル jobrun に行を入力します:

```
SELECT * FROM jobrun;

      runtime
-----
job_proc run at 2007-12-11 11:43:25
(1 row)
```

3.5.8 WHAT

WHAT プロシージャは、ジョブが実行するストアドプロシージャを変更します。

```
WHAT (<job> BINARY_INTEGER, <what> VARCHAR2)
```

Parameters

job

ストアドプロシージャを変更するジョブの識別子。

what

実行するストアドプロシージャの名前。

Examples

list_emp 手順を実行するようにジョブを変更します。

```
BEGIN
    DBMS_JOB.WHAT(104, 'list_emp;');
END;
```


3.6 DBMS_LOB

DBMS_LOB パッケージは、大きなオブジェクトを操作する機能を提供します。次の表に、サポートされている関数と手順を示します。

機能/手順	戻りタイプ	説明
APPEND(dest_lob IN OUT, src_lob)	該当なし	1 つの大きなオブジェクトを別のオブジェクトに追加します。
COMPARE(lob_1, lob_2 [, amount [, offset_1 [, offset_2]]])	INTEGER	2 つの大きなオブジェクトを比較します。
CONVERTOBLOB(dest_lob IN OUT, src_clob, amount, dest_offset IN OUT, src_offset IN OUT, blob_csid, lang_context IN OUT, <warning> OUT)	該当なし	文字データをバイナリに変換します。
CONVERTTOCLOB(dest_lob IN OUT, src_blob, amount, dest_offset IN OUT, src_offset IN OUT, blob_csid, lang_context IN OUT, warning OUT)	該当なし	バイナリデータを文字に変換します。
COPY(dest_lob IN OUT, src_lob, amount [, dest_offset [, src_offset]])	該当なし	1 つの大きなオブジェクトを別のオブジェクトにコピーします。
ERASE(lob_loc IN OUT, amount IN OUT [, offset])	該当なし	ラージオブジェクトを消去します。
GET_STORAGE_LIMIT(lob_loc)	INTEGER	ラージオブジェクトのストレージ制限を取得します。
GETLENGTH(lob_loc)	INTEGER	ラージオブジェクトの長さを取得します。
INSTR(lob_loc, pattern [, offset [, nth]])	INTEGER	offset から始まるラージオブジェクトのパターンの n 番目のオカレンスの位置を取得します
READ(lob_loc, amount IN OUT, offset, buffer OUT)	該当なし	ラージオブジェクトを読み取ります。
SUBSTR(lob_loc [, amount [, offset]])	RAW, VARCHAR2	ラージオブジェクトの一部を取得します。
TRIM(lob_loc IN OUT, newlen)	該当なし	ラージオブジェクトを指定の長さにトリミングします。
WRITE(lob_loc IN OUT, amount, offset, buffer)	該当なし	ラージオブジェクトにデータを書き込みます。
WRITEAPPEND(lob_loc IN OUT, amount, buffer)	該当なし	バッファからラージオブジェクトの最後にデータを書き込みます。

AdvancedServer の DBMS_LOB の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表にリストされている関数と手順のみがサポートされています。

次の表に、パッケージで使用可能なパブリック変数を示します。

Public Variables	Data Type	Value
compress_off	INTEGER	0
compress_on	INTEGER	1
deduplicate_off	INTEGER	0
deduplicate_on	INTEGER	4
default_csid	INTEGER	0
default_lang_ctx	INTEGER	0
encrypt_off	INTEGER	0
encrypt_on	INTEGER	1
file_readonly	INTEGER	0
lobmaxsize	INTEGER	1073741823
lob_readonly	INTEGER	0
lob_readwrite	INTEGER	1
no_warning	INTEGER	0
opt_compress	INTEGER	1
opt_deduplicate	INTEGER	4
opt_encrypt	INTEGER	2
warn_inconvertible_char	INTEGER	1

次のセクションでは、ラージオブジェクトが BLOBs の場合、長さとおフセットはバイト単位で測定されます。大きいオブジェクトが CLOBs の場合、長さとおフセットは文字で測定されます。

3.6.1 APPEND

APPEND プロシージャは、1 つの大きなオブジェクトを別のオブジェクトに追加する機能を提供します。両方のラージオブジェクトは同じタイプである必要があります。

```
APPEND(<dest_lob> IN OUT { BLOB | CLOB }, <src_lob> { BLOB | CLOB })
```

Parameters

dest_lob

宛先オブジェクトのラージオブジェクトロケータ。src_lob と同じデータ型でなければなりません。

src_lob

ソースオブジェクトのラージオブジェクトロケータ。dest_lob と同じデータ型でなければなりません。

3.6.2 COMPARE

COMPARE プロシージャは、指定されたオフセットで指定された長さの 2 つの大きなオブジェクトの正確なバイト単位の比較を実行します。比較するラージオブジェクトは同じデータ型である必要があります。

```
<status> INTEGER COMPARE(<lob_1> { BLOB | CLOB },  
    <lob_2> { BLOB | CLOB }  
    [, <amount> INTEGER [, <offset_1> INTEGER [, <offset_2> INTEGER ]]])
```

Parameters

lob_1

比較する最初のラージオブジェクトのラージオブジェクトロケータ。lob_2 と同じデータ型でなければなりません。

lob_2

比較する 2 番目のラージオブジェクトのラージオブジェクトロケータ。lob_1 と同じデータ型でなければなりません。

amount

ラージオブジェクトのデータタイプが BLOB の場合、amount バイトの比較が行われます。ラージオブジェクトのデータタイプが CLOB の場合、amount 文字に対して比較が行われます。デフォルトは、ラージオブジェクトの最大サイズです。

offset_1

比較を開始する最初のラージオブジェクト内に配置します。最初のバイト/文字はオフセット 1 です。デフォルトは 1 です。

offset_2

比較を開始する 2 番目のラージオブジェクト内に配置します。最初のバイト/文字はオフセット 1 です。デフォルトは 1 です。

status

両方の大きなオブジェクトが、指定されたオフセットに対して指定された長さでまったく同じ場合はゼロ。オブジェクトが同じでない場合、ゼロ以外。amount、offset_1、または offset_2 がゼロより小さい場合は NULL。

3.6.3 CONVERTTOBLOB

CONVERTTOBLOB プロシージャは、文字データをバイナリに変換する機能を提供します。

```
CONVERTTOBLOB(<dest_lob> IN OUT BLOB, <src_clob> CLOB,
  <amount> INTEGER, <dest_offset> IN OUT INTEGER,
  <src_offset> IN OUT INTEGER, <blob_csid> NUMBER,
  <lang_context> IN OUT INTEGER, <warning> OUT INTEGER)
```

Parameters

dest_lob

文字データの変換先の BLOB ラージオブジェクトロケーター。

src_clob

変換される文字データの CLOB ラージオブジェクトロケーター。

amount

変換する src_clob の文字数。

dest_offset IN

ソース CLOB の書き込みを開始する宛先 BLOB のバイト単位の位置。最初のバイトはオフセット 1 です。

dest_offset OUT

書き込み操作が完了した後の宛先 BLOB のバイト単位の位置。最初のバイトはオフセット 1 です。

src_offset IN

宛先 BLOB への変換を開始するソース CLOB 内の文字の位置。最初の文字はオフセット 1 です。

src_offset OUT

変換操作が完了した後のソース CLOB 内の文字の位置。最初の文字はオフセット 1 です。

blob_csid

変換された宛先 BLOB の文字セット ID。

lang_context IN

変換の言語コンテキスト。通常、この設定にはデフォルト値の 0 が使用されます。

lang_context OUT

変換完了後の言語コンテキスト。

warning

変換が成功した場合は 0、変換できない文字が検出された場合は 1。

3.6.4 CONVERTTOCLOB

CONVERTTOCLOB プロシージャは、バイナリデータを文字に変換する機能を提供します。

```
CONVERTTOCLOB(<dest_lob> IN OUT CLOB, <src_blob> BLOB,  
  <amount> INTEGER, <dest_offset> IN OUT INTEGER,  
  <src_offset> IN OUT INTEGER, <blob_csid> NUMBER,  
  <lang_context> IN OUT INTEGER, <warning> OUT INTEGER)
```

Parameters

dest_lob

バイナリデータの変換先の CLOB ラージオブジェクトロケター。

src_blob

変換されるバイナリデータの BLOB ラージオブジェクトロケター。

amount

変換する src_blob のバイト数。

dest_offset IN

ソース BLOB の書き込みを開始する宛先 CLOB の文字の位置。最初の文字はオフセット 1 です。

dest_offset OUT

書き込み操作が完了した後の宛先 CLOB の文字の位置。最初の文字はオフセット 1 です。

src_offset IN

宛先 CLOB への変換を開始するソース BLOB 内のバイト単位の位置。最初のバイトはオフセット 1 です。

src_offset OUT

変換操作が完了した後のソース BLOB のバイト単位の位置。最初のバイトはオフセット 1 です。

blob_csid

変換された宛先 CLOB の文字セット ID。

lang_context IN

変換の言語コンテキスト。通常、この設定にはデフォルト値の 0 が使用されます。

lang_context OUT

変換完了後の言語コンテキスト。

warning

変換が成功した場合は 0、変換できない文字が検出された場合は 1。

3.6.5 COPY

COPY プロシージャは、1 つの大きなオブジェクトを別の大きなオブジェクトにコピーする機能を提供します。ソースラージオブジェクトとデスティネーションラージオブジェクトは同じデータ型である必要があります。

```
COPY(<dest_lob> IN OUT { BLOB | CLOB }, <src_lob>
{ BLOB | CLOB },
<amount> INTEGER
[, <dest_offset> INTEGER [, <src_offset> INTEGER ]])
```

Parameters

dest_lob

src_lob のコピー先のラージオブジェクトのラージオブジェクトロケター。src_lob と同じデータ型でなければなりません。

src_lob

dest_lob にコピーされるラージオブジェクトのラージオブジェクトロケター。dest_lob と同じデータ型でなければなりません。

amount

コピーする src_lob のバイト数/文字数。

dest_offset

ソースラージオブジェクトの書き込みを開始するターゲットラージオブジェクト内の位置。最初の位置はオフセット 1 です。デフォルトは 1 です。

src_offset

コピー先のラージオブジェクトへのコピーを開始するソースラージオブジェクトの位置。最初の位置はオフセット 1 です。デフォルトは 1 です。

3.6.6 ERASE

ERASE プロシージャは、大きなオブジェクトの一部を消去する機能を提供します。ラージオブジェクトを消去するとは、指定された部分を BLOBs の場合はゼロバイトのフィラーで、CLOBs の場合はスペースで置き換えることです。ラージオブジェクトの実際のサイズは変更されません。

```
ERASE(<lob_loc> IN OUT { BLOB | CLOB }, <amount> IN OUT INTEGER  
[, <offset> INTEGER ])
```

Parameters

lob_loc

消去するラージオブジェクトのラージオブジェクトロケーター。

amount IN

消去するバイト/文字の数。

amount OUT

実際に消去されたバイト数/文字数。amount バイト/文字が消去される前にラージオブジェクトの終わりに達した場合、この値は入力値よりも小さくなります。

offset

消去を開始する大きなオブジェクト内の位置。最初のバイト/文字は位置 1 です。デフォルトは 1 です。

3.6.7 GET_STORAGE_LIMIT

GET_STORAGE_LIMIT 関数は、最大許容ラージオブジェクトの制限を返します。

```
<size> INTEGER GET_STORAGE_LIMIT(<lob_loc> BLOB)  
  
<size> INTEGER GET_STORAGE_LIMIT(<lob_loc> CLOB)
```


Parameters

size

このデータベース内のラージオブジェクトの最大許容サイズ。

lob_loc

このパラメーターは無視されますが、互換性のために含まれています。

3.6.8 GETLENGTH

GETLENGTH 関数は、ラージオブジェクトの長さを返します。

```
<amount> INTEGER GETLENGTH(<lob_loc> BLOB)
```

```
<amount> INTEGER GETLENGTH(<lob_loc> CLOB)
```

Parameters

lob_loc

長さを取得するラージオブジェクトのラージオブジェクトロケーター。

amount

BLOBs の場合はバイト単位のラージオブジェクトの長さ、CLOBs の場合は文字数。

3.6.9 INSTR

INSTR 関数は、ラージオブジェクト内で特定のパターンが **n** 番目に出現する場所を返します。

```
<position> INTEGER INSTR(<lob_loc> { BLOB | CLOB },
    <pattern> { RAW | VARCHAR2 } [, <offset> INTEGER [, <nth> INTEGER ]])
```

Parameters

lob_loc

パターンを検索するラージオブジェクトのラージオブジェクトロケーター。

pattern

ラージオブジェクトと一致するバイトまたは文字のパターン。lob_loc が BLOB の場合、lob.pattern は RAW でなければなりません。lob_loc が CLOB の場合、パターンは VARCHAR2 でなければなりません。

`offset`

`lob_loc` 内に位置して `pattern` の検索を開始します。最初のバイト/文字は位置 1 です。デフォルトは 1 です。

`nth`

`offset` で指定された位置から `pattern`、`nth` の回数を検索します。デフォルトは 1 です。

`position`

`offset` で指定された位置から `nth` で指定された `n` 番目の時刻に `pattern` が表示されるラージオブジェクト内の位置。

3.6.10 READ

READ プロシージャは、ラージオブジェクトの一部をバッファに読み込む機能を提供します。

```
READ(<lob_loc> { BLOB | CLOB }, <amount> IN OUT BINARY_INTEGER,  
     <offset> INTEGER, <buffer> OUT { RAW | VARCHAR2 })
```

Parameters

`lob_loc`

読み込まれるラージオブジェクトのラージオブジェクトロケータ。

`amount IN`

読み取るバイト数/文字数。

`amount OUT`

実際に読み取られたバイト数/文字数。読み取るデータがこれ以上ない場合、`amount` は 0 を返し、`DATA_NOT_FOUND` 例外がスローされます。

`offset`

読み始める位置。最初のバイト/文字は位置 1 です。

`buffer`

ラージオブジェクトを受け取る変数。`lob_loc` が `BLOB` の場合、`buffer` は `RAW` でなければなりません。`lob_loc` が `CLOB` の場合、`buffer` は `VARCHAR2` でなければなりません。

3.6.11 SUBSTR

SUBSTR 関数は、ラージオブジェクトの一部を返す機能を提供します。

```
<data> { RAW | VARCHAR2 } SUBSTR(<lob_loc> { BLOB | CLOB }
    [, <amount> INTEGER [, <offset> INTEGER ]])
```

Parameters

lob_loc

読み込まれるラージオブジェクトのラージオブジェクトロケーター。

amount

返されるバイト/文字の数。デフォルトは 32,767 です。

offset

データを返し始めるラージオブジェクト内の位置。最初のバイト/文字は位置 1 です。デフォルトは 1 です。

data

読み込まれるラージオブジェクトの返された部分。lob_loc が BLOB の場合、戻りデータ型は RAW です。lob_loc が CLOB の場合、戻りデータ型は VARCHAR2 です。

3.6.12 TRIM

TRIM プロシージャは、ラージオブジェクトを指定の長さに切り捨てる機能を提供します。

```
TRIM(<lob_loc> IN OUT { BLOB | CLOB }, <newlen> INTEGER)
```

Parameters

lob_loc

トリミングするラージオブジェクトのラージオブジェクトロケーター。

newlen

ラージオブジェクトがトリミングされるバイト数/文字数。

3.6.13 WRITE

WRITE プロシージャは、データをラージオブジェクトに書き込む機能を提供します。指定された長さの指定されたオフセットにあるラージオブジェクト内の既存のデータは、バッファで指定されたデータによって上書きされます。

```
WRITE(<lob_loc> IN OUT { BLOB | CLOB },
      <amount> BINARY_INTEGER,
      <offset> INTEGER, <buffer> { RAW | VARCHAR2 })
```

Parameters

lob_loc

書き込まれるラージオブジェクトのラージオブジェクトロケター。

amount

ラージオブジェクトに書き込まれる buffer のバイト数/文字数。

offset

書き込み操作を開始するための、ラージオブジェクトの開始からのバイト/文字単位のオフセット（起点は 1）。

buffer

ラージオブジェクトに書き込まれるデータが含まれます。lob_loc が BLOB の場合、buffer は RAW でなければなりません。lob_loc が CLOB の場合、buffer は VARCHAR2 でなければなりません。

3.6.14 WRITEAPPEND

WRITEAPPEND プロシージャは、ラージオブジェクトの最後にデータを追加する機能を提供します。

```
WRITEAPPEND(<lob_loc> IN OUT { BLOB | CLOB },
            <amount> BINARY_INTEGER, <buffer> { RAW | VARCHAR2 })
```

Parameters

lob_loc

データが追加されるラージオブジェクトのラージオブジェクトロケター。

amount

ラージオブジェクトに追加する buffer からのバイト数/文字数。

buffer

ラージオブジェクトに追加されるデータ。lob_loc が BLOB の場合、buffer は RAW でなければなりません。lob_loc が CLOB の場合、buffer は VARCHAR2 でなければなりません。

3.7 DBMS_LOCK

AdvancedServer は、DBMS_LOCK.SLEEP 手順のサポートを提供します。

機能/手順	戻りタイプ	説明
SLEEP (seconds)	該当なし	指定された数の seconds のセッションを中断します。

AdvancedServer の DBMS_LOCK の実装は、Oracle のバージョンと比較すると部分的な実装です。DBMS_LOCK.SLEEP のみがサポートされます。

3.7.1 SLEEP

SLEEP プロシージャは、指定された秒数の間、現在のセッションを中断します。

```
SLEEP (<seconds> NUMBER)
```

Parameters

seconds

seconds は、セッションを中断する秒数を指定します。seconds は小数値にすることができます。たとえば、1.75 と入力して、1/4 秒を指定します。

3.8 DBMS_MVIEW

DBMS_MVIEW パッケージの手順を使用して、マテリアライズドビューとその依存関係を管理および更新します。
AdvancedServer は、次の DBMS_MVIEW 手順のサポートを提供します。

手続き	戻りタイプ	説明
GET_MV_DEPENDENCIES(list VARCHAR2, deplist VARCHAR2);	該当なし	GET_MV_DEPENDENCIES プロシージャは、指定されたビューの依存関係のリストを返します。
REFRESH(list VARCHAR2, method VARCHAR2, rollback_seg VARCHAR2 , push_deferred_rpc BOOLEAN, refresh_after_errors BOOLEAN , purge_option NUMBER, parallelism NUMBER, heap_size NUMBER , atomic_refresh BOOLEAN , nested BOOLEAN);	該当なし	REFRESH プロシージャのこのバリエーションは、ビュー名のコンマ区切りリストで指定されたすべてのビューを更新します。
REFRESH(tab dbms_utility.uncl_array, method VARCHAR2, rollback_seg VARCHAR2, push_deferred_rpc BOOLEAN, refresh_after_errors BOOLEAN, purge_option NUMBER, parallelism NUMBER, heap_size NUMBER, atomic_refresh BOOLEAN, nested BOOLEAN);	該当なし	REFRESH プロシージャのこのバリエーションは、dbms_utility.uncl_array 値のテーブルで指定されたすべてのビューを更新します。
REFRESH_ALL_MVIEWS(number_of_failures BINARY_INTEGER, method VARCHAR2, rollback_seg VARCHAR2, refresh_after_errors BOOLEAN, atomic_refresh BOOLEAN);	該当なし	REFRESH_ALL_MVIEWS プロシージャは、すべてのマテリアライズドビューを更新します。
REFRESH_DEPENDENT(number_of_failures BINARY_INTEGER, list VARCHAR2, method VARCHAR2, rollback_seg VARCHAR2, refresh_after_errors BOOLEAN, atomic_refresh BOOLEAN, nested BOOLEAN);	該当なし	REFRESH_DEPENDENT プロシージャのこのバリエーションは、コンマ区切りリストにリストされているビューに依存するすべてのビューを更新します。
REFRESH_DEPENDENT(number_of_failures BINARY_INTEGER, tab dbms_utility. uncl_array, method VARCHAR2, rollback_seg VARCHAR2, refresh_after_errors BOOLEAN, atomic_refresh BOOLEAN, nested BOOLEAN);	該当なし	REFRESH_DEPENDENT プロシージャのこのバリエーションは、dbms_utility.uncl_array 値の表にリストされているビューに依存するすべてのビューを更新します。

AdvancedServer の DBMS_MVIEW の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表にリ

ストされている関数と手順のみがサポートされています。

3.8.1 GET_MV_DEPENDENCIES

マテリアライズドビューの名前を指定すると、GET_MV_DEPENDENCIES は指定したビューに依存するアイテムのリストを返します。署名は次のとおりです。

```
GET_MV_DEPENDENCIES (
    <list> IN VARCHAR2,
    <deplist> OUT VARCHAR2);
```

Parameters

list

list は、マテリアライズドビューの名前、またはマテリアライズドビュー名のコンマ区切りリストを指定します。

deplist

deplist は、スキーマで修飾された依存関係のコンマ区切りリストです。deplist は VARCHAR2 の値です。

Examples

次の例:

```
DECLARE
    deplist VARCHAR2(1000);
BEGIN
    DBMS_MVIEW.GET_MV_DEPENDENCIES('public.emp_view', deplist);
    DBMS_OUTPUT.PUT_LINE('deplist: ' || deplist);
END;
```

public.emp_view という名前のマテリアライズドビューの依存関係のリストを表示します。

3.8.2 REFRESH

REFRESH プロシージャを使用して、ビュー名のコンマ区切りリストまたは DBMS_UTILITY.UNCL_ARRAY 値の表で指定されたすべてのビューを更新します。プロシージャには 2 つの署名があります。ビュー名のコンマ区切りリストを指定する場合は、最初の形式を使用します。

```
REFRESH (
    <list> IN VARCHAR2,
    <method> IN VARCHAR2 DEFAULT NULL,
    <rollback_seg> IN VARCHAR2 DEFAULT NULL,
    <push_deferred_rpc> IN BOOLEAN DEFAULT TRUE,
    <refresh_after_errors> IN BOOLEAN DEFAULT FALSE,
```

(次のページに続く)

(前のページからの続き)

```
<purge_option> IN NUMBER DEFAULT 1,
<parallelism> IN NUMBER DEFAULT 0,
<heap_size> IN NUMBER DEFAULT 0,
<atomic_refresh> IN BOOLEAN DEFAULT TRUE,
<nested> IN BOOLEAN DEFAULT FALSE);
```

2 番目のフォームを使用して、DBMS_UTILITY.UNCL_ARRAY 値のテーブルにビュー名を指定します。

```
REFRESH (
  <tab> IN OUT DBMS_UTILITY.UNCL_ARRAY,
  <method> IN VARCHAR2 DEFAULT NULL,
  <rollback_seg> IN VARCHAR2 DEFAULT NULL,
  <push_deferred_rpc> IN BOOLEAN DEFAULT TRUE,
  <refresh_after_errors> IN BOOLEAN DEFAULT FALSE,
  <purge_option> IN NUMBER DEFAULT 1,
  <parallelism> IN NUMBER DEFAULT 0,
  <heap_size> IN NUMBER DEFAULT 0,
  <atomic_refresh> IN BOOLEAN DEFAULT TRUE,
  <nested> IN BOOLEAN DEFAULT FALSE);
```

Parameters

list

list は、マテリアライズドビューの名前を指定する VARCHAR2 値、またはマテリアライズドビュー名のコンマ区切りリストです。名前はスキーマで修飾される場合があります。

tab

tab は、マテリアライズドビューの名前を指定する DBMS_UTILITY.UNCL_ARRAY 値のテーブルです。

method

method は VARCHAR2 の値で、指定されたビューに適用される更新方法を指定します。サポートされる唯一の方法は C です。これにより、ビューの完全な更新が実行されます。

rollback_seg

rollback_seg は互換性のために受け入れられ、無視されます。デフォルトは NULL です。

push_deferred_rpc

push_deferred_rpc は互換性のために受け入れられ、無視されます。デフォルトは TRUE です。

refresh_after_errors

refresh_after_errors は互換性のために受け入れられ、無視されます。デフォルトは FALSE です。

purge_option

`purge_option` は互換性のために受け入れられ、無視されます。デフォルトは 1 です。

`parallelism`

`parallelism` は互換性のために受け入れられ、無視されます。デフォルトは 0 です。

`heap_size` IN NUMBER DEFAULT 0、

`heap_size` は互換性のために受け入れられ、無視されます。デフォルトは 0 です。

`atomic_refresh`

`atomic_refresh` は互換性のために受け入れられ、無視されます。デフォルトは TRUE です。

`nested`

`nested` は互換性のために受け入れられ、無視されます。デフォルトは FALSE です。

Examples

次の例では、`DBMS_MVIEW.REFRESH` を使用して `public.emp_view` マテリアライズドビューで COMPLETE リフレッシュを実行します。

```
EXEC DBMS_MVIEW.REFRESH(list => 'public.emp_view', method => 'C');
```

3.8.3 REFRESH_ALL_MVIEWS

`REFRESH_ALL_MVIEWS` 手順を使用して、ビューが依存するテーブルまたはビューが変更されてから更新されていないマテリアライズドビューを更新します。署名は次のとおりです。

```
REFRESH_ALL_MVIEWS(
  <number_of_failures> OUT BINARY_INTEGER,
  <method> IN VARCHAR2 DEFAULT NULL,
  <rollback_seg> IN VARCHAR2 DEFAULT NULL,
  <refresh_after_errors> IN BOOLEAN DEFAULT FALSE,
  <atomic_refresh> IN BOOLEAN DEFAULT TRUE);
```

Parameters

`number_of_failures`

`number_of_failures` は、リフレッシュ操作中に発生した障害の数を指定する `BINARY_INTEGER` です。

`method`

`method` は `VARCHAR2` の値で、指定されたビューに適用される更新方法を指定します。サポートされる唯一の方法は `C;` で、これによりビューの完全な更新が実行されます。

rollback_seg

rollback_seg は互換性のために受け入れられ、無視されます。デフォルトは NULL です。

refresh_after_errors

refresh_after_errors は互換性のために受け入れられ、無視されます。デフォルトは FALSE です。

atomic_refresh

atomic_refresh は互換性のために受け入れられ、無視されます。デフォルトは TRUE です。

Examples

次の例では、すべてのマテリアライズドビューで COMPLETE リフレッシュを実行します。

```
DECLARE
    errors INTEGER;
BEGIN
    DBMS_MVIEW.REFRESH_ALL_MVIEWS(errors, method => 'C');
END;
```

完了すると、errors には失敗の数が含まれます。

3.8.4 REFRESH_DEPENDENT

REFRESH_DEPENDENT プロシージャを使用して、プロシージャの呼び出しで指定されたビューに依存するすべてのマテリアルビューを更新します。コンマ区切りリストを指定するか、DBMS_UTILITY.UNCL_ARRAY 値のテーブルにビュー名を指定できます。

手順の最初の形式を使用して、コンマ区切りリストで指定されたビューに依存するすべてのマテリアルビューを更新します。

```
REFRESH_DEPENDENT (
    <number_of_failures> OUT BINARY_INTEGER,
    <list> IN VARCHAR2,
    <method> IN VARCHAR2 DEFAULT NULL,
    <rollback_seg> IN VARCHAR2 DEFAULT NULL
    <refresh_after_errors> IN BOOLEAN DEFAULT FALSE,
    <atomic_refresh> IN BOOLEAN DEFAULT TRUE,
    <nested> IN BOOLEAN DEFAULT FALSE);
```

手順の 2 番目の形式を使用して、DBMS_UTILITY.UNCL_ARRAY 値のテーブルで指定されたビューに依存するすべてのマテリアルビューを更新します。

```
REFRESH_DEPENDENT (
    <number_of_failures> OUT BINARY_INTEGER,
```

(次のページに続く)

(前のページからの続き)

```
<tab> IN DBMS_UTILITY.UNCL_ARRAY,
<method> IN VARCHAR2 DEFAULT NULL,
<rollback_seg> IN VARCHAR2 DEFAULT NULL,
<refresh_after_errors> IN BOOLEAN DEFAULT FALSE,
<atomic_refresh> IN BOOLEAN DEFAULT TRUE,
<nested> IN BOOLEAN DEFAULT FALSE);
```

Parameters

`number_of_failures`

`number_of_failures` は、リフレッシュ操作中に発生した障害の数を含む `BINARY_INTEGER` です。

`list`

`list` は、マテリアライズドビューの名前を指定する `VARCHAR2` 値、またはマテリアライズドビュー名のコンマ区切りリストです。名前はスキーマで修飾される場合があります。

`tab`

`tab` は、マテリアライズドビューの名前を指定する `DBMS_UTILITY.UNCL_ARRAY` 値のテーブルです。

`method`

`method` は `VARCHAR2` の値で、指定されたビューに適用される更新方法を指定します。サポートされる唯一の方法は `C` です。これにより、ビューの完全な更新が実行されます。

`rollback_seg`

`rollback_seg` は互換性のために受け入れられ、無視されます。デフォルトは `NULL` です。

`refresh_after_errors`

`refresh_after_errors` は互換性のために受け入れられ、無視されます。デフォルトは `FALSE` です。

`atomic_refresh`

`atomic_refresh` は互換性のために受け入れられ、無視されます。デフォルトは `TRUE` です。

`nested`

`nested` は互換性のために受け入れられ、無視されます。デフォルトは `FALSE` です。

Examples

次の例では、`public` スキーマにある `emp_view` という名前のマテリアライズドビューに依存するすべてのマテリアライズドビューで `COMPLETE` リフレッシュを実行します。

```
DECLARE
    errors INTEGER;
BEGIN
    DBMS_MVIEW.REFRESH_DEPENDENT(errors, list => 'public.emp_view', method =>
'C');
END;
```

完了すると、`errors` には失敗の数が含まれます。

3.9 DBMS_OUTPUT

DBMS_OUTPUT パッケージは、メッセージ（テキスト行）をメッセージバッファに送信したり、メッセージバッファからメッセージを取得したりする機能を提供します。メッセージバッファは、単一セッションに対してローカルです。DBMS_PIPE パッケージを使用して、セッション間でメッセージを送信します。

DBMS_OUTPUT パッケージで利用可能な手順と機能を次の表に示します。

機能/手順	戻りタイプ	説明
DISABLE	該当なし	メッセージを送受信する機能を無効にします。
ENABLE (buffer_size)	該当なし	メッセージを送受信する機能を有効にします。
GET_LINE (line OUT, status OUT)	該当なし	メッセージバッファから行を取得します。
GET_LINES (lines OUT, numlines IN OUT)	該当なし	メッセージバッファから複数の行を取得します。
NEW_LINE	該当なし	行末文字シーケンスを配置します。
PUT (item)	該当なし	行末文字シーケンスなしで部分的な行を配置します。
PUT_LINE (item)	該当なし	行末文字シーケンスで完全な行を配置します。
SERVEROUTPUT (stdout)	該当なし	PUT, PUT_LINE, または NEW_LINE からのメッセージを標準出力またはメッセージバッファに転送します。

次の表に、DBMS_OUTPUT パッケージで使用可能なパブリック変数を示します。

Public Variables	Data Type	Value	Description
chararr	TABLE		メッセージ行用。

3.9.1 CHARARR

CHARARR は複数のメッセージ行を保存するためのものです。

```
TYPE chararr IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

3.9.2 無効にする

DISABLE プロシージャは、メッセージバッファをクリアします。DISABLE プロシージャの実行時にバッファ内のメッセージにアクセスできなくなります。その後 PUT, PUT_LINE, または NEW_LINE 手順で送信されたメッセージはすべて破棄されます。PUT, PUT_LINE, または NEW_LINE 手順が実行され、メッセージが無効にされている場合、エラーは送信者に返されません。

ENABLE 手順または SERVEROUTPUT (TRUE) 手順を使用して、メッセージの送受信を再度有効にします。

```
DISABLE
```

Examples

この匿名ブロックは、現在のセッションでメッセージの送受信を無効にします。

```
BEGIN
    DBMS_OUTPUT.DISABLE;
END;
```

3.9.3 有効にする

ENABLE プロシージャを使用すると、メッセージバッファにメッセージを送信したり、メッセージバッファからメッセージを取得したりできます。SERVEROUTPUT (TRUE) を実行すると、ENABLE 手順も暗黙的に実行されます。

PUT, PUT_LINE, または NEW_LINE で送信されるメッセージの宛先は、SERVEROUTPUT の状態によって異なります。

- SERVEROUTPUT の最後の状態が TRUE の場合、メッセージはコマンドラインの標準出力に送られます。
- SERVEROUTPUT の最後の状態が FALSE の場合、メッセージはメッセージバッファに送られます。

```
ENABLE [ (<buffer_size> INTEGER) ]
```

Parameter

buffer_size

バイト単位のメッセージバッファの最大長。2000 未満の `buffer_size` が指定されている場合、バッファサイズは 2000 に設定されます。

Examples

次の匿名ブロックはメッセージを有効にします。 `SERVEROUTPUT (TRUE)` を設定すると、強制的に標準出力になります。

```
BEGIN
    DBMS_OUTPUT.ENABLE;
    DBMS_OUTPUT.SERVEROUTPUT (TRUE);
    DBMS_OUTPUT.PUT_LINE('Messages enabled');
END;

Messages enabled
```

`SERVEROUTPUT (TRUE)` を使用するだけで、同じ効果が得られます。

```
BEGIN
    DBMS_OUTPUT.SERVEROUTPUT (TRUE);
    DBMS_OUTPUT.PUT_LINE('Messages enabled');
END;

Messages enabled
```

次の匿名ブロックはメッセージを有効にしますが、 `SERVEROUTPUT (FALSE)` を設定するとメッセージがメッセージバッファに転送されます。

```
BEGIN
    DBMS_OUTPUT.ENABLE;
    DBMS_OUTPUT.SERVEROUTPUT (FALSE);
    DBMS_OUTPUT.PUT_LINE('Message sent to buffer');
END;
```

3.9.4 GET_LINE

`GET_LINE` プロシージャは、メッセージバッファからテキスト行を取得する機能を提供します。行末文字シーケンスで終了したテキストのみが取得されます。これは、 `PUT_LINE` を使用して生成された完全な行、または `NEW_LINE` 呼び出しに続く一連の `PUT` 呼び出しによって生成されます。

```
GET_LINE (<line> OUT VARCHAR2, <status> OUT INTEGER)
```

Parameters

`line`

メッセージバッファからテキスト行を受け取る変数。

status

メッセージバッファから行が返された場合は 0、返される行がなかった場合は 1。

Examples

次の匿名ブロックは、emp テーブルを各行のカンマ区切り文字列としてメッセージバッファに書き込みます。

```
EXEC DBMS_OUTPUT.SERVEROUTPUT(FALSE);

DECLARE
    v_emprec          VARCHAR2(120);
    CURSOR emp_cur IS SELECT * FROM emp ORDER BY empno;
BEGIN
    DBMS_OUTPUT.ENABLE;
    FOR i IN emp_cur LOOP
        v_emprec := i.empno || ',' || i.ename || ',' || i.job || ',' ||
            NVL(LTRIM(TO_CHAR(i.mgr,'9999')),') || ',' || i.hiredate ||
            ',' || i.sal || ',' ||
            NVL(LTRIM(TO_CHAR(i.comm,'9990.99')),') || ',' || i.deptno;
        DBMS_OUTPUT.PUT_LINE(v_emprec);
    END LOOP;
END;
```

次の匿名ブロックは、メッセージバッファを読み取り、前の例で書き込まれたメッセージを messages という名前のテーブルに挿入します。messages の行が表示されます。

```
CREATE TABLE messages (
    status          INTEGER,
    msg             VARCHAR2(100)
);

DECLARE
    v_line          VARCHAR2(100);
    v_status        INTEGER := 0;
BEGIN
    DBMS_OUTPUT.GET_LINE(v_line,v_status);
    WHILE v_status = 0 LOOP
        INSERT INTO messages VALUES(v_status, v_line);
        DBMS_OUTPUT.GET_LINE(v_line,v_status);
    END LOOP;
END;

SELECT msg FROM messages;

                msg
-----
```

(次のページに続く)

(前のページからの続き)

```

7369, SMITH, CLERK, 7902, 17-DEC-80 00:00:00, 800.00, , 20
7499, ALLEN, SALESMAN, 7698, 20-FEB-81 00:00:00, 1600.00, 300.00, 30
7521, WARD, SALESMAN, 7698, 22-FEB-81 00:00:00, 1250.00, 500.00, 30
7566, JONES, MANAGER, 7839, 02-APR-81 00:00:00, 2975.00, , 20
7654, MARTIN, SALESMAN, 7698, 28-SEP-81 00:00:00, 1250.00, 1400.00, 30
7698, BLAKE, MANAGER, 7839, 01-MAY-81 00:00:00, 2850.00, , 30
7782, CLARK, MANAGER, 7839, 09-JUN-81 00:00:00, 2450.00, , 10
7788, SCOTT, ANALYST, 7566, 19-APR-87 00:00:00, 3000.00, , 20
7839, KING, PRESIDENT, , 17-NOV-81 00:00:00, 5000.00, , 10
7844, TURNER, SALESMAN, 7698, 08-SEP-81 00:00:00, 1500.00, 0.00, 30
7876, ADAMS, CLERK, 7788, 23-MAY-87 00:00:00, 1100.00, , 20
7900, JAMES, CLERK, 7698, 03-DEC-81 00:00:00, 950.00, , 30
7902, FORD, ANALYST, 7566, 03-DEC-81 00:00:00, 3000.00, , 20
7934, MILLER, CLERK, 7782, 23-JAN-82 00:00:00, 1300.00, , 10
(14 rows)

```

3.9.5 GET_LINES

GET_LINES プロシージャは、1 行以上のテキストをメッセージバッファからコレクションに取得する機能を提供します。行末文字シーケンスで終了したテキストのみが取得されます。これは、PUT_LINE を使用して生成された完全な行、または NEW_LINE 呼び出しに続く一連の PUT 呼び出しによって生成されます。

```
GET_LINES(<lines> OUT CHARARR, <numlines> IN OUT INTEGER)
```

Parameters

lines

メッセージバッファからテキスト行を受け取るテーブル。lines. の説明については CHARARR を参照してください

numlines IN

メッセージバッファから取得する行数。

numlines OUT

メッセージバッファから取得された実際の行数。numlines の出力値が入力値よりも小さい場合、メッセージバッファに残っている行はありません。

Examples

次の例では、GET_LINES プロシージャを使用して、メッセージバッファに配置された emp テーブルのすべての行を配列に格納します。

```

EXEC DBMS_OUTPUT.SERVEROUTPUT(FALSE);

DECLARE
    v_emprec          VARCHAR2(120);
    CURSOR emp_cur IS SELECT * FROM emp ORDER BY empno;
BEGIN
    DBMS_OUTPUT.ENABLE;
    FOR i IN emp_cur LOOP
        v_emprec := i.empno || ',' || i.ename || ',' || i.job || ',' ||
            NVL(LTRIM(TO_CHAR(i.mgr, '9999')), '') || ',' || i.hiredate ||
            ',' || i.sal || ',' ||
            NVL(LTRIM(TO_CHAR(i.comm, '9990.99')), '') || ',' || i.deptno;
        DBMS_OUTPUT.PUT_LINE(v_emprec);
    END LOOP;
END;

DECLARE
    v_lines           DBMS_OUTPUT.CHARARR;
    v_numlines        INTEGER := 14;
    v_status           INTEGER := 0;
BEGIN
    DBMS_OUTPUT.GET_LINES(v_lines, v_numlines);
    FOR i IN 1..v_numlines LOOP
        INSERT INTO messages VALUES(v_numlines, v_lines(i));
    END LOOP;
END;

SELECT msg FROM messages;


```

msg

```

7369,SMITH,CLERK,7902,17-DEC-80 00:00:00,800.00,,20
7499,ALLEN,SALESMAN,7698,20-FEB-81 00:00:00,1600.00,300.00,30
7521,WARD,SALESMAN,7698,22-FEB-81 00:00:00,1250.00,500.00,30
7566,JONES,MANAGER,7839,02-APR-81 00:00:00,2975.00,,20
7654,MARTIN,SALESMAN,7698,28-SEP-81 00:00:00,1250.00,1400.00,30
7698,BLAKE,MANAGER,7839,01-MAY-81 00:00:00,2850.00,,30
7782,CLARK,MANAGER,7839,09-JUN-81 00:00:00,2450.00,,10
7788,SCOTT,ANALYST,7566,19-APR-87 00:00:00,3000.00,,20
7839,KING,PRESIDENT,,17-NOV-81 00:00:00,5000.00,,10
7844,TURNER,SALESMAN,7698,08-SEP-81 00:00:00,1500.00,0.00,30
7876,ADAMS,CLERK,7788,23-MAY-87 00:00:00,1100.00,,20
7900,JAMES,CLERK,7698,03-DEC-81 00:00:00,950.00,,30
7902,FORD,ANALYST,7566,03-DEC-81 00:00:00,3000.00,,20
7934,MILLER,CLERK,7782,23-JAN-82 00:00:00,1300.00,,10
(14 rows)

```

3.9.6 NEW_LINE

NEW_LINE プロシージャは、行末文字シーケンスをメッセージバッファに書き込みます。

```
NEW_LINE
```

Parameter

NEW_LINE プロシージャにはパラメータは必要ありません。

3.9.7 PUT

PUT プロシージャは、メッセージバッファに文字列を書き込みます。文字列の末尾に行末文字シーケンスは書き込まれません。NEW_LINE 手順を使用して、行末文字シーケンスを追加します。

```
PUT(<item> VARCHAR2)
```

Parameter

item

メッセージバッファに書き込まれるテキスト。

Examples

次の例では、PUT プロシージャを使用して、emp テーブルの従業員のカンマ区切りリストを表示します。

```
DECLARE
    CURSOR emp_cur IS SELECT * FROM emp ORDER BY empno;
BEGIN
    FOR i IN emp_cur LOOP
        DBMS_OUTPUT.PUT(i.empno);
        DBMS_OUTPUT.PUT(', ');
        DBMS_OUTPUT.PUT(i.ename);
        DBMS_OUTPUT.PUT(', ');
        DBMS_OUTPUT.PUT(i.job);
        DBMS_OUTPUT.PUT(', ');
        DBMS_OUTPUT.PUT(i.mgr);
        DBMS_OUTPUT.PUT(', ');
        DBMS_OUTPUT.PUT(i.hiredate);
        DBMS_OUTPUT.PUT(', ');
        DBMS_OUTPUT.PUT(i.sal);
        DBMS_OUTPUT.PUT(', ');
        DBMS_OUTPUT.PUT(i.comm);
        DBMS_OUTPUT.PUT(', ');
        DBMS_OUTPUT.PUT(i.deptno);
        DBMS_OUTPUT.NEW_LINE;
```

(次のページに続く)

(前のページからの続き)

```

END LOOP;
END;

7369, SMITH, CLERK, 7902, 17-DEC-80 00:00:00, 800.00, , 20
7499, ALLEN, SALESMAN, 7698, 20-FEB-81 00:00:00, 1600.00, 300.00, 30
7521, WARD, SALESMAN, 7698, 22-FEB-81 00:00:00, 1250.00, 500.00, 30
7566, JONES, MANAGER, 7839, 02-APR-81 00:00:00, 2975.00, , 20
7654, MARTIN, SALESMAN, 7698, 28-SEP-81 00:00:00, 1250.00, 1400.00, 30
7698, BLAKE, MANAGER, 7839, 01-MAY-81 00:00:00, 2850.00, , 30
7782, CLARK, MANAGER, 7839, 09-JUN-81 00:00:00, 2450.00, , 10
7788, SCOTT, ANALYST, 7566, 19-APR-87 00:00:00, 3000.00, , 20
7839, KING, PRESIDENT, , 17-NOV-81 00:00:00, 5000.00, , 10
7844, TURNER, SALESMAN, 7698, 08-SEP-81 00:00:00, 1500.00, 0.00, 30
7876, ADAMS, CLERK, 7788, 23-MAY-87 00:00:00, 1100.00, , 20
7900, JAMES, CLERK, 7698, 03-DEC-81 00:00:00, 950.00, , 30
7902, FORD, ANALYST, 7566, 03-DEC-81 00:00:00, 3000.00, , 20
7934, MILLER, CLERK, 7782, 23-JAN-82 00:00:00, 1300.00, , 10

```

3.9.8 PUT_LINE

PUT_LINE プロシージャは、行末文字シーケンスを含む単一行をメッセージバッファに書き込みます。

```
PUT_LINE(<item> VARCHAR2)
```

Parameter

item

メッセージバッファに書き込まれるテキスト。

Examples

次の例では、PUT_LINE プロシージャを使用して、emp テーブルの従業員のカンマ区切りリストを表示します。

```

DECLARE
    v_emprec          VARCHAR2(120);
    CURSOR emp_cur IS SELECT * FROM emp ORDER BY empno;
BEGIN
    FOR i IN emp_cur LOOP
        v_emprec := i.empno || ',' || i.ename || ',' || i.job || ',' ||
            NVL(LTRIM(TO_CHAR(i.mgr, '9999')), '') || ',' || i.hiredate ||
            ',' || i.sal || ',' ||
            NVL(LTRIM(TO_CHAR(i.comm, '9990.99')), '') || ',' || i.deptno;
        DBMS_OUTPUT.PUT_LINE(v_emprec);
    END LOOP;
END;

```

(次のページに続く)

(前のページからの続き)

```

7369, SMITH, CLERK, 7902, 17-DEC-80 00:00:00, 800.00, , 20
7499, ALLEN, SALESMAN, 7698, 20-FEB-81 00:00:00, 1600.00, 300.00, 30
7521, WARD, SALESMAN, 7698, 22-FEB-81 00:00:00, 1250.00, 500.00, 30
7566, JONES, MANAGER, 7839, 02-APR-81 00:00:00, 2975.00, , 20
7654, MARTIN, SALESMAN, 7698, 28-SEP-81 00:00:00, 1250.00, 1400.00, 30
7698, BLAKE, MANAGER, 7839, 01-MAY-81 00:00:00, 2850.00, , 30
7782, CLARK, MANAGER, 7839, 09-JUN-81 00:00:00, 2450.00, , 10
7788, SCOTT, ANALYST, 7566, 19-APR-87 00:00:00, 3000.00, , 20
7839, KING, PRESIDENT, , 17-NOV-81 00:00:00, 5000.00, , 10
7844, TURNER, SALESMAN, 7698, 08-SEP-81 00:00:00, 1500.00, 0.00, 30
7876, ADAMS, CLERK, 7788, 23-MAY-87 00:00:00, 1100.00, , 20
7900, JAMES, CLERK, 7698, 03-DEC-81 00:00:00, 950.00, , 30
7902, FORD, ANALYST, 7566, 03-DEC-81 00:00:00, 3000.00, , 20
7934, MILLER, CLERK, 7782, 23-JAN-82 00:00:00, 1300.00, , 10

```

3.9.9 SERVEROUTPUT

SERVEROUTPUT プロシージャは、コマンドラインの標準出力またはメッセージバッファにメッセージを送信する機能を提供します。SERVEROUTPUT (TRUE) を設定すると、ENABLE の暗黙的な実行も実行されます。

SERVEROUTPUT のデフォルト設定は実装依存です。たとえば、OracleSQL*Plus では、SERVEROUTPUT (FALSE) がデフォルトです。PSQL では、SERVEROUTPUT (TRUE) がデフォルトです。また、OracleSQL*Plus では、この設定は AdvancedServer に実装されているストアド・プロシージャではなく、SQL*Plus の SET コマンドを使用して制御されます。

```
SERVEROUTPUT (<stdout> BOOLEAN)
```

Oracle スタイルの表示出力を取得するには、postgresql.conf ファイルで dbms_output.serveroutput を FALSE に設定します。これにより、メッセージ出力が無効になります。デフォルトは TRUE で、メッセージ出力を有効にします。

Parameter

stdout

後続の PUT, PUT_LINE または NEW_LINE コマンドがコマンドラインの標準出力にテキストを直接送信する場合は TRUE に設定します。テキストをメッセージバッファに送信する場合は FALSE に設定します。

Examples

次の匿名ブロックは、最初のメッセージをコマンドラインに送信し、2 番目のメッセージをメッセージバッファに送信します。

```
BEGIN
    DBMS_OUTPUT.SERVEROUTPUT(TRUE);
    DBMS_OUTPUT.PUT_LINE('This message goes to the command line');
    DBMS_OUTPUT.SERVEROUTPUT(FALSE);
    DBMS_OUTPUT.PUT_LINE('This message goes to the message buffer');
END;
```

This message goes to the command line

同じセッション内で次の匿名ブロックが実行されると、前の例のメッセージバッファに保存されたメッセージがフラッシュされ、新しいメッセージとともにコマンドラインに表示されます。

```
BEGIN
    DBMS_OUTPUT.SERVEROUTPUT(TRUE);
    DBMS_OUTPUT.PUT_LINE('Flush messages from the buffer');
END;
```

This message goes to the message buffer
Flush messages from the buffer

3.10 DBMS_PIPE

DBMS_PIPE パッケージは、同じデータベースクラスターに接続されたセッション内またはセッション間でパイプを介してメッセージを送信する機能を提供します。

DBMS_PIPE パッケージで使用可能な手順と機能を次の表に示します。

機能/手順	戻りタイプ	説明
CREATE_PIPE(pipename [, maxpipesize] [, private])	INTEGER	private が「true」（デフォルト）の場合は明示的にプライベートパイプを作成し、private が「false」の場合はパブリックパイプを作成します。
NEXT_ITEM_TYPE	INTEGER	受信したメッセージの次のアイテムのデータ型を決定します。
PACK_MESSAGE(item)	該当なし	セッションのローカルメッセージバッファに item を配置します。
PURGE(pipename)	該当なし	指定されたパイプから未受信のメッセージを削除します。
RECEIVE_MESSAGE(pipename [, timeout])	INTEGER	指定されたパイプからメッセージを取得します。
REMOVE_PIPE(pipename)	INTEGER	明示的に作成されたパイプを削除します。
RESET_BUFFER	該当なし	ローカルメッセージバッファをリセットします。
SEND_MESSAGE(pipename [, timeout] [, maxpipesize])	INTEGER	パイプでメッセージを送信します。
UNIQUE_SESSION_NAME	VARCHAR2	一意のセッション名を取得します。
UNPACK_MESSAGE(item OUT)	該当なし	メッセージから次のデータ項目を取得して、タイプ互換変数 item に入れます。

パイプは、暗黙的または明示的に分類されます。「暗黙のパイプ」は、CREATE_PIPE 関数によって以前に作成されていないパイプ名を参照する場合に作成されます。たとえば、存在しないパイプ名を使用して SEND_MESSAGE 関数を実行すると、その名前で新しい暗黙のパイプが作成されます。「明示的なパイプ」は CREATE_PIPE 関数を使用して作成され、最初のパラメーターは新しいパイプのパイプ名を指定します。

パイプもプライベートまたはパブリックに分類されます。プライベートパイプには、パイプを作成したユーザーのみがアクセスできます。スーパーユーザーであっても、別のユーザーが作成したプライベートパイプにはアクセスできません。「パブリックパイプ」には、DBMS_PIPE パッケージにアクセスできるすべてのユーザーがアクセスできます。

パブリックパイプは、3 番目のパラメーターを FALSE に設定して CREATE_PIPE 関数を使用することによってのみ作成できます。CREATE_PIPE 関数を使用して、3 番目のパラメーターを TRUE に設定するか、3 番目のパ

ラメーターを省略することにより、プライベートパイプを作成できます。すべての暗黙のパイプはプライベートです。

メッセージの個々のデータ項目または「行」は、現在のセッションに固有の*ローカルメッセージバッファ*に最初に組み込まれます。PACK_MESSAGE プロシージャは、セッションのローカルメッセージバッファにメッセージを作成します。SEND_MESSAGE 関数を使用して、パイプを介してメッセージを送信します。

メッセージの受信には、逆の操作が含まれます。RECEIVE_MESSAGE 関数は、指定されたパイプからメッセージを取得するために使用されます。メッセージは、セッションのローカルメッセージバッファに書き込まれます。次に、UNPACK_MESSAGE プロシージャを使用して、メッセージデータアイテムをメッセージバッファからプログラム変数に転送します。パイプに複数のメッセージが含まれる場合、RECEIVE_MESSAGE はメッセージを*FIFO*（先入れ先出し）の順序で取得します。

各セッションは、PACK_MESSAGE プロシージャで作成されたメッセージと RECEIVE_MESSAGE 関数で取得されたメッセージ用に別々のメッセージバッファを保持します。したがって、メッセージは同じセッションで作成および受信できます。ただし、連続した RECEIVE_MESSAGE 呼び出しが行われた場合、最後の RECEIVE_MESSAGE 呼び出しからのメッセージのみがローカルメッセージバッファに保存されます。

3.10.1 CREATE_PIPE

CREATE_PIPE 関数は、指定された名前を持つ明示的なパブリックパイプまたは明示的なプライベートパイプを作成します。

```
<status> INTEGER CREATE_PIPE(<pipename> VARCHAR2
    [, <maxpipesize> INTEGER ] [, <private> BOOLEAN ])
```

Parameters

pipename

パイプの名前。

maxpipesize

パイプの最大容量（バイト単位）。デフォルトは 8192 バイトです。

private

FALSE に設定されている場合、パブリックパイプを作成します。TRUE に設定されている場合、プライベートパイプを作成するこれがデフォルトです。

status

操作によって返されたステータスコード。0 は作成が成功したことを示します。

Examples

次の例では、messages: という名前のプライベートパイプを作成します

```
DECLARE
    v_status      INTEGER;
BEGIN
    v_status := DBMS_PIPE.CREATE_PIPE('messages');
    DBMS_OUTPUT.PUT_LINE('CREATE_PIPE status: ' || v_status);
END;
CREATE_PIPE status: 0
```

次の例では、mailbox: という名前のパブリックパイプを作成します

```
DECLARE
    v_status      INTEGER;
BEGIN
    v_status := DBMS_PIPE.CREATE_PIPE('mailbox', 8192, FALSE);
    DBMS_OUTPUT.PUT_LINE('CREATE_PIPE status: ' || v_status);
END;
CREATE_PIPE status: 0
```

3.10.2 NEXT_ITEM_TYPE

NEXT_ITEM_TYPE 関数は、セッションのローカルメッセージバッファに取得されたメッセージ内の次のデータ項目のデータ型を識別する整数コードを返します。UNPACK_MESSAGE プロシージャを使用して各アイテムがローカルメッセージバッファから移動されると、NEXT_ITEM_TYPE 関数は次に利用可能なアイテムのデータ型コードを返します。メッセージにこれ以上アイテムが残っていない場合、コード 0 が返されます。

```
<typecode> INTEGER NEXT_ITEM_TYPE
```

Parameters

typecode

次の表に示すように、次のデータ項目のデータ型を識別するコード。

タイプコード	データ型
0	これ以上のデータ項目はありません
9	NUMBER
11	VARCHAR2
13	DATE
23	RAW

注釈: 表のタイプコードリストは、Oracle データベースと互換性がありません。Oracle は、データ型に異なる番号付けシーケンスを割り当てます。

Examples

次の例は、NUMBER アイテム、VARCHAR2 アイテム、DATE アイテム、RAW アイテムが詰め込まれたパイプを示しています。次に、2 番目の匿名ブロックは NEXT_ITEM_TYPE 関数を使用して各アイテムのタイプコードを表示します。

```
DECLARE
    v_number      NUMBER := 123;
    v_varchar     VARCHAR2(20) := 'Character data';
    v_date        DATE := SYSDATE;
    v_raw         RAW(4) := '21222324';
    v_status      INTEGER;
BEGIN
    DBMS_PIPE.PACK_MESSAGE(v_number);
    DBMS_PIPE.PACK_MESSAGE(v_varchar);
    DBMS_PIPE.PACK_MESSAGE(v_date);
    DBMS_PIPE.PACK_MESSAGE(v_raw);
```

(次のページに続く)

(前のページからの続き)

```

        v_status := DBMS_PIPE.SEND_MESSAGE('datatypes');
        DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE status: ' || v_status);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('SQLERRM: ' || SQLERRM);
        DBMS_OUTPUT.PUT_LINE('SQLCODE: ' || SQLCODE);
END;

SEND_MESSAGE status: 0

DECLARE
    v_number      NUMBER;
    v_varchar     VARCHAR2(20);
    v_date        DATE;
    v_timestamp   TIMESTAMP;
    v_raw         RAW(4);
    v_status      INTEGER;
BEGIN
    v_status := DBMS_PIPE.RECEIVE_MESSAGE('datatypes');
    DBMS_OUTPUT.PUT_LINE('RECEIVE_MESSAGE status: ' || v_status);
    DBMS_OUTPUT.PUT_LINE('-----');

    v_status := DBMS_PIPE.NEXT_ITEM_TYPE;
    DBMS_OUTPUT.PUT_LINE('NEXT_ITEM_TYPE: ' || v_status);
    DBMS_PIPE.UNPACK_MESSAGE(v_number);
    DBMS_OUTPUT.PUT_LINE('NUMBER Item      : ' || v_number);
    DBMS_OUTPUT.PUT_LINE('-----');

    v_status := DBMS_PIPE.NEXT_ITEM_TYPE;
    DBMS_OUTPUT.PUT_LINE('NEXT_ITEM_TYPE: ' || v_status);
    DBMS_PIPE.UNPACK_MESSAGE(v_varchar);
    DBMS_OUTPUT.PUT_LINE('VARCHAR2 Item : ' || v_varchar);
    DBMS_OUTPUT.PUT_LINE('-----');

    v_status := DBMS_PIPE.NEXT_ITEM_TYPE;
    DBMS_OUTPUT.PUT_LINE('NEXT_ITEM_TYPE: ' || v_status);
    DBMS_PIPE.UNPACK_MESSAGE(v_date);
    DBMS_OUTPUT.PUT_LINE('DATE Item      : ' || v_date);
    DBMS_OUTPUT.PUT_LINE('-----');

    v_status := DBMS_PIPE.NEXT_ITEM_TYPE;
    DBMS_OUTPUT.PUT_LINE('NEXT_ITEM_TYPE: ' || v_status);
    DBMS_PIPE.UNPACK_MESSAGE(v_raw);
    DBMS_OUTPUT.PUT_LINE('RAW Item       : ' || v_raw);
    DBMS_OUTPUT.PUT_LINE('-----');

    v_status := DBMS_PIPE.NEXT_ITEM_TYPE;
    DBMS_OUTPUT.PUT_LINE('NEXT_ITEM_TYPE: ' || v_status);

```

(次のページに続く)

(前のページからの続き)

```

    DBMS_OUTPUT.PUT_LINE('-----');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('SQLERRM: ' || SQLERRM);
        DBMS_OUTPUT.PUT_LINE('SQLCODE: ' || SQLCODE);
END;

RECEIVE_MESSAGE status: 0
-----
NEXT_ITEM_TYPE: 9
NUMBER Item    : 123
-----
NEXT_ITEM_TYPE: 11
VARCHAR2 Item  : Character data
-----
NEXT_ITEM_TYPE: 13
DATE Item      : 02-OCT-07 11:11:43
-----
NEXT_ITEM_TYPE: 23
RAW Item       : 21222324
-----
NEXT_ITEM_TYPE: 0

```

3.10.3 PACK_MESSAGE

PACK_MESSAGE プロシージャは、セッションのローカルメッセージバッファにデータ項目を配置します。SEND_MESSAGE 呼び出しを発行する前に、PACK_MESSAGE を少なくとも 1 回実行する必要があります。

```
PACK_MESSAGE(<item> { DATE | NUMBER | VARCHAR2 | RAW })
```

RECEIVE_MESSAGE コールを使用してメッセージを取得したら、UNPACK_MESSAGE 手順を使用してデータ項目を取得します。

Parameters

item

受け入れ可能なパラメーターデータ型のいずれかに評価される式。値は、セッションのローカルメッセージバッファに追加されます。

3.10.4 PURGE

PURGE プロシージャは、指定された暗黙のパイプから未受信のメッセージを削除します。

```
PURGE(<pipename> VARCHAR2)
```

REMOVE_PIPE 関数を使用して、明示的なパイプを削除します。

Parameters

pipename

パイプの名前。

Examples

2つのメッセージがパイプで送信されます。

```
DECLARE
    v_status          INTEGER;
BEGIN
    DBMS_PIPE.PACK_MESSAGE('Message #1');
    v_status := DBMS_PIPE.SEND_MESSAGE('pipe');
    DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE status: ' || v_status);

    DBMS_PIPE.PACK_MESSAGE('Message #2');
    v_status := DBMS_PIPE.SEND_MESSAGE('pipe');
    DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE status: ' || v_status);
END;

SEND_MESSAGE status: 0
SEND_MESSAGE status: 0
```

最初のメッセージを受信して解凍します。

```
DECLARE
    v_item            VARCHAR2(80);
    v_status          INTEGER;
BEGIN
    v_status := DBMS_PIPE.RECEIVE_MESSAGE('pipe',1);
    DBMS_OUTPUT.PUT_LINE('RECEIVE_MESSAGE status: ' || v_status);
    DBMS_PIPE.UNPACK_MESSAGE(v_item);
    DBMS_OUTPUT.PUT_LINE('Item: ' || v_item);
END;

RECEIVE_MESSAGE status: 0
Item: Message #1
```

パイプをページします。

```
EXEC DBMS_PIPE.PURGE('pipe');
```

次のメッセージを取得してください。RECEIVE_MESSAGE 呼び出しは、ステータスコード 1 を返し、メッセージが利用できなかったためタイムアウトしたことを示します。

```
DECLARE
    v_item          VARCHAR2(80);
    v_status        INTEGER;
BEGIN
    v_status := DBMS_PIPE.RECEIVE_MESSAGE('pipe',1);
    DBMS_OUTPUT.PUT_LINE('RECEIVE_MESSAGE status: ' || v_status);
END;

RECEIVE_MESSAGE status: 1
```

3.10.5 RECEIVE_MESSAGE

RECEIVE_MESSAGE 関数は、指定されたパイプからメッセージを取得します。

```
<status> INTEGER RECEIVE_MESSAGE(<pipename> VARCHAR2
    [, <timeout> INTEGER ])
```

Parameters

pipename

パイプの名前。

timeout

待機時間（秒）。デフォルトは 86400000（1000 日）です。

status

操作によって返されたステータスコード。

可能なステータスコードは次のとおりです。

状態コード	説明
0	成功
1	タイムアウト
2	メッセージが大きすぎます。

3.10.6 REMOVE_PIPE

REMOVE_PIPE 関数は、明示的なプライベートパイプまたは明示的なパブリックパイプを削除します。

```
<status> INTEGER REMOVE_PIPE(<pipename> VARCHAR2)
```

REMOVE_PIPE 関数を使用して、明示的に作成されたパイプ、つまり CREATE_PIPE 関数で作成されたパイプを削除します。

Parameters

pipename

パイプの名前。

status

操作によって返されたステータスコード。名前付きパイプが存在しない場合でも、ステータスコード 0 が返されます。

Examples

2 つのメッセージがパイプで送信されます。

```
DECLARE
    v_status          INTEGER;
BEGIN
    v_status := DBMS_PIPE.CREATE_PIPE('pipe');
    DBMS_OUTPUT.PUT_LINE('CREATE_PIPE status : ' || v_status);

    DBMS_PIPE.PACK_MESSAGE('Message #1');
    v_status := DBMS_PIPE.SEND_MESSAGE('pipe');
    DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE status: ' || v_status);

    DBMS_PIPE.PACK_MESSAGE('Message #2');
    v_status := DBMS_PIPE.SEND_MESSAGE('pipe');
    DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE status: ' || v_status);
END;

CREATE_PIPE status : 0
SEND_MESSAGE status: 0
SEND_MESSAGE status: 0
```

最初のメッセージを受信して解凍します。

```
DECLARE
    v_item            VARCHAR2(80);
    v_status          INTEGER;
```

(次のページに続く)

(前のページからの続き)

```

BEGIN
    v_status := DBMS_PIPE.RECEIVE_MESSAGE('pipe',1);
    DBMS_OUTPUT.PUT_LINE('RECEIVE_MESSAGE status: ' || v_status);
    DBMS_PIPE.UNPACK_MESSAGE(v_item);
    DBMS_OUTPUT.PUT_LINE('Item: ' || v_item);
END;

RECEIVE_MESSAGE status: 0
Item: Message #1

```

パイプを取り外します。

```

SELECT DBMS_PIPE.REMOVE_PIPE('pipe') FROM DUAL;

remove_pipe
-----
          0
(1 row)

```

次のメッセージを取得してください。RECEIVE_MESSAGE 呼び出しは、パイプが削除されたためにタイムアウトしたことを示すステータスコード 1 を返します。

```

DECLARE
    v_item          VARCHAR2(80);
    v_status        INTEGER;
BEGIN
    v_status := DBMS_PIPE.RECEIVE_MESSAGE('pipe',1);
    DBMS_OUTPUT.PUT_LINE('RECEIVE_MESSAGE status: ' || v_status);
END;

RECEIVE_MESSAGE status: 1

```

3.10.7 RESET_BUFFER

RESET_BUFFER プロシージャは、セッションのローカルメッセージバッファへの「ポインタ」をリセットして、バッファの先頭に戻します。これにより、後続の PACK_MESSAGE 呼び出しが、RESET_BUFFER 呼び出しの前にメッセージバッファに存在していたデータ項目を上書きするという効果があります。

```
RESET_BUFFER
```

Examples

John へのメッセージがローカルメッセージバッファに書き込まれます。RESET_BUFFER を呼び出すと、Bob へのメッセージに置き換えられます。メッセージはパイプで送信されます。

```

DECLARE
    v_status          INTEGER;
BEGIN
    DBMS_PIPE.PACK_MESSAGE('Hi, John');
    DBMS_PIPE.PACK_MESSAGE('Can you attend a meeting at 3:00, today?');
    DBMS_PIPE.PACK_MESSAGE('If not, is tomorrow at 8:30 ok with you?');
    DBMS_PIPE.RESET_BUFFER;
    DBMS_PIPE.PACK_MESSAGE('Hi, Bob');
    DBMS_PIPE.PACK_MESSAGE('Can you attend a meeting at 9:30, tomorrow?');
    v_status := DBMS_PIPE.SEND_MESSAGE('pipe');
    DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE status: ' || v_status);
END;

SEND_MESSAGE status: 0

```

Bob へのメッセージは、受信したメッセージに含まれています。

```

DECLARE
    v_item            VARCHAR2(80);
    v_status          INTEGER;
BEGIN
    v_status := DBMS_PIPE.RECEIVE_MESSAGE('pipe',1);
    DBMS_OUTPUT.PUT_LINE('RECEIVE_MESSAGE status: ' || v_status);
    DBMS_PIPE.UNPACK_MESSAGE(v_item);
    DBMS_OUTPUT.PUT_LINE('Item: ' || v_item);
    DBMS_PIPE.UNPACK_MESSAGE(v_item);
    DBMS_OUTPUT.PUT_LINE('Item: ' || v_item);
END;

RECEIVE_MESSAGE status: 0
Item: Hi, Bob
Item: Can you attend a meeting at 9:30, tomorrow?

```

3.10.8 SEND_MESSAGE

SEND_MESSAGE 関数は、セッションのローカルメッセージバッファから指定されたパイプにメッセージを送信します。

```

<status> SEND_MESSAGE(<pipename> VARCHAR2 [, <timeout> INTEGER ]
    [, <maxpipesize> INTEGER ])

```

Parameters

pipename

パイプの名前。

timeout

待機時間（秒）。デフォルトは 86400000（1000 日）です。

maxpipesize

パイプの最大容量（バイト単位）。デフォルトは 8192 バイトです。

status

操作によって返されたステータスコード。

可能なステータスコードは次のとおりです。

状態コード	説明
0	成功
1	タイムアウト
3	機能が中断されました

3.10.9 UNIQUE_SESSION_NAME

UNIQUE_SESSION_NAME 関数は、現在のセッションに固有の名前を返します。

```
<name> VARCHAR2 UNIQUE_SESSION_NAME
```

Parameters

name

一意のセッション名。

Examples

次の匿名ブロックは、一意のセッション名を取得して表示します。

```
DECLARE
    v_session      VARCHAR2(30);
BEGIN
    v_session := DBMS_PIPE.UNIQUE_SESSION_NAME;
    DBMS_OUTPUT.PUT_LINE('Session Name: ' || v_session);
END;

Session Name: PG$PIPE$5$2752
```

3.10.10 UNPACK_MESSAGE

UNPACK_MESSAGE プロシージャは、メッセージのデータ項目をローカルメッセージバッファから指定されたプログラム変数にコピーします。メッセージは、UNPACK_MESSAGE を使用する前に RECEIVE_MESSAGE 機能を使用してローカルメッセージバッファに配置する必要があります。

```
UNPACK_MESSAGE(<item> OUT { DATE | NUMBER | VARCHAR2 | RAW })
```

Parameters

item

ローカルメッセージバッファからデータ項目を受け取る型互換変数。

3.10.11 包括的な例

次の例では、パイプを「メールボックス」として使用します。メールボックスを作成し、マルチアイテムメッセージをメールボックスに追加し（最大 3 つのアイテム）、メールボックスの全内容を表示する手順は、mailbox という名前のパッケージに含まれています。

```
CREATE OR REPLACE PACKAGE mailbox
IS
    PROCEDURE create_mailbox;
    PROCEDURE add_message (
        p_mailbox    VARCHAR2,
        p_item_1     VARCHAR2,
        p_item_2     VARCHAR2 DEFAULT 'END',
        p_item_3     VARCHAR2 DEFAULT 'END'
    );
    PROCEDURE empty_mailbox (
        p_mailbox    VARCHAR2,
        p_waittime   INTEGER DEFAULT 10
    );
END mailbox;

CREATE OR REPLACE PACKAGE BODY mailbox
IS
    PROCEDURE create_mailbox
    IS
        v_mailbox    VARCHAR2(30);
        v_status     INTEGER;
    BEGIN
        v_mailbox := DBMS_PIPE.UNIQUE_SESSION_NAME;
        v_status := DBMS_PIPE.CREATE_PIPE(v_mailbox, 1000, FALSE);
        IF v_status = 0 THEN
            DBMS_OUTPUT.PUT_LINE('Created mailbox: ' || v_mailbox);
```

(次のページに続く)

(前のページからの続き)

```

ELSE
    DBMS_OUTPUT.PUT_LINE('CREATE_PIPE failed - status: ' ||
        v_status);
END IF;
END create_mailbox;

PROCEDURE add_message (
    p_mailbox    VARCHAR2,
    p_item_1     VARCHAR2,
    p_item_2     VARCHAR2 DEFAULT 'END',
    p_item_3     VARCHAR2 DEFAULT 'END'
)
IS
    v_item_cnt   INTEGER := 0;
    v_status     INTEGER;
BEGIN
    DBMS_PIPE.PACK_MESSAGE(p_item_1);
    v_item_cnt := 1;
    IF p_item_2 != 'END' THEN
        DBMS_PIPE.PACK_MESSAGE(p_item_2);
        v_item_cnt := v_item_cnt + 1;
    END IF;
    IF p_item_3 != 'END' THEN
        DBMS_PIPE.PACK_MESSAGE(p_item_3);
        v_item_cnt := v_item_cnt + 1;
    END IF;
    v_status := DBMS_PIPE.SEND_MESSAGE(p_mailbox);
    IF v_status = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Added message with ' || v_item_cnt ||
            ' item(s) to mailbox ' || p_mailbox);
    ELSE
        DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE in add_message failed - ' ||
            'status: ' || v_status);
    END IF;
END add_message;

PROCEDURE empty_mailbox (
    p_mailbox    VARCHAR2,
    p_waittime   INTEGER DEFAULT 10
)
IS
    v_msgno     INTEGER DEFAULT 0;
    v_itemno    INTEGER DEFAULT 0;
    v_item      VARCHAR2(100);
    v_status     INTEGER;
BEGIN
    v_status := DBMS_PIPE.RECEIVE_MESSAGE(p_mailbox, p_waittime);
    WHILE v_status = 0 LOOP

```

(次のページに続く)

(前のページからの続き)

```

v_msgno := v_msgno + 1;
DBMS_OUTPUT.PUT_LINE('***** Start message #' || v_msgno ||
  ' *****');
BEGIN
  LOOP
    v_status := DBMS_PIPE.NEXT_ITEM_TYPE;
    EXIT WHEN v_status = 0;
    DBMS_PIPE.UNPACK_MESSAGE(v_item);
    v_itemno := v_itemno + 1;
    DBMS_OUTPUT.PUT_LINE('Item #' || v_itemno || ': ' ||
      v_item);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('***** End message #' || v_msgno ||
    ' *****');
  DBMS_OUTPUT.PUT_LINE('*');
  v_itemno := 0;
  v_status := DBMS_PIPE.RECEIVE_MESSAGE(p_mailbox,1);
END;
END LOOP;
DBMS_OUTPUT.PUT_LINE('Number of messages received: ' || v_msgno);
v_status := DBMS_PIPE.REMOVE_PIPE(p_mailbox);
IF v_status = 0 THEN
  DBMS_OUTPUT.PUT_LINE('Deleted mailbox ' || p_mailbox);
ELSE
  DBMS_OUTPUT.PUT_LINE('Could not delete mailbox - status: '
    || v_status);
END IF;
END empty_mailbox;
END mailbox;
```

以下は、mailbox の手順の実行を示しています。最初の手順では、UNIQUE_SESSION_NAME 関数によって生成された名前を使用してパブリックパイプを作成します。

```
EXEC mailbox.create_mailbox;

Created mailbox: PG$PIPE$13$3940
```

メールボックス名を使用して、mailbox パッケージと DBMS_PIPE パッケージにアクセスできる同じデータベース内のユーザーはメッセージを追加できます。

```
EXEC mailbox.add_message('PG$PIPE$13$3940','Hi, John','Can you attend a
meeting at 3:00, today?','- Mary');

Added message with 3 item(s) to mailbox PG$PIPE$13$3940

EXEC mailbox.add_message('PG$PIPE$13$3940','Don't forget to submit your
report','Thanks','- Joe');
```

(次のページに続く)

(前のページからの続き)

```
Added message with 3 item(s) to mailbox PG$PIPE$13$3940
```

最後に、メールボックスの内容を空にすることができます。

```
EXEC mailbox.empty_mailbox('PG$PIPE$13$3940');

***** Start message #1 *****
Item #1: Hi, John
Item #2: Can you attend a meeting at 3:00, today?
Item #3: -- Mary
***** End message #1 *****
*
***** Start message #2 *****
Item #1: Don't forget to submit your report
Item #2: Thanks,
Item #3: Joe
***** End message #2 *****
*
Number of messages received: 2
Deleted mailbox PG$PIPE$13$3940
```


3.11 DBMS_PROFILER

DBMS_PROFILER パッケージは、パフォーマンスプロファイリングセッション中に実行される PL/pgSQL および SPL ステートメントに関するパフォーマンス情報を収集して保存します。以下にリストされている関数と手順を使用して、プロファイリングツールを制御します。

機能/手順	戻りタイプ	説明
FLUSH_DATA	ステータスコード または例外	セッションを終了せずに、現在のセッションで収集されたパフォーマンスデータをフラッシュします（プロファイリングは続行されます）。
GET_VERSION(major OUT, minor OUT)	該当なし	このパッケージのバージョン番号を返します。
INTERNAL_VERSION_CHECK	状態コード	プロファイラーの現在のバージョンが現在のデータベースで動作することを確認します。
PAUSE_PROFILER	ステータスコード または例外	データ収集を一時停止します。
RESUME_PROFILER	ステータスコード または例外	データ収集を再開します。
START_PROFILER(run_comment, run_comment1 [, run_number OUT])	ステータスコード または例外	データ収集を開始します。
STOP_PROFILER	ステータスコード または例外	データ収集を停止し、パフォーマンスデータを PLSQL_PROFILER_RAWDATA テーブルにフラッシュします。

DBMS_PROFILER パッケージ内の関数は、成功または失敗を示すステータスコードを返します。DBMS_PROFILER プロシージャは、障害が発生した場合にのみ例外を発生させます。関数によって返されるステータスコードとメッセージ、およびプロシージャによって発生する例外を次の表に示します。

状態コード	メッセージ	例外	説明
-1	error version	version_mismatch	プロファイラーのバージョンとデータベースに互換性がありません。
0	success	該当なし	操作は正常に完了しました。
1	error_param	profiler_error	操作が誤ったパラメーターを受け取りました。
2	error_io	profiler_error	データのフラッシュ操作が失敗しました。

3.11.1 FLUSH_DATA

FLUSH_DATA 関数/手順は、プロファイラーセッションを終了せずに、現在のセッションで収集されたデータをフラッシュします。データは、『AdvancedServerPerformanceFeaturesGuide』で説明されているテーブルにフラッシュされます。関数とプロシージャのシグネチャは次のとおりです。

```
<status> INTEGER FLUSH_DATA
```

```
FLUSH_DATA
```

Parameters

status

操作によって返されたステータスコード。

3.11.2 GET_VERSION

GET_VERSION プロシージャは DBMS_PROFILER のバージョンを返します。プロシージャのシグネチャは次のとおりです。

```
GET_VERSION(<major> OUT INTEGER, <minor> OUT INTEGER)
```

Parameters

major

DBMS_PROFILER のメジャーバージョン番号。

minor

DBMS_PROFILER のマイナーバージョン番号。

3.11.3 INTERNAL_VERSION_CHECK

INTERNAL_VERSION_CHECK 機能は、DBMS_PROFILER の現在のバージョンが現在のデータベースで動作することを確認します。関数のシグネチャは次のとおりです。

```
<status> INTEGER INTERNAL_VERSION_CHECK
```

Parameters

status

操作によって返されたステータスコード。

3.11.4 PAUSE_PROFILER

PAUSE_PROFILER 関数/手順は、プロファイリングセッションを一時停止します。関数とプロシージャのシグネチャは次のとおりです。

```
<status> INTEGER PAUSE_PROFILER

PAUSE_PROFILER
```

Parameters

status

操作によって返されたステータスコード。

3.11.5 RESUME_PROFILER

RESUME_PROFILER 関数/手順は、プロファイリングセッションを一時停止します。関数とプロシージャのシグネチャは次のとおりです。

```
<status> INTEGER RESUME_PROFILER

RESUME_PROFILER
```

Parameters

status

操作によって返されたステータスコード。

3.11.6 START_PROFILER

START_PROFILER 関数/手順は、データ収集セッションを開始します。関数とプロシージャのシグネチャは次のとおりです。

```
<status> INTEGER START_PROFILER(<run_comment> TEXT := SYSDATE,
    <run_comment1> TEXT := ' ' [, <run_number> OUT INTEGER ])

START_PROFILER(<run_comment> TEXT := SYSDATE,
    <run_comment1> TEXT := ' ' [, <run_number> OUT INTEGER ])
```

Parameters

run_comment

プロファイラーセッションのユーザー定義のコメント。デフォルト値は SYSDATE です。

run_comment1

プロファイラーセッション用の追加のユーザー定義コメント。デフォルト値は" です。

run_number

プロファイラーセッションのセッション番号。

status

操作によって返されたステータスコード。

3.11.7 STOP_PROFILER

STOP_PROFILER 関数/手順は、プロファイリングセッションを停止し、パフォーマンス情報を DBMS_PROFILER テーブルとビューにフラッシュします。関数とプロシージャのシグネチャは次のとおりです。

```
<status> INTEGER STOP_PROFILER
```

```
STOP_PROFILER
```

Parameters

status

操作によって返されたステータスコード。

3.11.8 Using DBMS_PROFILER

DBMS_PROFILER パッケージは、プロファイリングセッション中に実行される PL/pgSQL および SPL ステートメントに関するパフォーマンス情報を収集して保存します。プロファイラーによって提供されるテーブルとビューのパフォーマンス情報を確認できます。

DBMS_PROFILER は、プロファイリングセッション内で実行される PL/pgSQL または SPL ステートメントの各行のパフォーマンス関連のカウンターとタイマーのセットを記録することで機能します。カウンターとタイマーは SYS.PLSQL_PROFILER_DATA という名前のテーブルに保存されます。プロファイリングセッションを完了すると、DBMS_PROFILER はセッション内で実行された PL/pgSQL または SPL コードの各行のパフォーマンス統計テーブルに行を書き込みます。たとえば、次の機能を実行する場合:

```

1 - CREATE OR REPLACE FUNCTION getBalance(acctNumber INTEGER)
2 - RETURNS NUMERIC AS $$
3 - DECLARE
4 -     result NUMERIC;
5 - BEGIN
6 -     SELECT INTO result balance FROM acct WHERE id = acctNumber;
7 -
8 -     IF (result IS NULL) THEN
9 -         RAISE INFO 'Balance is null';
10-    END IF;
11-
12-    RETURN result;
13- END;
14- $$ LANGUAGE 'plpgsql';

```

DBMS_PROFILER は、getBalance() 関数内のコードの各行に 1 つの PLSQL_PROFILER_DATA エントリを追加します (空白行とコメントを含む)。SELECT ステートメントに対応するエントリは、1 回だけ実行されました。実行に必要な時間はごくわずかでした。一方、RAISE INFO ステートメントに対応するエントリは 1 回実行されるか、まったく実行されません (balance 列の値に応じて)。

この関数の一部の行には実行可能コードが含まれていないため、これらの行のパフォーマンス統計には常にゼロ値が含まれます。

プロファイリングセッションを開始するには、DBMS_PROFILER.START_PROFILER 関数 (またはプロシージャ) を呼び出します。START_PROFILER を呼び出すと、AdvancedServer は、プロファイラーを停止または一時停止するまで STOP_PROFILER または PAUSE_PROFILER を呼び出すまで、セッションが実行するすべての PL/pgSQL または SPL 関数、プロシージャ、トリガー、または匿名ブロックをプロファイリングします)。

プロファイラーを起動 (または再開) する場合、プロファイラーは START_PROFILER (または RESUME_PROFILER) の呼び出し後に開始される関数/手順/トリガーのパフォーマンス統計のみを収集することに注意することが重要です。

プロファイラーがアクティブになっている間、AdvancedServer はメモリ内に多数のタイマーとカウンターを記録します。STOP_PROFILER (または FLUSH_DATA) 関数/手順を呼び出すと、DBMS_PROFILER はこれらのタ

イマーとカウンターを 3 つのテーブルのセットに書き込みます。

- SYS.PLSQL_PROFILER_RAWDATA

セッション内で実行された各ステートメントのパフォーマンスカウンターとタイマーが含まれます。

- SYS.PLSQL_PROFILER_RUNS

各実行の要約が含まれます (PLSQL_PROFILER_RAWDATA で見つかった情報を集約します)。

- SYS.PLSQL_PROFILER_UNITS

セッション内で実行された各コード単位 (関数、プロシージャ、トリガー、または匿名ブロック) の概要が含まれます。

さらに、 DBMS_PROFILER は PLSQL_PROFILER_RAWDATA テーブルのサブセットを含むビュー SYS.PLSQL_PROFILER_DATA を定義します。

非スーパーユーザーはプロファイリング情報を収集できますが、スーパーユーザーがプロファイリングテーブル (SYS スキーマに格納されている) に対する特定の権限を付与しない限り、そのプロファイリング情報を表示できません。これにより、管理者が秘密にしたい情報を公開することなく、非特権ユーザーがパフォーマンス統計を収集できます。

3.11.8.1 DBMS_PROFILER の表およびビューの問合せ

次のステップバイステップの例では、 DBMS_PROFILER を使用して、 AdvancedServer で配布されるサンプルデータに含まれるプロシージャ、関数、およびトリガーのパフォーマンス情報を取得します。

1. EDB-PSQL コマンドラインを開き、 AdvancedServer データベースへの接続を確立します。 EXEC ステートメントを使用して、プロファイリングセッションを開始します。

```
acctg=# EXEC dbms_profiler.start_profiler('profile list_emp');  
  
EDB-SPL Procedure successfully completed
```

注釈: (start_profiler() の呼び出しには、 DBMS_PROFILER がプロファイラーセッションに関連付けられているコメントが含まれています)。

2. 次に、 list_emp 関数を呼び出します。

```
acctg=# SELECT list_emp();  
INFO:  EMPNO      ENAME  
INFO:  -----  
INFO:  7369        SMITH  
INFO:  7499        ALLEN
```

(次のページに続く)

(前のページからの続き)

```

INFO:  7521      WARD
INFO:  7566      JONES
INFO:  7654      MARTIN
INFO:  7698      BLAKE
INFO:  7782      CLARK
INFO:  7788      SCOTT
INFO:  7839      KING
INFO:  7844      TURNER
INFO:  7876      ADAMS
INFO:  7900      JAMES
INFO:  7902      FORD
INFO:  7934      MILLER
list_emp
-----

(1 row)

```

3. `dbms_profiler.stop_profiler`: を呼び出してプロファイリングセッションを停止します

```

acctg=# EXEC dbms_profiler.stop_profiler;

EDB-SPL Procedure successfully completed

```

4. `dbms_profiler.start_profiler` 機能を使用して新しいセッションを開始します (新しいコメントが続きます)。

```

acctg=# EXEC dbms_profiler.start_profiler('profile get_dept_name and
emp_sal_trig');

EDB-SPL Procedure successfully completed

```

5. `get_dept_name` 関数を呼び出します:

```

acctg=# SELECT get_dept_name(10);
get_dept_name
-----
ACCOUNTING
(1 row)

```

6. トリガーを実行させる UPDATE ステートメントを実行します。

```

acctg=# UPDATE memp SET sal = 500 WHERE empno = 7902;
INFO: Updating employee 7902
INFO: ..Old salary: 3000.00
INFO: ..New salary: 500.00
INFO: ..Raise: -2500.00

```

(次のページに続く)

(前のページからの続き)

```
INFO: User enterprisedb updated employee(s) on 04-FEB-14
UPDATE 1
```

7. プロファイリングセッションを終了し、パフォーマンス情報をプロファイリングテーブルにフラッシュします。

```
acctg=# EXEC dbms_profiler.stop_profiler;

EDB-SPL Procedure successfully completed
```

8. ここで、`plsql_profiler_runs` テーブルを照会して、`runid:` で並べられたプロファイリングセッションのリストを表示します

```
acctg=# SELECT * FROM plsql_profiler_runs;
 runid | related_run | run_owner  | run_date
 | run_comment | run_total_time | run_system_info
 | run_comment1 | spare1
-----+-----+-----+-----+-----+-----+-----
+-----+
      1 |             | enterprisedb | 04-FEB-14 09:32:48.874315 | profile
list_emp |             |             | 4154 |
 |             |             |
      2 |             | enterprisedb | 04-FEB-14 09:41:30.546503 | profile
get_dept_name and emp_sal_trig |             | 2088 |
 |             |
(2 rows)
```

9. `plsql_profiler_units` テーブルをクエリして、各ユニット（各関数、プロシージャ、またはトリガー）が消費した時間を表示します。

```
acctg=# SELECT * FROM plsql_profiler_units;
 runid | unit_number | unit_type | unit_owner |
 unit_name | unit_timestamp | total_time | spare1 | spare2
-----+-----+-----+-----+-----+-----+-----+-----
+-----+
      1 |      16999 | FUNCTION | enterprisedb |
list_emp() |             |             | 4 |
      2 |      17002 | FUNCTION | enterprisedb |
user_audit_trig() |             |             | 1 |
      2 |      17000 | FUNCTION | enterprisedb | get_dept_name(p_deptno
numeric) |             | 1 |
      2 |      17004 | FUNCTION | enterprisedb |
emp_sal_trig() |             |             | 1 |
(4 rows)
```

10. `plsql_profiler_rawdata` テーブルを照会して、待機イベントカウンターと待機イベント時間のリス

トを表示します。

```
acctg=# SELECT runid, sourcecode, func_oid, line_number, exec_count,
tuples_returned, time_total FROM plsql_profiler_rawdata;
```

runid	sourcecode				
func_oid	line_number	exec_count	tuples_returned	time_total	
-----+-----+-----+-----+-----+-----					
-----+-----+-----+-----+-----+-----					
1	DECLARE				
	16999	1	0	0	0
1	v_empno	NUMERIC(4);			
	16999	2	0	0	0
1	v_ename	VARCHAR(10);			
	16999	3	0	0	0
1	emp_cur CURSOR FOR				
	16999	4	0	0	0
1	SELECT empno, ename FROM memp ORDER BY empno;				
	16999	5	0	0	0
1	BEGIN				
	16999	6	0	0	0
1	OPEN emp_cur;				
	16999	7	0	0	0
1	RAISE INFO 'EMPNO ENAME';				
	16999	8	1	0	0.001621
1	RAISE INFO '-----';				
	16999	9	1	0	0.000301
1	LOOP				
	16999	10	1	0	4.6e-05
1	FETCH emp_cur INTO v_empno, v_ename;				
	16999	11	1	0	0.001114
1	EXIT WHEN NOT FOUND;				
	16999	12	15	0	0.000206
1	RAISE INFO '%', v_empno, v_ename;				
	16999	13	15	0	8.3e-05
1	END LOOP;				
	16999	14	14	0	0.000773
1	CLOSE emp_cur;				
	16999	15	0	0	0
1	RETURN;				
	16999	16	1	0	1e-05
1	END;				
	16999	17	1	0	0
1					
	16999	18	0	0	0
2	DECLARE				
	17002	1	0	0	0
2	v_action	VARCHAR(24);			
	17002	2	0	0	0

(次のページに続く)

(前のページからの続き)

```

2 |      v_text          TEXT;
|      17002 |          3 |          0 |          0 |          0
2 | BEGIN
|      17002 |          4 |          0 |          0 |          0
2 |      IF TG_OP = 'INSERT' THEN
|      17002 |          5 |          0 |          0 |          0
2 |          v_action := ' added employee(s) on ';
|      17002 |          6 |          1 |          0 |      0.000143
2 |      ELSIF TG_OP = 'UPDATE' THEN
|      17002 |          7 |          0 |          0 |          0
2 |          v_action := ' updated employee(s) on ';
|      17002 |          8 |          0 |          0 |          0
2 |      ELSIF TG_OP = 'DELETE' THEN
|      17002 |          9 |          1 |          0 |      3.2e-05
2 |          v_action := ' deleted employee(s) on ';
|      17002 |         10 |          0 |          0 |          0
2 |      END IF;
|      17002 |         11 |          0 |          0 |          0
2 |      v_text := 'User ' || USER || v_action || CURRENT_DATE;
|      17002 |         12 |          0 |          0 |          0
2 |      RAISE INFO ' %', v_text;
|      17002 |         13 |          1 |          0 |      0.000383
2 |      RETURN NULL;
|      17002 |         14 |          1 |          0 |      6.3e-05
2 | END;
|      17002 |         15 |          1 |          0 |      3.6e-05
2 |
|      17002 |         16 |          0 |          0 |          0
2 | DECLARE
|      17000 |          1 |          0 |          0 |          0
2 |      v_dname          VARCHAR(14);
|      17000 |          2 |          0 |          0 |          0
2 | BEGIN
|      17000 |          3 |          0 |          0 |          0
2 |      SELECT INTO v_dname dname FROM dept WHERE deptno = p_deptno;
|      17000 |          4 |          0 |          0 |          0
2 |      RETURN v_dname;
|      17000 |          5 |          1 |          0 |      0.000647
2 |      IF NOT FOUND THEN
|      17000 |          6 |          1 |          0 |      2.6e-05
2 |          RAISE INFO 'Invalid department number %', p_deptno;
|      17000 |          7 |          0 |          0 |          0
2 |          RETURN '';
|      17000 |          8 |          0 |          0 |          0
2 |      END IF;
|      17000 |          9 |          0 |          0 |          0
2 | END;
|      17000 |         10 |          0 |          0 |          0

```

(次のページに続く)

(前のページからの続き)

```

2 |
| 17000 |          11 |          0 |          0 |          0
2 | DECLARE
| 17004 |          1 |          0 |          0 |          0
2 |     sal_diff      NUMERIC(7,2);
| 17004 |          2 |          0 |          0 |          0
2 | BEGIN
| 17004 |          3 |          0 |          0 |          0
2 |     IF TG_OP = 'INSERT' THEN
| 17004 |          4 |          0 |          0 |          0
2 |         RAISE INFO 'Inserting employee %', NEW.empno;
| 17004 |          5 |          1 |          0 |      8.4e-05
2 |         RAISE INFO '..New salary: %', NEW.sal;
| 17004 |          6 |          0 |          0 |          0
2 |         RETURN NEW;
| 17004 |          7 |          0 |          0 |          0
2 |     END IF;
| 17004 |          8 |          0 |          0 |          0
2 |     IF TG_OP = 'UPDATE' THEN
| 17004 |          9 |          0 |          0 |          0
2 |         sal_diff := NEW.sal - OLD.sal;
| 17004 |         10 |          1 |          0 |    0.000355
2 |         RAISE INFO 'Updating employee %', OLD.empno;
| 17004 |         11 |          1 |          0 |    0.000177
2 |         RAISE INFO '..Old salary: %', OLD.sal;
| 17004 |         12 |          1 |          0 |    5.5e-05
2 |         RAISE INFO '..New salary: %', NEW.sal;
| 17004 |         13 |          1 |          0 |    3.1e-05
2 |         RAISE INFO '..Raise      : %', sal_diff;
| 17004 |         14 |          1 |          0 |    2.8e-05
2 |         RETURN NEW;
| 17004 |         15 |          1 |          0 |    2.7e-05
2 |     END IF;
| 17004 |         16 |          1 |          0 |    1e-06
2 |     IF TG_OP = 'DELETE' THEN
| 17004 |         17 |          0 |          0 |          0
2 |         RAISE INFO 'Deleting employee %', OLD.empno;
| 17004 |         18 |          0 |          0 |          0
2 |         RAISE INFO '..Old salary: %', OLD.sal;
| 17004 |         19 |          0 |          0 |          0
2 |         RETURN OLD;
| 17004 |         20 |          0 |          0 |          0
2 |     END IF;
| 17004 |         21 |          0 |          0 |          0
2 | END;
| 17004 |         22 |          0 |          0 |          0
2 |
| 17004 |         23 |          0 |          0 |          0

```

(次のページに続く)

(前のページからの続き)

(68 rows)

11. `plsql_profiler_data` ビューを照会して、`plsql_profiler_rawdata` テーブルにある情報のサブセットを確認します。

```
acctg=# SELECT * FROM plsql_profiler_data;
runid | unit_number | line# | total_occur | total_time | min_time | max_time
| spare1 | spare2 | spare3 | spare4
-----+-----+-----+-----+-----+-----+-----
--
1 | 16999 | 1 | 0 | 0 | 0 |
0 | | | |
1 | 16999 | 2 | 0 | 0 | 0 |
0 | | | |
1 | 16999 | 3 | 0 | 0 | 0 |
0 | | | |
1 | 16999 | 4 | 0 | 0 | 0 |
0 | | | |
1 | 16999 | 5 | 0 | 0 | 0 |
0 | | | |
1 | 16999 | 6 | 0 | 0 | 0 |
0 | | | |
1 | 16999 | 7 | 0 | 0 | 0 |
0 | | | |
1 | 16999 | 8 | 1 | 0.001621 | 0.001621 |
0.001621 | | | |
1 | 16999 | 9 | 1 | 0.000301 | 0.000301 |
0.000301 | | | |
1 | 16999 | 10 | 1 | 4.6e-05 | 4.6e-05 | 4.6e-
05 | | | |
1 | 16999 | 11 | 1 | 0.001114 | 0.001114 |
0.001114 | | | |
1 | 16999 | 12 | 15 | 0.000206 | 5e-06 | 7.8e-
05 | | | |
1 | 16999 | 13 | 15 | 8.3e-05 | 2e-06 | 4.7e-
05 | | | |
1 | 16999 | 14 | 14 | 0.000773 | 4.7e-05 |
0.000116 | | | |
1 | 16999 | 15 | 0 | 0 | 0 |
0 | | | |
1 | 16999 | 16 | 1 | 1e-05 | 1e-05 | 1e-
05 | | | |
1 | 16999 | 17 | 1 | 0 | 0 |
0 | | | |
1 | 16999 | 18 | 0 | 0 | 0 |
0 | | | |
2 | 17002 | 1 | 0 | 0 | 0 |
```

(次のページに続く)

(前のページからの続き)

0								
2		17002		2		0		0 0
0								
2		17002		3		0		0 0
0								
2		17002		4		0		0 0
0								
2		17002		5		0		0 0
0								
2		17002		6		1		0.000143 0.000143
0.000143								
2		17002		7		0		0 0
0								
2		17002		8		0		0 0
0								
2		17002		9		1		3.2e-05 3.2e-05 3.2e-
05								
2		17002		10		0		0 0
0								
2		17002		11		0		0 0
0								
2		17002		12		0		0 0
0								
2		17002		13		1		0.000383 0.000383
0.000383								
2		17002		14		1		6.3e-05 6.3e-05 6.3e-
05								
2		17002		15		1		3.6e-05 3.6e-05 3.6e-
05								
2		17002		16		0		0 0
0								
2		17000		1		0		0 0
0								
2		17000		2		0		0 0
0								
2		17000		3		0		0 0
0								
2		17000		4		0		0 0
0								
2		17000		5		1		0.000647 0.000647
0.000647								
2		17000		6		1		2.6e-05 2.6e-05 2.6e-
05								
2		17000		7		0		0 0
0								
2		17000		8		0		0 0
0								
2		17000		9		0		0 0

(次のページに続く)

(前のページからの続き)

0									
2		17000		10		0		0	
0									
2		17000		11		0		0	
0									
2		17004		1		0		0	
0									
2		17004		2		0		0	
0									
2		17004		3		0		0	
0									
2		17004		4		0		0	
0									
2		17004		5		1		8.4e-05	
05								8.4e-05	
2		17004		6		0		0	
0									
2		17004		7		0		0	
0									
2		17004		8		0		0	
0									
2		17004		9		0		0	
0									
2		17004		10		1		0.000355	
0.000355								0.000355	
2		17004		11		1		0.000177	
0.000177								0.000177	
2		17004		12		1		5.5e-05	
05								5.5e-05	
2		17004		13		1		3.1e-05	
05								3.1e-05	
2		17004		14		1		2.8e-05	
05								2.8e-05	
2		17004		15		1		2.7e-05	
05								2.7e-05	
2		17004		16		1		1e-06	
06								1e-06	
2		17004		17		0		0	
0									
2		17004		18		0		0	
0									
2		17004		19		0		0	
0									
2		17004		20		0		0	
0									
2		17004		21		0		0	
0									
2		17004		22		0		0	

(次のページに続く)

(前のページからの続き)

0				
2	17004	23	0	0 0
0				
(68 rows)				

3.11.8.2 DBMS_PROFILER - Reference

AdvancedServer インストーラーは、PL/SQL パフォーマンスプロファイル情報を確認するために照会できる次のテーブルとビューを作成します。

テーブル名	説明
PLSQL_PROFILER_RUNS	runid で整理された、すべてのプロファイラーの実行に関する情報を含むテーブル。
PLSQL_PROFILER_UNITS	ユニットごとに編成された、すべてのプロファイラー実行に関する情報を含むテーブル。
PLSQL_PROFILER_DATA	パフォーマンス統計を含むビュー。
PLSQL_PROFILER_RAWDATA	パフォーマンス統計 and を含むテーブル、DRITA カウンターおよびタイマーの拡張パフォーマンス統計。

3.11.8.2.1 PLSQL_PROFILER_RUNS

PLSQL_PROFILER_RUNS テーブルには次の列が含まれます。

コラム	データ型	説明
runid	INTEGER (NOT NULL)	一意の識別子 (plsql_profiler_runnumber)
related_run	INTEGER	関連する実行の runid。
run_owner	TEXT	プロファイリングセッションを記録したロール。
run_date	TIMESTAMP WITHOUT TIME ZONE	プロファイリングセッションの開始時刻。
run_comment	TEXT	この実行に関連するユーザーのコメント
run_total_time	BIGINT	実行時間 (マイクロ秒)
run_system_info	TEXT	現在未使用
run_comment1	TEXT	追加のユーザーコメント
spare1	TEXT	現在未使用

3.11.8.2.2 PLSQL_PROFILER_UNITS

PLSQL_PROFILER_UNITS テーブルには次の列が含まれます。

コラム	データ型	説明
runid	INTEGER	一意の識別子 (plsql_profiler_runnumber)
unit_number	OID	ユニットを識別する pg_proc テーブルの行の OID に対応します。
unit_type	TEXT	PL/SQL ファンクション、プロシージャ、トリガーまたは匿名ブロック
unit_owner	TEXT	ユニットを所有するロールの ID。
unit_name	TEXT	ユニットの完全な署名。
unit_timestamp	TIMESTAMP WITHOUT TIME ZONE	ユニットの作成日 (現在は NULL)。
total_time	BIGINT	ユニット内で費やされた時間 (ミリ秒単位)
spare1	BIGINT	現在未使用
spare2	BIGINT	現在未使用

3.11.8.2.3 PLSQL_PROFILER_DATA

PLSQL_PROFILER_DATA ビューには次の列が含まれます。

コラム	データ型	説明
runid	INTEGER	一意の識別子 (plsql_profiler_runnumber)
unit_number	OID	現在の行を含むユニットのオブジェクト ID。
line#	INTEGER	プロファイルされたワークロードの現在の行番号。
total_occur	BIGINT	行が実行された回数。
total_time	DOUBLE PRECISION	行の実行に費やした時間 (秒単位)
min_time	DOUBLE PRECISION	ラインの最小実行時間。
max_time	DOUBLE PRECISION	ラインの最大実行時間。
spare1	NUMBER	現在未使用
spare2	NUMBER	現在未使用
spare3	NUMBER	現在未使用
spare4	NUMBER	現在未使用

3.11.8.2.4 PLSQL_PROFILER_RAWDATA

PLSQL_PROFILER_RAWDATA テーブルには、PLSQL_PROFILER_DATA ビューにある統計情報と待機イベント情報、および DRITA カウンターとタイマーによって返されるパフォーマンス統計が含まれます。

コラム	データ型	説明
runid	INTEGER	実行識別子 (plsql_profiler_runnumber)。
sourcecode	TEXT	プロファイルされたコードの個々の行。
func_oid	OID	現在の行を含むユニットのオブジェクト ID。
line_number	INTEGER	プロファイルされたワークロードの現在の行番号。
exec_count	BIGINT	行が実行された回数。
tuples_returned	BIGINT	現在未使用
time_total	DOUBLE PRECISION	行の実行に費やした時間 (秒単位)
time_shortest	DOUBLE PRECISION	ラインの最小実行時間。
time_longest	DOUBLE PRECISION	ラインの最大実行時間。
num_scans	BIGINT	現在未使用
tuples_fetched	BIGINT	現在未使用
tuples_inserted	BIGINT	現在未使用
tuples_updated	BIGINT	現在未使用
tuples_deleted	BIGINT	現在未使用
blocks_fetched	BIGINT	現在未使用
blocks_hit	BIGINT	現在未使用
wal_write	BIGINT	サーバーは、先行書き込みログバッファーへの書き込みを待機しました (この値が高いと予想されます)。
wal_flush	BIGINT	サーバーは、先行書き込みログがディスクにフラッシュされるのを待っています。
wal_file_sync	BIGINT	サーバーは、先書きログがディスクに同期するのを待っています (デフォルトでは「fsync」である wal_sync_method パラメーターに関連しています。このパラメーターを open_sync に変更すると、パフォーマンスが向上します)。

次のページに続く

表 2 – 前のページからの続き

コラム	データ型	説明
db_file_read	BIGINT	サーバーは（ディスクからの）読み取りの完了を待機しました。
db_file_write	BIGINT	サーバーは（ディスクへの）書き込みの完了を待機しました。
db_file_sync	BIGINT	サーバーは、オペレーティングシステムがすべての変更をディスクにフラッシュするのを待っています。
db_file_extend	BIGINT	サーバーは、ファイルの末尾に新しいページを追加している間、オペレーティングシステムを待機しました。
sql_parse	BIGINT	現在未使用。
query_plan	BIGINT	サーバーがクエリプランを生成しました。
other_lwlock_acquire	BIGINT	サーバーは、データを保護するために他の軽量ロックを待機しています。
shared_plan_cache_collision	BIGINT	サーバーは shared_plan_cache_collision イベントの完了を待っています。
shared_plan_cache_insert	BIGINT	サーバーは shared_plan_cache_insert イベントの完了を待っています。
shared_plan_cache_hit	BIGINT	サーバーは shared_plan_cache_hit イベントの完了を待っています。
shared_plan_cache_miss	BIGINT	サーバーは shared_plan_cache_miss イベントの完了を待っています。
shared_plan_cache_lock	BIGINT	サーバーは shared_plan_cache_lock イベントの完了を待っています。
shared_plan_cache_busy	BIGINT	サーバーは shared_plan_cache_busy イベントの完了を待っています。
shmindexlock	BIGINT	サーバーは、共有メモリ内のスペースの検出または割り当てを待機しています。
oidgenlock	BIGINT	サーバーは、OID の割り当てまたは割り当てを待機しています。
xidgenlock	BIGINT	サーバーは、トランザクション ID の割り当てまたは割り当てを待機しています。

次のページに続く

表 2 – 前のページからの続き

コラム	データ型	説明
procarraylock	BIGINT	サーバーは、トランザクション終了時にスナップショットの取得またはトランザクション ID のクリアを待機しました。
sinvalreadlock	BIGINT	サーバーは、共有無効化キューからメッセージを取得または削除するのを待機しました。
sinvalwritelock	BIGINT	サーバーは、共有無効化キューにメッセージを追加するのを待っています。
walbufmappinglock	BIGINT	サーバーは、WAL バッファ内のページを置き換えるのを待っています。
walwritelock	BIGINT	サーバーは、WAL バッファがディスクに書き込まれるのを待っています。
controlfilelock	BIGINT	サーバーは、制御ファイルの読み取りまたは更新、または新しい WAL ファイルの作成を待機しています。
checkpointlock	BIGINT	サーバーはチェックポイントの実行を待機しています。
clogcontrollock	BIGINT	サーバーは、トランザクションステータスの読み取りまたは更新を待機しています。
subtranscontrollock	BIGINT	サーバーは、サブトランザクション情報の読み取りまたは更新を待機しています。
multixactgenlock	BIGINT	サーバーは、共有マルチステート状態の読み取りまたは更新を待機しています。
multixactoffsetcontrollock	BIGINT	サーバーは、multixact オフセットマッピングの読み取りまたは更新を待機しています。
multixactmembercontrollock	BIGINT	サーバーは、multixact メンバーマッピングの読み取りまたは更新を待機しています。
relcacheinitlock	BIGINT	サーバーは、関係キャッシュ初期化ファイルの読み取りまたは書き込みを待機しました。
checkpointercommlock	BIGINT	サーバーは、fsync 要求の管理を待機しています。
twophasestatelock	BIGINT	サーバーは、準備されたトランザクションの状態の読み取りまたは更新を待機しています。

次のページに続く

表 2 – 前のページからの続き

コラム	データ型	説明
tablespacecreatelock	BIGINT	サーバーは、テーブルスペースの作成または削除を待機しています。
btreevacuumlock	BIGINT	サーバーは、B ツリーインデックスのバキューム関連情報の読み取りまたは更新を待機しています。
addinshmeminitlock	BIGINT	サーバーは、共有メモリ内のスペース割り当ての管理を待機しています。
autovacuumlock	BIGINT	autovacuum ワーカーの現在の状態の読み取りまたは更新を待機している autovacuum ランチャー。
autovacuumschedulelock	BIGINT	サーバーは、バキューム用に選択されたテーブルが引き続きバキュームを必要とすることを確認するために待機しました。
syncscanlock	BIGINT	サーバーは、同期スキャンのテーブルでスキャンの開始場所を取得するのを待機しています。
relationmappinglock	BIGINT	サーバーは、カタログからファイルノードへのマッピングを保存するために使用される関係マップファイルの更新を待機しました。
asyncctllock	BIGINT	サーバーは、共有通知状態の読み取りまたは更新を待機しています。
asyncqueuelock	BIGINT	サーバーは、通知メッセージの読み取りまたは更新を待機しています。
serializableexacthashlock	BIGINT	サーバーは、シリアライズ可能なトランザクションに関する情報の取得または保存を待機しています。
serializablefinishedlistlock	BIGINT	サーバーは、終了したシリアル化可能なトランザクションのリストへのアクセスを待機しています。
serializablepredicatelocklistlock	BIGINT	サーバーは、シリアル化可能なトランザクションによって保持されているロックのリストに対する操作の実行を待機しています。
oldserxidlock	BIGINT	サーバーは、競合するシリアル化可能なトランザクションの読み取りまたは記録を待機しています。

次のページに続く

表 2 – 前のページからの続き

コラム	データ型	説明
syncreplock	BIGINT	サーバーは、同期レプリカに関する情報の読み取りまたは更新を待機しています。
backgroundworkerlock	BIGINT	サーバーは、バックグラウンドワーカーの状態の読み取りまたは更新を待機しています。
dynamicsharedmemorycontrollock	BIGINT	サーバーは、動的共有メモリ状態の読み取りまたは更新を待機しています。
autofilelock	BIGINT	サーバーは postgresql.auto.conf ファイルの更新を待機しています。
replicationslotallocationlock	BIGINT	サーバーは、複製スロットの割り当てまたは解放を待機しています。
replicationslotcontrollock	BIGINT	サーバーは、複製スロットの状態の読み取りまたは更新を待機しています。
committscontrollock	BIGINT	サーバーは、トランザクションコミットタイムスタンプの読み取りまたは更新を待機しています。
committslock	BIGINT	サーバーは、トランザクションのタイムスタンプに設定された最後の値の読み取りまたは更新を待機しています。
replicationoriginlock	BIGINT	サーバーは、複製元のセットアップ、ドロップ、または使用を待機しています。
multixacttruncationlock	BIGINT	サーバーは、多重情報の読み取りまたは切り捨てを待機しています。
oldsnapshottimemaplock	BIGINT	サーバーは、古いスナップショット制御情報の読み取りまたは更新を待機しています。
backendrandomlock	BIGINT	サーバーは、乱数の生成を待機しています。
logicalrepworkerlock	BIGINT	サーバーは、論理レプリケーションワーカーのアクションが完了するのを待機しました。
clogtruncationlock	BIGINT	サーバーは、先行書き込みログの切り捨てを待機しているか、先行書き込みログの切り捨てが終了するのを待っています。
bulkloadlock	BIGINT	サーバーは、bulkloadlock がデータを一括アップロードするのを待っています。

次のページに続く

表 2 – 前のページからの続き

コラム	データ型	説明
edbresourcemanagerlock	BIGINT	edbresourcemanagerlock は、edb リソースマネージャーロックモジュールに関する詳細を提供します。
wal_write_time	BIGINT	サーバーが wal_write 待機イベントが先書きログバッファに書き込むのを待機した時間（この値が高いと予想される）。
wal_flush_time	BIGINT	サーバーが wal_flush 待機イベントを待機してログを先書きしてディスクにフラッシュする時間。
wal_file_sync_time	BIGINT	サーバーが wal_file_sync 待機イベントを待ってディスクに同期するための先行書き込みログを待機した時間（デフォルトでは「fsync」である wal_sync_method パラメーターに関連-このパラメーターを変更することでパフォーマンスを向上させることができます）open_sync へ）。
db_file_read_time	BIGINT	サーバーが（ディスクからの）読み取りの完了のために db_file_read 待機イベントを待機した時間。
db_file_write_time	BIGINT	サーバーが（ディスクへの）書き込みの完了のために db_file_write 待機イベントを待機した時間。
db_file_sync_time	BIGINT	サーバーがすべての変更をディスクに同期するために db_file_sync 待機イベントを待機した時間。
db_file_extend_time	BIGINT	サーバーがファイルの最後に新しいページを追加する際に db_file_extend 待機イベントを待機した時間。
sql_parse_time	BIGINT	サーバーが SQL ステートメントを解析するために sql_parse 待機イベントを待機した時間。
query_plan_time	BIGINT	サーバーが query_plan 待機イベントを待って SQL ステートメントの実行計画を計算した時間。
other_lwlock_acquire_time	BIGINT	サーバーがデータを保護するために other_lwlock_acquire 待機イベントを待機した時間。

次のページに続く

表 2 – 前のページからの続き

コラム	データ型	説明
shared_plan_cache_collision_time	BIGINT	サーバーが shared_plan_cache_collision 待機イベントを待機した時間。
shared_plan_cache_insert_time	BIGINT	サーバーが shared_plan_cache_insert 待機イベントを待機した時間。
shared_plan_cache_hit_time	BIGINT	サーバーが shared_plan_cache_hit 待機イベントを待機した時間。
shared_plan_cache_miss_time	BIGINT	サーバーが shared_plan_cache_miss 待機イベントを待機した時間。
shared_plan_cache_lock_time	BIGINT	サーバーが shared_plan_cache_lock 待機イベントを待機した時間。
shared_plan_cache_busy_time	BIGINT	サーバーが shared_plan_cache_busy 待機イベントを待機した時間。
shmindexlock_time	BIGINT	サーバーが shmindexlock 待機イベントを待機して、共有メモリ内のスペースを見つけるか割り当てる時間。
oidgenlock_time	BIGINT	サーバーが oidgenlock 待機イベントを待機して OID を割り当てるか割り当てる時間。
xidgenlock_time	BIGINT	サーバーがトランザクション ID の割り当てまたは割り当てのために xidgenlock 待機イベントを待機した時間。
procarraylock_time	BIGINT	トランザクション終了時にトランザクション ID をクリアするためにサーバーが procarraylock 待機イベントを待機した時間。
sinvalreadlock_time	BIGINT	サーバーが共有無効化キューからメッセージを取得または削除するために sinvalreadlock 待機イベントを待機した時間。
sinvalwritelock_time	BIGINT	サーバーが共有無効化キューにメッセージを追加するために sinvalwritelock 待機イベントを待機した時間。

次のページに続く

表 2 – 前のページからの続き

コラム	データ型	説明
walbufmappinglock_time	BIGINT	サーバーが walbufmappinglock 待機イベントを待って WAL バッファ内のページを置き換える時間。
walwritelock_time	BIGINT	サーバーが WAL バッファをディスクに書き込むための walwritelock 待機イベントを待機した時間。
controlfilelock_time	BIGINT	サーバーが controlfilelock 待機イベントを待って制御ファイルを読み取ったり更新したり、新しい WAL ファイルを作成したりした時間。
checkpointlock_time	BIGINT	サーバーがチェックポイントを実行するために checkpointlock 待機イベントを待機した時間。
clogcontrollock_time	BIGINT	サーバーがトランザクションステータスの読み取りまたは更新のために clogcontrollock 待機イベントを待機した時間。
subtranscontrollock_time	BIGINT	サーバーがサブトランザクション情報を読み取りまたは更新するために subtranscontrollock 待機イベントを待機した時間。
multixactgenlock_time	BIGINT	サーバーが multixactgenlock 待機イベントを待機して、共有マルチステート状態を読み取ったり更新したりした時間。
multixactoffsetcontrollock_time	BIGINT	サーバーが multixactoffsetcontrollock 待機イベントを待機して、multixact オフセットマッピングの読み取りまたは更新を行った時間。
multixactmembercontrollock_time	BIGINT	サーバーが multixact メンバーマッピングを読み取りまたは更新するために multixactmembercontrollock 待機イベントを待機した時間。
relcacheinitlock_time	BIGINT	サーバーが relcacheinitlock 待機イベントがリレーションキャッシュ初期化ファイルを読み書きするのを待機した時間。

次のページに続く

表 2 – 前のページからの続き

コラム	データ型	説明
checkpointintercommlock_time	BIGINT	サーバーが fsync 要求を管理するために checkpointintercommlock 待機イベントを待機した時間。
twophasestatelock_time	BIGINT	サーバーが準備済みトランザクションの状態を読み取りまたは更新するために twophasestatelock 待機イベントを待機した時間。
tablespacecreatelock_time	BIGINT	サーバーが tablespacecreatelock 待機イベントがテーブルスペースを作成または削除するのを待機した時間。
btreevacuumlock_time	BIGINT	B ツリーインデックスのバキューム関連情報を読み取りまたは更新するために、サーバーが btreevacuumlock 待機イベントを待機した時間。
addinshmeminitlock_time	BIGINT	共有メモリ内のスペース割り当てを管理するために、サーバーが addinshmeminitlock 待機イベントを待機した時間。
autovacuumlock_time	BIGINT	自動バキュームワーカーの現在の状態を読み取りまたは更新するために、サーバーが autovacuumlock 待機イベントを待機した時間。
autovacuumschedulelock_time	BIGINT	バキューム用に選択されたテーブルが引き続きバキューム処理を必要とすることを保証するために、サーバーが autovacuumschedulelock 待機イベントを待機した時間。
syncscanlock_time	BIGINT	同期スキャンのテーブルでスキャンの開始位置を取得するために、サーバーが syncscanlock 待機イベントを待機した時間。
relationmappinglock_time	BIGINT	サーバーが relationmappinglock 待機イベントを待機して、カタログからファイルノードへのマッピングを保存するために使用される関係マップファイルを更新した時間。

次のページに続く

表 2 – 前のページからの続き

コラム	データ型	説明
<code>asyncctllock_time</code>	BIGINT	サーバーが <code>asyncctllock</code> 待機イベントが共有通知状態を読み取ったり更新したりするのを待機した時間。
<code>asyncqueueunlock_time</code>	BIGINT	サーバーが通知メッセージを読み取りまたは更新するための <code>asyncqueueunlock</code> 待機イベントを待機した時間。
<code>serializableexacthashlock_time</code>	BIGINT	シリアル化可能なトランザクションに関する情報を取得または保存するために、サーバーが <code>serializableexacthashlock</code> 待機イベントを待機した時間。
<code>serializablefinishedlistlock_time</code>	BIGINT	サーバーが <code>serializablefinishedlistlock</code> 待機イベントを待機して、終了したシリアル化可能なトランザクションのリストにアクセスした時間。
<code>serializablepredicatelocklistlock_time</code>	BIGINT	サーバーが、シリアル化可能なトランザクションによって保持されているロックのリストに対して操作を実行するために <code>serializablepredicatelocklistlock</code> 待機イベントを待機した時間。
<code>oldserxidlock_time</code>	BIGINT	競合する直列化可能なトランザクションを読み取りまたは記録するために、サーバーが <code>oldserxidlock</code> 待機イベントを待機した時間。
<code>syncreplock_time</code>	BIGINT	同期レプリカに関する情報を読み取りまたは更新するために、サーバーが <code>syncreplock</code> 待機イベントを待機した時間。
<code>backgroundworkerlock_time</code>	BIGINT	サーバーが <code>backgroundworkerlock</code> 待機イベントを待ってバックグラウンドワーカーの状態を読み取ったり更新したりした時間。
<code>dynamicsharedmemorycontrollock_time</code>	BIGINT	サーバーが <code>dynamicsharedmemorycontrollock</code> 待機イベントを待機して、動的共有メモリの状態を読み取りまたは更新した時間。

次のページに続く

表 2 – 前のページからの続き

コラム	データ型	説明
autofilelock_time	BIGINT	サーバーが postgresql.auto.conf ファイルを更新する autofilelock 待機イベントを待機した時間。
replactionslotallocationlock_time	BIGINT	サーバーが replactionslotallocationlock 待機イベントを待機して複製スロットを割り当てたり解放したりした時間。
replactionslotcontrollock_time	BIGINT	サーバーが replactionslotcontrollock 待機イベントを待機して複製スロットの状態を読み取ったり更新したりした時間。
committscontrollock_time	BIGINT	サーバーがトランザクションコミットタイムスタンプの読み取りまたは更新のために committscontrollock 待機イベントを待機した時間。
committslock_time	BIGINT	トランザクションタイムスタンプに設定された最後の値を読み取りまたは更新するために、サーバーが committslock 待機イベントを待機した時間。
replicationoriginlock_time	BIGINT	サーバーが replicationoriginlock 待機イベントが複製元をセットアップ、ドロップ、または使用するのを待機した時間。
multixacttruncationlock_time	BIGINT	サーバーが multixacttruncationlock 待機イベントがマルチサクト情報を読み取りまたは切り捨てるのを待機した時間。
oldsnapshottimemaplock_time	BIGINT	サーバーが oldsnapshottimemaplock 待機イベントを待機して、古いスナップショット制御情報を読み取ったり更新したりした時間。
backendrandomlock_time	BIGINT	サーバーが backendrandomlock 待機イベントを待って乱数を生成した時間。

次のページに続く

表 2 – 前のページからの続き

コラム	データ型	説明
logicalrepworkerlock_time	BIGINT	論理レプリケーションワーカーに対するアクションの logicalrepworkerlock 待機イベントが終了するまでサーバーが待機した時間。
clogtruncationlock_time	BIGINT	サーバーが clogtruncationlock 待機イベントを待って先書きログを切り捨てるか、先書きログの切り捨てが終了するのを待っている時間。
bulkloadlock_time	BIGINT	サーバーが bulkloadlock 待機イベントを待ってデータを一括アップロードした時間。
edbresourcemanagerlock_time	BIGINT	サーバーが edbresourcemanagerlock 待機イベントを待機した時間。
totalwaits	BIGINT	イベント待機の総数。
totalwaittime	BIGINT	イベントの待機に費やされた合計時間。

3.12 DBMS_RANDOM

DBMS_RANDOM パッケージには、ランダムな値を生成するためのいくつかのメソッドが用意されています。DBMS_RANDOM パッケージで利用可能な手順と機能を次の表に示します。

機能/手順	戻りタイプ	説明
INITIALIZE (val)	該当なし	指定されたシード value で DBMS_RANDOM パッケージを初期化します。非推奨ですが、下位互換性のためにサポートされています。
NORMAL ()	NUMBER	ランダムな NUMBER を返します。
RANDOM	INTEGER	-2^{31} 以上 2^{31} 未満の値を持つランダムな INTEGER を返します。非推奨ですが、下位互換性のためにサポートされています。
SEED (val)	該当なし	指定された value でシードをリセットします。
SEED (val)	該当なし	指定された value でシードをリセットします。
STRING (opt, len)	VARCHAR2	ランダムな文字列を返します。
TERMINATE	該当なし	TERMINATE は効果がありません。非推奨ですが、下位互換性のためにサポートされています。
VALUE	NUMBER	0 以上 1 未満の値を持つ乱数を、38 桁の精度で返します。
VALUE (low, high)	NUMBER	low 以上 high 未満の値を持つ乱数を返します。

3.12.1 INITIALIZE

INITIALIZE プロシージャは、DBMS_RANDOM パッケージをシード値で初期化します。署名は次のとおりです。

```
INITIALIZE(<val> IN INTEGER)
```

この手順は非推奨と見なされる必要があります。下位互換性のためにのみ含まれています。

Parameters

val

val は DBMS_RANDOM パッケージアルゴリズムで使用するシード値です。

Example

次のコードスニペットは、シード値 6475 で DBMS_RANDOM パッケージを初期化する INITIALIZE プロシージャの呼び出しを示しています。

```
DBMS_RANDOM.INITIALIZE(6475);
```

3.12.2 NORMAL

NORMAL 関数は、NUMBER タイプの乱数を返します。署名は次のとおりです。

```
<result> NUMBER NORMAL()
```

Parameters

result

result は NUMBER タイプのランダムな値です。

Example

次のコードスニペットは、NORMAL 関数の呼び出しを示しています。

```
x:= DBMS_RANDOM.NORMAL();
```

3.12.3 RANDOM

RANDOM 関数は、 -2^{31} 以上 2^{31} 未満のランダムな INTEGER 値を返します。署名は次のとおりです。

```
<result> INTEGER RANDOM()
```

この関数は非推奨と見なされるべきです。下位互換性のためにのみ含まれています。

Parameters

result

result は INTEGER タイプのランダムな値です。

Example

次のコードスニペットは、RANDOM 関数の呼び出しを示しています。呼び出しは乱数を返します:

```
x := DBMS_RANDOM.RANDOM();
```

3.12.4 SEED

SEED 手順の最初の形式は、DBMS_RANDOM パッケージのシード値を INTEGER 値にリセットします。SEED 手順は 2 つの形式で利用できます。最初のフォームの署名は次のとおりです。

```
SEED(<val> IN INTEGER)
```

Parameters

val

val は DBMS_RANDOM パッケージアルゴリズムで使用するシード値です。

Example

次のコードスニペットは、SEED プロシージャの呼び出しを示しています。呼び出しは、シード値を 8495 に設定します。

```
DBMS_RANDOM.SEED(8495);
```

3.12.5 SEED

SEED プロシージャの 2 番目の形式は、DBMS_RANDOM パッケージのシード値を文字列値でリセットします。SEED 手順は 2 つの形式で利用できます。2 番目の形式の署名は次のとおりです。

```
SEED(<val> IN VARCHAR2)
```

Parameters

val

val は DBMS_RANDOM パッケージアルゴリズムで使用するシード値です。

Example

次のコードスニペットは、SEED プロシージャの呼び出しを示しています。呼び出しはシード値を abc123 に設定します。

```
DBMS_RANDOM.SEED('abc123');
```

3.12.6 STRING

STRING 関数は、ユーザー指定の形式でランダムな VARCHAR2 文字列を返します。STRING 関数のシグネチャは次のとおりです。

```
<result> VARCHAR2 STRING(<opt> IN CHAR, <len> IN NUMBER)
```

Parameters

opt

返される文字列のフォーマットオプション。option は:

Option	Specifies Formatting Option
u または U	大文字のアルファ文字列
l または L	小文字のアルファ文字列
a または A	大/小文字混合ストリング
x または X	大文字の英数字文字列
p または P	印刷可能な文字

len

返される文字列の長さ。

result

result は VARCHAR2 タイプのランダムな値です。

Example

次のコードスニペットは、STRING 関数の呼び出しを示しています。呼び出しは、10 文字のランダムな英数字文字列を返します。

```
x := DBMS_RANDOM.STRING('X', 10);
```

3.12.7 TERMINATE

TERMINATE プロシージャは効果がありません。署名は次のとおりです。

```
TERMINATE
```

TERMINATE 手順は非推奨と見なされる必要があります。この手順は互換性のためにのみサポートされています。

3.12.8 価値

VALUE 関数は、38 桁の精度で、0 以上 1 未満のランダムな NUMBER を返します。VALUE 関数には 2 つの形式があります。最初のフォームの署名は次のとおりです。

```
<result> NUMBER VALUE()
```

Parameters

result

result は NUMBER タイプのランダムな値です。

Example

次のコードスニペットは、VALUE 関数の呼び出しを示しています。呼び出しはランダムな NUMBER を返します:

```
x := DBMS_RANDOM.VALUE();
```

3.12.9 価値

VALUE 関数は、ユーザー指定の境界の間の値を持つランダムな NUMBER を返します。VALUE 関数には 2 つの形式があります。2 番目の形式の署名は次のとおりです。

```
<result> NUMBER VALUE(<low> IN NUMBER, <high> IN NUMBER)
```

Parameters

low

low は、ランダム値の下限を指定します。ランダム値は low に等しい場合があります。

high

high は、ランダム値の上限を指定します。ランダム値は high 未満になります。

result

result は NUMBER タイプのランダムな値です。

Example

次のコードスニペットは、VALUE 関数の呼び出しを示しています。呼び出しは、1 以上で 100 未満の値を持つランダムな NUMBER を返します。

```
x := DBMS_RANDOM.VALUE(1, 100);
```

3.13 DBMS_REDACT

DBMS_REDACT パッケージは、クエリによって返されたデータの編集またはマスキングを可能にします。DBMS_REDACT パッケージには、ポリシーの作成、ポリシーの変更、ポリシーの有効化、ポリシーの無効化、およびポリシーの削除の手順が含まれています。DBMS_REDACT パッケージで利用可能な手順を次の表に示します。

機能/手順	機能または手順	戻りタイプ	説明
ADD_POLICY(object_schema, object_name, policy_name, policy_description, column_name, column_description, function_type, function_parameters, expression, enable, regexp_pattern, regexp_replace_string, regexp_position, regexp_occurence, regexp_match_parameter, custom_function_expression)	手続き	該当なし	データ編集ポリシーを追加します。
ALTER_POLICY(object_schema, object_name, policy_name, action, column_name, function_type, function_parameters, expression, regexp_pattern, regexp_replace_string, regexp_position, regexp_occurence, regexp_match_parameter, policy_description, column_description, custom_function_expression)	手続き	該当なし	既存のデータ改訂ポリシーを変更します。
DISABLE_POLICY(object_schema, object_name, policy_name)	手続き	該当なし	既存のデータ改訂ポリシーを無効にします。
ENABLE_POLICY(object_schema, object_name, policy_name)	手続き	該当なし	以前に無効にしたデータ修正ポリシーを有効にします。
DROP_POLICY(object_schema, object_name, policy_name)	手続き	該当なし	データ編集ポリシーを削除します。
UPDATE_FULL_REDACTION_VALUES(number_val, binfloat_val, bindouble_val, char_val, varchar_val, nchar_val, nvarchar_val, datecol_val, ts_val, tswtz_val, blob_val, clob_val, nclob_val)	手続き	該当なし	指定したデータ型の完全な墨消しデフォルト値を更新します。

データリダクション機能は DBMS_REDACT パッケージを使用して、テーブルの列タイプとリダクションタイプに基づいて列のデータをリダクションするポリシーまたは条件を定義します。

データ編集ポリシーを作成または変更するには、テーブルの所有者である必要があります。ユーザーは、すべての列のリダクションポリシーから免除されます。これらのポリシーは、テーブル所有者またはスーパーユーザーがデフォルトです。

3.13.1 Using DBMS_REDACT Constants and Function Parameters

DBMS_REDACT パッケージは定数を使用し、データ編集タイプのいずれかを使用して列データを編集します。改訂タイプは、`dbms_redact.add_policy` および `dbms_redact.alter_policy` 手順の `function_type` パラメータに基づいて決定できます。次の表は、`dbms_redact.add_policy` および `dbms_redact.alter_policy` の `function_type` パラメーターの値を示しています。

定数	種類	価値	説明
NONE	INTEGER	0	リダクションなし、テーブルに対するクエリの結果への影響なし。
FULL	INTEGER	1	完全な編集。列データのすべての値を編集します。
PARTIAL	INTEGER	2	部分的な編集。列データの一部を編集します。
RANDOM	INTEGER	4	ランダムリダクション。各クエリは、列のデータ型に応じて異なるランダム値になります。
REGEXP	INTEGER	5	正規表現ベースの墨消し、墨消しするデータのパターンを検索します。
CUSTOM	INTEGER	99	カスタム編集タイプ。

次の表に、`dbms_redact.alter_policy` の `action` パラメーターの値を示します。

定数	種類	価値	説明
ADD_COLUMN	INTEGER	1	改訂ポリシーに列を追加します。
DROP_COLUMN	INTEGER	2	リダクションポリシーから列を削除します。
MODIFY_EXPRESSION	INTEGER	3	リダクションポリシーの表現を変更します。式が BOOLEAN 値から TRUE に評価されると、墨消しが適用されます。
MODIFY_COLUMN	INTEGER	4	リダクションポリシーの列を変更して、リダクション関数のタイプまたは関数パラメーターを変更します。
SET_POLICY_DESCRIPTION	INTEGER	5	墨消しポリシーの説明を設定します。
SET_COLUMN_DESCRIPTION	INTEGER	6	列に対して実行されるリダクションの説明を設定します。

部分的なデータリダクションを使用すると、列データの一部のみをリダクションできます。部分的なリダクションを使用するには、`dbms_redact.add_policy` プロシージャの `function_type` パラメーターを

`dbms_redact.partial` に設定し、`function_parameters` パラメーターを使用して部分的なリダクションの動作を指定する必要があります。

データ修正機能は、次のデータ型を使用するポリシーを構成するための事前定義された形式を提供します。

- Character
- Number
- Datetime

次の表は、データ型に関する部分的なリダクションのフォーマット記述子を強調しています。以下に説明する例は、文字列データ型（このシナリオでは社会保障番号（SSN））、Number データ型、および DATE データ型の墨消しを実行する方法を示しています。

データ型	フォーマット記述子	説明	例
キ ャ ラ ク タ ー	REDACT_PARTIAL_INPUT_FORMAT	入 力 形 式 を指定しま す。入力文 字列の各文 字に V を 入 力 す る と、編集さ れ る 可 能 性がありま す。空白ス ペースやハ イフンなど の区切り文 字と見なす ことができ る入力文字 列の各文字 に F を入 力します。	123-45-6789 などの SSN 文 字列の最初の 5 桁をマスク するために 'VVVFVVVFVVVV, VVV-VV-VVVV,X,1,5' を検 討し、それをフォーマットする ためにハイフンを追加すると、 XXX-6789. などの文字列にな ります。123-45-6789 などの SSN 文字列に一致するフィール ド値 VVVFVVVFVVVV。

次のページに続く

表 3 – 前のページからの続き

	REDACT_PARTIAL_OUTPUT_FORMAT	出力形式を指定します。入力文字列の各文字に V を入力すると、編集される可能性があります。入力形式の各 F 文字をハイフンやその他の区切り文字などの文字に置き換えます。	フィールド値 VVV-VV-VVVV を使用して、SSN 文字列を XXX-XX-6789 に編集し、X は REDACT_PARTIAL_MASKCHAR フィールドから取得できます。
	REDACT_PARTIAL_MASKCHAR	墨消しに使用する文字を指定します。	SSN 文字列を XXX-XX-6789 に編集するための値 X。
	REDACT_PARTIAL_MASKFROM	入力フォーマット内のどの V からリダクションを開始するかを指定します。	VVVFVVVFVVV の入力形式の最初の V から始まる XXX-XX-6789 などの文字列への SSN 文字列を編集するための値 1。
	REDACT_PARTIAL_MASKTO	入力フォーマット内で、リダクションを終了する V を指定します。	入力フォーマット VVVFVVVFVVV 内の 5 番目の V までの SSN スtring を XXX-XX-6789 などの String に編集するための値 5。

次のページに続く

表 3 – 前のページからの続き

数	REDACT_PARTIAL_MASKCHAR	0～9 の範囲で表示される文字を指定します。	社会保障番号 123456789 の最初の 5 桁を 999996789 に編集するための ‘9, 1, 5’。
	REDACT_PARTIAL_MASKFROM	リダクションの開始桁位置を指定します。	
	REDACT_PARTIAL_MASKTO	リダクションの終了桁位置を指定します。	
日時	REDACT_PARTIAL_DATE_MONTH	‘m’ は月を編集します。特定の月をマスクするには、‘m#’ を指定します。# は、1 と 12 の間の番号で指定された月を示します。	m3 は 3 月として表示されます。
	REDACT_PARTIAL_DATE_DAY	‘d’ は月の日付を編集します。月の日でマスクするには、小文字の d に 1-31 を追加します。	d3 は 03 と表示されます。

次のページに続く

表 3 – 前のページからの続き

	REDACT_PARTIAL_DATE_YEAR	‘y’ は年を編集します。年をマスクするには、小文字の y に 1-9999 を追加します。	y1960 は 60 と表示されます。
	REDACT_PARTIAL_DATE_HOUR	‘h’ は時間を編集します。時間をマスクするには、小文字の h に 0-23 を追加します。	h18 は 18 と表示されます。
	REDACT_PARTIAL_DATE_MINUTE	‘m’ は分を編集します。分でマスクするには、小文字の m に 0-59 を追加します。	m20 は 20 と表示されます。
	REDACT_PARTIAL_DATE_SECOND	‘s’ は2番目を編集します。秒でマスクするには、小文字の s に 0-59 を追加します。	s40 は 40 と表示されます。

次の表は、部分的な改訂に使用できる `function_parameters` 値を表しています。

関数パラメーター	データ型	価値	説明
----------	------	----	----

次のページに続く

表 4 – 前のページからの続き

REDACT_US_SSN_F5	VARCHAR2	'VVVFVVVFVVV, VVV-VV-VVVV,X,1,5'	SSN の最初の 5 つの数字 を編集します。 Example: 番 号 123-45-6789 は XXX-XX-6789 に な ります。
REDACT_US_SSN_L4	VARCHAR2	'VVVFVVVFVVV, VVV-VV-VVVV,X,6,9'	SSN の最後の 4 つの数字 を編集します。 Example: 番 号 123-45-6789 は 123-45-XXXX に な ります。
REDACT_US_SSN_ENTIRE	VARCHAR2	'VVVFVVVFVVV, VVV-VV-VVVV,X,1,9'	SSN 全体を編集します。 Example: 番 号 123-45-6789 は XXX-XX-XXXX に な ります。
REDACT_NUM_US_SSN_F5	VARCHAR2	'9,1,5'	列が数値データ型の場合、 SSN の最初の 5 つの数値 を編集します。 Example: 番 号 123456789 は 999996789 に なり ま す。
REDACT_NUM_US_SSN_L4	VARCHAR2	'9,6,9'	列が数値データ型の場合、 SSN の最後の 4 つの数値 を編集します。 Example: 番 号 123456789 は 123459999 に なり ま す。
REDACT_NUM_US_SSN_ENTIRE	VARCHAR2	'9,1,9'	列が数値データ型の場合、 SSN 全体を編集します。 Example: 番 号 123456789 は 999999999 に なり ま す。

次のページに続く

表 4 – 前のページからの続き

REDACT_ZIP_CODE	VARCHAR2	'VVVVV,VVVV,X,1,5'	5 桁の郵便番号を編集します。 Example: 12345 は XXXXX になります。
REDACT_NUM_ZIP_CODE	VARCHAR2	'9,1,5'	列が数値データ型の場合、5 桁の郵便番号を編集します。 Example: 12345 は 99999 になります。
REDACT_CCN16_F12	VARCHAR2	'VVVVFVVVVFVVVVFVVV,VVVV-VVVV-VVVV-VVV,*,1,12'	16 桁のクレジットカード番号を編集し、4 桁のみを表示します。 Example: 1234 567890002358 は **** -**** -**** -2358 になります。
REDACT_DATE_MILLENNIUM	VARCHAR2	'm1d1y2000'	DD-MM-YY 形式の日付を編集します。 Example: すべての日付を 01-JAN-2000 に編集します。
REDACT_DATE_EPOCH	VARCHAR2	'm1d1y1970'	すべての日付を 01-JAN-70 に編集します。
REDACT_AMEX_CCN_FORMATTED	VARCHAR2	'VVVVFVVVVVFVVVVV,VVVV-VVVVVV-VVVV,*,1,10'	AmericianExpress クレジットカード番号を編集し、最後の 5 桁を除き、数字を*に置き換えます。 Example: クレジットカード番号 1234 567890 34500 は **** * 34500 になります。

次のページに続く

表 4 – 前のページからの続き

REDACT_AMEX_CCN_NUMBER	VARCHAR2	'0,1,10'	AmericianExpress クレジットカード番号を編集し、最後の 5 桁を除いて数字を 0 に置き換えます。 Example: クレジットカード番号 1234 567890 34500 は 0000 000000 34500 になります。
REDACT_SIN_FORMATTED	VARCHAR2	'VVVFVVVFVVV, VVV-VVV-VVV,*,1,6'	最初の 6 桁を*で置き換えることにより、社会保険番号を編集します。 Example: 123-456-789 は ***-*** -789 になります。
REDACT_SIN_NUMBER	VARCHAR2	'9,1,6'	最初の 6 桁を 9 に置き換えて、社会保険番号を編集します。 Example: 123456789 は 999999789 になります。
REDACT_SIN_UNFORMATTED	VARCHAR2	'VVVVVVVVV, VVVVVVVVV,*,1,6'	最初の 6 桁を/*に置き換えて、社会保険番号を編集します。 Example: 123456789 は **** * 789 になります。
REDACT_CCN_FORMATTED	VARCHAR2	'VVVFVVVFVVVFVVVFVVV, VVVV-VVVV-VVVV-VVVV, *,1,12'	クレジットカード番号を*で編集し、4 桁のみを表示します。 Example: クレジットカード番号 1234-5678-9000-4671 は **** -**** -**** -4671 になります。

次のページに続く

表 4 – 前のページからの続き

REDACT_CCN_NUMBER	VARCHAR2	'9,1,12'	最後の4桁を除く0でクレジットカード番号を編集します。 Example: クレジットカード番号1234567890004671は0000000000004671になります
REDACT_NA_PHONE_FORMATTED	VARCHAR2	'VVVFVVVFVVVV, VVV-VVV-VVVV,X,4,10'	北米の電話番号を市外局番を残してXで編集します。 Example: 123-456-7890 は123-XXX-XXXXになります。
REDACT_NA_PHONE_NUMBER	VARCHAR2	'0,4,10'	北米の電話番号を市外局番を残して0で編集します。 Example: 1234567890 は1230000000になります。
REDACT_NA_PHONE_UNFORMATTED	VARCHAR2	'VVVVVVVVVV, VVVVVVVVVV,X,4,10'	北米の電話番号を市外局番を残してXで編集します。 Example: 1234567890 は123XXXXXXXになります。
REDACT_UK_NIN_FORMATTED	VARCHAR2	'VVFVVFVVFVVFV,VV VV VV V,X,3,8'	英国国民保険番号をXで編集しますが、アルファベット文字は残します。 Example: NY 22 01 34 D はNY XX XX XX Dになります。
REDACT_UK_NIN_UNFORMATTED	VARCHAR2	'VVVVVVVVV, VVVVVVVVV,X,3,8'	英国国民保険番号をXで編集しますが、アルファベット文字は残します。 Example: NY220134D はNYXXXXXXDになります。

正規表現ベースの墨消しは、墨消しするデータのパターンを検索します。 `regexp_pattern` は、`regexp_replace_string` が値を変更するために値を検索します。次の表は、REGEXP ベースの編集集中に使用できる `regexp_pattern` の値を示しています。

関数のパラメーターと説明	データ型	価値
RE_PATTERN_CC_L6_T4: 先行 6 桁と末尾 4 桁を含むクレジットカード番号の中桁を検索します。 形式で使用する <code>regexp_replace_string</code> 設定は <code>RE_REDACT_CC_MIDDLE_DIGITS</code> であり、識別されたパターンを <code>RE_REDACT_CC_MIDDLE_DIGITS</code> パラメーターで指定された文字に置き換えます。	VARCHAR2	' (//d//d//d//d//d) (//d//d//d*) (//d//d//d//d) '
RE_PATTERN_ANY_DIGIT: 任意の数字を検索し、識別されたパターンを <code>regexp_replace_string</code> パラメーターの以下の値で指定された文字に置き換えます。 <code>regexp_replace_string=>RE_REDACT_WITH_SINGLE_X</code> (一致した数字を X 文字に置き換えます)。 <code>regexp_replace_string=>RE_REDACT_WITH_SINGLE_1</code> (一致した数字を 1 文字に置き換えます)。	VARCHAR2	'//d'
RE_PATTERN_US_PHONE: 米国の電話番号を検索し、識別されたパターンを <code>regexp_replace_string</code> パラメーターで指定された文字に置き換えます。 <code>regexp_replace_string=>RE_REDACT_US_PHONE_L7</code> (電話番号を検索し、最後の 7 桁を置き換えます)。	VARCHAR2	' (// (//d//d//d//) //d//d//d) - (//d//d//d) - (//d//d//d//d) '

次のページに続く

表 5 – 前のページからの続き

関数のパラメーターと説明	データ型	価値
RE_PATTERN_EMAIL_ADDRESS: 電子メールアドレスを検索し、識別されたパターンを、<code>regexp_replace_string</code> パラメーターの次の値で指定された文字で置き換えます。 <code>regexp_replace_string=>RE_REDACT_EMAIL_NAME</code> (電子メールアドレスを検索し、電子メールユーザー名を編集します)。 <code>regexp_replace_string=>RE_REDACT_EMAIL_DOMAIN</code> (電子メールアドレスを検索し、電子メールアドレスを編集します)。 <code>regexp_replace_string=>RE_REDACT_EMAIL_ENTIRE</code> (メールアドレスを検索し、メールアドレス全体を編集します)。	VARCHAR2	' ([A-Za-z0-9._%+-]) @ ([A-Za-z0-9.-]+)/. [A-Za-z]{2,4}) '
RE_PATTERN_IP_ADDRESS: IP アドレスを検索し、識別されたパターンを <code>regexp_replace_string</code> パラメーターで指定された文字に置き換えます。使用される <code>regexp_replace_string</code> パラメーターは <code>RE_REDACT_IP_L3</code> であり、IP アドレスの最後のセクションを 999 に置き換え、編集されたことを示します。	VARCHAR2	' (//d{1,3}//. //d{1,3}//. //d{1,3}) //. //d{1,3}'
RE_PATTERN_AMEX_CCN: AmericanExpress クレジットカード番号を検索します。使用される <code>regexp_replace_string</code> パラメーターは <code>RE_REDACT_AMEX_CCN</code> で、最後の 5 桁を除くすべての数字を編集します。	VARCHAR2	' .* (//d/d//d//d//d) \$'

次のページに続く

表 5 – 前のページからの続き

関数のパラメーターと説明	データ型	価値
RE_PATTERN_CCN :AmericanExpress クレジットカード以外のクレジットカード 番号を検索します。使用される <code>regexp_replace_string</code> パラメーターは RE_REDACT_CCN で、最後の 4 桁を除 くすべての数字を編集します。	VARCHAR2	'。 <code>* (//d//d//d//d) \$</code> '
RE_PATTERN_US_SSN :SSN 番号を検 索し、識別されたパターンを <code>regexp_replace_string</code> パラメーターで指 定された文字で置き換えます。 「//1-XXX-XXXX」または 「XXX-XXX-//3」は、値「123-45-6789」 に対してそれぞれ 123-XXX-XXXX また は XXX-XXX-6789 を返します。	VARCHAR2	' (<code>//d//d//d</code>) - (<code>//d//d</code>) - (<code>//d//d//d//d</code>) '

次の表は、REGEXP ベースの編集集中に使用できる `regexp_replace_string` 値を示しています。

関数パラメーター	データ型	価値	説明
RE_REDACT_CC_MIDDLE_DIGITS	VARCHAR2	'/1XXXXXX/ 3'	RE_PATTERN_CC_L6_T4 形 式の <code>regexp_pattern</code> パラ メータに従ってクレジットカード 番号の中央の数字を編集し、 編集された各文字を X に置き換 えます。 Example: クレジットカード番 号 1234 5678 9000 2490 は 1234 56XX XXXX 2490 になります。

次のページに続く

表 6 – 前のページからの続き

RE_REDACT_WITH_SINGLE_X	VARCHAR2	'X'	RE_PATTERN_ANY_DIGIT 形式の regexp_pattern パラメータを設定することで指定された各一致パターンの単一の X 文字でデータを置き換えます。 Example: クレジットカード番号 1234 5678 9000 2490 は XXXX XXXX XXXX XXXX になります。
RE_REDACT_WITH_SINGLE_1	VARCHAR2	'1'	regexp_pattern パラメータを RE_PATTERN_ANY_DIGIT 形式に設定することで指定されたように、データを各データ桁の単一の 1 桁に置き換えます。 Example: クレジットカード番号 1234 5678 9000 2490 は 1111 1111 1111 1111 になります。
RE_REDACT_US_PHONE_L7	VARCHAR2	'/1-XXX-XXXX'	RE_PATTERN_US_PHONE 形式の regexp_pattern パラメータに従って米国電話番号の最後の 7 桁を編集し、編集された各文字を X に置き換えます。 Example: 電話番号 123-444-5900 は 123-XXX-XXXX になります。

次のページに続く

表 6 – 前のページからの続き

RE_REDACT_EMAIL_NAME	VARCHAR2	'xxxx@/2'	RE_PATTERN_EMAIL_ADDRESS 形式の regexp_pattern パラメータに従って電子メール名を編集し、電子メールのユーザー名を 4 つの x 文字に置き換えます。 Example: メールアドレス sjohn@example.com は xxxx@example.com になります。
RE_REDACT_EMAIL_DOMAIN	VARCHAR2	'/1@xxxxx.com'	RE_PATTERN_EMAIL_ADDRESS 形式の regexp_pattern パラメータに従って電子メールアドレスメイン名を編集し、ドメインを 5 つの x 文字に置き換えます。 Example: メールアドレス sjohn@example.com は sjohn@xxxxx.com になります。
RE_REDACT_EMAIL_ENTIRE	VARCHAR2	'xxxx@xxxxx.com'	RE_PATTERN_EMAIL_ADDRESS 形式の regexp_pattern パラメータに従ってメールアドレス全体を編集し、メールアドレスを x 文字に置き換えます。 Example: メールアドレス sjohn@example.com は xxxx@xxxxx.com になります。
RE_REDACT_IP_L3	VARCHAR2	'/1.999'	RE_PATTERN_IP_ADDRESS 形式の regexp_pattern パラメーターに従って、IP アドレスの最後の 3 桁を編集します。 Example: IP アドレス 172.0.1.258 は 172.0.1.999 になりますが、これは無効な IP アドレスです。

次のページに続く

表 6 – 前のページからの続き

RE_REDACT_AMEX_CCN	VARCHAR2	'**** * ** /1'	AmericanExpress クレジット カード番号の最初の 10 桁 を、RE_PATTERN_AMEX_CCN 形式の regexp_pattern パ ラメーターに従って編集 します。 regexp_pattern 123456789062816 は **** **** ** 62816 になります。
RE_REDACT_CCN	VARCHAR2	'**** * **** /1'	regexp_pattern パ ラメーターで指定された RE_PATTERN_CCN 形式の クレジットカード番号の最初の 12 桁を編集します。 Example: 8749012678345671 は **** **** **** 5671 に なります。

次の表に、REGEXP ベースの編集集中に使用できる regexp_position 値と regexp_occurence 値を示します。

関数パラメーター	データ型	価値	説明
RE_BEGINNING	INTEGER	1	検索を開始する必要がある文字の位置を指定します。デフォルトでは、値は 1 で、source_char の最初の文字から検索が開始されることを示します。

関数パラメーター	データ型	価値	説明
RE_ALL	INTEGER	0	部分文字列の置換オカレンスを指定します。値が 0 の場合、一致する各部分文字列の置換が発生します。
RE_FIRST	INTEGER	1	部分文字列の置換オカレンスを指定します。値が 1 の場合、最初に一致した部分文字列の置換が発生します。

次の表は、関数のデフォルトの一致動作を変更できる REGEXP ベースの編集集中に使用できる regexp_match_parameter 値を示しています。

関数パラメーター	データ型	価値	説明
RE_CASE_SENSITIVE	VARCHAR2	'c'	大文字と小文字を区別する一致を指定します。
RE_CASE_INSENSITIVE	VARCHAR2	'i'	大文字と小文字を区別しないマッチングを指定します。
RE_MULTIPLE_LINES	VARCHAR2	'm'	ソース文字列を複数行として扱いますが、このパラメーターを省略すると、単一行として示されます。
RE_NEWLINE_WILDCARD	VARCHAR2	'n'	ピリオド (。) を指定しますが、このパラメーターを省略すると、ピリオドは改行文字と一致しません。
RE_IGNORE_WHITESPACE	VARCHAR2	'x'	空白文字を無視します。

注釈: 数値型の列に基づいて墨消しポリシーを作成する場合は、墨消し後の結果が数値であることを確認し、それに応じて置換文字列を設定して実行時エラーを回避します。

注釈: 文字タイプの列に基づいて改訂ポリシーを作成する場合、改訂後の結果の長さが列タイプと互換性があることを確認し、それに応じて置換文字列を設定してランタイムエラーを回避します。

3.13.2 ADD_POLICY

add_policy プロシージャは、テーブルの新しいデータ改訂ポリシーを作成します。

```
PROCEDURE add_policy (
    <object_schema>          IN VARCHAR2 DEFAULT NULL,
    <object_name>             IN VARCHAR2,
    <policy_name>             IN VARCHAR2,
    <policy_description>      IN VARCHAR2 DEFAULT NULL,
    <column_name>             IN VARCHAR2 DEFAULT NULL,
    <column_description>      IN VARCHAR2 DEFAULT NULL,
    <function_type>           IN INTEGER DEFAULT DBMS_REDACT.FULL,
    <function_parameters>     IN VARCHAR2 DEFAULT NULL,
    <expression>              IN VARCHAR2,
    <enable>                  IN BOOLEAN DEFAULT TRUE,
    <regexp_pattern>          IN VARCHAR2 DEFAULT NULL,
    <regexp_replace_string>   IN VARCHAR2 DEFAULT NULL,
    <regexp_position>         IN INTEGER DEFAULT DBMS_REDACT.RE_BEGINNING,
    <regexp_occurrence>      IN INTEGER DEFAULT DBMS_REDACT.RE_ALL,
```

(次のページに続く)

(前のページからの続き)

```

<regexp_match_parameter>      IN VARCHAR2 DEFAULT NULL,
<custom_function_expression>  IN VARCHAR2 DEFAULT NULL
)

```

Parameters

object_schema

オブジェクトが存在し、データ改訂ポリシーが適用されるスキーマの名前を指定します。NULL を指定すると、search_path の設定で指定された順序で特定のオブジェクトが検索されます。

object_name

データ編集ポリシーが作成されるテーブルの名前。

policy_name

追加するポリシーの名前。ポリシーが作成されるテーブルに対して policy_name が一意であることを確認します。

policy_description

リダクションポリシーの説明を指定します。

column_name

リダクションポリシーが適用される列の名前。複数の列を編集するには、alter_policy 手順を使用して列を追加します。

column_description

編集する列の説明。column_description はサポートされていませんが、列の説明を指定すると、警告メッセージが表示されます。

function_type

使用する墨消し機能のタイプ。可能な値は NONE, FULL, PARTIAL, RANDOM, REGEXP および CUSTOM です。

function_parameters

パーティションリダクションの関数パラメーターを指定し、部分的なリダクションにのみ適用できます。

expression

テーブルのブール式を指定し、ポリシーの適用方法を決定します。このポリシー式が TRUE に評価されると、墨消しが発生します。

enable

TRUE に設定すると、ポリシーは作成時に有効になります。デフォルトは TRUE に設定されています。FALSE に設定すると、ポリシーは無効になりますが、`enable_policy` プロシージャを呼び出すことでポリシーを有効にできます。

`regexp_pattern`

データを編集する正規表現パターンを指定します。 `regexp_pattern` が一致しない場合、NULL 値が返されます。

`regexp_replace_string`

置換文字列値を指定します。

`regexp_position`

検索を開始する必要がある文字の位置を指定します。デフォルトでは、関数パラメーターは `RE_BEGINNING` です。

`regexp_occurrence`

部分文字列の置換オカレンスを指定します。定数が `RE_ALL` の場合、一致する各部分文字列の置換が発生します。定数が `RE_FIRST` の場合、最初に一致した部分文字列の置換が発生します。

`regexp_match_parameter`

関数のデフォルトの一致動作を変更します。可能な `regexp_match_parameter` 定数は `'RE_CASE_SENSITIVE'` , `'RE_CASE_INSENSITIVE'` , `'RE_MULTIPLE_LINES'` , `'RE_NEWLINE_WILDCARD'` , `'RE_IGNORE_WHITESPACE'` にすることができます。

注釈: `constants`、`function_parameters`、または `regexp`（正規表現）の詳細については、`DBMS_REDACT Constants and Function Parameters` の使用を参照してください。

`custom_function_expression`

`custom_function_expression` は、`CUSTOM` 編集タイプにのみ適用されます。`custom_function_expression` は、`schema_name.function_name (argument1, ...)` などのパラメーターを持つスキーマ修飾関数である関数式です。これにより、ユーザーはリダクションロジックを使用して列データをリダクションできます。

Example

次の例は、ポリシーを作成し、`payment_details_tab` テーブルの `customer_id` 列の値に完全な墨消しを使用する方法を示しています。

```
edb=# CREATE TABLE payment_details_tab (
customer_id NUMBER          NOT NULL,
```

(次のページに続く)

(前のページからの続き)

```

card_string VARCHAR2(19) NOT NULL);
CREATE TABLE

edb=# BEGIN
  INSERT INTO payment_details_tab VALUES (4000, '1234-1234-1234-1234');
  INSERT INTO payment_details_tab VALUES (4001, '2345-2345-2345-2345');
END;

EDB-SPL Procedure successfully completed

edb=# CREATE USER redact_user;
CREATE ROLE
edb=# GRANT SELECT ON payment_details_tab TO redact_user;
GRANT

\c edb base_user

BEGIN
  DBMS_REDACT.add_policy(
    object_schema      => 'public',
    object_name        => 'payment_details_tab',
    policy_name        => 'redactPolicy_001',
    policy_description  => 'redactPolicy_001 for payment_details_tab
table',
    column_name        => 'customer_id',
    function_type       => DBMS_REDACT.full,
    expression         => '1=1',
    enable             => TRUE);
END;

```

編集結果:

```

edb=# \c edb redact_user
You are now connected to database "edb" as user "redact_user".

edb=> select customer_id from payment_details_tab order by 1;
 customer_id
-----
          0
          0
(2 rows)

```


3.13.3 ALTER_POLICY

alter_policy プロシージャは、テーブルの既存のデータ改訂ポリシーを変更または変更します。

```
PROCEDURE alter_policy (
    <object_schema>          IN VARCHAR2 DEFAULT NULL,
    <object_name>            IN VARCHAR2,
    <policy_name>            IN VARCHAR2,
    <action>                 IN INTEGER DEFAULT DBMS_REDACT.ADD_COLUMN,
    <column_name>           IN VARCHAR2 DEFAULT NULL,
    <function_type>          IN INTEGER DEFAULT DBMS_REDACT.FULL,
    <function_parameters>    IN VARCHAR2 DEFAULT NULL,
    <expression>             IN VARCHAR2 DEFAULT NULL,
    <regexp_pattern>         IN VARCHAR2 DEFAULT NULL,
    <regexp_replace_string>  IN VARCHAR2 DEFAULT NULL,
    <regexp_position>        IN INTEGER DEFAULT DBMS_REDACT.RE_BEGINNING,
    <regexp_occurrence>     IN INTEGER DEFAULT DBMS_REDACT.RE_ALL,
    <regexp_match_parameter> IN VARCHAR2 DEFAULT NULL,
    <policy_description>     IN VARCHAR2 DEFAULT NULL,
    <column_description>     IN VARCHAR2 DEFAULT NULL,
    <custom_function_expression> IN VARCHAR2 DEFAULT NULL
)
```

Parameters

object_schema

オブジェクトが存在し、データ改訂ポリシーが変更されるスキーマの名前を指定します。NULL を指定すると、search_path の設定で指定された順序で特定のオブジェクトが検索されます。

object_name

データ改訂ポリシーを変更するテーブルの名前。

policy_name

変更するポリシーの名前。

action

実行するアクション。アクションパラメータの詳細については、DBMS_REDACT Constants and Function Parameters を参照してください。

column_name

リダクションポリシーが適用される列の名前。

function_type

使用する墨消し機能のタイプ。可能な値は NONE, FULL, PARTIAL, RANDOM, REGEXP および CUSTOM です。

`function_parameters`

リダクション関数の関数パラメーターを指定します。

`expression`

テーブルのブール式を指定し、ポリシーの適用方法を決定します。このポリシー式が TRUE に評価されると、墨消しが発生します。

`regexp_pattern`

正規表現を使用してデータを編集できるようにします。 `regexp_pattern` がデータと一致しない場合、NULL 値が返されます。

`regexp_replace_string`

置換文字列値を指定します。

`regexp_position`

検索を開始する必要がある文字の位置を指定します。デフォルトでは、関数パラメーターは RE_BEGINNING です。

`regexp_occurence`

部分文字列の置換オカレンスを指定します。定数が RE_ALL の場合、一致する各部分文字列の置換が発生します。定数が RE_FIRST の場合、最初に一致した部分文字列の置換が発生します。

`regexp_match_parameter`

関数のデフォルトの一致動作を変更します。可能な `regexp_match_parameter` 定数は 'RE_CASE_SENSITIVE' , 'RE_CASE_INSENSITIVE' , 'RE_MULTIPLE_LINES' , 'RE_NEWLINE_WILDCARD' , 'RE_IGNORE_WHITESPACE' にすることができます。

注釈: `constants`, `function_parameters` または `regexp` (正規表現) の詳細については、Using DBMS_REDACT Constants and Function Parameters を参照してください。

`policy_description`

リダクションポリシーの説明を指定します。

`column_description`

編集する列の説明。 `column_description` はサポートされていませんが、列の説明を指定すると、警告メッセージが表示されます。

custom_function_expression

custom_function_expression は、CUSTOM 編集タイプにのみ適用されます。
 custom_function_expression は、schema_name.function_name (argument1, ...) などのパラメーターを持つスキーマ修飾関数である関数式です。これにより、ユーザーはリダクションロジックを使用して列データをリダクションできます。

Example

次の例は、payment_details_tab テーブルの card_string（通常はクレジットカード番号）列の値のポリシー部分改訂を変更する方法を示しています。

```
\c edb base _user

BEGIN
  DBMS_REDACT.alter_policy (
    object_schema      => 'public',
    object_name        => 'payment_details_tab',
    policy_name        => 'redactPolicy_001',
    action              => DBMS_REDACT.ADD_COLUMN,
    column_name        => 'card_string',
    function_type       => DBMS_REDACT.partial,
    function_parameters => DBMS_REDACT.REDACT_CCN16_F12);
END;
```

編集結果:

```
edb=# \c - redact_user
You are now connected to database "edb" as user "redact_user".
edb=> SELECT * FROM payment_details_tab;
 customer_id |      card_string
-----+-----
           0 | *****-1234
           0 | *****-2345
(2 rows)
```

3.13.4 DISABLE_POLICY

disable_policy プロシージャは、既存のデータ改訂ポリシーを無効にします。

```
PROCEDURE disable_policy (
  <object_schema>      IN VARCHAR2 DEFAULT NULL,
  <object_name>        IN VARCHAR2,
  <policy_name>        IN VARCHAR2
)
```

Parameters

object_schema

オブジェクトが存在し、データ改訂ポリシーが適用されるスキーマの名前を指定します。NULL を指定すると、search_path の設定で指定された順序で特定のオブジェクトが検索されます。

object_name

データ改訂ポリシーを無効にするテーブルの名前。

policy_name

無効にするポリシーの名前。

Example

次の例は、ポリシーを無効にする方法を示しています。

```
\c edb base_user

BEGIN
  DBMS_REDACT.disable_policy(
    object_schema => 'public',
    object_name   => 'payment_details_tab',
    policy_name   => 'redactPolicy_001');
END;
```

編集結果:ポリシーを無効にした後、データは編集されなくなりました。

3.13.5 ENABLE_POLICY

enable_policy 手順は、以前に無効にされたデータ改訂ポリシーを有効にします。

```
PROCEDURE enable_policy (
  <object_schema>          IN VARCHAR2 DEFAULT NULL,
  <object_name>            IN VARCHAR2,
  <policy_name>            IN VARCHAR2
)
```

Parameters

object_schema

オブジェクトが存在し、データ改訂ポリシーが適用されるスキーマの名前を指定します。NULL を指定すると、search_path の設定で指定された順序で特定のオブジェクトが検索されます。

object_name

データ改訂ポリシーを有効にするテーブルの名前。

policy_name

有効にするポリシーの名前。

Example

次の例は、ポリシーを有効にする方法を示しています。

```
\c edb base_user

BEGIN
    DBMS_REDACT.enable_policy(
        object_schema => 'public',
        object_name => 'payment_details_tab',
        policy_name => 'redactPolicy_001');
END;
```

編集結果:ポリシーを有効にした後、データが編集されます。

3.13.6 DROP_POLICY

drop_policy プロシージャは、テーブルからマスキングポリシーを削除することにより、データ改訂ポリシーを削除します。

```
PROCEDURE drop_policy (
    <object_schema>          IN VARCHAR2 DEFAULT NULL,
    <object_name>            IN VARCHAR2,
    <policy_name>            IN VARCHAR2
)
```

Parameters

object_schema

オブジェクトが存在し、データ改訂ポリシーが適用されるスキーマの名前を指定します。NULL を指定すると、search_path の設定で指定された順序で特定のオブジェクトが検索されます。

object_name

データ修正ポリシーを削除するテーブルの名前。

policy_name

削除するポリシーの名前。

Example

次の例は、ポリシーを削除する方法を示しています。

```
\c edb base_user

BEGIN
    DBMS_REDACT.drop_policy(
        object_schema => 'public',
        object_name => 'payment_details_tab',
        policy_name => 'redactPolicy_001');
END
```

編集結果:サーバーは指定されたポリシーを削除します。

3.13.7 UPDATE_FULL_REDACTION_VALUES

update_full_redaction_values 手順は、データ改訂ポリシーのデフォルトの表示値を更新します。これらのデフォルト値は、完全な改訂タイプを使用する redaction_values_for_type_full ビューを使用して表示できます。

```
PROCEDURE update_full_redaction_values (
    <number_val>          IN NUMBER          DEFAULT NULL,
    <binfloat_val>        IN FLOAT4          DEFAULT NULL,
    <bindouble_val>       IN FLOAT8          DEFAULT NULL,
    <char_val>            IN CHAR            DEFAULT NULL,
    <varchar_val>         IN VARCHAR2        DEFAULT NULL,
    <nchar_val>           IN NCHAR           DEFAULT NULL,
    <nvarchar_val>        IN NVARCHAR2       DEFAULT NULL,
    <datecol_val>         IN DATE            DEFAULT NULL,
    <ts_val>              IN TIMESTAMP       DEFAULT NULL,
    <tswtz_val>           IN TIMESTAMPTZ     DEFAULT NULL,
    <blob_val>            IN BLOB            DEFAULT NULL,
    <clob_val>            IN CLOB            DEFAULT NULL,
    <nclob_val>           IN CLOB            DEFAULT NULL
)
```

Parameters

number_val

NUMBER データ型の列のデフォルト値を更新します。

binfloat_val

FLOAT4 データ型はランダムな値です。バイナリ浮動データ型はサポートされていません。

bindouble_val

FLOAT8 データ型はランダムな値です。binarydouble データ型はサポートされていません。

char_val

CHAR データ型の列のデフォルト値を更新します。

varchar_val

VARCHAR2 データ型の列のデフォルト値を更新します。

nchar_val

nchar_val は CHAR データ型にマッピングされ、CHAR 値を返します。

nvarchar_val

nvarchar_val は VARCHAR2 データ型にマッピングされ、VARCHAR 値を返します。

datecol_val

DATE データ型の列のデフォルト値を更新します。

ts_val

TIMESTAMP データ型の列のデフォルト値を更新します。

tswtz_val

TIMESTAMPTZ データ型の列のデフォルト値を更新します。

blob_val

BLOB データ型の列のデフォルト値を更新します。

clob_val

CLOB データ型の列のデフォルト値を更新します。

nclob_val

nclob_val は CLOB データ型にマッピングされ、CLOB 値を返します。

Example

次の例は、完全な墨消し値を更新する方法を示していますが、値を更新する前に次のことができます。

以下に示すように、redaction_values_for_type_full ビューを使用してデフォルト値を表示します。

```
edb=# \x
Expanded display is on.
edb=# SELECT number_value, char_value, varchar_value, date_value,
        timestamp_value, timestamp_with_time_zone_value, blob_value,
        clob_value
FROM redaction_values_for_type_full;
-[ RECORD 1 ]-----+-----
number_value          | 0
```

(次のページに続く)

(前のページからの続き)

```

char_value          |
varchar_value       |
date_value          | 01-JAN-01 00:00:00
timestamp_value     | 01-JAN-01 01:00:00
timestamp_with_time_zone_value | 31-DEC-00 20:00:00 -05:00
blob_value          | \x5b72656461637465645d
clob_value          | [redacted]
(1 row)

```

次に、完全な墨消しタイプのデフォルト値を更新します。NULL 値は無視されます。

```

\c edb base_user

edb=# BEGIN
      DBMS_REDACT.update_full_redaction_values (
        number_val => 9999999,
        char_val   => 'Z',
        varchar_val => 'V',
        datecol_val => to_date('17/10/2018', 'DD/MM/YYYY'),
        ts_val     => to_timestamp('17/10/2018 11:12:13', 'DD/MM/YYYY HH24:MI:SS'),
        tswtz_val  => NULL,
        blob_val   => 'NEW REDACTED VALUE',
        clob_val   => 'NEW REDACTED VALUE');
END;

```

redaction_values_for_type_full ビューを使用して、更新された値を確認できるようになりました。

```

EDB-SPL Procedure successfully completed
edb=# SELECT number_value, char_value, varchar_value, date_value,
        timestamp_value, timestamp_with_time_zone_value, blob_value,
        clob_value
FROM redaction_values_for_type_full;
-[ RECORD 1 ]-----+-----
number_value          | 9999999
char_value            | Z
varchar_value         | V
date_value            | 17-OCT-18 00:00:00
timestamp_value       | 17-OCT-18 11:12:13
timestamp_with_time_zone_value | 31-DEC-00 20:00:00 -05:00
blob_value            | \x4e45572052454441435445442056414c5545
clob_value            | NEW REDACTED VALUE
(1 row)

```

編集結果:

```

edb=# \c edb redact_user
You are now connected to database "edb" as user "redact_user".

```

(次のページに続く)

(前のページからの続き)

```
edb=> select * from payment_details_tab order by 1;
 customer_id | card_string
-----+-----
      9999999 | V
      9999999 | V
(2 rows)
```

3.14 DBMS_RLS

DBMS_RLS パッケージを使用すると、特定の AdvancedServer データベースオブジェクトに仮想プライベートデータベースを実装できます。

機能/手順	機能または手順	戻りタイプ	説明
<code>ADD_POLICY(object_schema, object_name, policy_name, function_schema, policy_function [, statement_types [, update_check [, enable [, static_policy [, policy_type [, long_predicate [, sec_relevant_cols [, sec_relevant_cols_opt]]]]]]]])</code>	手続き	該当なし	セキュリティポリシーをデータベースオブジェクトに追加します。
<code>DROP_POLICY(object_schema, object_name, policy_name)</code>	手続き	該当なし	データベースオブジェクトからセキュリティポリシーを削除します。
<code>ENABLE_POLICY(object_schema, object_name, policy_name, enable)</code>	手続き	該当なし	セキュリティポリシーを有効または無効にします。

AdvancedServer の DBMS_RLS の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表にリストされている関数と手順のみがサポートされています。

仮想プライベートデータベース*は、セキュリティポリシーを使用したきめ細かいアクセス制御の一種です。仮想プライベートデータベースの*詳細なアクセス制御は、セキュリティポリシーで定義された特定の行までデータへのアクセスを制御できることを意味します。

セキュリティポリシーをエンコードするルールは、*policyfunction**で定義されます。これは、特定の入力パラメーターと戻り値を持つ *SPL* 関数です。*セキュリティポリシーは、ポリシー関数と特定のデータベースオブジェクト（通常はテーブル）の名前付き関連付けです。

注釈: AdvancedServer では、SQL、PL/pgSQL、SPL など、AdvancedServer がサポートする任意の言語でポリシー関数を作成できます。

注釈: 現在 AdvancedServerVirtualPrivateDatabase でサポートされているデータベースオブジェクトはテーブルです。ポリシーをビューまたはシノニムに適用することはできません。

仮想プライベートデータベースを使用する利点は次のとおりです。

- きめ細かいレベルのセキュリティを提供します。GRANT コマンドによって与えられるデータベースオブジェクトレベルの特権は、データベースオブジェクトのインスタンス全体へのアクセス特権を決定しますが、仮想プライベートデータベースは、データベースオブジェクトインスタンスの個々の行に対するアクセス制御を提供します。
- SQL コマンドのタイプ (INSERT, UPDATE, DELETE または SELECT) に応じて、異なるセキュリティポリシーを適用できます。
- セキュリティポリシーは、データベースオブジェクトにアクセスするアプリケーションのセッションユーザーなどの要因に応じて、データベースオブジェクトに影響を与える該当する SQL コマンドごとに動的に変化します。
- セキュリティポリシーの呼び出しは、データベースオブジェクトにアクセスするすべてのアプリケーションに対して透過的であるため、セキュリティポリシーを適用するために個々のアプリケーションを変更する必要はありません。
- セキュリティポリシーが有効になると、アプリケーション (新しいアプリケーションを含む) がセキュリティポリシーを回避することはできません。ただし、以下に記載されているシステム権限が必要です。
- スーパーユーザーであっても、以下に記載されているシステム権限を除き、セキュリティポリシーを回避することはできません。

注釈: セキュリティポリシーを回避できる唯一の方法は、EXEMPT ACCESS POLICY システム権限がユーザーに付与されている場合です。この特権を持つユーザーはデータベース内のすべてのポリシーから除外されるため、EXEMPT ACCESS POLICY 特権は細心の注意を払って付与する必要があります。

DBMS_RLS パッケージは、ポリシーの作成、ポリシーの削除、ポリシーの有効化、およびポリシーの無効化の手順を提供します。

仮想プライベートデータベースを実装するプロセスは次のとおりです。

- ポリシー関数を作成します。関数には、VARCHAR2 タイプの 2 つの入力パラメーターが必要です。最初の入力パラメーターは、ポリシーが適用されるデータベースオブジェクトを含むスキーマ用であり、2 番目の入力パラメーターはそのデータベースオブジェクトの名前用です。関数には VARCHAR2 の戻り値の型が必要です。関数は、WHERE 句の述語の形式で文字列を返す必要があります。この述語は、データベースオブジェクトに作用する SQL コマンドに AND 条件として動的に追加されます。したがって、ポリシー関数の述部を満たさない行は、SQL コマンドの結果セットから除外されます。

- `ADD_POLICY` 手順を使用して、新しいポリシーを定義します。これは、ポリシー機能とデータベースオブジェクトの関連付けです。 `ADD_POLICY` 手順では、ポリシーを適用する SQL コマンドのタイプ (`INSERT`, `UPDATE`, `DELETE` または `SELECT`), ポリシーの作成時にポリシーを有効にするかどうか、およびポリシーは、新しく挿入された行または更新された行の変更されたイメージに適用する必要があります。
- `ENABLE_POLICY` 手順を使用して、既存のポリシーを無効または有効にします。
- `DROP_POLICY` 手順を使用して、既存のポリシーを削除します。 `DROP_POLICY` プロシージャは、ポリシー関数または関連するデータベースオブジェクトを削除しません。

作成されたポリシーは、Oracle データベースと互換性のあるカタログビューで表示できます: `ALL_POLICIES`, `DBA_POLICIES` または `USER_POLICIES`。サポートされている互換ビューは、次の EDBWeb サイトで入手可能な「OracleDevelopersCatalogViews のデータベース互換性ガイド」にリストされています。

<https://www.enterprisedb.com/edb-docs/>

`SYS_CONTEXT` 機能は `DBMS_RLS` とともに使用されることがよくあります。署名は次のとおりです。

```
SYS_CONTEXT(<namespace>, <attribute>)
```

どこで:

`namespace` は `VARCHAR2` です。受け入れられる唯一の値は `USERENV` です。その他の値は `NULL` を返します。

`attribute` は `VARCHAR2` です。 `attribute` は:

属性値	等価値
<code>SESSION_USER</code>	<code>pg_catalog.session_user</code>
<code>CURRENT_USER</code>	<code>pg_catalog.current_user</code>
<code>CURRENT_SCHEMA</code>	<code>pg_catalog.current_schema</code>
ホスト	<code>pg_catalog.inet_host</code>
<code>IP_ADDRESS</code>	<code>pg_catalog.inet_client_addr</code>
<code>SERVER_HOST</code>	<code>pg_catalog.inet_server_addr</code>

注釈: `DBMS_RLS` パッケージの説明に使用される例は、テーブルに対するすべての特権が付与された `salesmgr` という名前のロールとともに、AdvancedServer で提供されるサンプル `emp` テーブルの変更されたコピーに基づいています。次のように、`vpemp` という名前の `emp` テーブルと `salesmgr` ロールの変更されたコピーを作成できます。

```

CREATE TABLE public.vpemp AS SELECT empno, ename, job, sal, comm, deptno
FROM emp;
ALTER TABLE vpemp ADD authid VARCHAR2(12);
UPDATE vpemp SET authid = 'researchmgr' WHERE deptno = 20;
UPDATE vpemp SET authid = 'salesmgr' WHERE deptno = 30;
SELECT * FROM vpemp;

```

empno	ename	job	sal	comm	deptno	authid
7782	CLARK	MANAGER	2450.00		10	
7839	KING	PRESIDENT	5000.00		10	
7934	MILLER	CLERK	1300.00		10	
7369	SMITH	CLERK	800.00		20	researchmgr
7566	JONES	MANAGER	2975.00		20	researchmgr
7788	SCOTT	ANALYST	3000.00		20	researchmgr
7876	ADAMS	CLERK	1100.00		20	researchmgr
7902	FORD	ANALYST	3000.00		20	researchmgr
7499	ALLEN	SALESMAN	1600.00	300.00	30	salesmgr
7521	WARD	SALESMAN	1250.00	500.00	30	salesmgr
7654	MARTIN	SALESMAN	1250.00	1400.00	30	salesmgr
7698	BLAKE	MANAGER	2850.00		30	salesmgr
7844	TURNER	SALESMAN	1500.00	0.00	30	salesmgr
7900	JAMES	CLERK	950.00		30	salesmgr

(14 rows)

```

CREATE ROLE salesmgr WITH LOGIN PASSWORD 'password';
GRANT ALL ON vpemp TO salesmgr;

```

3.14.1 ADD_POLICY

ADD_POLICY プロシージャは、ポリシー関数をデータベースオブジェクトに関連付けることにより、新しいポリシーを作成します。

この手順を実行するには、スーパーユーザーである必要があります。

```
ADD_POLICY(<object_schema> VARCHAR2, <object_name> VARCHAR2,  
  <policy_name> VARCHAR2, <function_schema> VARCHAR2,  
  <policy_function> VARCHAR2  
  [, <statement_types> VARCHAR2  
  [, <update_check> BOOLEAN  
  [, <enable> BOOLEAN  
  [, <static_policy> BOOLEAN  
  [, <policy_type> INTEGER  
  [, <long_predicate> BOOLEAN  
  [, <sec_relevant_cols> VARCHAR2  
  [, <sec_relevant_cols_opt> INTEGER ]]]]]])
```

Parameters

object_schema

ポリシーが適用されるデータベースオブジェクトを含むスキーマの名前。

object_name

ポリシーが適用されるデータベースオブジェクトの名前。特定のデータベースオブジェクトには、複数のポリシーが適用されている場合があります。

policy_name

ポリシーに割り当てられた名前。データベースオブジェクト（object_schema と object_name で識別される）とポリシー名の組み合わせは、データベース内で一意である必要があります。

function_schema

ポリシー関数を含むスキーマの名前。

注釈: ポリシー関数はパッケージに属する場合があります。この場合、function_schema にはパッケージが定義されているスキーマの名前が含まれている必要があります。

policy_function

セキュリティポリシーのルールを定義する SPL 関数の名前。同じ機能を複数のポリシーで指定できます。

注釈: ポリシー関数はパッケージに属している場合があります。その場合、`policy_function` にはドット表記のパッケージ名（つまり `package_name.function_name`）も含める必要があります。

`statement_types`

ポリシーが適用される SQL コマンドのコンマ区切りリスト。有効な SQL コマンドは `INSERT`, `UPDATE`, `DELETE` および `SELECT` です。デフォルトは `INSERT,UPDATE,DELETE,SELECT` です。

注釈: `AdvancedServer` は、ステートメントタイプとして `INDEX` を受け入れますが、無視されます。ポリシーは `AdvancedServer` のインデックス操作に適用されません。

`update_check`

`INSERT` および `UPDATE` の SQL コマンドにのみ適用されます。

`TRUE` に設定すると、ポリシーは新しく挿入された行と更新された行の変更されたイメージに適用されます。新規または変更された行のいずれかがポリシー関数の述部に従って修飾されない場合、`INSERT` または `UPDATE` コマンドは例外をスローし、`INSERT` または `UPDATE` コマンドによって行が挿入または変更されません。

`FALSE` に設定すると、ポリシーは新しく挿入された行または更新された行の変更されたイメージに適用されません。したがって、新しく挿入された行は、同じポリシーを呼び出す後続の SQL コマンドの結果セットに表示されない場合があります。同様に、`UPDATE` コマンドの前にポリシーに従って修飾された行は、同じポリシーを呼び出す後続の SQL コマンドの結果セットに表示されない場合があります。

デフォルトは `FALSE` です。

`enable`

`TRUE` に設定すると、ポリシーが有効になり、`statement_types` パラメーターで指定された SQL コマンドに適用されます。`FALSE` に設定すると、ポリシーは無効になり、SQL コマンドには適用されません。ポリシーは、`ENABLE_POLICY` 手順を使用して有効にできます。デフォルトは `TRUE` です。

`static_policy`

Oracle では、`TRUE` に設定すると、ポリシーは `*static*` になります。つまり、ポリシー関数は、データベースオブジェクトのポリシーによって初めて呼び出されたときに、データベースオブジェクトごとに 1 回評価されます。結果のポリシー関数述語文字列はメモリに保存され、データベースサーバーインスタンスの実行中にそのデータベースオブジェクトに対するそのポリシーのすべての呼び出しに再利用されます。

`FALSE` に設定すると、ポリシーは `*dynamic*` になります。これは、ポリシー関数が再評価され、ポリシーのすべての呼び出しに対してポリシー関数の述語文字列が再生成されることを意味します。

デフォルトは `FALSE` です。

注釈: Oracle10g では、`policy_type` パラメーターが導入されましたが、これは `static_policy` パラメーターに代わるものです。Oracle では、`policy_type` パラメーターがデフォルト値の `NULL` に設定されていない場合、`policy_type` パラメーター設定は `static_policy` 設定をオーバーライドします。

注釈: `static_policy` の設定は `AdvancedServer` によって無視されます。`AdvancedServer` は、`static_policy` パラメーターの設定に関係なく、動的ポリシーのみを実装します。

`policy_type`

Oracle では、ポリシー関数がいつ再評価されるか、したがって、ポリシー関数によって返される述語文字列が変更されるかどうか、およびいつ変更されるかを決定します。デフォルトは `NULL` です。

注釈: このパラメーターの設定は、`AdvancedServer` によって無視されます。`AdvancedServer` は常に動的ポリシーを想定しています。

`long_predicate`

Oracle では、`TRUE` に設定すると最大 32K バイトの述語が許可されます。それ以外の場合、述語は 4000 バイトに制限されます。デフォルトは `FALSE` です。

注釈: このパラメーターの設定は、`AdvancedServer` によって無視されます。`AdvancedServer` ポリシー関数は、あらゆる実用的な目的のために、無制限の長さの述語を返すことができます。

`sec_relevant_cols`

`object_name` の列のコンマ区切りリスト。リストされた列に*列レベルの仮想プライベートデータベース*を提供します。リストされた列のいずれかが `statement_types` にリストされたタイプの SQL コマンドで参照されている場合、ポリシーが実施されます。そのような列が参照されていない場合、ポリシーは適用されません。

デフォルトは `NULL` で、データベースオブジェクトのすべての列が `sec_relevant_cols` に含まれている場合と同じ効果があります。

`sec_relevant_cols_opt`

Oracle では、`sec_relevant_cols_opt` が値 1) の `DBMS_RLS.ALL_ROWS` (INTEGER 定数に設定されている場合、`sec_relevant_cols` にリストされている列は、適用されたポリシー述語が `false` であるすべての行で `NULL` を返します。(`sec_relevant_cols_opt` が `DBMS_RLS.ALL_ROWS` に設定されていない場合、これらの行は結果セットでまったく返されません。) デフォルトは `NULL` です。

注 釈: AdvancedServer は DBMS_RLS.ALL_ROWS 機能をサポートしていません。sec_relevant_cols_opt が DBMS_RLS.ALL_ROWS (INTEGER 値 1) に設定されている場合、AdvancedServer はエラーをスローします。

Examples

この例では、次のポリシー関数を使用します。

```
CREATE OR REPLACE FUNCTION verify_session_user (
    p_schema      VARCHAR2,
    p_object      VARCHAR2
)
RETURN VARCHAR2
IS
BEGIN
    RETURN 'authid = SYS_CONTEXT(''USERENV'', 'SESSION_USER')';
END;
```

この関数は、ADD_POLICY プロシージャで指定されたタイプの SQL コマンドの WHERE 句に追加される述語 authid = SYS_CONTEXT('USERENV', 'SESSION_USER') を生成します。

これにより、SQL コマンドの効果は、authid 列の内容がセッションユーザーと同じである行に限定されます。

注釈: この例では、SYS_CONTEXT 関数を使用してログインユーザー名を返します。Oracle では、SYS_CONTEXT 関数を使用して*アプリケーションコンテキスト*の属性を返します。SYS_CONTEXT 関数の最初のパラメーターはアプリケーションコンテキストの名前で、2 番目のパラメーターはアプリケーションコンテキスト内で設定された属性の名前です。USERENV は、現在のセッションを記述する特別な組み込み名前空間です。AdvancedServer はアプリケーションコンテキストをサポートしていませんが、SYS_CONTEXT 機能のこの特定の使用法のみをサポートしています。

次の匿名ブロックは、ADD_POLICY プロシージャを呼び出して、INSERT、UPDATE または DELETE の SQL コマンドが vpemp」の表。

```
DECLARE
    v_object_schema      VARCHAR2(30) := 'public';
    v_object_name        VARCHAR2(30) := 'vpemp';
    v_policy_name        VARCHAR2(30) := 'secure_update';
    v_function_schema    VARCHAR2(30) := 'enterprisedb';
    v_policy_function     VARCHAR2(30) := 'verify_session_user';
    v_statement_types    VARCHAR2(30) := 'INSERT,UPDATE,DELETE';
    v_update_check       BOOLEAN      := TRUE;
    v_enable             BOOLEAN      := TRUE;
```

(次のページに続く)

(前のページからの続き)

```

BEGIN
    DBMS_RLS.ADD_POLICY(
        v_object_schema,
        v_object_name,
        v_policy_name,
        v_function_schema,
        v_policy_function,
        v_statement_types,
        v_update_check,
        v_enable
    );
END;

```

ポリシーが正常に作成されると、ユーザー salesmgr によってターミナルセッションが開始されます。次のクエリは、vpemp テーブルの内容を示しています。

```

edb=# \c edb salesmgr
Password for user salesmgr:
You are now connected to database "edb" as user "salesmgr".
edb=> SELECT * FROM vpemp;

```

empno	ename	job	sal	comm	deptno	authid
7782	CLARK	MANAGER	2450.00		10	
7839	KING	PRESIDENT	5000.00		10	
7934	MILLER	CLERK	1300.00		10	
7369	SMITH	CLERK	800.00		20	researchmgr
7566	JONES	MANAGER	2975.00		20	researchmgr
7788	SCOTT	ANALYST	3000.00		20	researchmgr
7876	ADAMS	CLERK	1100.00		20	researchmgr
7902	FORD	ANALYST	3000.00		20	researchmgr
7499	ALLEN	SALESMAN	1600.00	300.00	30	salesmgr
7521	WARD	SALESMAN	1250.00	500.00	30	salesmgr
7654	MARTIN	SALESMAN	1250.00	1400.00	30	salesmgr
7698	BLAKE	MANAGER	2850.00		30	salesmgr
7844	TURNER	SALESMAN	1500.00	0.00	30	salesmgr
7900	JAMES	CLERK	950.00		30	salesmgr

(14 rows)

修飾されていない UPDATE コマンド（WHERE 節なし）が salesmgr ユーザーによって発行されます。

```

edb=> UPDATE vpemp SET comm = sal * .75;
UPDATE 6

```

テーブル内のすべての行を更新する代わりに、ポリシーは、ポリシー関数の述語 authid = SYS_CONTEXT('USERENV', 'SESSION_USER') で指定された authid 列に値 salesmgr が含まれる行のみに更新の効果を制限します。

次のクエリは、comm 列が authid に salesmgr が含まれる行に対してのみ変更されたことを示しています。他のすべての行は変更されません。

```
edb=> SELECT * FROM vpemp;
 empno | ename   | job       | sal   | comm   | deptno | authid
-----+-----+-----+-----+-----+-----+-----
  7782 | CLARK   | MANAGER   | 2450.00 |      |    10 |
  7839 | KING    | PRESIDENT | 5000.00 |      |    10 |
  7934 | MILLER  | CLERK     | 1300.00 |      |    10 |
  7369 | SMITH   | CLERK     |  800.00 |      |    20 | researchmgr
  7566 | JONES   | MANAGER   | 2975.00 |      |    20 | researchmgr
  7788 | SCOTT   | ANALYST   | 3000.00 |      |    20 | researchmgr
  7876 | ADAMS   | CLERK     | 1100.00 |      |    20 | researchmgr
  7902 | FORD    | ANALYST   | 3000.00 |      |    20 | researchmgr
  7499 | ALLEN   | SALESMAN  | 1600.00 | 1200.00 |    30 | salesmgr
  7521 | WARD    | SALESMAN  | 1250.00 |  937.50 |    30 | salesmgr
  7654 | MARTIN  | SALESMAN  | 1250.00 |  937.50 |    30 | salesmgr
  7698 | BLAKE   | MANAGER   | 2850.00 | 2137.50 |    30 | salesmgr
  7844 | TURNER  | SALESMAN  | 1500.00 | 1125.00 |    30 | salesmgr
  7900 | JAMES   | CLERK     |  950.00 |  712.50 |    30 | salesmgr
(14 rows)
```

さらに、ADD_POLICY 手順で update_check パラメーターが TRUE に設定されているため、authid 列に指定された値 researchmgr がセッションと一致しないため、次の INSERT コマンドは例外をスローします salesmgr であるため、ポリシーに失敗します。

```
edb=> INSERT INTO vpemp VALUES (9001,'SMITH','ANALYST',3200.00,NULL,20,
'researchmgr');
ERROR:  policy with check option violation
DETAIL:  Policy predicate was evaluated to FALSE with the updated values
```

update_check が FALSE に設定されていた場合、先行する INSERT コマンドは成功したでしょう。

次の例は、sec_relevant_cols パラメーターを使用して、特定の列が SQL コマンドで参照されている場合にのみポリシーを適用する方法を示しています。この例では、従業員の給与が 2000 未満の行を選択する次のポリシー関数を使用されます。

```
CREATE OR REPLACE FUNCTION sal_lt_2000 (
    p_schema    VARCHAR2,
    p_object     VARCHAR2
)
RETURN VARCHAR2
IS
BEGIN
    RETURN 'sal < 2000';
END
```

SELECT コマンドに列 sal または comm が含まれる場合にのみ適用されるように、ポリシーが作成されます。

```

DECLARE
    v_object_schema      VARCHAR2(30) := 'public';
    v_object_name         VARCHAR2(30) := 'vpemp';
    v_policy_name         VARCHAR2(30) := 'secure_salary';
    v_function_schema     VARCHAR2(30) := 'enterprisedb';
    v_policy_function     VARCHAR2(30) := 'sal_lt_2000';
    v_statement_types     VARCHAR2(30) := 'SELECT';
    v_sec_relevant_cols   VARCHAR2(30) := 'sal,comm';
BEGIN
    DBMS_RLS.ADD_POLICY(
        v_object_schema,
        v_object_name,
        v_policy_name,
        v_function_schema,
        v_policy_function,
        v_statement_types,
        sec_relevant_cols => v_sec_relevant_cols
    );
END;

```

クエリが sal 列または comm 列を参照しない場合、ポリシーは適用されません。次のクエリは、テーブル vpemp の 14 行すべてを返します。

```

edb=# SELECT empno, ename, job, deptno, authid FROM vpemp;
 empno | ename  |   job   | deptno |  authid
-----+-----+-----+-----+-----
  7782 | CLARK  | MANAGER |    10 |
  7839 | KING   | PRESIDENT |    10 |
  7934 | MILLER | CLERK   |    10 |
  7369 | SMITH  | CLERK   |    20 | researchmgr
  7566 | JONES  | MANAGER |    20 | researchmgr
  7788 | SCOTT  | ANALYST |    20 | researchmgr
  7876 | ADAMS  | CLERK   |    20 | researchmgr
  7902 | FORD   | ANALYST |    20 | researchmgr
  7499 | ALLEN  | SALESMAN |    30 | salesmgr
  7521 | WARD   | SALESMAN |    30 | salesmgr
  7654 | MARTIN | SALESMAN |    30 | salesmgr
  7698 | BLAKE  | MANAGER |    30 | salesmgr
  7844 | TURNER | SALESMAN |    30 | salesmgr
  7900 | JAMES  | CLERK   |    30 | salesmgr
(14 rows)

```

クエリが sal または comm 列を参照する場合、ポリシーはクエリに適用され、次のように sal が 2000 以上の行を削除します。

```

edb=# SELECT empno, ename, job, sal, comm, deptno, authid FROM vpemp;
 empno | ename  |   job   |  sal  |  comm  | deptno |  authid

```

(次のページに続く)

(前のページからの続き)

7934		MILLER		CLERK		1300.00				10		
7369		SMITH		CLERK		800.00				20		researchmgr
7876		ADAMS		CLERK		1100.00				20		researchmgr
7499		ALLEN		SALESMAN		1600.00		1200.00		30		salesmgr
7521		WARD		SALESMAN		1250.00		937.50		30		salesmgr
7654		MARTIN		SALESMAN		1250.00		937.50		30		salesmgr
7844		TURNER		SALESMAN		1500.00		1125.00		30		salesmgr
7900		JAMES		CLERK		950.00		712.50		30		salesmgr

(8 rows)

3.14.2 DROP_POLICY

DROP_POLICY 手順は、既存のポリシーを削除します。ポリシーに関連付けられたポリシー関数とデータベースオブジェクトは、DROP_POLICY 手順では削除されません。

この手順を実行するには、スーパーユーザーである必要があります。

```
DROP_POLICY(<object_schema> VARCHAR2, <object_name> VARCHAR2,
            <policy_name> VARCHAR2)
```

Parameters

object_schema

ポリシーが適用されるデータベースオブジェクトを含むスキーマの名前。

object_name

ポリシーが適用されるデータベースオブジェクトの名前。

policy_name

削除するポリシーの名前。

Examples

次の例では、テーブル public.vpemp のポリシー secure_update を削除します。

```
DECLARE
    v_object_schema    VARCHAR2(30) := 'public';
    v_object_name      VARCHAR2(30) := 'vpemp';
    v_policy_name      VARCHAR2(30) := 'secure_update';
BEGIN
    DBMS_RLS.DROP_POLICY(
        v_object_schema,
```

(次のページに続く)

(前のページからの続き)

```

        v_object_name,
        v_policy_name
    );
END;
```

3.14.3 ENABLE_POLICY

ENABLE_POLICY プロシージャは、指定されたデータベースオブジェクトの既存のポリシーを有効または無効にします。

この手順を実行するには、スーパーユーザーである必要があります。

```
ENABLE_POLICY(<object_schema> VARCHAR2, <object_name> VARCHAR2,
              <policy_name> VARCHAR2, <enable> BOOLEAN)
```

Parameters

object_schema

ポリシーが適用されるデータベースオブジェクトを含むスキーマの名前。

object_name

ポリシーが適用されるデータベースオブジェクトの名前。

policy_name

有効または無効にするポリシーの名前。

enable

TRUE に設定すると、ポリシーが有効になります。FALSE に設定すると、ポリシーは無効になります。

Examples

次の例では、テーブル public.vpemp のポリシー secure_update を無効にします。

```
DECLARE
    v_object_schema    VARCHAR2(30) := 'public';
    v_object_name      VARCHAR2(30) := 'vpemp';
    v_policy_name      VARCHAR2(30) := 'secure_update';
    v_enable           BOOLEAN := FALSE;
BEGIN
    DBMS_RLS.ENABLE_POLICY(
        v_object_schema,
        v_object_name,
        v_policy_name,
```

(次のページに続く)

(前のページからの続き)

```
        v_enable
    );
END;
```

3.15 DBMS_SCHEDULER

DBMS_SCHEDULER パッケージは、Oracle スタイルのジョブ、プログラム、およびジョブスケジュールを作成および管理する方法を提供します。DBMS_SCHEDULER パッケージは、次の機能と手順を実装しています。

機能/手順	戻りタイプ	説明
CREATE_JOB(job_name, job_type, job_action, number_of_arguments, start_date, repeat_interval, end_date, job_class, enabled, auto_drop, comments)	該当なし	CREATE_JOB 手順の最初の形式を使用してジョブを作成し、パラメーターを使用してプログラムとスケジュールの詳細を指定します。
CREATE_JOB(job_name, program_name, schedule_name, job_class, enabled, auto_drop, comments)	該当なし	CREATE_JOB の 2 番目の形式を使用して、名前付きプログラムと名前付きスケジュールを使用するジョブを作成します。
CREATE_PROGRAM(program_name, program_type, program_action, number_of_arguments, enabled, comments)	該当なし	CREATE_PROGRAM を使用してプログラムを作成します。
CREATE_SCHEDULE(schedule_name, start_date, repeat_interval, end_date, comments)	該当なし	CREATE_SCHEDULE 手順を使用して、スケジュールを作成します。
DEFINE_PROGRAM_ARGUMENT(program_name, argument_position, argument_name, argument_type, default_value, out_argument)	該当なし	DEFINE_PROGRAM_ARGUMENT プロシージャの最初の形式を使用して、デフォルト値を持つプログラム引数を定義します。
DEFINE_PROGRAM_ARGUMENT(program_name, argument_position, argument_name, argument_type, out_argument)	該当なし	DEFINE_PROGRAM_ARGUMENT プロシージャの最初の形式を使用して、デフォルト値を持たないプログラム引数を定義します。
DISABLE(name, force, commit_semantics)	該当なし	DISABLE 手順を使用して、ジョブまたはプログラムを無効にします。
DROP_JOB(job_name, force, defer, commit_semantics)	該当なし	DROP_JOB 手順を使用して、ジョブを削除します。
DROP_PROGRAM(program_name, force)	該当なし	DROP_PROGRAM 手順を使用して、プログラムをドロップします。
DROP_PROGRAM_ARGUMENT(program_name, argument_position)	該当なし	DROP_PROGRAM_ARGUMENT の最初の形式を使用して、引数の位置を指定してプログラム引数を削除します。

次のページに続く

表 7 – 前のページからの続き

DROP_PROGRAM_ARGUMENT (program_name, argument_name)	該 当 な し	引数名を指定してプログラム引数を削除する には、DROP_PROGRAM_ARGUMENT の 2 番目の形式を使用します。
DROP_SCHEDULE (schedule_name, force)	該 当 な し	DROP SCHEDULE 手順を使用して、スケ ジュールを削除します。
ENABLE (name, commit_semantics)	該 当 な し	ENABLE コマンドを使用して、プログラム またはジョブを有効にします。
EVALUATE_CALENDAR_STRING (calendar_string, start_date, return_date_after, next_run_date)	該 当 な し	EVALUATE_CALENDAR_STRING を使用 して、ユーザー定義のカレンダースケ ジュールで記述された実行日を確認します。
RUN_JOB (job_name, use_current_session, manually)	該 当 な し	RUN_JOB プロシージャを使用して、ジョ ブをすぐに実行します。
SET_JOB_ARGUMENT_VALUE (job_name, argument_position, argument_value)	該 当 な し	SET_JOB_ARGUMENT 値の最初の形式を 使用して、引数の位置によって記述される ジョブ引数の値を設定します。
SET_JOB_ARGUMENT_VALUE (job_name, argument_name, argument_value)	該 当 な し	SET_JOB_ARGUMENT 値の 2 番目の形式 を使用して、引数の名前で記述されたジョ ブ引数の値を設定します。

AdvancedServer の DBMS_SCHEDULER の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表
にリストされている関数と手順のみがサポートされています。

DBMS_SCHEDULER パッケージは pgAgent サービスに依存しています。DBMS_SCHEDULER を使用する前に、
サーバーに pgAgent サービスをインストールして実行する必要があります。

DBMS_SCHEDULER を使用する前に、データベーススーパーユーザーは DBMS_SCHEDULER プログラム、スケ
ジュール、およびジョブが格納されるカタログテーブルを作成する必要があります。psql クライアントを使用し
てデータベースに接続し、コマンドを呼び出します。

```
CREATE EXTENSION dbms_scheduler;
```

デフォルトでは、dbms_scheduler 拡張子は「AdvancedServer インストール」の下 contrib/
dbms_scheduler_ext サブディレクトリにあります。

DBMS_SCHEDULER テーブルを作成した後は、スーパーユーザーのみがデータベースのダンプまたはリロードを
実行できることに注意してください。

3.15.1 Using Calendar Syntax to Specify a Repeating Interval

CREATE_JOB および CREATE_SCHEDULE の手順では、Oracle スタイルのカレンダー構文を使用して、ジョブまたはスケジュールが繰り返される間隔を定義します。各手順の repeat_interval パラメータでスケジューリング情報を提供する必要があります。

repeat_interval は、スケジュールされたジョブの実行間隔を定義する値（または一連の値）です。各値はトークンで構成され、その後に等号、スケジュールが実行される単位が続きます。複数のトークン値はセミコロン (;) で区切る必要があります。

たとえば、次の値:

```
FREQ=DAILY; BYDAY=MON, TUE, WED, THU, FRI; BYHOUR=17; BYMINUTE=45
```

毎週夜 5:45 に実行されるスケジュールを定義します。

次の表で説明されているトークンの種類と構文は、AdvancedServer でサポートされています。

トークンの種類	構文	有効な値
FREQ	FREQ=predefined_interval	predefined_interval は YEARLY, MONTHLY, WEEKLY, DAILY, HOURLY, MINUTELY のいずれかです。SECONDLY キーワードはサポートされていません。
BYMONTH	BYMONTH=month(, month)...	month は月名の 3 文字の略語です: JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC
BYMONTH	BYMONTH=month(, month).. .	month は月を表す数値です: 1 2 3 4 5 6 7 8 9 10 11 12
BYMONTHDAY	BYMONTHDAY=day_of_month	day_of_month は 1 から 31 までの値です
BYDAY	BYDAY=weekday	weekday は、曜日を表す 3 文字の略語または 1 桁の値です。
		Monday MON 1
		Tuesday TUE 2
		Wednesday WED 3
		Thursday THU 4
		Friday FRI 5
		Saturday SAT 6

次のページに続く

表 8 – 前のページからの続き

トークンの種類	構文	有効な値
		Sunday SUN 7
BYDATE	BYDATE=date (, date)...	日付は YYYYMMDD です。 YYYY は年を表す 4 桁の年です。 MM は 2 桁の月を表し、 DD は 2 桁の日を表しています。
BYDATE	BYDATE=date (, date)...	日付は MMDD です。 MM は 2 桁の月を表し、 DD は日を表す 2 桁の日です
BYHOUR	BYHOUR=hour	hour は 0 から 23 までの値です。
BYMINUTE	BYMINUTE=minute	minute は 0 から 59 までの値です。

3.15.2 CREATE_JOB

CREATE_JOB 手順を使用してジョブを作成します。手順には 2 つの形式があります。プロシージャの最初の形式は、ジョブ定義内のスケジュールと、ジョブの実行時に呼び出されるジョブアクションを指定します。

```
CREATE_JOB (
  <job_name> IN VARCHAR2,
  <job_type> IN VARCHAR2,
  <job_action> IN VARCHAR2,
  <number_of_arguments> IN PLS_INTEGER DEFAULT 0,
  <start_date> IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  <repeat_interval> IN VARCHAR2 DEFAULT NULL,
  <end_date> IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  <job_class> IN VARCHAR2 DEFAULT 'DEFAULT_JOB_CLASS',
  <enabled> IN BOOLEAN DEFAULT FALSE,
  <auto_drop> IN BOOLEAN DEFAULT TRUE,
  <comments> IN VARCHAR2 DEFAULT NULL)
```

2 番目の形式では、ジョブスケジュールを使用して、ジョブを実行するスケジュールを指定し、ジョブの実行時に実行するプログラムの名前を指定します。

```
CREATE_JOB (
  <job_name> IN VARCHAR2,
  <program_name> IN VARCHAR2,
  <schedule_name> IN VARCHAR2,
```

(次のページに続く)

(前のページからの続き)

```
<job_class> IN VARCHAR2 DEFAULT 'DEFAULT_JOB_CLASS',  
<enabled> IN BOOLEAN DEFAULT FALSE,  
<auto_drop> IN BOOLEAN DEFAULT TRUE,  
<comments> IN VARCHAR2 DEFAULT NULL)
```

Parameters

job_name

job_name は、作成されるジョブのスキーマ修飾名をオプションで指定します。

job_type

job_type はジョブのタイプを指定します。CREATE_JOB の現在の実装は、PLSQL_BLOCK または STORED_PROCEDURE のジョブタイプをサポートしています。

job_action

job_type が PLSQL_BLOCK の場合、job_action はジョブの実行時に呼び出される PL/SQL ブロックのコンテンツを指定します。ブロックはセミコロン (;) で終了する必要があります。

job_type が STORED_PROCEDURE の場合、job_action はオプションでスキーマ修飾されたプロシージャ名を指定します。

number_of_arguments

number_of_arguments は、ジョブが期待する引数の数を指定する INTEGER 値です。デフォルトは 0 です。

start_date

start_date は TIMESTAMP WITH TIME ZONE の値で、ジョブの実行が最初にスケジュールされる時間を指定します。デフォルト値は NULL で、ジョブが有効になったときにジョブを実行するようにスケジュールする必要があることを示します。

repeat_interval

repeat_interval は、ジョブを繰り返す頻度を指定する VARCHAR2 の値です。repeat_interval が指定されていない場合、ジョブは 1 回だけ実行されます。デフォルト値は NULL です。

end_date

end_date は TIMESTAMP WITH TIME ZONE の値で、それ以降はジョブが実行されなくなるまでの時間を指定します。日付が指定されている場合、end_date は start_date の後になければなりません。デフォルト値は NULL です。

end_date が指定されておらず、repeat_interval が指定されている場合、ジョブは無効になるまで無期限に繰り返されることに注意してください。

program_name

program_name は、ジョブによって実行されるプログラムの名前です。

schedule_name

schedule_name は、ジョブに関連付けられたスケジュールの名前です。

job_class

job_class は互換性のために受け入れられ、無視されます。

enabled

enabled は、作成時にジョブを有効にするかどうかを指定する BOOLEAN 値です。デフォルトでは、ジョブは enabled が FALSE に設定された無効な状態で作成されます。ジョブを有効にするには、ジョブの作成時に TRUE の値を指定するか、DBMS_SCHEDULER.ENABLE 手順でジョブを有効にします。

auto_drop

auto_drop パラメーターは互換性のために受け入れられ、無視されます。デフォルトでは、ジョブのステータスは end_date で指定された時間が経過すると DISABLED に変更されます。

comments

comments パラメーターを使用して、ジョブに関するコメントを指定します。

Example

次の例は、CREATE_JOB プロシージャの呼び出しを示しています。

```
EXEC
DBMS_SCHEDULER.CREATE_JOB (
  job_name      => 'update_log',
  job_type      => 'PLSQL_BLOCK',
  job_action    => 'BEGIN INSERT INTO my_log VALUES(current_timestamp);
                  END;',
  start_date    => '01-JUN-15 09:00:00.000000',
  repeat_interval => 'FREQ=DAILY;BYDAY=MON,TUE,WED,THU,FRI;BYHOUR=17;',
  end_date      => NULL,
  enabled       => TRUE,
  comments      => 'This job adds a row to the my_log table.');
```

コードフラグメントは、update_log という名前のジョブを作成し、毎週夜 5 時に実行します。ジョブは、現在のタイムスタンプをログファイル (my_log) に挿入する PL/SQL ブロックを実行します。end_date が指定されていないため、DBMS_SCHEDULER.DISABLE 手順で無効になるまでジョブが実行されます。

3.15.3 CREATE_PROGRAM

CREATE_PROGRAM 手順を使用して DBMS_SCHEDULER プログラムを作成します。署名は次のとおりです。

```
CREATE_PROGRAM(  
  <program_name> IN VARCHAR2,  
  <program_type> IN VARCHAR2,  
  <program_action> IN VARCHAR2,  
  <number_of_arguments> IN PLS_INTEGER DEFAULT 0,  
  <enabled> IN BOOLEAN DEFAULT FALSE,  
  <comments> IN VARCHAR2 DEFAULT NULL)
```

Parameters

program_name

program_name は、作成中のプログラムの名前を指定します。

program_type

program_type はプログラムのタイプを指定します。現在の CREATE_PROGRAM の実装は、PLSQL_BLOCK または PROCEDURE の program_type をサポートしています。

program_action

program_type が PLSQL_BLOCK、program_action の場合、プログラムの起動時に実行される PL/SQL ブロックが含まれます。PL/SQL ブロックはセミコロン (;) で終了する必要があります。

program_type が PROCEDURE の場合、program_action にはストアドプロシージャの名前が含まれます。

number_of_arguments

program_type が PLSQL_BLOCK の場合、この引数は無視されます。

program_type が PROCEDURE の場合、number_of_arguments はプロシージャに必要な引数の数を指定します。デフォルト値は 0 です。

enabled

enabled は、プログラムを作成するかどうかを指定します。

- enabled が TRUE の場合、プログラムは有効に作成されます。
- enabled が FALSE の場合、プログラムは無効に作成されます。DBMS_SCHEDULER.ENABLE プログラムを使用して、無効なプログラムを有効にします。

デフォルト値は FALSE です。

comments

`comments` パラメーターを使用して、プログラムに関するコメントを指定します。デフォルトでは、このパラメーターは `NULL` です。

Example

次の `CREATE_PROGRAM` プロシージャの呼び出しにより、`update_log` という名前のプログラムが作成されます。

```
EXEC
DBMS_SCHEDULER.CREATE_PROGRAM (
  program_name      => 'update_log',
  program_type      => 'PLSQL_BLOCK',
  program_action    => 'BEGIN INSERT INTO my_log VALUES(current_timestamp);
                        END;',
  enabled           => TRUE,
  comment           => 'This program adds a row to the my_log table.');
```

`update_log` は、現在の日付と時刻を含む行を `my_log` テーブルに追加する `PL/SQL` ブロックです。`CREATE_PROGRAM` プロシージャを実行すると、プログラムが有効になります。

3.15.4 CREATE_SCHEDULE

`CREATE_SCHEDULE` 手順を使用して、ジョブスケジュールを作成します。`CREATE_SCHEDULE` プロシージャのシグネチャは次のとおりです。

```
CREATE_SCHEDULE (
  <schedule_name> IN VARCHAR2,
  <start_date> IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  <repeat_interval> IN VARCHAR2,
  <end_date> IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  <comments> IN VARCHAR2 DEFAULT NULL)
```

Parameters

`schedule_name`

`schedule_name` はスケジュールの名前を指定します。

`start_date`

`start_date` は `TIMESTAMP WITH TIME ZONE` の値で、スケジュールを実行する資格がある日時を指定します。`start_date` が指定されていない場合、ジョブが有効になった日付が `start_date` として使用されます。デフォルトでは、`start_date` は `NULL` です。

`repeat_interval`

`repeat_interval` は、ジョブを繰り返す頻度を指定する `VARCHAR2` の値です。 `repeat_interval` が指定されていない場合、ジョブは `start_date` で指定された日に 1 回だけ実行されます。

注釈: `start_date` または `repeat_interval` のいずれかの値を指定する必要があります。 `start_date` と `repeat_interval` の両方が `NULL` の場合、サーバーはエラーを返します。

`end_date`

`end_date` は `TIMESTAMP WITH TIME ZONE` の値で、それ以降はスケジュールが実行されなくなる時間を指定します。日付が指定されている場合、`end_date` は `start_date` の後になければなりません。デフォルト値は `NULL` です。

注釈: `repeat_interval` が指定され、`end_date` が指定されていない場合、スケジュールは無効になるまで無期限に繰り返されます。

`comments`

`comments` パラメーターを使用して、スケジュールに関するコメントを指定します。デフォルトでは、このパラメーターは `NULL` です。

Example

次のコードは、`CREATE_SCHEDULE` を呼び出して `weeknights_at_5` という名前のスケジュールを作成します。

```
EXEC
DBMS_SCHEDULER.CREATE_SCHEDULE (
  schedule_name    => 'weeknights_at_5',
  start_date       => '01-JUN-13 09:00:00.000000',
  repeat_interval  => 'FREQ=DAILY;BYDAY=MON,TUE,WED,THU,FRI;BYHOUR=17;',
  comments         => 'This schedule executes each weeknight at 5:00');
```

スケジュールは 2013 年 6 月 1 日以降の平日の午後 5 時に実行されます。 `end_date` が指定されていないため、スケジュールは `DBMS_SCHEDULER.DISABLE` で無効になるまで無期限に実行されます。

3.15.5 DEFINE_PROGRAM_ARGUMENT

DEFINE_PROGRAM_ARGUMENT プロシージャを使用して、プログラムの引数を定義します。DEFINE_PROGRAM_ARGUMENT 手順には 2 つの形式があります。最初の形式は、デフォルト値で引数を定義します:

```
DEFINE_PROGRAM_ARGUMENT (
    <program_name> IN VARCHAR2,
    <argument_position> IN PLS_INTEGER,
    <argument_name> IN VARCHAR2 DEFAULT NULL,
    <argument_type> IN VARCHAR2,
    <default_value> IN VARCHAR2,
    <out_argument> IN BOOLEAN DEFAULT FALSE)
```

2 番目の形式は、デフォルト値なしで引数を定義します。

```
DEFINE_PROGRAM_ARGUMENT (
    <program_name> IN VARCHAR2,
    <argument_position> IN PLS_INTEGER,
    <argument_name> IN VARCHAR2 DEFAULT NULL,
    <argument_type> IN VARCHAR2,
    <out_argument> IN BOOLEAN DEFAULT FALSE)
```

Parameters

program_name

program_name は、引数が属するプログラムの名前です。

argument_position

argument_position は、引数がプログラムに渡されるとき引数の位置を指定します。

argument_name

argument_name は、引数のオプション名を指定します。デフォルトでは、argument_name は NULL です。

argument_type IN VARCHAR2

argument_type は引数のデータ型を指定します。

default_value

default_value は、引数に割り当てられるデフォルト値を指定します。default_value は、ジョブの実行時にジョブで指定された値によってオーバーライドされます。

out_argument IN BOOLEAN DEFAULT FALSE

out_argument は現在使用されていません。指定する場合、値は FALSE でなければなりません。

Example

次のコードは、`DEFINE_PROGRAM_ARGUMENT` プロシージャを使用して、`add_emp` という名前のプログラムの最初と 2 番目の引数を定義します。

```
EXEC
DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT (
  program_name      => 'add_emp',
  argument_position  => 1,
  argument_name      => 'dept_no',
  argument_type      => 'INTEGER',
  default_value      => '20');
EXEC
DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT (
  program_name      => 'add_emp',
  argument_position  => 2,
  argument_name      => 'emp_name',
  argument_type      => 'VARCHAR2');
```

最初の引数は、`dept_no` という名前の `INTEGER` 値で、デフォルト値は 20 です。2 番目の引数は、`emp_name` という名前の `VARCHAR2` 値です。2 番目の引数にはデフォルト値がありません。

3.15.6 無効にする

`DISABLE` 手順を使用して、プログラムまたはジョブを無効にします。`DISABLE` プロシージャのシグネチャは次のとおりです。

```
DISABLE (
  <name> IN VARCHAR2,
  <force> IN BOOLEAN DEFAULT FALSE,
  <commit_semantics> IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR')
```

Parameters

`name`

`name` は、無効にするプログラムまたはジョブの名前を指定します。

`force`

`force` は互換性のために受け入れられ、無視されます。

`commit_semantics`

`commit_semantics` は、プログラムまたはジョブを無効にしているときに発生したエラーの処理方法をサーバーに指示します。デフォルトでは、`commit_semantics` は `STOP_ON_FIRST_ERROR` に設定さ

れ、エラーが発生したときにサーバーを停止するように指示します。エラーの前に正常に無効化されたプログラムまたはジョブはすべてディスクにコミットされます。

TRANSACTIONAL および ABSORB_ERRORS キーワードは互換性のために受け入れられ、無視されます。

Example

次の DISABLE プロシージャの呼び出しは、update_emp という名前のプログラムを無効にします。

```
DBMS_SCHEDULER.DISABLE('update_emp');
```

3.15.7 DROP_JOB

DROP_JOB 手順を使用してジョブを DROP、ジョブに属するすべての引数を DROP にし、将来のジョブ実行を排除します。プロシージャのシグネチャは次のとおりです。

```
DROP_JOB (
  <job_name> IN VARCHAR2,
  <force> IN BOOLEAN DEFAULT FALSE,
  <defer> IN BOOLEAN DEFAULT FALSE,
  <commit_semantics> IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR')
```

Parameters

job_name

job_name は、ドロップされるジョブの名前を指定します。

force

force は互換性のために受け入れられ、無視されます。

defer

defer は互換性のために受け入れられ、無視されます。

commit_semantics

commit_semantics は、プログラムまたはジョブのドロップ中に発生したエラーの処理方法をサーバーに指示します。デフォルトでは、commit_semantics は STOP_ON_FIRST_ERROR に設定され、エラーが発生したときにサーバーを停止するように指示します。

TRANSACTIONAL および ABSORB_ERRORS キーワードは互換性のために受け入れられ、無視されます。

Example

次の DROP_JOB を呼び出すと、update_log という名前のジョブが削除されます。

```
DBMS_SCHEDULER.DROP_JOB('update_log');
```

3.15.8 DROP_PROGRAM

プログラムをドロップする DROP_PROGRAM 手順。DROP_PROGRAM プロシージャのシグネチャは次のとおりです。

```
DROP_PROGRAM(  
  <program_name> IN VARCHAR2,  
  <force> IN BOOLEAN DEFAULT FALSE)
```

Parameters

program_name

program_name は、ドロップされるプログラムの名前を指定します。

force

force は BOOLEAN 値で、サーバーに依存ジョブを持つプログラムの処理方法を指示します。

FALSE を指定すると、プログラムがジョブによって参照されている場合にエラーを返すようサーバーに指示します。

TRUE を指定すると、プログラムを削除する前にプログラムを参照するジョブを無効にするようサーバーに指示します。

デフォルト値は FALSE です。

Example

次の DROP_PROGRAM を呼び出すと、update_emp という名前のジョブが削除されます。

```
DBMS_SCHEDULER.DROP_PROGRAM('update_emp');
```

3.15.9 DROP_PROGRAM_ARGUMENT

DROP_PROGRAM_ARGUMENT プロシージャを使用して、プログラム引数を削除します。DROP_PROGRAM_ARGUMENT 手順には 2 つの形式があります。最初の形式では、引数の位置を使用して、削除する引数を指定します。

```
DROP_PROGRAM_ARGUMENT(  
  <program_name> IN VARCHAR2,  
  <argument_position> IN PLS_INTEGER)
```

2 番目の形式は引数名を取ります:

```
DROP_PROGRAM_ARGUMENT (
    <program_name> IN VARCHAR2,
    <argument_name> IN VARCHAR2)
```

Parameters

program_name

program_name は、変更されるプログラムの名前を指定します。

argument_position

argument_position は、ドロップされる引数の位置を指定します。

argument_name

argument_name は、ドロップされる引数の名前を指定します。

Examples

次の DROP_PROGRAM_ARGUMENT を呼び出すと、update_emp プログラムの最初の引数が削除されます。

```
DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT ('update_emp', 1);
```

次の DROP_PROGRAM_ARGUMENT を呼び出すと、emp_name という名前の引数が削除されます。

```
DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (update_emp', 'emp_name');
```

3.15.10 DROP_SCHEDULE

DROP_SCHEDULE 手順を使用して、スケジュールを削除します。署名は次のとおりです。

```
DROP_SCHEDULE (
    <schedule_name> IN VARCHAR2,
    <force> IN BOOLEAN DEFAULT FALSE)
```

Parameters

schedule_name

schedule_name は、ドロップされるスケジュールの名前を指定します。

force

force は、指定されたスケジュールがジョブによって参照される場合のサーバーの動作を指定します。

- FALSE を指定すると、指定したスケジュールがジョブによって参照されている場合にエラーを返すようサーバーに指示します。これがデフォルトの動作です。
- TRUE を指定して、スケジュールを削除する前に、指定されたスケジュールを使用するジョブを無効にするようサーバーに指示します。実行中のジョブは、スケジュールが削除される前に完了することができます。

Example

次の DROP_SCHEDULE を呼び出すと、weeknights_at_5 という名前のスケジュールが削除されます。

```
DBMS_SCHEDULER.DROP_SCHEDULE('weeknights_at_5', TRUE);
```

サーバーは、スケジュールを削除する前に、スケジュールを使用するジョブを無効にします。

3.15.11 有効にする

ENABLE の手順を使用して、無効なプログラムまたはジョブを有効にします。

ENABLE プロシージャのシグネチャは次のとおりです。

```
ENABLE (  
    <name> IN VARCHAR2,  
    <commit_semantics> IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR')
```

Parameters

name

name は、有効にするプログラムまたはジョブの名前を指定します。

commit_semantics

commit_semantics は、プログラムまたはジョブの有効化中に発生したエラーの処理方法をサーバーに指示します。デフォルトでは、commit_semantics は STOP_ON_FIRST_ERROR に設定され、エラーが発生したときにサーバーを停止するように指示します。

TRANSACTIONAL および ABSORB_ERRORS キーワードは互換性のために受け入れられ、無視されます。

Example

次の DBMS_SCHEDULER.ENABLE を呼び出すと、update_emp プログラムが有効になります。

```
DBMS_SCHEDULER.ENABLE('update_emp');
```

3.15.12 EVALUATE_CALENDAR_STRING

CREATE_SCHEDULE 手順でスケジュールを作成するときに指定した repeat_interval 値を評価するには、EVALUATE_CALENDAR_STRING 手順を使用します。EVALUATE_CALENDAR_STRING プロシージャは、実際にジョブをスケジュールせずに、指定されたスケジュールが実行される日付と時刻を返します。

EVALUATE_CALENDAR_STRING プロシージャのシグネチャは次のとおりです。

```
EVALUATE_CALENDAR_STRING (
    <calendar_string> IN VARCHAR2,
    <start_date> IN TIMESTAMP WITH TIME ZONE,
    <return_date_after> IN TIMESTAMP WITH TIME ZONE,
    <next_run_date> OUT TIMESTAMP WITH TIME ZONE)
```

Parameters

calendar_string

calendar_string は、評価中の repeat_interval を説明するカレンダー文字列です。

start_date IN TIMESTAMP WITH TIME ZONE

start_date は、repeat_interval が有効になる日時です。

return_date_after

repeat_interval を評価するときに EVALUATE_CALENDAR_STRING が開始日として使用する日付と時刻を指定するには、return_date_after パラメーターを使用します。

たとえば、return_date_after の値に 01-APR-13 09.00.00.000000, EVALUATE_CALENDAR_STRING を指定すると、2013 年 4 月 1 日以降のスケジュールの最初の反復の日付と時刻が返されます。

next_run_date OUT TIMESTAMP WITH TIME ZONE

next_run_date は OUT パラメーターで、return_date_after パラメーターで指定された日付の後のスケジュールの最初のおカレンスが含まれます。

Example

次の例では、カレンダー文字列を評価し、2013 年 6 月 15 日以降にスケジュールが実行される最初の日付と時刻を返します。

```
DECLARE
    result      TIMESTAMP;
BEGIN

    DBMS_SCHEDULER.EVALUATE_CALENDAR_STRING
```

(次のページに続く)

(前のページからの続き)

```
(
  'FREQ=DAILY;BYDAY=MON,TUE,WED,THU,FRI;BYHOUR=17;',
  '15-JUN-2013', NULL, result
);

DBMS_OUTPUT.PUT_LINE('next_run_date: ' || result);
END;
/

next_run_date: 17-JUN-13 05.00.00.000000 PM
```

2013 年 6 月 15 日は土曜日です。スケジュールは 2013 年 6 月 17 日月曜日の午後 5 時まで実行されません。

3.15.13 RUN_JOB

RUN_JOB 手順を使用して、ジョブをすぐに実行します。RUN_JOB プロシージャのシグネチャは次のとおりです。

```
RUN_JOB (
  <job_name> IN VARCHAR2,
  <use_current_session> IN BOOLEAN DEFAULT TRUE
```

Parameters

job_name

job_name は、実行するジョブの名前を指定します。

use_current_session

デフォルトでは、ジョブは現在のセッションで実行されます。指定する場合、use_current_session を TRUE に設定する必要があります。use_current_session が FALSE に設定されている場合、AdvancedServer はエラーを返します。

Example

次の RUN_JOB を呼び出すと、update_log という名前のジョブが実行されます。

```
DBMS_SCHEDULER.RUN_JOB('update_log', TRUE);
```

2 番目の引数として TRUE の値を渡すと、サーバーは現在のセッションでジョブを呼び出すように指示されます。

3.15.14 SET_JOB_ARGUMENT_VALUE

SET_JOB_ARGUMENT_VALUE プロシージャを使用して、引数の値を指定します。SET_JOB_ARGUMENT_VALUE 手順には 2 つの形式があります。最初の形式は、位置によって変更される引数を指定します。

```
SET_JOB_ARGUMENT_VALUE (
    <job_name> IN VARCHAR2,
    <argument_position> IN PLS_INTEGER,
    <argument_value> IN VARCHAR2)
```

2 番目の形式では、引数名を使用して、変更する引数を指定します。

```
SET_JOB_ARGUMENT_VALUE (
    <job_name> IN VARCHAR2,
    <argument_name> IN VARCHAR2,
    <argument_value> IN VARCHAR2)
```

SET_JOB_ARGUMENT_VALUE プロシージャによって設定された引数値は、デフォルトで設定された値をオーバーライドします。

Parameters

job_name

job_name は、変更された引数が属するジョブの名前を指定します。

argument_position

argument_position を使用して、値を設定する引数の位置を指定します。

argument_name

argument_name を使用して、値を設定する名前で引数を指定します。

argument_value

argument_value は、引数の新しい値を指定します。

Examples

次の例では、30 の値を update_emp ジョブの最初の引数に割り当てます。

```
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('update_emp', 1, '30');
```

次の例では、emp_name 引数を SMITH に設定しています。

```
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('update_emp', 'emp_name', 'SMITH');
```

3.16 DBMS_SESSION

AdvancedServer は、次の DBMS_SESSION.SET_ROLE 手順のサポートを提供します。

機能/手順	戻りタイプ	説明
SET_ROLE(role_cmd)	該当なし	SET_ROLE ステートメントに続いて role_cmd で指定された文字列値を実行します。

AdvancedServer の DBMS_SESSION の実装は、Oracle のバージョンと比較すると部分的な実装です。DBMS_SESSION.SET_ROLE のみがサポートされます。

3.16.1 SET_ROLE

SET_ROLE プロシージャは、role_cmd で指定されたロールに現在のセッションユーザーを設定します。SET_ROLE 手順を呼び出した後、現在のセッションは指定されたロールに割り当てられた権限を使用します。プロシージャのシグネチャは次のとおりです。

```
SET_ROLE(<role_cmd>)
```

SET_ROLE プロシージャは、role_cmd に指定された値を SET_ROLE ステートメントに追加してから、ステートメントを呼び出します。

Parameters

role_cmd

role_cmd は、文字列値の形式でロール名を指定します。

Example

次の SET_ROLE プロシージャの呼び出しは、SET_ROLE コマンドを呼び出して、現在のセッションユーザーの ID をマネージャーに設定します。

```
edb=# exec DBMS_SESSION.SET_ROLE('manager');
```

3.17 DBMS_SQL

DBMS_SQL パッケージは、Oracle データベースと EDB 動的 SQL 機能との互換性があるアプリケーションインターフェイスを提供します。DBMS_SQL を使用すると、実行時に（アプリケーションを作成するときではなく）クエリやその他のコマンドを作成できます。EDBAdvancedServer は、動的 SQL のネイティブサポートを提供します。DBMS_SQL は、アプリケーションを変更せずに、Oracle データベースと互換性のある方法で動的 SQL を使用する方法を提供します。

DBMS_SQL は、動的 SQL ステートメントを実行するときに現在のユーザーの権限を引き継ぎます。

機能/手順	機能または手順	戻りタイプ	説明
BIND_VARIABLE(c, name, value [, out_value_size])	手続き	該当なし	値を変数にバインドします。
BIND_VARIABLE_CHAR(c, name, value [, out_value_size])	手続き	該当なし	CHAR 値を変数にバインドします。
BIND_VARIABLE_RAW(c, name, value [, out_value_size])	手続き	該当なし	RAW 値を変数にバインドします。
CLOSE_CURSOR(c IN OUT)	手続き	該当なし	カーソルを閉じます。
COLUMN_VALUE(c, position, value OUT [, column_error OUT [, actual_length OUT]])	手続き	該当なし	列値を変数に返します。
COLUMN_VALUE_CHAR(c, position, value OUT [, column_error OUT [, actual_length OUT]])	手続き	該当なし	CHAR 列の値を変数に返します。
COLUMN_VALUE_RAW(c, position, value OUT [, column_error OUT [, actual_length OUT]])	手続き	該当なし	RAW 列の値を変数に返します。
COLUMN_VALUE_LONG(c, position, length, offset, value OUT, value_length OUT)	手続き	該当なし	LONG 列の値の一部を変数に返します。
DEFINE_COLUMN(c, position, column [, column_size])	手続き	該当なし	SELECT リストに列を定義します。
DEFINE_COLUMN_CHAR(c, position, column, column_size)	手続き	該当なし	SELECT リストの CHAR 列を定義します。
DEFINE_COLUMN_RAW(c, position, column, column_size)	手続き	該当なし	SELECT リストの RAW 列を定義します。
DEFINE_COLUMN_LONG(c, position)	手続き	該当なし	SELECT リストの LONG 列を定義します。
DESCRIBE_COLUMNS	手続き	該当なし	カーソル結果セットを保持する列を定義します。
EXECUTE(c)	機能	整数	カーソルを実行します。
EXECUTE_AND_FETCH(c [, exact])	機能	整数	カーソルを実行し、単一の行をフェッチします。
FETCH_ROWS(c)	機能	整数	カーソルから行をフェッチします。
IS_OPEN(c)	機能	ブーリアン	カーソルが開いているかどうかを確認します。
LAST_ROW_COUNT	機能	整数	フェッチされた行の累積数を返します。
LAST_ERROR_POSITION	機能	整数	エラーが発生した SQL ステートメントの位置を返します。
OPEN_CURSOR	機能	整数	カーソルを開きます。

AdvancedServer の DBMS_SQL の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表にリストされている関数と手順のみがサポートされています。

次の表に、DBMS_SQL パッケージで使用可能なパブリック変数を示します。

パブリック変数	データ型	価値	説明
native	INTEGER	1	Oracle 構文との互換性のために提供されています。詳細については、DBMS_SQL.PARSE を参照してください。
V6	INTEGER	2	Oracle 構文との互換性のために提供されています。詳細については、DBMS_SQL.PARSE を参照してください。
V7	INTEGER	3	Oracle 構文との互換性のために提供されています。詳細については、DBMS_SQL.PARSE を参照してください。

3.17.1 BIND_VARIABLE

BIND_VARIABLE プロシージャは、値を SQL コマンドの IN または IN OUT バインド変数に関連付ける機能を提供します。

```

BIND_VARIABLE(<c> INTEGER, <name> VARCHAR2,
  <value> { BLOB | CLOB | DATE | FLOAT | INTEGER | NUMBER |
           TIMESTAMP | VARCHAR2 }
  [, <out_value_size> INTEGER ])
```

Parameters

c

バインド変数を使用した SQL コマンドのカーソルのカーソル ID。

name

SQL コマンドのバインド変数の名前。

value

割り当てられる値。

out_value_size

name が IN OUT 変数の場合、出力値の最大長を定義します。指定しない場合、value の長さが想定されます。

Examples

次の匿名ブロックはバインド変数を使用して、emp 安定版に行を挿入します。

```

DECLARE
  curid          INTEGER;
  v_sql          VARCHAR2(150) := 'INSERT INTO emp VALUES ' ||
    '(:p_empno, :p_ename, :p_job, :p_mgr, ' ||
    ':p_hiredate, :p_sal, :p_comm, :p_deptno)';
  v_empno        emp.empno%TYPE;
  v_ename        emp.ename%TYPE;
  v_job          emp.job%TYPE;
  v_mgr          emp.mgr%TYPE;
  v_hiredate      emp.hiredate%TYPE;
  v_sal          emp.sal%TYPE;
  v_comm         emp.comm%TYPE;
  v_deptno       emp.deptno%TYPE;
  v_status       INTEGER;
BEGIN
  curid := DBMS_SQL.OPEN_CURSOR;
```

(次のページに続く)

(前のページからの続き)

```

DBMS_SQL.PARSE (curid,v_sql,DBMS_SQL.native);
v_empno      := 9001;
v_ename      := 'JONES';
v_job        := 'SALESMAN';
v_mgr        := 7369;
v_hiredate   := TO_DATE('13-DEC-07','DD-MON-YY');
v_sal        := 8500.00;
v_comm       := 1500.00;
v_deptno     := 40;
DBMS_SQL.BIND_VARIABLE (curid,':p_empno',v_empno);
DBMS_SQL.BIND_VARIABLE (curid,':p_ename',v_ename);
DBMS_SQL.BIND_VARIABLE (curid,':p_job',v_job);
DBMS_SQL.BIND_VARIABLE (curid,':p_mgr',v_mgr);
DBMS_SQL.BIND_VARIABLE (curid,':p_hiredate',v_hiredate);
DBMS_SQL.BIND_VARIABLE (curid,':p_sal',v_sal);
DBMS_SQL.BIND_VARIABLE (curid,':p_comm',v_comm);
DBMS_SQL.BIND_VARIABLE (curid,':p_deptno',v_deptno);
v_status := DBMS_SQL.EXECUTE(curid);
DBMS_OUTPUT.PUT_LINE('Number of rows processed: ' || v_status);
DBMS_SQL.CLOSE_CURSOR (curid);
END;

Number of rows processed: 1

```

3.17.2 BIND_VARIABLE_CHAR

BIND_VARIABLE_CHAR プロシージャは、SQL コマンドで CHAR 値を IN または IN OUT バインド変数に関連付ける機能を提供します。

```

BIND_VARIABLE_CHAR(<c> INTEGER, <name> VARCHAR2, <value> CHAR
[, <out_value_size> INTEGER ])

```

Parameters

c

バインド変数を使用した SQL コマンドのカーソルのカーソル ID。

name

SQL コマンドのバインド変数の名前。

value

割り当てられるタイプ CHAR の値。

out_value_size

name が IN OUT 変数の場合、出力値の最大長を定義します。指定しない場合、value の長さが想定されます。

3.17.3 BIND_VARIABLE_RAW

BIND_VARIABLE_RAW プロシージャは、SQL コマンドで RAW 値を IN または IN OUT バインド変数に関連付ける機能を提供します。

```
BIND_VARIABLE_RAW(<c> INTEGER, <name> VARCHAR2, <value> RAW  
[, <out_value_size> INTEGER ])
```

Parameters

c

バインド変数を使用した SQL コマンドのカーソルのカーソル ID。

name

SQL コマンドのバインド変数の名前。

value

割り当てられるタイプ RAW の値。

out_value_size

name が IN OUT 変数の場合、出力値の最大長を定義します。指定しない場合、value の長さが想定されます。

3.17.4 CLOSE_CURSOR

CLOSE_CURSOR プロシージャは、開いているカーソルを閉じます。カーソルに割り当てられたリソースが解放され、使用できなくなります。

```
CLOSE_CURSOR(<c> IN OUT INTEGER)
```

Parameters

c

閉じられるカーソルのカーソル ID。

Examples

次の例では、以前に開いたカーソルを閉じます。


```

DECLARE
    curid          INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
        .
        .
        .
    DBMS_SQL.CLOSE_CURSOR(curid);
END;

```

3.17.5 COLUMN_VALUE

COLUMN_VALUE プロシージャは、カーソルから値を受け取る変数を定義します。

```

COLUMN_VALUE(<c> INTEGER, <position> INTEGER, <value> OUT { BLOB |
    CLOB | DATE | FLOAT | INTEGER | NUMBER | TIMESTAMP | VARCHAR2 }
[, <column_error> OUT NUMBER [, <actual_length> OUT INTEGER ]])

```

Parameters

c

定義されている変数にデータを返すカーソルのカーソル ID。

position

返されたデータのカーソル内の位置。カーソルの最初の値は位置 1 です。

value

以前のフェッチ呼び出しによってカーソルで返されたデータを受け取る変数。

column_error

列に関連付けられているエラー番号（ある場合）。

actual_length

切り捨て前のデータの実際の長さ。

Examples

次の例は、COLUMN_VALUE プロシージャを使用してカーソルから値を受け取る匿名ブロックの一部を示しています。

```

DECLARE
    curid          INTEGER;
    v_empno        NUMBER(4);

```

(次のページに続く)

(前のページからの続き)

```

v_ename          VARCHAR2(10);
v_hiredate        DATE;
v_sal             NUMBER(7,2);
v_comm           NUMBER(7,2);
v_sql            VARCHAR2(50) := 'SELECT empno, ename, hiredate, sal, ' ||
                                'comm FROM emp';
v_status         INTEGER;
BEGIN
    .
    .
    .
    LOOP
        v_status := DBMS_SQL.FETCH_ROWS(curid);
        EXIT WHEN v_status = 0;
        DBMS_SQL.COLUMN_VALUE(curid,1,v_empno);
        DBMS_SQL.COLUMN_VALUE(curid,2,v_ename);
        DBMS_SQL.COLUMN_VALUE(curid,3,v_hiredate);
        DBMS_SQL.COLUMN_VALUE(curid,4,v_sal);
        DBMS_SQL.COLUMN_VALUE(curid,4,v_sal);
        DBMS_SQL.COLUMN_VALUE(curid,5,v_comm);
        DBMS_OUTPUT.PUT_LINE(v_empno || ' ' || RPAD(v_ename,10) || ' ' ||
            TO_CHAR(v_hiredate,'yyyy-mm-dd') || ' ' ||
            TO_CHAR(v_sal,'9,999.99') || ' ' ||
            TO_CHAR(NVL(v_comm,0),'9,999.99'));
    END LOOP;
    DBMS_SQL.CLOSE_CURSOR(curid);
END;
```

3.17.6 COLUMN_VALUE_CHAR

COLUMN_VALUE_CHAR プロシージャは、カーソルから CHAR 値を受け取る変数を定義します。

```

COLUMN_VALUE_CHAR(<c> INTEGER, <position> INTEGER, <value> OUT CHAR
[, <column_error> OUT NUMBER [, <actual_length> OUT INTEGER ]])
```

Parameters

c

定義されている変数にデータを返すカーソルのカーソル ID。

position

返されたデータのカーソル内の位置。カーソルの最初の値は位置 1 です。

value

前のフェッチ呼び出しによってカーソルで返されたデータを受け取るデータ型 CHAR の変数。

column_error

列に関連付けられているエラー番号（ある場合）。

actual_length

切り捨て前のデータの実際の長さ。

3.17.7 COLUMN_VALUE_RAW

COLUMN_VALUE_RAW プロシージャは、カーソルから RAW 値を受け取る変数を定義します。

```
COLUMN_VALUE_RAW(<c> INTEGER, <position> INTEGER, <value> OUT RAW
[, <column_error> OUT NUMBER [, <actual_length> OUT INTEGER ]])
```

Parameters

c

定義されている変数にデータを返すカーソルのカーソル ID。

position

返されたデータのカーソル内の位置。カーソルの最初の値は位置 1 です。

value

前のフェッチ呼び出しによってカーソルで返されたデータを受け取るデータ型 RAW の変数。

column_error

列に関連付けられているエラー番号（ある場合）。

actual_length

切り捨て前のデータの実際の長さ。

3.17.8 COLUMN_VALUE_LONG

COLUMN_VALUE_LONG プロシージャは、LONG 列の値の一部を返します。

```
COLUMN_VALUE_LONG(<c> INTEGER, <position> INTEGER, <length> INTEGER,
<offset> INTEGER, <value> OUT VARCHAR2, <value_length> OUT INTEGER)
```

Parameters

c

値を取得するカーソルのカーソル ID。

position

値を取得する列の位置。

length

取得する long 値のバイト数。

offset

フェッチの開始のための長いフィールドへのオフセット。

value

列の値。

value_length

value に返されるバイト数。

例を参照するには、[*DEFINE_COLUMN_LONG*](#) を参照してください。

3.17.9 DEFINE_COLUMN

DEFINE_COLUMN プロシージャは、カーソルで返されて取得される SELECT リストの列または式を定義します。

```
DEFINE_COLUMN(<c> INTEGER, <position> INTEGER, <column> { BLOB |  
  CLOB | DATE | FLOAT | INTEGER | NUMBER | TIMESTAMP | VARCHAR2 }  
[, <column_size> INTEGER ])
```

Parameters

c

SELECT コマンドに関連付けられたカーソルのカーソル ID。

position

定義されている SELECT リスト内の列または式の位置。

column

SELECT リストの position の位置にある列または式と同じデータ型の変数。

column_size

返されるデータの最大長。column が VARCHAR2 の場合のみ、column_size を指定する必要があります。column_size を超える返されるデータは column_size 文字に切り捨てられます。

Examples

次に、emp テーブルの empno, ename, hiredate, sal 列と comm 列が DEFINE_COLUMN 手順でどのように定義されるかを示します。

```
DECLARE
    curid          INTEGER;
    v_empno        NUMBER(4);
    v_ename        VARCHAR2(10);
    v_hiredate     DATE;
    v_sal          NUMBER(7,2);
    v_comm         NUMBER(7,2);
    v_sql          VARCHAR2(50) := 'SELECT empno, ename, hiredate, sal, ' ||
                                   'comm FROM emp';
    v_status       INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
    DBMS_SQL.DEFINE_COLUMN(curid, 1, v_empno);
    DBMS_SQL.DEFINE_COLUMN(curid, 2, v_ename, 10);
    DBMS_SQL.DEFINE_COLUMN(curid, 3, v_hiredate);
    DBMS_SQL.DEFINE_COLUMN(curid, 4, v_sal);
    DBMS_SQL.DEFINE_COLUMN(curid, 5, v_comm);
    .
    .
    .
END;
```

以下は、まったく同じ結果を生成する前の例の代替を示しています。データ型の長さは関係ないことに注意してください。v_num が NUMBER(1) として定義されていても、empno, sal および comm 列はそれぞれ NUMBER(4) および NUMBER(7,2) に相当するデータを返します。「12」手順の宣言は、適切な最大サイズです。ename 列は、「11」に対して宣言された「10」というデータ型宣言ではなく、DEFINE_COLUMN コールの length パラメータで定義された最大 10 文字のデータを返します。返されるデータの実際のサイズは、「12」手順によって決まります。

```
DECLARE
    curid          INTEGER;
    v_num          NUMBER(1);
    v_varchar      VARCHAR2(1);
    v_date         DATE;
    v_sql          VARCHAR2(50) := 'SELECT empno, ename, hiredate, sal, ' ||
                                   'comm FROM emp';
    v_status       INTEGER;
BEGIN
```

(次のページに続く)

(前のページからの続き)

```
curid := DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE(curid,v_sql,DBMS_SQL.native);
DBMS_SQL.DEFINE_COLUMN(curid,1,v_num);
DBMS_SQL.DEFINE_COLUMN(curid,2,v_varchar,10);
DBMS_SQL.DEFINE_COLUMN(curid,3,v_date);
DBMS_SQL.DEFINE_COLUMN(curid,4,v_num);
DBMS_SQL.DEFINE_COLUMN(curid,5,v_num);
      .
      .
      .
END;
```

3.17.10 DEFINE_COLUMN_CHAR

DEFINE_COLUMN_CHAR プロシージャは、カーソルで返されて取得される SELECT リストの CHAR 列または式を定義します。

```
DEFINE_COLUMN_CHAR(<c> INTEGER, <position> INTEGER, <column>
CHAR, <column_size> INTEGER)
```

Parameters

c

SELECT コマンドに関連付けられたカーソルのカーソル ID。

position

定義されている SELECT リスト内の列または式の位置。

column

CHAR 変数。

column_size

返されるデータの最大長。返された column_size を超えるデータは column_size 文字に切り捨てられます。

3.17.11 DEFINE_COLUMN_RAW

DEFINE_COLUMN_RAW プロシージャは、カーソルで返されて取得される SELECT リストの RAW 列または式を定義します。

```
DEFINE_COLUMN_RAW(<c> INTEGER, <position> INTEGER, <column> RAW,
                  <column_size> INTEGER)
```

Parameters

c

SELECT コマンドに関連付けられたカーソルのカーソル ID。

position

定義されている SELECT リスト内の列または式の位置。

column

RAW 変数。

column_size

返されるデータの最大長。返された column_size を超えるデータは column_size 文字に切り捨てられます。

3.17.12 DEFINE_COLUMN_LONG

DEFINE_COLUMN_LONG プロシージャは、SELECT カーソルの長い列を定義します。

```
DEFINE_COLUMN_LONG(<c> INTEGER, <position> INTEGER)
```

Parameters

c

選択するように定義された行のカーソルのカーソル ID。

position

定義されている行の列の位置。

Examples

次の例は、DEFINE_COLUMN_LONG プロシージャを使用して SELECT リストの長い列を定義し、プロシージャ COLUMN_VALUE_LONG を使用して LONG 列の値の一部を変数に返す匿名ブロックを示しています。

```
DECLARE
    curid          INTEGER;
    v_ename        VARCHAR(20);
    sql_stmt       VARCHAR2(50) := 'SELECT ename ' || ' FROM emp WHERE empno
                                = 7844';

    v_status       INTEGER;
    v_length       INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(curid, sql_stmt, DBMS_SQL.native);
    DBMS_SQL.DEFINE_COLUMN_LONG(curid, 1);
    v_status := DBMS_SQL.EXECUTE(curid);
    v_status := DBMS_SQL.FETCH_ROWS(curid);
    DBMS_SQL.COLUMN_VALUE_LONG(curid, 1, 7, 0, v_ename, v_length);
    DBMS_OUTPUT.PUT_LINE('ename: ' || v_ename || ' & length: ' || v_length);
    DBMS_SQL.CLOSE_CURSOR(curid);
END;

ename: TURNER & length: 6
```

3.17.13 DESCRIBE COLUMNS

DESCRIBE_COLUMNS プロシージャは、カーソルが返す列を示します。

```
DESCRIBE_COLUMNS(<c> INTEGER, <col_cnt> OUT INTEGER, <desc_t> OUT
DESC_TAB);
```

Parameters

c

カーソルのカーソル ID。

col_cnt

カーソル結果セットの列数。

desc_tab

カーソルによって返される各列の説明を含むテーブル。説明は DESC_REC タイプで、次の値が含まれます。

列名	種類
col_type	整数
col_max_len	整数
col_name	VARCHAR2 (128)
col_name_len	整数
col_schema_name	VARCHAR2 (128)
col_schema_name_len	整数
col_precision	整数
col_scale	整数
col_charsetid	整数
col_charsetform	整数
col_null_ok	ブーリアン

3.17.14 EXECUTE

EXECUTE 関数は、解析された SQL コマンドまたは SPL ブロックを実行します。

```
<status> INTEGER EXECUTE (<c> INTEGER)
```

Parameters

c

実行する解析済み SQL コマンドまたは SPL ブロックのカーソル ID。

status

SQL コマンドが DELETE, INSERT または UPDATE の場合に処理された行の数。status は他のすべてのコマンドでは意味がありません。

Examples

次の匿名ブロックは、dept テーブルに行を挿入します。

```
DECLARE
    curid          INTEGER;
    v_sql          VARCHAR2(50);
    v_status       INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
    v_sql := 'INSERT INTO dept VALUES (50, ''HR'', ''LOS ANGELES'')';
    DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
    v_status := DBMS_SQL.EXECUTE(curid);
    DBMS_OUTPUT.PUT_LINE('Number of rows processed: ' || v_status);
```

(次のページに続く)

(前のページからの続き)

```
DBMS_SQL.CLOSE_CURSOR(curid);
END;
```

3.17.15 EXECUTE_AND_FETCH

関数 EXECUTE_AND_FETCH は解析された SELECT コマンドを実行し、1 行をフェッチします。

```
<status> INTEGER EXECUTE_AND_FETCH(<c> INTEGER
[, <exact> BOOLEAN ])
```

Parameters

c

実行する SELECT コマンドのカーソルのカーソル ID。

exact

TRUE に設定すると、結果セットの行数が 1 に正確に等しくない場合に例外がスローされます。FALSE に設定すると、例外はスローされません。デフォルトは FALSE です。exact が TRUE で、結果セットに行がない場合、NO_DATA_FOUND 例外がスローされます。exact が TRUE で、結果セットに複数の行がある場合、TOO_MANY_ROWS 例外がスローされます。

status

行が正常にフェッチされた場合は 1 を返し、フェッチする行がない場合は 0 を返します。例外がスローされた場合、値は返されません。

Examples

次のストアドプロシージャは、EXECUTE_AND_FETCH 関数を使用して、従業員の名前を使用して 1 人の従業員を取得します。従業員が見つからない場合、または同じ名前の従業員が複数いる場合、例外がスローされます。

```
CREATE OR REPLACE PROCEDURE select_by_name (
    p_ename          emp.ename%TYPE
)
IS
    curid            INTEGER;
    v_empno          emp.empno%TYPE;
    v_hiredate       emp.hiredate%TYPE;
    v_sal            emp.sal%TYPE;
    v_comm           emp.comm%TYPE;
    v_dname          dept.dname%TYPE;
    v_disp_date      VARCHAR2(10);
    v_sql            VARCHAR2(120) := 'SELECT empno, hiredate, sal, ' ||
```

(次のページに続く)

(前のページからの続き)

```

        'NVL(comm, 0), dname ' ||
        'FROM emp e, dept d ' ||
        'WHERE ename = :p_ename ' ||
        'AND e.deptno = d.deptno';

    v_status          INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(curid,v_sql,DBMS_SQL.native);
    DBMS_SQL.BIND_VARIABLE(curid,':p_ename',UPPER(p_ename));
    DBMS_SQL.DEFINE_COLUMN(curid,1,v_empno);
    DBMS_SQL.DEFINE_COLUMN(curid,2,v_hiredate);
    DBMS_SQL.DEFINE_COLUMN(curid,3,v_sal);
    DBMS_SQL.DEFINE_COLUMN(curid,4,v_comm);
    DBMS_SQL.DEFINE_COLUMN(curid,5,v_dname,14);
    v_status := DBMS_SQL.EXECUTE_AND_FETCH(curid,TRUE);
    DBMS_SQL.COLUMN_VALUE(curid,1,v_empno);
    DBMS_SQL.COLUMN_VALUE(curid,2,v_hiredate);
    DBMS_SQL.COLUMN_VALUE(curid,3,v_sal);
    DBMS_SQL.COLUMN_VALUE(curid,4,v_comm);
    DBMS_SQL.COLUMN_VALUE(curid,5,v_dname);
    v_disp_date := TO_CHAR(v_hiredate, 'MM/DD/YYYY');
    DBMS_OUTPUT.PUT_LINE('Number      : ' || v_empno);
    DBMS_OUTPUT.PUT_LINE('Name        : ' || UPPER(p_ename));
    DBMS_OUTPUT.PUT_LINE('Hire Date   : ' || v_disp_date);
    DBMS_OUTPUT.PUT_LINE('Salary      : ' || v_sal);
    DBMS_OUTPUT.PUT_LINE('Commission: ' || v_comm);
    DBMS_OUTPUT.PUT_LINE('Department: ' || v_dname);
    DBMS_SQL.CLOSE_CURSOR(curid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employee ' || p_ename || ' not found');
        DBMS_SQL.CLOSE_CURSOR(curid);
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Too many employees named, ' ||
            p_ename || ', found');
        DBMS_SQL.CLOSE_CURSOR(curid);
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('The following is SQLERRM:');
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
        DBMS_OUTPUT.PUT_LINE('The following is SQLCODE:');
        DBMS_OUTPUT.PUT_LINE(SQLCODE);
        DBMS_SQL.CLOSE_CURSOR(curid);
END;

EXEC select_by_name('MARTIN')

Number      : 7654
Name        : MARTIN

```

(次のページに続く)

(前のページからの続き)

```
Hire Date : 09/28/1981
Salary    : 1250
Commission: 1400
Department: SALES
```

3.17.16 FETCH_ROWS

FETCH_ROWS 関数は、カーソルから行を取得します。

```
<status> INTEGER FETCH_ROWS(<c> INTEGER)
```

Parameters

c

行をフェッチするカーソルのカーソル ID。

status

行が正常に取得された場合は 1、取得する行がもうない場合は 0 を返します。

Examples

次の例では、emp テーブルから行をフェッチし、結果を表示します。

```
DECLARE
    curid          INTEGER;
    v_empno        NUMBER(4);
    v_ename        VARCHAR2(10);
    v_hiredate     DATE;
    v_sal          NUMBER(7,2);
    v_comm         NUMBER(7,2);
    v_sql          VARCHAR2(50) := 'SELECT empno, ename, hiredate, sal, ' ||
                                'comm FROM emp';
    v_status       INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
    DBMS_SQL.DEFINE_COLUMN(curid, 1, v_empno);
    DBMS_SQL.DEFINE_COLUMN(curid, 2, v_ename, 10);
    DBMS_SQL.DEFINE_COLUMN(curid, 3, v_hiredate);
    DBMS_SQL.DEFINE_COLUMN(curid, 4, v_sal);
    DBMS_SQL.DEFINE_COLUMN(curid, 5, v_comm);

    v_status := DBMS_SQL.EXECUTE(curid);
    DBMS_OUTPUT.PUT_LINE('EMPNO   ENAME      HIREDATE    SAL      COMM');
```

(次のページに続く)

(前のページからの続き)

```

DBMS_OUTPUT.PUT_LINE('-----      -----      -----      -----      ' ||
'-----');
LOOP
    v_status := DBMS_SQL.FETCH_ROWS(curid);
    EXIT WHEN v_status = 0;
    DBMS_SQL.COLUMN_VALUE(curid,1,v_empno);
    DBMS_SQL.COLUMN_VALUE(curid,2,v_ename);
    DBMS_SQL.COLUMN_VALUE(curid,3,v_hiredate);
    DBMS_SQL.COLUMN_VALUE(curid,4,v_sal);
    DBMS_SQL.COLUMN_VALUE(curid,4,v_sal);
    DBMS_SQL.COLUMN_VALUE(curid,5,v_comm);
    DBMS_OUTPUT.PUT_LINE(v_empno || ' ' || RPAD(v_ename,10) || ' ' ||
        TO_CHAR(v_hiredate,'yyyy-mm-dd') || ' ' ||
        TO_CHAR(v_sal,'9,999.99') || ' ' ||
        TO_CHAR(NVL(v_comm,0),'9,999.99'));
END LOOP;
DBMS_SQL.CLOSE_CURSOR(curid);
END;
```

EMPNO	ENAME	HIREDATE	SAL	COMM
7369	SMITH	1980-12-17	800.00	.00
7499	ALLEN	1981-02-20	1,600.00	300.00
7521	WARD	1981-02-22	1,250.00	500.00
7566	JONES	1981-04-02	2,975.00	.00
7654	MARTIN	1981-09-28	1,250.00	1,400.00
7698	BLAKE	1981-05-01	2,850.00	.00
7782	CLARK	1981-06-09	2,450.00	.00
7788	SCOTT	1987-04-19	3,000.00	.00
7839	KING	1981-11-17	5,000.00	.00
7844	TURNER	1981-09-08	1,500.00	.00
7876	ADAMS	1987-05-23	1,100.00	.00
7900	JAMES	1981-12-03	950.00	.00
7902	FORD	1981-12-03	3,000.00	.00
7934	MILLER	1982-01-23	1,300.00	.00

3.17.17 IS_OPEN

IS_OPEN 関数は、指定されたカーソルが開いているかどうかをテストする機能を提供します。

```
<status> BOOLEAN IS_OPEN(<c> INTEGER)
```

Parameters

c

テストするカーソルのカーソル ID。

status

カーソルが開いている場合は TRUE に、カーソルが開いていない場合は FALSE に設定します。

3.17.18 LAST_ROW_COUNT

LAST_ROW_COUNT 関数は、現在フェッチされている行の数を返します。

```
<rowcnt> INTEGER LAST_ROW_COUNT
```

Parameters

rowcnt

これまでにフェッチされた行の数。

Examples

次の例では、LAST_ROW_COUNT 関数を使用して、クエリでフェッチされた行の総数を表示します。

```
DECLARE
    curid          sINTEGER;
    v_empno        NUMBER(4);
    v_ename        VARCHAR2(10);
    v_hiredate     DATE;
    v_sal          NUMBER(7,2);
    v_comm         NUMBER(7,2);
    v_sql          VARCHAR2(50) := 'SELECT empno, ename, hiredate, sal, ' ||
                                   'comm FROM emp';
    v_status       INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(curid,v_sql,DBMS_SQL.native);
    DBMS_SQL.DEFINE_COLUMN(curid,1,v_empno);
    DBMS_SQL.DEFINE_COLUMN(curid,2,v_ename,10);
    DBMS_SQL.DEFINE_COLUMN(curid,3,v_hiredate);
    DBMS_SQL.DEFINE_COLUMN(curid,4,v_sal);
    DBMS_SQL.DEFINE_COLUMN(curid,5,v_comm);

    v_status := DBMS_SQL.EXECUTE(curid);
    DBMS_OUTPUT.PUT_LINE('EMPNO   ENAME       HIREDATE    SAL        COMM');
    DBMS_OUTPUT.PUT_LINE('-----');
    LOOP
        v_status := DBMS_SQL.FETCH_ROWS(curid);
        EXIT WHEN v_status = 0;
```

(次のページに続く)

(前のページからの続き)

```

        DBMS_SQL.COLUMN_VALUE(curid,1,v_empno);
        DBMS_SQL.COLUMN_VALUE(curid,2,v_ename);
        DBMS_SQL.COLUMN_VALUE(curid,3,v_hiredate);
        DBMS_SQL.COLUMN_VALUE(curid,4,v_sal);
        DBMS_SQL.COLUMN_VALUE(curid,4,v_sal);
        DBMS_SQL.COLUMN_VALUE(curid,5,v_comm);
        DBMS_OUTPUT.PUT_LINE(v_empno || ' ' || RPAD(v_ename,10) || ' ' ||
            TO_CHAR(v_hiredate,'yyyy-mm-dd') || ' ' ||
            TO_CHAR(v_sal,'9,999.99') || ' ' ||
            TO_CHAR(NVL(v_comm,0),'9,999.99'));
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Number of rows: ' || DBMS_SQL.LAST_ROW_COUNT);
    DBMS_SQL.CLOSE_CURSOR(curid);
END;
```

EMPNO	ENAME	HIREDATE	SAL	COMM
7369	SMITH	1980-12-17	800.00	.00
7499	ALLEN	1981-02-20	1,600.00	300.00
7521	WARD	1981-02-22	1,250.00	500.00
7566	JONES	1981-04-02	2,975.00	.00
7654	MARTIN	1981-09-28	1,250.00	1,400.00
7698	BLAKE	1981-05-01	2,850.00	.00
7782	CLARK	1981-06-09	2,450.00	.00
7788	SCOTT	1987-04-19	3,000.00	.00
7839	KING	1981-11-17	5,000.00	.00
7844	TURNER	1981-09-08	1,500.00	.00
7876	ADAMS	1987-05-23	1,100.00	.00
7900	JAMES	1981-12-03	950.00	.00
7902	FORD	1981-12-03	3,000.00	.00
7934	MILLER	1982-01-23	1,300.00	.00

Number of rows: 14

3.17.19 LAST_ERROR_POSITION

LAST_ERROR_POSITION 関数は、エラーが発生した SQL ステートメントテキストのバイトオフセットを示す INTEGER 値を返します。SQL ステートメントの最初の文字のエラー位置は 1 です。

```
LAST_ERROR_POSITION RETURN INTEGER;
```

Examples

次の例は、LAST_ERROR_POSITION 関数でエラー位置を返す匿名ブロックを示しています。

```
DECLARE
    curid          INTEGER;
    sql_stmt       VARCHAR2(50) := 'SELECT empno FROM not_exist_table';
    v_position     INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(curid, sql_stmt, DBMS_SQL.native);
EXCEPTION WHEN OTHERS THEN
    v_position := DBMS_SQL.LAST_ERROR_POSITION;
    DBMS_OUTPUT.PUT_LINE('error position = ' || v_position);
    DBMS_SQL.CLOSE_CURSOR(curid);
END;

error position = 19
```

3.17.20 OPEN_CURSOR

OPEN_CURSOR 関数は新しいカーソルを作成します。動的 SQL ステートメントを解析および実行するには、カーソルを使用する必要があります。カーソルを開くと、同じまたは異なる SQL ステートメントで再利用できます。カーソルは、再利用するために閉じて再度開く必要はありません。

```
<c> INTEGER OPEN_CURSOR
```

Parameters

c

新しく作成されたカーソルに関連付けられたカーソル ID 番号。

Examples

次の例では、新しいカーソルを作成します。

```
DECLARE
    curid          INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
    .
    .
    .
END;
```


3.17.21 PARSE

PARSE プロシージャは、SQL コマンドまたは SPL ブロックを解析します。SQL コマンドが DDL コマンドの場合、すぐに実行され、EXECUTE 関数を実行する必要はありません。

```
PARSE (<c> INTEGER, <statement> VARCHAR2, <language_flag> INTEGER)
```

Parameters

c

開いているカーソルのカーソル ID。

statement

解析する SQL コマンドまたは SPL ブロック。SQL コマンドはセミコロンターミネータで終了してはなりませんが、SPL ブロックにはセミコロンターミネータが必要です。

language_flag

Oracle 構文との互換性のために提供される言語フラグ。DBMS_SQL.V6、DBMS_SQL.V7 または DBMS_SQL.native を使用します。このフラグは無視され、すべての構文は EDBAdvancedServer 形式であると想定されます。

Examples

次の匿名ブロックは、job という名前のテーブルを作成します。DDL ステートメントは PARSE プロシージャによってすぐに実行され、別の EXECUTE ステップを必要としないことに注意してください。

```
DECLARE
    curid          INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(curid, 'CREATE TABLE job (jobno NUMBER(3), ' ||
        'jname VARCHAR2(9))', DBMS_SQL.native);
    DBMS_SQL.CLOSE_CURSOR(curid);
END;
```

以下は、job テーブルに 2 つの行を挿入します。

```
DECLARE
    curid          INTEGER;
    v_sql          VARCHAR2(50);
    v_status       INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
    v_sql := 'INSERT INTO job VALUES (100, 'ANALYST')';
    DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
```

(次のページに続く)

(前のページからの続き)

```
v_status := DBMS_SQL.EXECUTE(curid);
DBMS_OUTPUT.PUT_LINE('Number of rows processed: ' || v_status);
v_sql := 'INSERT INTO job VALUES (200, ''CLERK'')';
DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
v_status := DBMS_SQL.EXECUTE(curid);
DBMS_OUTPUT.PUT_LINE('Number of rows processed: ' || v_status);
DBMS_SQL.CLOSE_CURSOR(curid);
END;

Number of rows processed: 1
Number of rows processed: 1
```

次の匿名ブロックは、DBMS_SQL パッケージを使用して、2 つの INSERT ステートメントを含むブロックを実行します。ブロックの終わりには終了セミコロンが含まれていますが、前の例では、個々の INSERT ステートメントには終了セミコロンがありません。

```
DECLARE
    curid          INTEGER;
    v_sql          VARCHAR2(100);
    v_status       INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
    v_sql := 'BEGIN ' ||
        'INSERT INTO job VALUES (300, ''MANAGER''); ' ||
        'INSERT INTO job VALUES (400, ''SALESMAN''); ' ||
        'END;';
    DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
    v_status := DBMS_SQL.EXECUTE(curid);
    DBMS_SQL.CLOSE_CURSOR(curid);
END;
```

3.18 DBMS_UTILITY

DBMS_UTILITY パッケージは、次のさまざまなユーティリティプログラムのサポートを提供します。

機能/手順	機能または手順	戻りタイプ	説明
<code>ANALYZE_DATABASE (method [, estimate_rows [, estimate_percent [, method_opt]])</code>	手続き	該当なし	データベーステーブルを分析します。
<code>ANALYZE_PART_OBJECT (schema, object_name [, object_type [, command_type [, command_opt [, sample_clause]]])</code>	手続き	該当なし	パーティションテーブルを分析します。
<code>ANALYZE_SCHEMA (schema, method [, estimate_rows [, estimate_percent [, method_opt]])</code>	手続き	該当なし	スキーマテーブルを分析します。
<code>CANONICALIZE (name, canon_name OUT, canon_len)</code>	手続き	該当なし	文字列を正規化します。たとえば、空白を取り除きます。
<code>COMMA_TO_TABLE (list, tablen OUT, tab OUT)</code>	手続き	該当なし	名前のコンマ区切りリストを名前の表に変換します。
<code>DB_VERSION (version OUT, compatibility OUT)</code>	手続き	該当なし	データベースのバージョンを取得します。
<code>EXEC_DDL_STATEMENT (parse_string)</code>	手続き	該当なし	DDL ステートメントを実行します。
<code>FORMAT_CALL_STACK</code>	機能	TEXT	現在の呼び出しスタックをフォーマットします。
<code>GET_CPU_TIME</code>	機能	NUMBER	現在の CPU 時間を取得します。
<code>GET_DEPENDENCY (type, schema, name)</code>	手続き	該当なし	指定されたオブジェクトに依存するオブジェクトを取得します。
<code>GET_HASH_VALUE (name, base, hash_size)</code>	機能	NUMBER	ハッシュ値を計算します。
<code>GET_PARAMETER_VALUE (parnam, intval OUT, strval OUT)</code>	手続き	BINARY_INTEGER	データベース初期化パラメータ設定を取得します。
<code>GET_TIME</code>	機能	NUMBER	現在の時刻を取得します。
<code>NAME_TOKENIZE (name, a OUT, b OUT, c OUT, dblink OUT, nextpos OUT)</code>	手続き	該当なし	指定された名前をそのコンポーネント部分に解析します。
<code>TABLE_TO_COMMA (tab, tablen OUT, list OUT)</code>	手続き	該当なし	名前の表をコンマ区切りリストに変換します。

AdvancedServer の DBMS_UTILITY の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表にリストされている関数と手順のみがサポートされています。

次の表に、DBMS_UTILITY パッケージで使用可能なパブリック変数を示します。

パブリック変数	データ型	価値	説明
inv_error_on_restrictions	PLS_INTEGER	1	INVALIDATE 手順で使用されます。
lname_array	TABLE		長い名前のリスト用。
uncl_array	TABLE		ユーザーと名前のリスト用。

3.18.1 LNAME_ARRAY

LNAME_ARRAY は、完全修飾名を含む長い名前のリストを保存するためのものです。

```
TYPE lname_array IS `TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;
```

3.18.2 UNCL_ARRAY

UNCL_ARRAY は、ユーザーと名前のリストを保存するためのものです。

```
TYPE uncl_array IS TABLE OF VARCHAR2(227) INDEX BY BINARY_INTEGER;
```

3.18.3 ANALYZE_DATABASE, ANALYZE_SCHEMA and ANALYZE_PART_OBJECT

ANALYZE_DATABASE(), ANALYZE_SCHEMA() and ANALYZE_PART_OBJECT() プロシージャは、データベース内のテーブルに関する統計を収集する機能を提供します。ANALYZE ステートメントを実行すると、Postgres はデータをテーブルでサンプリングし、分布統計を pg_statistics system テーブルに記録します。

ANALYZE_DATABASE, ANALYZE_SCHEMA と ANALYZE_PART_OBJECT は、主に処理されるテーブルの数が異なります。

- ANALYZE_DATABASE は、現在のデータベース内のすべてのスキーマのすべてのテーブルを分析します。
- ANALYZE_SCHEMA は、特定のスキーマ（現在のデータベース内）のすべてのテーブルを分析します。
- ANALYZE_PART_OBJECT は単一のテーブルを分析します。

ANALYZE コマンドの構文は次のとおりです。

```
ANALYZE_DATABASE(<method> VARCHAR2 [, <estimate_rows> NUMBER  
[, <estimate_percent> NUMBER [, <method_opt> VARCHAR2 ]]])  
  
ANALYZE_SCHEMA(<schema> VARCHAR2, <method> VARCHAR2  
[, <estimate_rows> NUMBER [, <estimate_percent> NUMBER  
[, <method_opt> VARCHAR2 ]]])  
  
ANALYZE_PART_OBJECT(<schema> VARCHAR2, <object_name> VARCHAR2  
[, <object_type> CHAR [, <command_type> CHAR  
[, <command_opt> VARCHAR2 [, <sample_clause> ]]]])
```

Parameters - ANALYZE_DATABASE および ANALYZE_SCHEMA

method

メソッドは、「12」プロシージャが pg_statistics テーブルに入力するか、pg_statistics テーブルからエントリを削除するかを決定します。DELETE の方法を指定した場合、「12」手順は pg_statistics から関連する行を削除します。「10」または「11」の方法を指定すると、「12」プロシージャはテーブル（または複数のテーブル）を分析し、pg_statistics に分布情報を記録します。「10」と「11」の間に違いはありません。どちらのメソッドも Postgres の「12」ステートメントを実行します。他のすべてのパラメータは検証されてから無視されます。

estimate_rows

推定統計のベースとなる行の数。メソッドが ESTIMATE の場合、estimate_rows または estimate_percent のいずれかを指定する必要があります。

この引数は無視されますが、互換性のために含まれています。

estimate_percent

推定統計の基礎となる行の割合。メソッドが ESTIMATE の場合、estimate_rows または estimate_percent のいずれかを指定する必要があります。

この引数は無視されますが、互換性のために含まれています。

method_opt

分析するオブジェクトタイプ。以下の任意の組み合わせ:

[FOR TABLE]

[FOR ALL [INDEXED] COLUMNS] [SIZE n]

[FOR ALL INDEXES]

この引数は無視されますが、互換性のために含まれています。

Parameters - ANALYZE_PART_OBJECT

schema

オブジェクトが分析されるスキーマの名前。

object_name

分析するパーティション化されたオブジェクトの名前。

object_type

分析するオブジェクトのタイプ。有効な値は、T-テーブル、I-インデックスです。

この引数は無視されますが、互換性のために含まれています。

command_type

実行する分析機能のタイプ。有効な値は次のとおりです。E - sample_clause 句の指定された行数または行の割合に基づいて推定統計を収集します。C - 正確な統計を計算します。または V - パーティションの構造と整合性を検証します。

この引数は無視されますが、互換性のために含まれています。

command_opt

command_type C または E の場合、次の任意の組み合わせを使用できます。

[FOR TABLE]

[FOR ALL COLUMNS]

[FOR ALL LOCAL INDEXES]

command_type V の場合、object_type が T の場合は CASCADE になります。

この引数は無視されますが、互換性のために含まれています。

sample_clause

command_type が E の場合、次の句が含まれており、推定の基になる行数または割合または行数を指定します。

SAMPLE n { ROWS | PERCENT }

この引数は無視されますが、互換性のために含まれています。

3.18.4 CANONICALIZE

CANONICALIZE プロシージャは、入力文字列に対して次の操作を実行します。

- 文字列が二重引用符で囲まれていない場合、正当な識別子の文字を使用していることを確認します。そうでない場合、例外がスローされます。文字列が二重引用符で囲まれている場合、すべての文字が許可されます。
- 文字列が二重引用符で囲まれておらず、ピリオドが含まれていない場合、すべてのアルファベット文字を大文字にし、先頭と末尾のスペースを削除します。
- 文字列が二重引用符で囲まれており、ピリオドが含まれていない場合は、二重引用符を取り除きます。
- 文字列にピリオドが含まれ、文字列の一部が二重引用符で囲まれていない場合、文字列の各部分を大文字にし、各部分を二重引用符で囲みます。
- 文字列にピリオドが含まれ、文字列の一部が二重引用符で囲まれている場合、二重引用符を含む二重引用符で囲まれた部分を変更せずに返し、二重引用符で囲まれていない二重引用符で囲まれていない部分を返します。


```
CANONICALIZE(<name> VARCHAR2, <canon_name> OUT VARCHAR2,
             <canon_len> BINARY_INTEGER)
```

Parameters

name

正規化される文字列。

canon_name

正規化された文字列。

canon_len

最初の文字から始まる正規化する name のバイト数。

Examples

次の手順は、入力パラメーターに CANONICALIZE 手順を適用し、結果を表示します。

```
CREATE OR REPLACE PROCEDURE canonicalize (
    p_name      VARCHAR2,
    p_length    BINARY_INTEGER DEFAULT 30
)
IS
    v_canon     VARCHAR2(100);
BEGIN
    DBMS_UTILITY.CANONICALIZE(p_name,v_canon,p_length);
    DBMS_OUTPUT.PUT_LINE('Canonicalized name ==>' || v_canon || '<==');
    DBMS_OUTPUT.PUT_LINE('Length: ' || LENGTH(v_canon));
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('SQLERRM: ' || SQLERRM);
        DBMS_OUTPUT.PUT_LINE('SQLCODE: ' || SQLCODE);
END;

EXEC canonicalize('Identifier')
Canonicalized name ==>IDENTIFIER<==
Length: 10

EXEC canonicalize('"Identifier"')
Canonicalized name ==>Identifier<==
Length: 10

EXEC canonicalize('"_+142% "')
Canonicalized name ==>_+142%<==
Length: 6
```

(次のページに続く)

(前のページからの続き)

```
EXEC canonicalize('abc.def.ghi')
Canonicalized name ==>"ABC"."DEF"."GHI"<==
Length: 17

EXEC canonicalize('"abc.def.ghi"')
Canonicalized name ==>abc.def.ghi<==
Length: 11

EXEC canonicalize('"abc".def."ghi"')
Canonicalized name ==>"abc"."DEF"."ghi"<==
Length: 17

EXEC canonicalize('"abc.def".ghi')
Canonicalized name ==>"abc.def"."GHI"<==
Length: 15
```

3.18.5 COMMA_TO_TABLE

COMMA_TO_TABLE プロシージャは、名前のカンマ区切りリストを名前のテーブルに変換します。リスト内の各エントリはテーブルエントリになります。名前は有効な識別子としてフォーマットする必要があります。

```
COMMA_TO_TABLE(<list> VARCHAR2, <tablen> OUT BINARY_INTEGER,
               <tab> OUT { LNAME_ARRAY | UNCL_ARRAY })
```

Parameters

list

名前のカンマ区切りリスト。

tablen

tab のエントリ数。

tab

list の個々の名前を含むテーブル。

LNAME_ARRAY

DBMS_UTILITY LNAME_ARRAY (*LNAME_ARRAY* セクションで説明)。

UNCL_ARRAY

DBMS_UTILITY UNCL_ARRAY (*UNCL_ARRAY* セクションで説明)。

Examples

次の手順では、COMMA_TO_TABLE 手順を使用して名前のリストをテーブルに変換します。その後、テーブルエントリが表示されます。

```
CREATE OR REPLACE PROCEDURE comma_to_table (
    p_list      VARCHAR2
)
IS
    r_lname     DBMS_UTILITY.LNAME_ARRAY;
    v_length    BINARY_INTEGER;
BEGIN
    DBMS_UTILITY.COMMA_TO_TABLE(p_list,v_length,r_lname);
    FOR i IN 1..v_length LOOP
        DBMS_OUTPUT.PUT_LINE(r_lname(i));
    END LOOP;
END;

EXEC comma_to_table('edb.dept, edb.emp, edb.jobhist')

edb.dept
edb.emp
edb.jobhist
```

3.18.6 DB_VERSION

DB_VERSION プロシージャは、データベースのバージョン番号を返します。

```
DB_VERSION(<version> OUT VARCHAR2, <compatibility> OUT VARCHAR2)
```

Parameters

version

データベースのバージョン番号。

compatibility

データベースの互換性設定。(その意味に関して実装定義されること。)

Examples

次の匿名ブロックは、データベースのバージョン情報を表示します。

```
DECLARE
    v_version    VARCHAR2(150);
    v_compat     VARCHAR2(150);
BEGIN
    DBMS_UTILITY.DB_VERSION(v_version,v_compat);
```

(次のページに続く)

(前のページからの続き)

```

DBMS_OUTPUT.PUT_LINE('Version: ' || v_version);
DBMS_OUTPUT.PUT_LINE('Compatibility: ' || v_compat);
END;

Version: EnterpriseDB 10.0.0 on i686-pc-linux-gnu, compiled by GCC gcc
(GCC) 4.1.2 20080704 (Red Hat 4.1.2-48), 32-bit
Compatibility: EnterpriseDB 10.0.0 on i686-pc-linux-gnu, compiled by GCC
gcc (GCC) 4.1.220080704 (Red Hat 4.1.2-48), 32-bit

```

3.18.7 EXEC_DDL_STATEMENT

EXEC_DDL_STATEMENT は、DDL コマンドを実行する機能を提供します。

```
EXEC_DDL_STATEMENT(<parse_string> VARCHAR2)
```

Parameters

parse_string

実行される DDL コマンド。

Examples

次の匿名ブロックは job テーブルを作成します。

```

BEGIN
  DBMS_UTILITY.EXEC_DDL_STATEMENT(
    'CREATE TABLE job (' ||
      'jobno NUMBER(3),' ||
      'jname VARCHAR2(9)) '
  );
END;

```

parse_string に有効な DDL ステートメントが含まれていない場合、AdvancedServer は次のエラーを返します。

```

edb=#^^c2^^a0 exec dbms_utility.exec_ddl_statement('select rownum from dual');
ERROR:^^c2^^a0 EDB-20001: 'parse_string' must be a valid DDL statement

```

この場合、AdvancedServer の動作は Oracle とは異なります。Oracle は、文句なしに無効な parse_string を受け入れます。

3.18.8 FORMAT_CALL_STACK

FORMAT_CALL_STACK 関数は、現在の呼び出しスタックのフォーマットされたコンテンツを返します。

```
DBMS_UTILITY.FORMAT_CALL_STACK
return VARCHAR2
```

この関数をストアドプロシージャ、関数、またはパッケージで使用して、現在の呼び出しスタックを読み取り可能な形式で返すことができます。この関数は、デバッグの目的に役立ちます。

3.18.9 GET_CPU_TIME

GET_CPU_TIME 関数は、任意の時点からの CPU 時間を 100 分の 1 秒単位で返します。

```
<cputime> NUMBER GET_CPU_TIME
```

Parameters

cputime

CPU 時間の 100 分の 1 秒の数。

Examples

次の SELECT コマンドは、現在の CPU 時間を取得します。これは、603 の 100 分の 1 秒または.0603 秒です。

```
SELECT DBMS_UTILITY.GET_CPU_TIME FROM DUAL;

get_cpu_time
-----
          603
```

3.18.10 GET_DEPENDENCY

GET_DEPENDENCY プロシージャは、指定されたオブジェクトに依存するオブジェクトをリストする機能を提供します。GET_DEPENDENCY は、関数またはプロシージャの依存関係を示しません。

```
GET_DEPENDENCY(<type> VARCHAR2, <schema> VARCHAR2,
               <name> VARCHAR2)
```

Parameters

type

name のオブジェクトタイプ。有効な値は INDEX, PACKAGE, PACKAGE BODY, SEQUENCE, TABLE, TRIGGER, TYPE および VIEW です。

schema

name が存在するスキーマの名前。

name

依存関係を取得するオブジェクトの名前。

Examples

次の匿名ブロックは、EMP テーブルの依存関係を見つけます。

```
BEGIN
    DBMS_UTILITY.GET_DEPENDENCY('TABLE','public','EMP');
END;

DEPENDENCIES ON public.EMP
-----
*TABLE public.EMP()
*  CONSTRAINT c public.emp()
*  CONSTRAINT f public.emp()
*  CONSTRAINT p public.emp()
*  TYPE public.emp()
*  CONSTRAINT c public.emp()
*  CONSTRAINT f public.jobhist()
*  VIEW .empname_view()
```

3.18.11 GET_HASH_VALUE

GET_HASH_VALUE 関数は、特定の文字列のハッシュ値を計算する機能を提供します。

```
<hash> NUMBER GET_HASH_VALUE(<name> VARCHAR2, <base> NUMBER,
    <hash_size> NUMBER)
```

Parameters

name

ハッシュ値を計算する対象の文字列。

base

ハッシュ値が生成される開始値。

hash_size

目的のハッシュテーブルのハッシュ値の数。

hash

生成されたハッシュ値。

Examples

次の匿名ブロックは、emp テーブルの ename 列を使用してハッシュ値のテーブルを作成し、ハッシュ値とともにキーを表示します。ハッシュ値は 100 から始まり、最大 1024 の個別の値があります。

```
DECLARE
    v_hash          NUMBER;
    TYPE hash_tab IS TABLE OF NUMBER INDEX BY VARCHAR2(10);
    r_hash          HASH_TAB;
    CURSOR emp_cur IS SELECT ename FROM emp;
BEGIN
    FOR r_emp IN emp_cur LOOP
        r_hash(r_emp.ename.. code-block:: text) :=
            DBMS_UTILITY.GET_HASH_VALUE(r_emp.ename,100,1024);
    END LOOP;
    FOR r_emp IN emp_cur LOOP
        DBMS_OUTPUT.PUT_LINE(RPAD(r_emp.ename,10) || ' ' ||
            r_hash(r_emp.ename));
    END LOOP;
END;
```

SMITH	377
ALLEN	740
WARD	718.. code-block:: text
JONES	131
MARTIN	176
BLAKE	568
CLARK	621
SCOTT	1097
KING	235
TURNER	850
ADAMS	156
JAMES	942
FORD	775
MILLER	148

3.18.12 GET_PARAMETER_VALUE

GET_PARAMETER_VALUE プロシージャは、データベース初期化パラメータ設定を取得する機能を提供します。

```
<status> BINARY_INTEGER GET_PARAMETER_VALUE (<parnam> VARCHAR2,
      <intval> OUT INTEGER, <strval> OUT VARCHAR2)``
```

Parameters

parnam

値が返されるパラメーターの名前。パラメータは pg_settings システムビューに一覧表示されます。

intval

整数パラメータの値または strval の長さ。

strval

文字列パラメーターの値。

status

パラメータ値が INTEGER または BOOLEAN の場合、0 を返します。パラメータ値が文字列の場合、1 を返します。

Examples

次の匿名ブロックは、2 つの初期化パラメーターの値を示しています。

```
DECLARE
    v_intval      INTEGER;
    v_strval      VARCHAR2(80);
BEGIN
    DBMS_UTILITY.GET_PARAMETER_VALUE('max_fsm_pages', v_intval, v_strval);
    DBMS_OUTPUT.PUT_LINE('max_fsm_pages' || ': ' || v_intval);
    DBMS_UTILITY.GET_PARAMETER_VALUE('client_encoding', v_intval, v_strval);
    DBMS_OUTPUT.PUT_LINE('client_encoding' || ': ' || v_strval);
END;

max_fsm_pages: 72625
client_encoding: SQL_ASCII
```


3.18.13 GET_TIME

GET_TIME 関数は、現在の時刻を 100 分の 1 秒単位で返す機能を提供します。

```
<time> NUMBER GET_TIME
```

Parameters

time

プログラムが開始されてからの 100 分の 1 秒の数。

Examples

次の例は、GET_TIME 関数の呼び出しを示しています。

```
SELECT DBMS_UTILITY.GET_TIME FROM DUAL;

get_time
-----
1555860

SELECT DBMS_UTILITY.GET_TIME FROM DUAL;

get_time
-----
1556037
```

3.18.14 NAME_TOKENIZE

NAME_TOKENIZE プロシージャは、名前をそのコンポーネント部分に解析します。二重引用符のない名前は大文字になります。二重引用符は、二重引用符で名前から削除されます。

```
NAME_TOKENIZE(<name> VARCHAR2, <a> OUT VARCHAR2,
              <b> OUT VARCHAR2, <c> OUT VARCHAR2, <dblink> OUT VARCHAR2,
              <nextpos> OUT BINARY_INTEGER)
```

Parameters

name

次の形式の名前を含む文字列:

```
a[.b[.c]][@dblink ]
```

a

左端のコンポーネントを返します。

b

2 番目のコンポーネントがあれば、それを返します。

c

3 番目のコンポーネントがあれば、それを返します。

dblink

データベースのリンク名を返します。

nextpos

名前で解析された最後の文字の位置。

Examples

次のストアードプロシージャを使用して、さまざまな名前の NAME_TOKENIZE プロシージャの返されたパラメータ値を表示します。

```
CREATE OR REPLACE PROCEDURE name_tokenize (  
    p_name          VARCHAR2  
)  
IS  
    v_a             VARCHAR2(30);  
    v_b             VARCHAR2(30);  
    v_c             VARCHAR2(30);  
    v_dblink        VARCHAR2(30);  
    v_nextpos       BINARY_INTEGER;  
BEGIN  
    DBMS_UTILITY.NAME_TOKENIZE(p_name,v_a,v_b,v_c,v_dblink,v_nextpos);  
    DBMS_OUTPUT.PUT_LINE('name      : ' || p_name);  
    DBMS_OUTPUT.PUT_LINE('a        : ' || v_a);  
    DBMS_OUTPUT.PUT_LINE('b        : ' || v_b);  
    DBMS_OUTPUT.PUT_LINE('c        : ' || v_c);  
    DBMS_OUTPUT.PUT_LINE('dblink   : ' || v_dblink);  
    DBMS_OUTPUT.PUT_LINE('nextpos: ' || v_nextpos);  
END;
```

名前 emp をトークン化する:

```
BEGIN  
    name_tokenize('emp');  
END;  
  
name      : emp  
a         : EMP  
b         :  
c         :
```

(次のページに続く)

(前のページからの続き)

```
dblink :
nextpos: 3
```

名前 edb.list_emp のトークン化:

```
BEGIN
    name_tokenize('edb.list_emp');
END;

name      : edb.list_emp
a         : EDB
b         : LIST_EMP
c         :
dblink    :
nextpos: 12
```

名前のトークン化、 /"edb"/."Emp_Admin"/.update_emp_sal:

```
BEGIN
    name_tokenize('"edb"."Emp_Admin".update_emp_sal');
END;

name      : "edb"."Emp_Admin".update_emp_sal
a         : edb
b         : Emp_Admin
c         : UPDATE_EMP_SAL
dblink    :
nextpos: 32
```

名前 edb.emp@edb_dblink をトークン化する:

```
BEGIN
    name_tokenize('edb.emp@edb_dblink');
END;

name      : edb.emp@edb_dblink
a         : EDB
b         : EMP
c         :
dblink    : EDB_DBLINK
nextpos: 18
```

3.18.15 TABLE_TO_COMMA

TABLE_TO_COMMA プロシージャは、名前のテーブルをコンマ区切りの名前のリストに変換します。各テーブルエントリはリストエントリになります。名前は有効な識別子としてフォーマットする必要があります。

```
TABLE_TO_COMMA(<tab> { LNAME_ARRAY | UNCL_ARRAY },
               <tablen> OUT BINARY_INTEGER, <list> OUT VARCHAR2)
```

Parameters

tab

名前を含むテーブル。

LNAME_ARRAY

DBMS_UTILITY LNAME_ARRAY (*LNAMEARRAY* セクションで説明されています)。

UNCL_ARRAY

DBMS_UTILITY UNCL_ARRAY (*UNCL_ARRAY* セクションで説明)。

tablen

list のエントリ数。

list

tab からの名前のカンマ区切りリスト。

Examples

次の例では、最初に COMMA_TO_TABLE プロシージャを使用して、コンマ区切りリストをテーブルに変換します。TABLE_TO_COMMA プロシージャは、テーブルをコンマ区切りリストに変換して表示します。

```
CREATE OR REPLACE PROCEDURE table_to_comma (
    p_list      VARCHAR2
)
IS
    r_lname     DBMS_UTILITY.LNAME_ARRAY;
    v_length    BINARY_INTEGER;
    v_listlen   BINARY_INTEGER;
    v_list      VARCHAR2(80);
BEGIN
    DBMS_UTILITY.COMMA_TO_TABLE(p_list, v_length, r_lname);
    DBMS_OUTPUT.PUT_LINE('Table Entries');
    DBMS_OUTPUT.PUT_LINE('-----');
    FOR i IN 1..v_length LOOP
        DBMS_OUTPUT.PUT_LINE(r_lname(i));
    END LOOP;
```

(次のページに続く)

(前のページからの続き)

```
DBMS_OUTPUT.PUT_LINE('-----');
DBMS_UTILITY.TABLE_TO_COMMA(r_lname,v_listlen,v_list);
DBMS_OUTPUT.PUT_LINE('Comma-Delimited List: ' || v_list);
END;

EXEC table_to_comma('edb.dept, edb.emp, edb.jobhist')

Table Entries
-----
edb.dept
edb.emp
edb.jobhist
-----
Comma-Delimited List: edb.dept, edb.emp, edb.jobhist
```

3.19 UTL_ENCODE

UTL_ENCODE パッケージは、データをエンコードおよびデコードする方法を提供します。AdvancedServer は、次の機能と手順をサポートしています。

機能/手順	戻りタイプ	説明
BASE64_DECODE(r)	RAW	BASE64_DECODE 関数を使用して、Base64 エンコード文字列を元の RAW 値に変換します。
BASE64_ENCODE(r)	RAW	BASE64_ENCODE 関数を使用して、RAW 文字列をエンコードされた Base64 値に変換します。
BASE64_ENCODE(loid)	TEXT	BASE64_ENCODE 関数を使用して、TEXT 文字列をエンコードされた Base64 値に変換します。
MIMEHEADER_DECODE(buf)	VARCHAR2	MIMEHEADER_DECODE 関数を使用して、エンコードされた MIMEHEADER 形式の文字列を元の値に変換します。
MIMEHEADER_ENCODE(buf, encode_charset, encoding)	VARCHAR2	MIMEHEADER_ENCODE 関数を使用して、MIMEHEADER 形式の文字列を変換およびエンコードします。
QUOTED_PRINTABLE_DECODE(r)	RAW	QUOTED_PRINTABLE_DECODE 関数を使用して、エンコードされた文字列を RAW 値に変換します。
QUOTED_PRINTABLE_ENCODE(r)	RAW	QUOTED_PRINTABLE_ENCODE 関数を使用して、入力文字列を引用符付きの印刷可能な RAW value に変換します。
TEXT_DECODE(buf, encode_charset, encoding)	VARCHAR2	TEXT_DECODE 関数を使用して、TEXT_ENCODE でエンコードされた文字列をデコードします。
TEXT_ENCODE(buf, encode_charset, encoding)	VARCHAR2	TEXT_ENCODE 関数を使用して、文字列をユーザー指定の文字セットに変換してから、文字列をエンコードします。
UUDECODE(r)	RAW	UUDECODE 関数を使用して、uuencode エンコード文字列を RAW 値に変換します。
UUENCODE(r, type, filename, permission)	RAW	UUENCODE 関数を使用して、RAW 文字列をエンコードされた uuencode 値に変換します。

3.19.1 BASE64_DECODE

BASE64_DECODE 関数を使用して、Base64 でエンコードされた文字列を、BASE64_ENCODE で元々エンコードされた元の値に変換します。署名は次のとおりです。

```
BASE64_DECODE (<r> IN RAW)
```

この関数は RAW 値を返します。

Parameters

r

r は、RAW 形式に変換される Base64 エンコードデータを含む文字列です。

Examples

次の例を実行する前に、コマンドを呼び出します。

```
SET bytea_output = escape;
```

このコマンドは、印刷できない文字をエスケープし、BYTEA または RAW の値を読み取り可能な形式で画面に表示するようにサーバーに指示します。詳細については、次の Web サイトで入手可能な Postgres コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/datatype-binary.html>

次の例では、最初にテキスト abc: を含む文字列をエンコード (BASE64_ENCODE を使用) してからデコード (BASE64_DECODE を使用) します。

```
edb=# SELECT UTL_ENCODE.BASE64_ENCODE(CAST ('abc' AS RAW));
 base64_encode
-----
 YWJj
(1 row)

edb=# SELECT UTL_ENCODE.BASE64_DECODE(CAST ('YWJj' AS RAW));
 base64_decode
-----
 abc
(1 row)
```

3.19.2 BASE64_ENCODE

「RFC4648 で説明されているように」「64」形式の文字列を変換およびエンコードするには、BASE64_ENCODE 関数を使用します。この機能は、UTL_SMTP パッケージを使用して送信する MIME メールを作成するときに役立ちます。BASE64_ENCODE 関数には 2 つのシグネチャがあります。

```
BASE64_ENCODE (<r> IN RAW)
```

そして

```
BASE64_ENCODE (<loid> IN OID)
```

この関数は、RAW 値または OID を返します。

Parameters

r

r は、Base64 に変換される RAW 文字列を指定します。

loid

loid は、Base64 に変換されるラージオブジェクトのオブジェクト ID を指定します。

Examples

次の例を実行する前に、コマンドを呼び出します。

```
SET bytea_output = escape;
```

このコマンドは、印刷できない文字をエスケープし、BYTEA または RAW の値を読み取り可能な形式で画面に表示するようにサーバーに指示します。詳細については、次の Web サイトで入手可能な Postgres コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/datatype-binary.html>

次の例では、最初にテキスト abc: を含む文字列をエンコード (BASE64_ENCODE を使用) してからデコード (BASE64_DECODE を使用) します。

```
edb=# SELECT UTL_ENCODE.BASE64_ENCODE(CAST ('abc' AS RAW));
 base64_encode
-----
 YWJj
(1 row)

edb=# SELECT UTL_ENCODE.BASE64_DECODE(CAST ('YWJj' AS RAW));
 base64_decode
-----
```

(次のページに続く)

(前のページからの続き)

```
abc
(1 row)
```

3.19.3 MIMEHEADER_DECODE

MIMEHEADER_DECODE 関数を使用して、MIMEHEADER_ENCODE 関数によってエンコードされた値をデコードします。署名は次のとおりです。

```
MIMEHEADER_DECODE(<buf> IN VARCHAR2)
```

この関数は VARCHAR2 値を返します。

Parameters

buf

buf には、デコードされる MIMEHEADER_ENCODE) でエンコードされた値が含まれます。

Examples

次の例では、MIMEHEADER_ENCODE および MIMEHEADER_DECODE 関数を使用して、最初に文字列をエンコードし、次にデコードします。

```
edb=# SELECT UTL_ENCODE.MIMEHEADER_ENCODE('What is the date?') FROM DUAL;
      mimeheader_encode
-----
=?UTF8?Q?What is the date??=
(1 row)

edb=# SELECT UTL_ENCODE.MIMEHEADER_DECODE('=?UTF8?Q?What is the date??=')
FROM DUAL;
      mimeheader_decode
-----
      What is the date?
(1 row)
```

3.19.4 MIMEHEADER_ENCODE

MIMEHEADER_ENCODE 関数を使用して文字列を MIME ヘッダー形式に変換し、文字列をエンコードします。署名は次のとおりです。

```
MIMEHEADER_ENCODE(<buf> IN VARCHAR2, <encode_charset> IN VARCHAR2
DEFAULT NULL, <encoding> IN INTEGER DEFAULT NULL)
```

この関数は VARCHAR2 値を返します。

Parameters

buf

buf には、フォーマットおよびエンコードされる文字列が含まれます。文字列は VARCHAR2 値です。

encode_charset

encode_charset は、文字列がフォーマットおよびエンコードされる前に変換される文字セットを指定します。デフォルト値は NULL です。

encoding

encoding は、文字列をエンコードするときに使用されるエンコードタイプを指定します。以下を指定できます。

- quoted-printable エンコードを有効にするには Q。値を指定しない場合、MIMEHEADER_ENCODE は quoted-printable エンコードを使用します。
- base-64 エンコードを有効にするには B。

Examples

次の例では、MIMEHEADER_ENCODE および MIMEHEADER_DECODE 関数を使用して、最初に文字列をエンコードし、次にデコードします。

```
edb=# SELECT UTL_ENCODE.MIMEHEADER_ENCODE('What is the date?') FROM DUAL;
      mimeheader_encode
-----
=?UTF8?Q?What is the date??=
(1 row)

edb=# SELECT UTL_ENCODE.MIMEHEADER_DECODE('=?UTF8?Q?What is the date??=')
FROM DUAL;
      mimeheader_decode
-----
      What is the date?
(1 row)
```

3.19.5 QUOTED_PRINTABLE_DECODE

QUOTED_PRINTABLE_DECODE 関数を使用して、エンコードされた quoted-printable 文字列をデコードされた RAW 文字列に変換します。

署名は次のとおりです。

```
QUOTED_PRINTABLE_DECODE(<r> IN RAW)
```

この関数は RAW 値を返します。

Parameters

r

r には、デコードされるエンコードされた文字列が含まれます。文字列は QUOTED_PRINTABLE_ENCODE でエンコードされた RAW 値です。

Examples

次の例を実行する前に、コマンドを呼び出します。

```
SET bytea_output = escape;
```

このコマンドは、印刷できない文字をエスケープし、BYTEA または RAW の値を読み取り可能な形式で画面に表示するようにサーバーに指示します。詳細については、次の Web サイトで入手可能な Postgres コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/datatype-binary.html>

次の例では、最初に文字列をエンコードしてからデコードします。

```
edb=# SELECT UTL_ENCODE.QUOTED_PRINTABLE_ENCODE('E=mc2') FROM DUAL;
quoted_printable_encode
-----
E=3Dmc2
(1 row)

edb=# SELECT UTL_ENCODE.QUOTED_PRINTABLE_DECODE('E=3Dmc2') FROM DUAL;
quoted_printable_decode
-----
E=mc2
(1 row)
```

3.19.6 QUOTED_PRINTABLE_ENCODE

QUOTED_PRINTABLE_ENCODE 関数を使用して、文字列を quoted-printable 形式で変換およびエンコードします。署名は次のとおりです。

```
QUOTED_PRINTABLE_ENCODE(<r> IN RAW)
```

この関数は RAW 値を返します。

Parameters

r

r には、quoted-printable 形式でエンコードされる文字列（RAW 値）が含まれます。

Examples

次の例を実行する前に、コマンドを呼び出します。

```
SET bytea_output = escape;
```

このコマンドは、印刷できない文字をエスケープし、BYTEA または RAW の値を読み取り可能な形式で画面に表示するようにサーバーに指示します。詳細については、次の Web サイトで入手可能な Postgres コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/datatype-binary.html>

次の例では、最初に文字列をエンコードしてからデコードします。

```
edb=# SELECT UTL_ENCODE.QUOTED_PRINTABLE_ENCODE('E=mc2') FROM DUAL;
quoted_printable_encode
-----
E=3Dmc2
(1 row)

edb=# SELECT UTL_ENCODE.QUOTED_PRINTABLE_DECODE('E=3Dmc2') FROM DUAL;
quoted_printable_decode
-----
E=mc2
(1 row)
```

3.19.7 TEXT_DECODE

TEXT_DECODE 関数を使用して、エンコードされた文字列を TEXT_ENCODE 関数によって元々エンコードされた VARCHAR2 値に変換およびデコードします。署名は次のとおりです。

```
TEXT_DECODE(<buf> IN VARCHAR2, <encode_charset> IN VARCHAR2 DEFAULT
NULL, <encoding> IN PLS_INTEGER DEFAULT NULL)
```

この関数は VARCHAR2 値を返します。

Parameters

buf

buf には、TEXT_ENCODE でエンコードされた元の値に変換されるエンコードされた文字列が含まれます。

encode_charset

encode_charset は、エンコード前に文字列が変換される文字セットを指定します。デフォルト値は NULL です。

encoding

encoding は TEXT_DECODE で使用されるエンコードタイプを指定します。指定してください:

- UTL_ENCODE.BASE64 は base-64 エンコーディングを指定します。
- UTL_ENCODE.QUOTED_PRINTABLE は、引用符付きの印刷可能なエンコードを指定します。これがデフォルトです。

Examples

次の例では、TEXT_ENCODE および TEXT_DECODE 関数を使用して、最初に文字列をエンコードし、次にデコードします。

```
edb=# SELECT UTL_ENCODE.TEXT_ENCODE('What is the date?', 'BIG5',
UTL_ENCODE.BASE64) FROM DUAL;
      text_encode
-----
V2hhdCBpcyB0aGUgZGF0ZT8=
(1 row)

edb=# SELECT UTL_ENCODE.TEXT_DECODE('V2hhdCBpcyB0aGUgZGF0ZT8=', 'BIG5',
UTL_ENCODE.BASE64) FROM DUAL;
      text_decode
-----
What is the date?
(1 row)
```

3.19.8 TEXT_ENCODE

TEXT_ENCODE 関数を使用して、文字列をユーザー指定の文字セットに変換してから、文字列をエンコードします。署名は次のとおりです。

```
TEXT_DECODE(<buf> IN VARCHAR2, <encode_charset> IN VARCHAR2 DEFAULT
NULL, <encoding> IN PLS_INTEGER DEFAULT NULL)
```

この関数は VARCHAR2 値を返します。

Parameters

buf

buf には、指定された文字セットに変換され、TEXT_ENCODE でエンコードされるエンコードされた文字列が含まれます。

encode_charset

encode_charset は、エンコード前に値が変換される文字セットを指定します。デフォルト値は NULL です。

encoding

encoding は TEXT_ENCODE で使用されるエンコードタイプを指定します。指定してください:

- UTL_ENCODE.BASE64 は base-64 エンコーディングを指定します。
- UTL_ENCODE.QUOTED_PRINTABLE は、引用符付きの印刷可能なエンコードを指定します。これがデフォルトです。

Examples

次の例では、TEXT_ENCODE および TEXT_DECODE 関数を使用して、最初に文字列をエンコードし、次にデコードします。

```
edb=# SELECT UTL_ENCODE.TEXT_ENCODE('What is the date?', 'BIG5',
UTL_ENCODE.BASE64) FROM DUAL;
          text_encode
-----
V2hhdBpCyB0aGUgZGF0ZT8=
(1 row)

edb=# SELECT UTL_ENCODE.TEXT_DECODE('V2hhdBpCyB0aGUgZGF0ZT8=', 'BIG5',
UTL_ENCODE.BASE64) FROM DUAL;
          text_decode
-----
What is the date?
(1 row)
```

3.19.9 UUDECODE

UUDECODE 関数を使用して、`uuencode` エンコード文字列を UUENCODE 関数によって元々エンコードされた RAW 値に変換およびデコードします。署名は次のとおりです。

```
UUDECODE (<r> IN RAW)
```

この関数は RAW 値を返します。

AdvancedServer の UUDECODE 関数を使用して、`UTL_ENCODE.UUENCODE` 関数の Oracle 実装によって作成された `uuencoded` データをデコードする場合、Advanced を呼び出す前に、まず AdvancedServer 構成パラメーター `utl_encode.uudecode_redwood` を TRUE に設定する必要があります Oracle が作成したデータに対するサーバー UUDECODE 機能。(たとえば、`uuencode` されたデータを含む Oracle テーブルを AdvancedServer データベースに移行した場合、この状況が発生する可能性があります。)

Oracle バージョンの UUENCODE 関数によって作成された `uuencode` されたデータは、AdvancedServer の UUENCODE 関数によって作成された `uuencode` されたデータとは異なる形式になります。その結果、構成パラメーター `utl_encode.uudecode_redwood` が TRUE に設定されていない限り、Oracle `uuencoded` データで AdvancedServer UUDECODE 機能を使用しようとするとエラーが発生します。

ただし、AdvancedServer の UUENCODE 機能によって作成された `uuencode` されたデータに対して AdvancedServer の UUDECODE 機能を使用している場合は、`utl_encode.uudecode_redwood` をデフォルト設定の FALSE に設定する必要があります。

Parameters

`r`

`r` には、RAW に変換される `uuencode` された文字列が含まれます。

Examples

次の例を実行する前に、コマンドを呼び出します。

```
SET bytea_output = escape;
```

このコマンドは、印刷できない文字をエスケープし、BYTEA または RAW の値を読み取り可能な形式で画面に表示するようにサーバーに指示します。詳細については、次の Web サイトで入手可能な Postgres コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/datatype-binary.html>

次の例では、UUENCODE と UUDECODE を使用して、文字列を最初にエンコードしてからデコードします。

```
edb=# SET bytea_output = escape;
SET
edb=# SELECT UTL_ENCODE.UUENCODE('What is the date?') FROM DUAL;
```

(次のページに続く)

(前のページからの続き)

```

                                uuencode
-----
begin 0 uuencode.txt\01215VAA="!I<R!T:&4@9&%T93\\` \012` \012end\012
(1 row)

edb=# SELECT UTL_ENCODE.UUENCODE
edb=# ('begin 0 uuencode.txt\01215VAA="!I<R!T:&4@9&%T93\\` \012` \012end\012')
edb=# FROM DUAL;
          uuencode
-----
What is the date?
(1 row)

```

3.19.10 UUENCODE

UUENCODE 関数を使用して、RAW データを **uuencode** 形式のエンコードされた文字列に変換します。署名は次のとおりです。

```

UUENCODE(<r> IN RAW, <type> IN INTEGER DEFAULT 1, <filename> IN
VARCHAR2 DEFAULT NULL, <permission> IN VARCHAR2 DEFAULT NULL)

```

この関数は RAW 値を返します。

Parameters

r

r には、**uuencode** 形式に変換される RAW 文字列が含まれます。

type

type は、返される **uuencode** された文字列のタイプを指定する INTEGER 値または定数です。デフォルト値は 1 です。可能な値は次のとおりです。

Value	Constant
1	完成する
2	header_piece
3	ミドルピース
4	エンドピース

filename

filename は、エンコードされたフォームに埋め込むファイル名を指定する VARCHAR2 の値です。ファイル名を指定しない場合、UUENCODE にはエンコード形式で **uuencode.txt** のファイル名が含まれます。

permission

permission は許可モードを指定する VARCHAR2 です。デフォルト値は NULL です。

Examples

次の例を実行する前に、コマンドを呼び出します。

```
SETbytea_output=escape;
```

このコマンドは、印刷できない文字をエスケープし、BYTEA または RAW の値を読み取り可能な形式で画面に表示するようにサーバーに指示します。詳細については、次の **Web** サイトで入手可能な **Postgres** コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/datatype-binary.html>

次の例では、UUENCODE と UUDECODE を使用して、文字列を最初にエンコードしてからデコードします。

```
edb=# SET bytea_output = escape;
SET
edb=# SELECT UTL_ENCODE.UUENCODE('What is the date?') FROM DUAL;
                uuencode
-----
begin 0 uuencode.txt\01215VAA="!I<R!T:&4@9&%T93\\` \012` \012end\012
(1 row)

edb=# SELECT UTL_ENCODE.UUDECODE
edb-# ('begin 0 uuencode.txt\01215VAA="!I<R!T:&4@9&%T93\\` \012` \012end\012')
edb-# FROM DUAL;
        uudecode
-----
What is the date?
(1 row)
```

3.20 UTL_FILE

UTL_FILE パッケージは、オペレーティングシステムのファイルシステム上のファイルを読み書きする機能を提供します。非スーパーユーザーは、パッケージ内の関数または手順を使用する前に、スーパーユーザーによって UTL_FILE パッケージに対する EXECUTE 特権を付与されている必要があります。たとえば、次のコマンドは、ユーザー mary に特権を付与します。

```
GRANT EXECUTE ON PACKAGE SYS.UTL_FILE TO mary;
```

また、オペレーティングシステムのユーザー名 enterprisedb には、UTL_FILE 関数および手順を使用してアクセスするディレクトリおよびファイルに対する適切な読み取りおよび/または書き込み権限が必要です。必要なファイル許可が設定されていない場合、UTL_FILE 関数またはプロシージャで例外がスローされます。

書き込みまたは読み取り対象のファイルへのハンドルは、ファイルを参照するために使用されます。ファイルハンドルは、UTL_FILE.FILE_TYPE という名前の UTL_FILE パッケージ内のパブリック変数によって定義されます。FOPEN 関数を呼び出して返されたファイルハンドルを受け取るには、FILE_TYPE 型の変数を宣言する必要があります。ファイルハンドルは、ファイルに対する以降のすべての操作に使用されます。

ファイルシステム上のディレクトリへの参照は、CREATE DIRECTORY コマンドを使用してディレクトリに割り当てられたディレクトリ名またはエイリアスを使用して行われます。

UTL_FILE パッケージで使用可能な手順と機能を次の表に示します。

機能/手順	戻りタイプ	説明
FCLOSE(file IN OUT)	該当なし	file で識別される指定されたファイルを閉じます。
FCLOSE_ALL	該当なし	開いているファイルをすべて閉じます。
FCOPY(location, filename, dest_dir, dest_file [, start_line [, end_line]])	該当なし	location で識別されるディレクトリ内の filename を、start_line 行から end_line 行まで、ディレクトリ dest_dir 内のファイル dest_file にコピーします。
FFLUSH(file)	該当なし	バッファ内のデータを file で識別されるファイル内のディスクに強制的に書き込みます。
FOPEN(location, filename, open_mode [, max_linesize])	FILE_TYPE	location で識別されるディレクトリ内のファイル filename を開きます。
FREMOVE(location, filename)	該当なし	指定したファイルをファイルシステムから削除します。
FRENAME(location, filename, dest_dir, dest_file [, overwrite])	該当なし	指定されたファイルの名前を変更します。
GET_LINE(file, buffer OUT)	該当なし	file で識別されるファイルからテキスト行を変数 buffer に読み込みます。
IS_OPEN(file)	BOOLEAN	指定されたファイルが開いているかどうかを決定します。
NEW_LINE(file [, lines])	該当なし	行末文字シーケンスをファイルに書き込みます。
PUT(file, buffer)	該当なし	指定されたファイルに buffer を書き込みます。PUT は行末文字シーケンスを書き込みません。
PUT_LINE(file, buffer)	該当なし	指定されたファイルに buffer を書き込みます。行末文字シーケンスは、PUT_LINE 手順によって追加されます。
PUTF(file, format [, arg1] [, ...])	該当なし	指定されたファイルにフォーマットされた文字列を書き込みます。最大 5 つの置換パラメーター、arg1, ..., arg5 を指定して format の置換を指定できます。

AdvancedServer の UTL_FILE の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表にリストされている関数と手順のみがサポートされています。

UTL_FILE Exception Codes

UTL_FILE プロシージャまたは関数の呼び出しで例外が発生した場合、条件名を使用して例外をキャッチできます。UTL_FILE パッケージは、Oracle データベースと互換性のある次の例外コードを報告します。

Exception Code	Condition name
-29283	無効な操作
-29285	write_error
-29284	read_error
-29282	invalid_filehandle
-29287	invalid_maxlinesize
-29281	invalid_mode
-29280	invalid_path

3.20.1 Setting File Permissions with utl_file.umask

UTL_FILE 関数またはプロシージャがファイルを作成するとき、次のようにデフォルトのファイル許可があります。

```
-rw----- 1 enterprisedb enterprisedb 21 Jul 24 16:08 utlfile
```

enterprisedb グループに属するユーザーおよび他のすべてのユーザーに対するすべての権限が拒否されることに注意してください。enterprisedb ユーザーのみが、作成されたファイルに対する読み取りおよび書き込み権限を持ちます。

UTL_FILE 関数および手順によって作成されたファイルに異なるファイル許可セットを設定する場合は、utl_file.umask 構成パラメーターを設定することでこれを実現できます。

utl_file.umask パラメータは、Linux の umask コマンドと同様の方法で、ファイルモード作成マスク、または単に*mask*を設定します。これは、AdvancedServer UTL_FILE パッケージ内でのみ使用されます。

注釈: utl_file.umask パラメーターは、Windows システムではサポートされていません。

utl_file.umask に指定される値は、Linux の umask コマンドに有効な 3 文字または 4 文字の 8 進数文字列です。この設定により、UTL_FILE 関数および手順によって作成されたファイルの権限が決まります。(ファイル許可と umask コマンドの使用法については、Linux または Unix システムに関する情報源を参照してください。)

以下は、ファイル許可を utl_file.umask で設定する例です。

まず、UTL_FILE パッケージが使用するファイルシステムのディレクトリを設定します。オペレーティングシステムアカウント enterprisedb または postgres のどちらか適切な方が、ディレクトリで読み書きできることを確認してください。

```
mkdir /tmp/utldir
chmod 777 /tmp/utldir
```

psql で CREATE DIRECTORY コマンドを発行して、前の手順で作成したファイルシステムディレクトリを使用してディレクトリデータベースオブジェクトを作成します。

```
CREATE DIRECTORY utldir AS '/tmp/utldir';
```

utl_file.umask 構成パラメーターを設定します。次の設定では、ファイルの所有者に許可が与えられます。グループユーザーと他のユーザーには、実行権限以外のすべての権限が許可されます。

```
SET utl_file.umask TO '0011';
```

utl_file.umask パラメーターが目的の値に設定されている同じセッションで、UTL_FILE 関数と手順を実行します。

```

DECLARE
    v_utlfile      UTL_FILE.FILE_TYPE;
    v_directory    VARCHAR2(50) := 'utldir';
    v_filename     VARCHAR2(20) := 'utlfile';
BEGIN
    v_utlfile := UTL_FILE.FOPEN(v_directory, v_filename, 'w');
    UTL_FILE.PUT_LINE(v_utlfile, 'Simple one-line file');
    DBMS_OUTPUT.PUT_LINE('Created file: ' || v_filename);
    UTL_FILE.FCLOSE(v_utlfile);
END;

```

結果のファイルのアクセス許可設定は、グループユーザーと他のユーザーが、ファイルとファイル所有者の読み取りおよび書き込みアクセス許可を持っていることを示しています。

```

$ pwd
/tmp/utldir
$ ls -l
total 4
-rw-rw-rw- 1 enterprisedb enterprisedb 21 Jul 24 16:04 utlfile

```

このパラメーターは、ALTER ROLE コマンドを使用してロールごとに、ALTER DATABASE コマンドを使用してデータベースごとに、または postgresql.conf ファイルに設定することでデータベースサーバーインスタンス全体に設定することもできます。

3.20.2 FCLOSE

FCLOSE プロシージャは、開いているファイルを閉じます。

```
FCLOSE(<file> IN OUT FILE_TYPE)
```

Parameters

file

閉じられるファイルのファイルハンドルを含む FILE_TYPE 型の変数。

3.20.3 FCLOSE_ALL

FCLOSE_ALL プロシージャは、開いているファイルをすべて閉じます。クローズするオープンファイルがない場合でも、プロシージャは正常に実行されます。

```
FCLOSE_ALL
```

3.20.4 FCOPY

FCOPY プロシージャは、あるファイルから別のファイルにテキストをコピーします。

```
FCOPY(<location> VARCHAR2, <filename> VARCHAR2,
      <dest_dir> VARCHAR2, <dest_file> VARCHAR2
      [, <start_line> PLS_INTEGER [, <end_line> PLS_INTEGER ] ])
```

Parameters

location

コピーするファイルを含むディレクトリの pg_catalog.edb_dir.dirname に格納されているディレクトリ名。

filename

コピーするソースファイルの名前。

dest_dir

pg_catalog.edb_dir.dirname に保存されている、ファイルのコピー先ディレクトリのディレクトリ名。

dest_file

宛先ファイルの名前。

start_line

コピーを開始するソースファイルの行番号。デフォルトは 1 です。

end_line

コピーするソースファイルの最後の行の行番号。省略またはヌルの場合、コピーはファイルの最終行に移動します。

Examples

以下は、emp テーブルの従業員のカンマ区切りリストを含む C:/TEMP/EMPDIR/empfile.csv ファイルのコピーを作成します。コピー empcopy.csv がリストされます。

```
CREATE DIRECTORY empdir AS 'C:/TEMP/EMPDIR';

DECLARE
    v_empfile      UTL_FILE.FILE_TYPE;
    v_src_dir      VARCHAR2(50) := 'empdir';
    v_src_file     VARCHAR2(20) := 'empfile.csv';
    v_dest_dir     VARCHAR2(50) := 'empdir';
```

(次のページに続く)

(前のページからの続き)

```

v_dest_file      VARCHAR2(20) := 'empcopy.csv';
v_emprec         VARCHAR2(120);
v_count          INTEGER := 0;
BEGIN
    UTL_FILE.FCOPY(v_src_dir,v_src_file,v_dest_dir,v_dest_file);
    v_empfile := UTL_FILE.FOPEN(v_dest_dir,v_dest_file,'r');
    DBMS_OUTPUT.PUT_LINE('The following is the destination file, '' ||
        v_dest_file || ''');
    LOOP
        UTL_FILE.GET_LINE(v_empfile,v_emprec);
        DBMS_OUTPUT.PUT_LINE(v_emprec);
        v_count := v_count + 1;
    END LOOP;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            UTL_FILE.FCLOSE(v_empfile);
            DBMS_OUTPUT.PUT_LINE(v_count || ' records retrieved');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('SQLERRM: ' || SQLERRM);
            DBMS_OUTPUT.PUT_LINE('SQLCODE: ' || SQLCODE);
END;
```

```

The following is the destination file, 'empcopy.csv'
7369,SMITH,CLERK,7902,17-DEC-80,800,,20
7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30
7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30
7566,JONES,MANAGER,7839,02-APR-81,2975,,20
7654,MARTIN,SALESMAN,7698,28-SEP-81,1250,1400,30
7698,BLAKE,MANAGER,7839,01-MAY-81,2850,,30
7782,CLARK,MANAGER,7839,09-JUN-81,2450,,10
7788,SCOTT,ANALYST,7566,19-APR-87,3000,,20
7839,KING,PRESIDENT,,17-NOV-81,5000,,10
7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30
7876,ADAMS,CLERK,7788,23-MAY-87,1100,,20
7900,JAMES,CLERK,7698,03-DEC-81,950,,30
7902,FORD,ANALYST,7566,03-DEC-81,3000,,20
7934,MILLER,CLERK,7782,23-JAN-82,1300,,10
14 records retrieved
```


3.20.5 FFLUSH

FFLUSH プロシーダは、書き込みバッファからファイルに未書き込みデータをフラッシュします。

```
FFLUSH(<file> FILE_TYPE)
```

Parameters

`file`

ファイルハンドルを含む `FILE_TYPE` 型の変数。

Examples

NEW_LINE プロシーダが呼び出された後、各行がフラッシュされます。

```
DECLARE
    v_empfile      UTL_FILE.FILE_TYPE;
    v_directory    VARCHAR2(50) := 'empdir';
    v_filename     VARCHAR2(20) := 'empfile.csv';
    CURSOR emp_cur IS SELECT * FROM emp ORDER BY empno;
BEGIN
    v_empfile := UTL_FILE.FOPEN(v_directory,v_filename,'w');
    FOR i IN emp_cur LOOP
        UTL_FILE.PUT(v_empfile,i.empno);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.ename);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.job);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.mgr);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.hiredate);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.sal);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.comm);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.deptno);
        UTL_FILE.NEW_LINE(v_empfile);
        UTL_FILE.FFLUSH(v_empfile);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Created file: ' || v_filename);
    UTL_FILE.FCLOSE(v_empfile);
END;
```

3.20.6 FOPEN

FOPEN 関数は、I/O 用のファイルを開きます。

```
filetype FILE_TYPE FOPEN(<location> VARCHAR2,  
    <filename> VARCHAR2,<open_mode> VARCHAR2  
    [, <max_linesize> BINARY_INTEGER ])
```

Parameters

location

pg_catalog.edb_dir.dirname に保存されている、開くファイルを含むディレクトリのディレクトリ名。

filename

開くファイルの名前。

open_mode

ファイルが開かれるモード。モードは次のとおりです。a-ファイルに追加します。r-ファイルから読み取ります。w-ファイルに書き込みます。

max_linesize

文字の行の最大サイズ。読み取りモードでは、max_linesize を超える行を読み取ろうとすると例外がスローされます。書き込みモードと追加モードでは、max_linesize を超える行を書き込もうとすると、例外がスローされます。行末文字は、最大行サイズを超えているかどうかの判断には含まれません。この動作は、Oracle データベースと互換性がありません。Oracle は、行末文字をカウントします。

filetype

開かれたファイルのファイルハンドルを含む FILE_TYPE タイプの変数。

3.20.7 REMOVE

REMOVE プロシージャは、システムからファイルを削除します。

```
REMOVE(<location> VARCHAR2, <filename> VARCHAR2)
```

削除するファイルが存在しない場合、例外がスローされます。

Parameters

location

削除するファイルを含むディレクトリの `pg_catalog.edb_dir.dirname` に格納されているディレクトリ名。

`filename`

削除するファイルの名前。

Examples

以下はファイル `empfile.csv` を削除します。

```
DECLARE
    v_directory    VARCHAR2(50) := 'empdir';
    v_filename     VARCHAR2(20) := 'empfile.csv';
BEGIN
    UTL_FILE.FREMOVE(v_directory,v_filename);
    DBMS_OUTPUT.PUT_LINE('Removed file: ' || v_filename);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('SQLERRM: ' || SQLERRM);
        DBMS_OUTPUT.PUT_LINE('SQLCODE: ' || SQLCODE);
END;
```

Removed file: empfile.csv

3.20.8 FRENAME

`FRENAME` プロシージャは、指定されたファイルの名前を変更します。これにより、ファイルをある場所から別の場所に効果的に移動できます。

```
FRENAME(<location> VARCHAR2, <filename> VARCHAR2,
        <dest_dir> VARCHAR2, <dest_file> VARCHAR2,
        [ <overwrite> BOOLEAN ])
```

Parameters

`location`

名前を変更するファイルを含むディレクトリの `pg_catalog.edb_dir.dirname` に格納されているディレクトリ名。

`filename`

名前を変更するソースファイルの名前。

`dest_dir`

pg_catalog.edb_dir.dirname に保存されている、名前が変更されたファイルが存在するディレクトリのディレクトリ名。

dest_file

元のファイルの新しい名前。

overwrite

TRUE に設定されている場合は dest_dir の dest_file という名前の既存ファイルを置き換えます。FALSE に設定されている場合は例外がスローされます。これがデフォルトです。

Examples

以下は、emp テーブルの従業員のカンマ区切りリストを含むファイル C:/TEMP/EMPDIR/empfile.csv の名前を変更します。名前が変更されたファイル C:/TEMP/NEWDIR/newemp.csv が一覧表示されます。

```
CREATE DIRECTORY "newdir" AS 'C:/TEMP/NEWDIR';

DECLARE
    v_empfile      UTL_FILE.FILE_TYPE;
    v_src_dir      VARCHAR2(50) := 'empdir';
    v_src_file     VARCHAR2(20) := 'empfile.csv';
    v_dest_dir     VARCHAR2(50) := 'newdir';
    v_dest_file    VARCHAR2(50) := 'newemp.csv';
    v_replace      BOOLEAN := FALSE;
    v_emprec       VARCHAR2(120);
    v_count        INTEGER := 0;
BEGIN
    UTL_FILE.FRENAME(v_src_dir,v_src_file,v_dest_dir,
        v_dest_file,v_replace);
    v_empfile := UTL_FILE.FOPEN(v_dest_dir,v_dest_file,'r');
    DBMS_OUTPUT.PUT_LINE('The following is the renamed file, '' ||
        v_dest_file || ''');
    LOOP
        UTL_FILE.GET_LINE(v_empfile,v_emprec);
        DBMS_OUTPUT.PUT_LINE(v_emprec);
        v_count := v_count + 1;
    END LOOP;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            UTL_FILE.FCLOSE(v_empfile);
            DBMS_OUTPUT.PUT_LINE(v_count || ' records retrieved');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('SQLERRM: ' || SQLERRM);
            DBMS_OUTPUT.PUT_LINE('SQLCODE: ' || SQLCODE);
END;

The following is the renamed file, 'newemp.csv'
```

(次のページに続く)

(前のページからの続き)

```

7369, SMITH, CLERK, 7902, 17-DEC-80 00:00:00, 800.00, , 20
7499, ALLEN, SALESMAN, 7698, 20-FEB-81 00:00:00, 1600.00, 300.00, 30
7521, WARD, SALESMAN, 7698, 22-FEB-81 00:00:00, 1250.00, 500.00, 30
7566, JONES, MANAGER, 7839, 02-APR-81 00:00:00, 2975.00, , 20
7654, MARTIN, SALESMAN, 7698, 28-SEP-81 00:00:00, 1250.00, 1400.00, 30
7698, BLAKE, MANAGER, 7839, 01-MAY-81 00:00:00, 2850.00, , 30
7782, CLARK, MANAGER, 7839, 09-JUN-81 00:00:00, 2450.00, , 10
7788, SCOTT, ANALYST, 7566, 19-APR-87 00:00:00, 3000.00, , 20
7839, KING, PRESIDENT, , 17-NOV-81 00:00:00, 5000.00, , 10
7844, TURNER, SALESMAN, 7698, 08-SEP-81 00:00:00, 1500.00, 0.00, 30
7876, ADAMS, CLERK, 7788, 23-MAY-87 00:00:00, 1100.00, , 20
7900, JAMES, CLERK, 7698, 03-DEC-81 00:00:00, 950.00, , 30
7902, FORD, ANALYST, 7566, 03-DEC-81 00:00:00, 3000.00, , 20
7934, MILLER, CLERK, 7782, 23-JAN-82 00:00:00, 1300.00, , 10
14 records retrieved

```

3.20.9 GET_LINE

GET_LINE プロシージャは、指定されたファイルからテキストの行を読み取りますが、行末ターミネータは含みません。読み取る行がなくなると、NO_DATA_FOUND 例外がスローされます。

```
GET_LINE(<file> FILE_TYPE, <buffer> OUT VARCHAR2)
```

Parameters

file

開かれたファイルのファイルハンドルを含む FILE_TYPE タイプの変数。

buffer

ファイルから行を受け取る変数。

Examples

次の匿名ブロックは、ファイル empfile.csv のレコードを読み取り、表示します。

```

DECLARE
    v_empfile      UTL_FILE.FILE_TYPE;
    v_directory    VARCHAR2(50) := 'empdir';
    v_filename     VARCHAR2(20) := 'empfile.csv';
    v_emprec       VARCHAR2(120);
    v_count        INTEGER := 0;
BEGIN
    v_empfile := UTL_FILE.FOPEN(v_directory, v_filename, 'r');
    LOOP

```

(次のページに続く)

(前のページからの続き)

```

        UTL_FILE.GET_LINE(v_empfile,v_emprec);
        DBMS_OUTPUT.PUT_LINE(v_emprec);
        v_count := v_count + 1;
    END LOOP;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            UTL_FILE.FCLOSE(v_empfile);
            DBMS_OUTPUT.PUT_LINE('End of file ' || v_filename || ' - ' ||
                v_count || ' records retrieved');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('SQLERRM: ' || SQLERRM);
            DBMS_OUTPUT.PUT_LINE('SQLCODE: ' || SQLCODE);
END;
```

```

7369, SMITH, CLERK, 7902, 17-DEC-80 00:00:00, 800.00, , 20
7499, ALLEN, SALESMAN, 7698, 20-FEB-81 00:00:00, 1600.00, 300.00, 30
7521, WARD, SALESMAN, 7698, 22-FEB-81 00:00:00, 1250.00, 500.00, 30
7566, JONES, MANAGER, 7839, 02-APR-81 00:00:00, 2975.00, , 20
7654, MARTIN, SALESMAN, 7698, 28-SEP-81 00:00:00, 1250.00, 1400.00, 30
7698, BLAKE, MANAGER, 7839, 01-MAY-81 00:00:00, 2850.00, , 30
7782, CLARK, MANAGER, 7839, 09-JUN-81 00:00:00, 2450.00, , 10
7788, SCOTT, ANALYST, 7566, 19-APR-87 00:00:00, 3000.00, , 20
7839, KING, PRESIDENT, , 17-NOV-81 00:00:00, 5000.00, , 10
7844, TURNER, SALESMAN, 7698, 08-SEP-81 00:00:00, 1500.00, 0.00, 30
7876, ADAMS, CLERK, 7788, 23-MAY-87 00:00:00, 1100.00, , 20
7900, JAMES, CLERK, 7698, 03-DEC-81 00:00:00, 950.00, , 30
7902, FORD, ANALYST, 7566, 03-DEC-81 00:00:00, 3000.00, , 20
7934, MILLER, CLERK, 7782, 23-JAN-82 00:00:00, 1300.00, , 10
End of file empfile.csv - 14 records retrieved
```

3.20.10 IS_OPEN

IS_OPEN 関数は、指定されたファイルが開いているかどうかを判断します。

```
<status> BOOLEAN IS_OPEN(<file> FILE_TYPE)
```

Parameters

file

テストするファイルのファイルハンドルを含む FILE_TYPE タイプの変数。

status

指定されたファイルが開いている場合は TRUE、開いていない場合は FALSE。

3.20.11 NEW_LINE

NEW_LINE プロシージャは、ファイルに行末文字シーケンスを書き込みます。

```
NEW_LINE(<file> FILE_TYPE [, <lines> INTEGER ])
```

Parameters

file

行末文字シーケンスが書き込まれるファイルのファイルハンドルを含む FILE_TYPE タイプの変数。

lines

書き込まれる行末文字シーケンスの数。デフォルトは 1 です。

Examples

従業員レコードのダブルスペースリストを含むファイルが書き込まれます。

```
DECLARE
    v_empfile      UTL_FILE.FILE_TYPE;
    v_directory    VARCHAR2(50) := 'empdir';
    v_filename     VARCHAR2(20) := 'empfile.csv';
    CURSOR emp_cur IS SELECT * FROM emp ORDER BY empno;
BEGIN
    v_empfile := UTL_FILE.FOPEN(v_directory,v_filename,'w');
    FOR i IN emp_cur LOOP
        UTL_FILE.PUT(v_empfile,i.empno);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.ename);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.job);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.mgr);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.hiredate);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.sal);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.comm);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.deptno);
        UTL_FILE.NEW_LINE(v_empfile,2);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Created file: ' || v_filename);
    UTL_FILE.FCLOSE(v_empfile);
END;
```

(次のページに続く)

(前のページからの続き)

```
Created file: empfile.csv
```

このファイルが表示されます:

```
C:\TEMP\EMPDIR>TYPE empfile.csv

7369,SMITH,CLERK,7902,17-DEC-80 00:00:00,800.00,,20

7499,ALLEN,SALESMAN,7698,20-FEB-81 00:00:00,1600.00,300.00,30

7521,WARD,SALESMAN,7698,22-FEB-81 00:00:00,1250.00,500.00,30

7566,JONES,MANAGER,7839,02-APR-81 00:00:00,2975.00,,20

7654,MARTIN,SALESMAN,7698,28-SEP-81 00:00:00,1250.00,1400.00,30

7698,BLAKE,MANAGER,7839,01-MAY-81 00:00:00,2850.00,,30

7782,CLARK,MANAGER,7839,09-JUN-81 00:00:00,2450.00,,10

7788,SCOTT,ANALYST,7566,19-APR-87 00:00:00,3000.00,,20

7839,KING,PRESIDENT,,17-NOV-81 00:00:00,5000.00,,10

7844,TURNER,SALESMAN,7698,08-SEP-81 00:00:00,1500.00,0.00,30

7876,ADAMS,CLERK,7788,23-MAY-87 00:00:00,1100.00,,20

7900,JAMES,CLERK,7698,03-DEC-81 00:00:00,950.00,,30

7902,FORD,ANALYST,7566,03-DEC-81 00:00:00,3000.00,,20

7934,MILLER,CLERK,7782,23-JAN-82 00:00:00,1300.00,,10
```

3.20.12 PUT

PUT プロシージャは、指定されたファイルに文字列を書き込みます。文字列の末尾に行末文字シーケンスは書き込まれません。NEW_LINE 手順を使用して、行末文字シーケンスを追加します。

```
PUT(<file> FILE_TYPE, <buffer> { DATE | NUMBER | TIMESTAMP |
    VARCHAR2 })
```

Parameters

file

指定された文字列が書き込まれるファイルのファイルハンドルを含む FILE_TYPE 型の変数。

buffer

指定されたファイルに書き込まれるテキスト。

Examples

次の例では、PUT プロシージャを使用して、emp テーブルから従業員のカンマ区切りファイルを作成します。

```
DECLARE
    v_empfile      UTL_FILE.FILE_TYPE;
    v_directory    VARCHAR2(50) := 'empdir';
    v_filename     VARCHAR2(20) := 'empfile.csv';
    CURSOR emp_cur IS SELECT * FROM emp ORDER BY empno;
BEGIN
    v_empfile := UTL_FILE.FOPEN(v_directory,v_filename,'w');
    FOR i IN emp_cur LOOP
        UTL_FILE.PUT(v_empfile,i.empno);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.ename);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.job);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.mgr);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.hiredate);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.sal);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.comm);
        UTL_FILE.PUT(v_empfile,',');
        UTL_FILE.PUT(v_empfile,i.deptno);
        UTL_FILE.NEW_LINE(v_empfile);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Created file: ' || v_filename);
    UTL_FILE.FCLOSE(v_empfile);
END;
```

Created file: empfile.csv

以下は、上記で作成された empfile.csv の内容です。

```
C:\TEMP\EMPDIR>TYPE empfile.csv

7369,SMITH,CLERK,7902,17-DEC-80 00:00:00,800.00,,20
7499,ALLEN,SALESMAN,7698,20-FEB-81 00:00:00,1600.00,300.00,30
7521,WARD,SALESMAN,7698,22-FEB-81 00:00:00,1250.00,500.00,30
7566,JONES,MANAGER,7839,02-APR-81 00:00:00,2975.00,,20
```

(次のページに続く)

(前のページからの続き)

```

7654,MARTIN,SALESMAN,7698,28-SEP-81 00:00:00,1250.00,1400.00,30
7698,BLAKE,MANAGER,7839,01-MAY-81 00:00:00,2850.00,,30
7782,CLARK,MANAGER,7839,09-JUN-81 00:00:00,2450.00,,10
7788,SCOTT,ANALYST,7566,19-APR-87 00:00:00,3000.00,,20
7839,KING,PRESIDENT,,17-NOV-81 00:00:00,5000.00,,10
7844,TURNER,SALESMAN,7698,08-SEP-81 00:00:00,1500.00,0.00,30
7876,ADAMS,CLERK,7788,23-MAY-87 00:00:00,1100.00,,20
7900,JAMES,CLERK,7698,03-DEC-81 00:00:00,950.00,,30
7902,FORD,ANALYST,7566,03-DEC-81 00:00:00,3000.00,,20
7934,MILLER,CLERK,7782,23-JAN-82 00:00:00,1300.00,,10

```

3.20.13 PUT_LINE

PUT_LINE プロシージャは、行末文字シーケンスを含む特定のファイルに単一行を書き込みます。

```

PUT_LINE(<file> FILE_TYPE,
        <buffer> {DATE|NUMBER|TIMESTAMP|VARCHAR2})

```

Parameters

file

指定された行が書き込まれるファイルのファイルハンドルを含む FILE_TYPE タイプの変数。

buffer

指定されたファイルに書き込まれるテキスト。

Examples

次の例では、PUT_LINE プロシージャを使用して、emp テーブルから従業員のカンマ区切りファイルを作成します。

```

DECLARE
    v_empfile      UTL_FILE.FILE_TYPE;
    v_directory    VARCHAR2(50) := 'empdir';
    v_filename     VARCHAR2(20) := 'empfile.csv';
    v_emprec       VARCHAR2(120);
    CURSOR emp_cur IS SELECT * FROM emp ORDER BY empno;
BEGIN
    v_empfile := UTL_FILE.FOPEN(v_directory,v_filename,'w');
    FOR i IN emp_cur LOOP
        v_emprec := i.empno || ',' || i.ename || ',' || i.job || ',' ||
            NVL(LTRIM(TO_CHAR(i.mgr,'9999')),') || ',' || i.hiredate ||
            ',' || i.sal || ',' ||
            NVL(LTRIM(TO_CHAR(i.comm,'9990.99')),') || ',' || i.deptno;
    
```

(次のページに続く)

(前のページからの続き)

```

        UTL_FILE.PUT_LINE(v_empfile,v_emprec);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Created file: ' || v_filename);
    UTL_FILE.FCLOSE(v_empfile);
END;
```

以下は、上記で作成された empfile.csv の内容です。

```

C:\TEMP\EMPDIR>TYPE empfile.csv

7369,SMITH,CLERK,7902,17-DEC-80 00:00:00,800.00,,20
7499,ALLEN,SALESMAN,7698,20-FEB-81 00:00:00,1600.00,300.00,30
7521,WARD,SALESMAN,7698,22-FEB-81 00:00:00,1250.00,500.00,30
7566,JONES,MANAGER,7839,02-APR-81 00:00:00,2975.00,,20
7654,MARTIN,SALESMAN,7698,28-SEP-81 00:00:00,1250.00,1400.00,30
7698,BLAKE,MANAGER,7839,01-MAY-81 00:00:00,2850.00,,30
7782,CLARK,MANAGER,7839,09-JUN-81 00:00:00,2450.00,,10
7788,SCOTT,ANALYST,7566,19-APR-87 00:00:00,3000.00,,20
7839,KING,PRESIDENT,,17-NOV-81 00:00:00,5000.00,,10
7844,TURNER,SALESMAN,7698,08-SEP-81 00:00:00,1500.00,0.00,30
7876,ADAMS,CLERK,7788,23-MAY-87 00:00:00,1100.00,,20
7900,JAMES,CLERK,7698,03-DEC-81 00:00:00,950.00,,30
7902,FORD,ANALYST,7566,03-DEC-81 00:00:00,3000.00,,20
7934,MILLER,CLERK,7782,23-JAN-82 00:00:00,1300.00,,10
```

3.20.14 PUTF

PUTF プロシージャは、指定されたファイルにフォーマットされた文字列を書き込みます。

```

PUTF(<file> FILE_TYPE, <format> VARCHAR2 [, <arg1> VARCHAR2]
    [, ...])
```

Parameters

file

フォーマットされた行が書き込まれるファイルのファイルハンドルを含む FILE_TYPE 型の変数。

format

ファイルに書き込まれるテキストをフォーマットする文字列。特殊文字シーケンス %s は、arg の値に置き換えられます。特殊文字シーケンス /n は、改行を示します。ただし、AdvancedServer では、改行文字を 1 つではなく 2 つの連続したバックスラッシュ (//n) で指定する必要があります。この特性は、Oracle データベースと互換性がありません。

arg1

最大で 5 つの引数、arg1、.../arg5、%s が出現するたびにフォーマット文字列で置換されます。最初の引数は %s の最初の出現に置き換えられ、2 番目の引数は %s の 2 番目の出現に置き換えられます。

Examples

次の匿名ブロックは、emp テーブルのデータを含むフォーマットされた出力を生成します。Oracle データベースと互換性のないフォーマット文字列の改行文字シーケンスには、E リテラル構文と二重バックスラッシュを使用していることに注意してください。

```
DECLARE
    v_empfile      UTL_FILE.FILE_TYPE;
    v_directory    VARCHAR2(50) := 'empdir';
    v_filename     VARCHAR2(20) := 'empfile.csv';
    v_format       VARCHAR2(200);
    CURSOR emp_cur IS SELECT * FROM emp ORDER BY empno;
BEGIN
    v_format := E'%s %s, %s\nSalary: $%s Commission: $%s\n\n';
    v_empfile := UTL_FILE.FOPEN(v_directory,v_filename,'w');
    FOR i IN emp_cur LOOP
        UTL_FILE.PUTF(v_empfile,v_format,i.empno,i.ename,i.job,i.sal,
            NVL(i.comm,0));
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Created file: ' || v_filename);
    UTL_FILE.FCLOSE(v_empfile);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('SQLERRM: ' || SQLERRM);
        DBMS_OUTPUT.PUT_LINE('SQLCODE: ' || SQLCODE);
END;
```

Created file: empfile.csv

以下は、上記で作成された empfile.csv の内容です。

```
C:\TEMP\EMPDIR>TYPE empfile.csv
7369 SMITH, CLERK
Salary: $800.00 Commission: $0
7499 ALLEN, SALESMAN
Salary: $1600.00 Commission: $300.00
7521 WARD, SALESMAN
Salary: $1250.00 Commission: $500.00
7566 JONES, MANAGER
Salary: $2975.00 Commission: $0
7654 MARTIN, SALESMAN
Salary: $1250.00 Commission: $1400.00
7698 BLAKE, MANAGER
Salary: $2850.00 Commission: $0
7782 CLARK, MANAGER
```

(次のページに続く)

(前のページからの続き)

```
Salary: $2450.00 Commission: $0
7788 SCOTT, ANALYST
Salary: $3000.00 Commission: $0
7839 KING, PRESIDENT
Salary: $5000.00 Commission: $0
7844 TURNER, SALESMAN
Salary: $1500.00 Commission: $0.00
7876 ADAMS, CLERK
Salary: $1100.00 Commission: $0
7900 JAMES, CLERK
Salary: $950.00 Commission: $0
7902 FORD, ANALYST
Salary: $3000.00 Commission: $0
7934 MILLER, CLERK
Salary: $1300.00 Commission: $0
```

3.21 UTL_HTTP

UTL_HTTP パッケージは、HTTP または HTTPS プロトコルを使用して、URL で見つかった情報を取得する方法を提供します。AdvancedServer は、次の機能と手順をサポートしています。

機能/手順	戻りタイプ	説明
BEGIN_REQUEST(url, method, http_version)	UTL_HTTP.REQ	新しい HTTP 要求を開始します。
END_REQUEST(r IN OUT)	該当なし	完了する前に HTTP 要求を終了します。
END_RESPONSE(r IN OUT)	該当なし	HTTP 応答を終了します。
END_OF_BODY(r IN OUT)	該当なし	パッケージ本体を終了
GET_BODY_CHARSET	VARCHAR2	将来の HTTP 要求の本文のデフォルトの文字セットを返します。
GET_BODY_CHARSET(charset OUT)	該当なし	将来の HTTP 要求の本文のデフォルトの文字セットを返します。
GET_FOLLOW_REDIRECT(max_redirects OUT)	該当なし	許可されるリダイレクトの最大数の現在の設定。
GET_HEADER(r IN OUT, n, name OUT, value OUT)	該当なし	HTTP 応答の n 番目のヘッダーを返します。
GET_HEADER_BY_NAME(r IN OUT, name, value OUT, n)	該当なし	指定された名前の HTTP 応答ヘッダーを返します。
GET_HEADER_COUNT(r IN OUT)	INTEGER	HTTP 応答ヘッダーの数を返します。
GET_RESPONSE(r IN OUT)	UTL_HTTP.RESP	HTTP 応答を返します。
GET_RESPONSE_ERROR_CHECK(enable OUT)	該当なし	応答エラーチェックが設定されているかどうかを返します。
GET_TRANSFER_TIMEOUT(timeout OUT)	該当なし	HTTP 要求の転送タイムアウト設定を返します。
READ_LINE(r IN OUT, data OUT, remove_crlf)	該当なし	行の終わりまでテキスト形式の HTTP 応答本文を返します。
READ_RAW(r IN OUT, data OUT, len)	該当なし	指定されたバイト数について、HTTP 応答の本文をバイナリ形式で返します。
READ_TEXT(r IN OUT, data OUT, len)	該当なし	指定した文字数の HTTP 応答本文をテキスト形式で返します。

次のページに続く

表 9 – 前のページからの続き

REQUEST(url)	VARCHAR2	Web ページのコンテンツを返します。
REQUEST_PIECES(url, max_pieces)	UTL_HTTP. HTML_PIECES	URL から取得した 2000 バイトのセグメントのテーブルを返します。
SET_BODY_CHARSET(charset)	該当なし	将来の HTTP リクエストの本文のデフォルトの文字セットを設定します。
SET_FOLLOW_REDIRECT(max_redirects)	該当なし	リダイレクト命令に従う最大回数を設定します。
SET_FOLLOW_REDIRECT(r IN OUT, max_redirects)	該当なし	個々のリクエストのリダイレクト命令に従う最大回数を設定します。
SET_HEADER(r IN OUT, name, value)	該当なし	HTTP 要求ヘッダーを設定します。
SET_RESPONSE_ERROR_CHECK(enable)	該当なし	HTTP4xx および 5xx ステータスコードをエラーとして扱うかどうかを決定します。
SET_TRANSFER_TIMEOUT(timeout)	該当なし	HTTP 要求のデフォルトの転送タイムアウト値を設定します。
SET_TRANSFER_TIMEOUT(r IN OUT, timeout)	該当なし	個々の HTTP 要求の転送タイムアウト値を設定します。
WRITE_LINE(r IN OUT, data)	該当なし	CRLF 終了データを TEXT 形式で HTTP 要求本文に書き込みます。
WRITE_RAW(r IN OUT, data)	該当なし	HTTP リクエストの本文にデータをバイナリ形式で書き込みます。
WRITE_TEXT(r IN OUT, data)	該当なし	データを TEXT 形式で HTTP リクエスト本文に書き込みます。

AdvancedServer の UTL_HTTP の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表にリストされている関数と手順のみがサポートされています。

注釈: AdvancedServer では、HTTP 4xx または HTTP 5xx の応答によりデータベースエラーが生成されます。Oracle では、これは構成可能ですが、デフォルトでは FALSE です。

AdvancedServer では、UTL_HTTP テキストインターフェイスは、ダウンロードされたデータがデータベースエンコーディングであると想定しています。現在使用可能なインターフェイスはすべてテキストインターフェイスです。Oracle では、エンコーディングは HTTP ヘッダーから検出されます。ヘッダーがない場合、デフォルトは設定可能で、デフォルトは ISO-8859-1 です。

AdvancedServer は、受信したすべての Cookie を無視します。

Oracle で発生する可能性のある UTL_HTTP 例外は、AdvancedServer によって認識されません。また、AdvancedServer から返されるエラーコードは、Oracle から返されるエラーコードとは異なります。

UTL_HTTP Exception Codes

UTL_HTTP プロシージャまたは関数の呼び出しで例外が発生した場合、条件名を使用して例外をキャッチできます。UTL_HTTP パッケージは、Oracle データベースと互換性のある次の例外コードを報告します。

Exception Code	Condition Name	Description	Raised Where
-29266	END_OF_BODY	HTTP 応答本文の終わりに達しました	READ_LINE, READ_RAW および READ_TEXT 機能

UTL_HTTP.END_OF_BODY 例外を使用するには、まずインストールディレクトリの contrib/utl_http ディレクトリから utl_http_public.sql ファイルを実行する必要があります。

UTL_HTTP で使用できるさまざまなパブリック定数があります。これらは次の表にリストされています。

次の表には、HTTP バージョンとポート割り当てを定義する UTL_HTTP パブリック定数が含まれています。

HTTP VERSIONS	
HTTP_VERSION_1_0	CONSTANTVARCHAR2 (64) := 'HTTP/1.0';
HTTP_VERSION_1_1	CONSTANTVARCHAR2 (64) := 'HTTP/1.1';
STANDARD PORT ASSIGNMENTS	
DEFAULT_HTTP_PORT	CONSTANTINTEGER:=80;
DEFAULT_HTTPS_PORT	定数 INTEGER:=443;

次の表には、UTL_HTTP のパブリックステータスコード定数が含まれています。

1XX INFORMATIONAL	
HTTP_CONTINUE	定数 INTEGER:=100;
HTTP_SWITCHING_PROTOCOLS	定数 INTEGER:=101;
HTTP_PROCESSING	定数 INTEGER:=102;
2XX SUCCESS	
HTTP_OK	定数 INTEGER:=200;
HTTP_CREATED	CONSTANTINTEGER:=201;
HTTP_ACCEPTED	CONSTANTINTEGER:=202;
HTTP_NON_AUTHORITATIVE_INFO	CONSTANTINTEGER:=203;
HTTP_NO_CONTENT	定数 INTEGER:=204;
HTTP_RESET_CONTENT	定数 INTEGER:=205;
HTTP_PARTIAL_CONTENT	CONSTANTINTEGER:=206;
HTTP_MULTI_STATUS	CONSTANTINTEGER:=207;
HTTP_ALREADY_REPORTED	定数 INTEGER:=208;
HTTP_IM_USED	定数 INTEGER:=226;
3XX REDIRECTION	
HTTP_MULTIPLE_CHOICES	定数 INTEGER:=300;
HTTP_MOVED_PERMANENTLY	CONSTANTINTEGER:=301;
HTTP_FOUND	CONSTANTINTEGER:=302;
HTTP_SEE_OTHER	CONSTANTINTEGER:=303;
HTTP_NOT_MODIFIED	CONSTANTINTEGER:=304;
HTTP_USE_PROXY	CONSTANTINTEGER:=305;
HTTP_SWITCH_PROXY	CONSTANTINTEGER:=306;
HTTP_TEMPORARY_REDIRECT	CONSTANTINTEGER:=307;
HTTP_PERMANENT_REDIRECT	CONSTANTINTEGER:=308;

4XX CLIENT ERROR	
HTTP_BAD_REQUEST	定数 INTEGER:=400;
HTTP_UNAUTHORIZED	定数 INTEGER:=401;
HTTP_PAYMENT_REQUIRED	定数 INTEGER:=402;
HTTP_FORBIDDEN	CONSTANTINTEGER:=403;
HTTP_NOT_FOUND	CONSTANTINTEGER:=404;
HTTP_METHOD_NOT_ALLOWED	定数 INTEGER:=405;
HTTP_NOT_ACCEPTABLE	CONSTANTINTEGER:=406;
HTTP_PROXY_AUTH_REQUIRED	定数 INTEGER:=407;
HTTP_REQUEST_TIME_OUT	定数 INTEGER:=408;
HTTP_CONFLICT	定数 INTEGER:=409;
HTTP_GONE	CONSTANTINTEGER:=410;
HTTP_LENGTH_REQUIRED	定数 INTEGER:=411;
HTTP_PRECONDITION_FAILED	CONSTANTINTEGER:=412;
HTTP_REQUEST_ENTITY_TOO_LARGE	定数 INTEGER:=413;
HTTP_REQUEST_URI_TOO_LARGE	定数 INTEGER:=414;
HTTP_UNSUPPORTED_MEDIA_TYPE	CONSTANTINTEGER:=415;
HTTP_REQ_RANGE_NOT_SATISFIABLE	定数 INTEGER:=416;
HTTP_EXPECTATION_FAILED	CONSTANTINTEGER:=417;
HTTP_I_AM_A_TEAPOT	定数 INTEGER:=418;
HTTP_AUTHENTICATION_TIME_OUT	定数 INTEGER:=419;
HTTP_ENHANCE_YOUR_CALM	定数 INTEGER:=420;
HTTP_UNPROCESSABLE_ENTITY	定数 INTEGER:=422;
HTTP_LOCKED	定数 INTEGER:=423;
HTTP_FAILED_DEPENDENCY	CONSTANTINTEGER:=424;
HTTP_UNORDERED_COLLECTION	定数 INTEGER:=425;
HTTP_UPGRADE_REQUIRED	CONSTANTINTEGER:=426;
HTTP_PRECONDITION_REQUIRED	CONSTANTINTEGER:=428;
HTTP_TOO_MANY_REQUESTS	定数 INTEGER:=429;
HTTP_REQUEST_HEADER_FIELDS_TOO_LARGE	定数 INTEGER:=431;
HTTP_NO_RESPONSE	定数 INTEGER:=444;
HTTP_RETRY_WITH	定数 INTEGER:=449;
HTTP_BLOCKED_BY_WINDOWS_PARENTAL_CONTROLS	定数 INTEGER:=450;
HTTP_REDIRECT	定数 INTEGER:=451;
HTTP_REQUEST_HEADER_TOO_LARGE	定数 INTEGER:=494;
HTTP_CERT_ERROR	定数 INTEGER:=495;
HTTP_NO_CERT	CONSTANTINTEGER:=496;
HTTP_HTTP_TO_HTTPS	定数 INTEGER:=497;

次のページに続く

表 10 – 前のページからの続き

4XX CLIENT ERROR	
HTTP_CLIENT_CLOSED_REQUEST	CONSTANTINTEGER:=499;

5XX SERVER ERROR	
HTTP_INTERNAL_SERVER_ERROR	定数 INTEGER:=500;
HTTP_NOT_IMPLEMENTED	定数 INTEGER:=501;
HTTP_BAD_GATEWAY	CONSTANTINTEGER:=502;
HTTP_SERVICE_UNAVAILABLE	定数 INTEGER:=503;
HTTP_GATEWAY_TIME_OUT	定数 INTEGER:=504;
HTTP_VERSION_NOT_SUPPORTED	CONSTANTINTEGER:=505;
HTTP_VARIANT_ALSO_NEGOTIATES	定数 INTEGER:=506;
HTTP_INSUFFICIENT_STORAGE	CONSTANTINTEGER:=507;
HTTP_LOOP_DETECTED	CONSTANTINTEGER:=508;
HTTP_BANDWIDTH_LIMIT_EXCEEDED	CONSTANTINTEGER:=509;
HTTP_NOT_EXTENDED	CONSTANTINTEGER:=510;
HTTP_NETWORK_AUTHENTICATION_REQUIRED	定数 INTEGER:=511;
HTTP_NETWORK_READ_TIME_OUT_ERROR	定数 INTEGER:=598;
HTTP_NETWORK_CONNECT_TIME_OUT_ERROR	定数 INTEGER:=599;

3.21.1 HTML_PIECES

UTL_HTTP パッケージは、HTML_PIECES というタイプを宣言します。これは、BINARY_INTEGER でインデックス付けされた VARCHAR2 (2000) タイプのテーブルです。このタイプの値は、REQUEST_PIECES 関数によって返されます。

```
TYPE html_pieces IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;
```

3.21.2 REQ

REQ レコードタイプは、各 HTTP リクエストに関する情報を保持します。

```
TYPE req IS RECORD (
    url            VARCHAR2(32767),      -- URL to be accessed
    method         VARCHAR2(64),        -- HTTP method
    http_version   VARCHAR2(64),        -- HTTP version
    private_hndl   INTEGER               -- Holds handle for this request
);
```

3.21.3 RESP

RESP レコードタイプは、各 HTTP 要求からの応答に関する情報を保持します。

```
TYPE resp IS RECORD (
    status_code    INTEGER,              -- HTTP status code
    reason_phrase  VARCHAR2(256),        -- HTTP response reason phrase
    http_version   VARCHAR2(64),        -- HTTP version
    private_hndl   INTEGER               -- Holds handle for this response
);
```

3.21.4 BEGIN_REQUEST

BEGIN_REQUEST 関数は、新しい HTTP 要求を開始します。指定した URL で Web サーバーへのネットワーク接続が確立されます。署名は次のとおりです。

```
BEGIN_REQUEST(<url> IN VARCHAR2, <method> IN VARCHAR2 DEFAULT
'GET ', <http_version> IN VARCHAR2 DEFAULT NULL) RETURN
UTL_HTTP.REQ
```

BEGIN_REQUEST 関数は、UTL_HTTP.REQ タイプのレコードを返します。

Parameters

`url`

`url` は UTL_HTTP がコンテンツを返す `UniformResourceLocator` です。

`method`

`method` は使用する HTTP メソッドです。デフォルトは GET です。

`http_version`

`http_version` は、リクエストを送信する HTTP プロトコルのバージョンです。指定する値は HTTP/1.0 または HTTP/1.1 のいずれかです。デフォルトは `null` です。この場合、UTL_HTTP パッケージでサポートされている最新の HTTP プロトコルバージョン 1.1 が使用されます。

3.21.5 END_REQUEST

END_REQUEST プロシージャは HTTP 要求を終了します。END_REQUEST 手順を使用して、HTTP 要求を完了せずに応答を待機せずに終了します。通常のプロセスでは、リクエストを開始し、レスポンスを取得してからレスポンスを閉じます。署名は次のとおりです。

`END_REQUEST(<r> IN OUT UTL_HTTP.REQ)`

Parameters

`r`

`r` は HTTP 要求レコードです。

3.21.6 END_RESPONSE

END_RESPONSE プロシージャは HTTP 応答を終了します。END_RESPONSE プロシージャは、HTTP 要求と応答を完了します。これは、要求および応答プロセスを終了する通常の方法です。署名は次のとおりです。

`END_RESPONSE(<r> IN OUT UTL_HTTP.RESP)`

Parameters

`r`

`r` は HTTP 応答レコードです。

3.21.7 GET_BODY_CHARSET

GET_BODY_CHARSET プログラムは、手続きと関数の両方の形式で利用できます。GET_BODY_CHARSET を呼び出すと、将来の HTTP リクエストの本文のデフォルトの文字セットが返されます。

プロシージャのシグネチャは次のとおりです。

```
GET_BODY_CHARSET(<charset> OUT VARCHAR2)
```

関数のシグネチャは次のとおりです。

```
GET_BODY_CHARSET() RETURN VARCHAR2
```

この関数は VARCHAR2 値を返します。

Parameters

charset

charset は本文の文字セットです。

Examples

以下は GET_BODY_CHARSET 関数の例です。

```
edb=# SELECT UTL_HTTP.GET_BODY_CHARSET() FROM DUAL;
get_body_charset
-----
ISO-8859-1
(1 row)
```

3.21.8 GET_FOLLOW_REDIRECT

GET_FOLLOW_REDIRECT プロシージャは、許可されるリダイレクトの最大数の現在の設定を返します。署名は次のとおりです。

```
GET_FOLLOW_REDIRECT(<max_redirects> OUT INTEGER)
```

Parameters

max_redirects

max_redirects は、許可されるリダイレクトの最大数です。

3.21.9 GET_HEADER

GET_HEADER プロシージャは、HTTP 応答の nth ヘッダーを返します。署名は次のとおりです。

```
GET_HEADER(<r> IN OUT UTL_HTTP.RESP, <n> INTEGER, <name> OUT
VARCHAR2, <value> OUT VARCHAR2)
```

Parameters

r

r は HTTP 応答レコードです。

n

n は、取得する HTTP 応答レコードの nth ヘッダーです。

name

name は応答ヘッダーの名前です。

value

value は応答ヘッダーの値です。

Examples

次の例では、ヘッダーカウントを取得してからヘッダーを取得します。

```
DECLARE
    v_req          UTL_HTTP.REQ;
    v_resp         UTL_HTTP.RESP;
    v_name         VARCHAR2(30);
    v_value        VARCHAR2(200);
    v_header_cnt   INTEGER;
BEGIN
    -- Initiate request and get response
    v_req := UTL_HTTP.BEGIN_REQUEST('www.enterprisedb.com');
    v_resp := UTL_HTTP.GET_RESPONSE(v_req);

    -- Get header count
    v_header_cnt := UTL_HTTP.GET_HEADER_COUNT(v_resp);
    DBMS_OUTPUT.PUT_LINE('Header Count: ' || v_header_cnt);

    -- Get all headers
    FOR i IN 1 .. v_header_cnt LOOP
        UTL_HTTP.GET_HEADER(v_resp, i, v_name, v_value);
        DBMS_OUTPUT.PUT_LINE(v_name || ': ' || v_value);
    END LOOP;
```

(次のページに続く)

(前のページからの続き)

```
-- Terminate request
    UTL_HTTP.END_RESPONSE(v_resp);
END;
```

以下は、この例の出力です。

```
Header Count: 23
Age: 570
Cache-Control: must-revalidate
Content-Type: text/html; charset=utf-8
Date: Wed, 30 Apr 2015 14:57:52 GMT
ETag: "aab02f2bd2d696eed817ca89ef411dda"
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Last-Modified: Wed, 30 Apr 2015 14:15:49 GMT
RTSS: 1-1307-3
Server: Apache/2.2.3 (Red Hat)
Set-Cookie:
SESS2771d0952de2a1a84d322a262e0c173c=jnluljletmdi5gg4lh8hakvs01;
expires=Fri, 23-May-2015 18:21:43 GMT; path=/; domain=.enterprisedb.com
Vary: Accept-Encoding
Via: 1.1 varnish
X-EDB-Backend: ec
X-EDB-Cache: HIT
X-EDB-Cache-Address: 10.31.162.212
X-EDB-Cache-Server: ip-10-31-162-212
X-EDB-Cache-TTL: 600.000
X-EDB-Cacheable: MAYBE: The user has a cookie of some sort. Maybe it's
double choc-chip!
X-EDB-Do-GZIP: false
X-Powered-By: PHP/5.2.17
X-Varnish: 484508634 484506789
transfer-encoding: chunked
Connection: keep-alive
```

3.21.10 GET_HEADER_BY_NAME

GET_HEADER_BY_NAME プロシージャは、指定された名前に従って HTTP 応答のヘッダーを返します。署名は次のとおりです。

```
GET_HEADER_BY_NAME(<r> IN OUT UTL_HTTP.RESP, <name> VARCHAR2,
<value> OUT VARCHAR2, <n> INTEGER DEFAULT 1)
```

Parameters

r

`r` は HTTP 応答レコードです。

`name`

`name` は取得する応答ヘッダーの名前です。

`value`

`value` は応答ヘッダーの値です。

`n`

`n` は、`name` で指定された値に従って取得する HTTP 応答レコードの `nth` ヘッダーです。デフォルトは 1 です。

Examples

次の例では、`Content-Type` のヘッダーを取得します。

```
DECLARE
    v_req          UTL_HTTP.REQ;
    v_resp         UTL_HTTP.RESP;
    v_name         VARCHAR2(30) := 'Content-Type';
    v_value        VARCHAR2(200);
BEGIN
    v_req := UTL_HTTP.BEGIN_REQUEST('www.enterprisedb.com');
    v_resp := UTL_HTTP.GET_RESPONSE(v_req);
    UTL_HTTP.GET_HEADER_BY_NAME(v_resp, v_name, v_value);
    DBMS_OUTPUT.PUT_LINE(v_name || ': ' || v_value);
    UTL_HTTP.END_RESPONSE(v_resp);
END;
```

Content-Type: text/html; charset=utf-8

3.21.11 GET_HEADER_COUNT

`GET_HEADER_COUNT` 関数は、HTTP 応答ヘッダーの数を返します。署名は次のとおりです。

```
GET_HEADER_COUNT(<r> IN OUT UTL_HTTP.RESP) RETURN INTEGER
```

この関数は `INTEGER` 値を返します。

Parameters

`r`

`r` は HTTP 応答レコードです。

3.21.12 GET_RESPONSE

GET_RESPONSE 関数はネットワーク要求を送信し、HTTP 応答を返します。署名は次のとおりです。

```
GET_RESPONSE(<r> IN OUT UTL_HTTP.REQ) RETURN UTL_HTTP.RESP
```

この関数は UTL_HTTP.RESP レコードを返します。

Parameters

r

r は HTTP 要求レコードです。

3.21.13 GET_RESPONSE_ERROR_CHECK

GET_RESPONSE_ERROR_CHECK プロシージャは、応答エラーチェックが設定されているかどうかを返します。署名は次のとおりです。

```
GET_RESPONSE_ERROR_CHECK(<enable> OUT BOOLEAN)
```

Parameters

enable

応答エラーチェックが設定されている場合、enable は TRUE を返し、そうでない場合は FALSE を返します。

3.21.14 GET_TRANSFER_TIMEOUT

GET_TRANSFER_TIMEOUT プロシージャは、HTTP 要求の現在のデフォルトの転送タイムアウト設定を返します。署名は次のとおりです。

```
GET_TRANSFER_TIMEOUT(<timeout> OUT INTEGER)
```

Parameters

timeout

timeout は、秒単位の転送タイムアウト設定です。

3.21.15 READ_LINE

READ_LINE プロシージャは、行の終わりに達するまでテキスト形式で HTTP 応答本文からデータを返します。CR 文字、LF 文字、CR LF シーケンス、または応答本文の終わりが行の終わりを構成します。署名は次のとおりです。

```
READ_LINE(<r> IN OUT UTL_HTTP.RESP, <data> OUT VARCHAR2,
<remove_crlf> BOOLEAN DEFAULT FALSE)
```

Parameters

r

r は HTTP 応答レコードです。

data

data はテキスト形式の応答本文です。

remove_crlf

改行文字を削除するには remove_crlf を TRUE に設定し、そうでない場合は FALSE に設定します。デフォルトは FALSE です。

Examples

次の例では、指定した Web サイトの本文を取得して表示します。

```
DECLARE
    v_req          UTL_HTTP.REQ;
    v_resp         UTL_HTTP.RESP;
    v_value        VARCHAR2(1024);
BEGIN
    v_req := UTL_HTTP.BEGIN_REQUEST('http://www.enterprisedb.com');
    v_resp := UTL_HTTP.GET_RESPONSE(v_req);
    LOOP
        UTL_HTTP.READ_LINE(v_resp, v_value, TRUE);
        DBMS_OUTPUT.PUT_LINE(v_value);
    END LOOP;
EXCEPTION
    WHEN OTHERS THEN
        UTL_HTTP.END_RESPONSE(v_resp);
END;
```

出力は次のとおりです。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

(次のページに続く)

(前のページからの続き)

```

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en" dir="ltr">

  <!-- _____ HEAD _____ -->

  <head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

    <title>EnterpriseDB | The Postgres Database Company</title>

    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta name="keywords" content="postgres, postgresql, postgresql installer,
mysql migration, open source database, training, replication" />
<meta name="description" content="The leader in open source database
products, services, support, training and expertise based on PostgreSQL.
Free downloads, documentation, and tutorials." />
<meta name="abstract" content="The Enterprise PostgreSQL Company" />
<link rel="EditURI" type="application/rsd+xml" title="RSD" href="http://
www.enterprisedb.com/blogapi/rsd" />
<link rel="alternate" type="application/rss+xml" title="EnterpriseDB RSS"
href="http://www.enterprisedb.com/rss.xml" />
<link rel="shortcut icon" href="/sites/all/themes/edb_pixelcrayons/
favicon.ico" type="image/x-icon" />
    <link type="text/css" rel="stylesheet" media="all" href="/sites/default/
files/css/css_db1ladabae0aed6b79a2c3c52def4754.css" />
<!--[if IE 6]>
<link type="text/css" rel="stylesheet" media="all" href="/sites/all/themes/
oho_basic/css/ie6.css?g" />
<![endif]-->
<!--[if IE 7]>
<link type="text/css" rel="stylesheet" media="all" href="/sites/all/themes/
oho_basic/css/ie7.css?g" />
<![endif]-->
    <script type="text/javascript" src="/sites/default/files/js/
js_74d97b1176812e2fd6e43d62503a5204.js"></script>
<script type="text/javascript">
<!--//--><![CDATA[//><!--

```

3.21.16 READ_RAW

READ_RAW プロシージャは、HTTP 応答本文からデータをバイナリ形式で返します。返されるバイト数は、len パラメーターで指定されます。署名は次のとおりです。

```
READ_RAW(<r> IN OUT UTL_HTTP.RESP, <data> OUT RAW, <len> INTEGER)
```

Parameters

3.21. UTL_HTTP

`r`

`r` は HTTP 応答レコードです。

`data`

`data` は、バイナリ形式の応答本文です。

`len`

返されるデータのバイト数に `len` を設定します。

Examples

次の例では、最初の 150 バイトをバイナリ形式で取得して表示します。

```
DECLARE
    v_req          UTL_HTTP.REQ;
    v_resp         UTL_HTTP.RESP;
    v_data         RAW;
BEGIN
    v_req := UTL_HTTP.BEGIN_REQUEST('http://www.enterprisedb.com');
    v_resp := UTL_HTTP.GET_RESPONSE(v_req);
    UTL_HTTP.READ_RAW(v_resp, v_data, 150);
    DBMS_OUTPUT.PUT_LINE(v_data);
    UTL_HTTP.END_RESPONSE(v_resp);
END;
```

以下は、この例の出力です。

```
\x3c21444f43545950452068746d6c205055424c494320222d2f2f5733432f2f4454442058485
44d4c20312e30205374726963742f2f454e220d0a20222687474703a2f2f7777772e77332e6f
72672f54522f7868746d6c312f4454442f7868746d6c312d7374726963742e647464223e0d0a3
c68746d6c20786d6c6e733d22687474703a2f2f7777772e77332e6f72672f313939392f
```

3.21.17 READ_TEXT

`READ_TEXT` プロシージャは、HTTP 応答本文からデータをテキスト形式で返します。返される文字の最大数は、`len` パラメーターで指定されます。署名は次のとおりです。

```
READ_TEXT(<r> IN OUT UTL_HTTP.RESP, <data> OUT VARCHAR2, <len> INTEGER)
```

Parameters

`r`

`r` は HTTP 応答レコードです。

data

data はテキスト形式の応答本文です。

len

len を返される最大文字数に設定します。

Examples

次の例では、最初の 150 文字を取得します。

```
DECLARE
    v_req          UTL_HTTP.REQ;
    v_resp         UTL_HTTP.RESP;
    v_data         VARCHAR2(150);
BEGIN
    v_req := UTL_HTTP.BEGIN_REQUEST('http://www.enterprisedb.com');
    v_resp := UTL_HTTP.GET_RESPONSE(v_req);
    UTL_HTTP.READ_TEXT(v_resp, v_data, 150);
    DBMS_OUTPUT.PUT_LINE(v_data);
    UTL_HTTP.END_RESPONSE(v_resp);
END;
```

出力は次のとおりです。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/
```

3.21.18 REQUEST

REQUEST 関数は、ユーザー指定の URL から取得した最初の 2000 バイトを返します。署名は次のとおりです。

```
REQUEST(<url> IN VARCHAR2) RETURN VARCHAR2
```

指定された URL で見つかったデータが 2000 バイトより長い場合、残りは破棄されます。指定された URL で見つかったデータが 2000 バイトより短い場合、結果は 2000 バイトより短くなります。

Parameters

url

url は UTL_HTTP がコンテンツを返す UniformResourceLocator です。

Example

次のコマンドは、EDBWeb サイトから取得した最初の 2000 バイトを返します。

```
SELECT UTL_HTTP.REQUEST('http://www.enterprisedb.com/') FROM DUAL;
```

3.21.19 REQUEST_PIECES

REQUEST_PIECES 関数は、URL から取得した 2000 バイトのセグメントのテーブルを返します。署名は次のとおりです。

```
REQUEST_PIECES(<url> IN VARCHAR2, <max_pieces> NUMBER IN  
DEFAULT 32767) RETURN UTL_HTTP.HTML_PIECES
```

Parameters

url

url は UTL_HTTP がコンテンツを返す UniformResourceLocator です。

max_pieces

max_pieces は、REQUEST_PIECES 関数が返す 2000 バイトセグメントの最大数を指定します。max_pieces で指定された url で使用可能な単位よりも多くの単位を指定すると、最終単位のバイト数が少なくなります。

Example

次の例は、EDBWeb サイトから取得した最初の 4 つの 2000 バイトセグメントを返します。

```
DECLARE  
    result UTL_HTTP.HTML_PIECES;  
BEGIN  
result := UTL_HTTP.REQUEST_PIECES('http://www.enterprisedb.com/', 4);  
END
```

3.21.20 SET_BODY_CHARSET

SET_BODY_CHARSET プロシージャは、将来の HTTP リクエストの本文のデフォルト文字セットを設定します。署名は次のとおりです。

```
SET_BODY_CHARSET(<charset> VARCHAR2 DEFAULT NULL)
```

Parameters

charset

charset は、将来のリクエストの本文の文字セットです。デフォルトはヌルです。この場合、データベースの文字セットが想定されます。

3.21.21 SET_FOLLOW_REDIRECT

SET_FOLLOW_REDIRECT プロシージャは、この要求または将来の要求への応答で HTTP リダイレクト命令に従う最大回数を設定します。この手順には 2 つのシグネチャがあります。

```
SET_FOLLOW_REDIRECT(<max_redirects> IN INTEGER DEFAULT 3)
```

そして

```
SET_FOLLOW_REDIRECT(<r> IN OUT UTL_HTTP.REQ, <max_redirects>  
IN INTEGER DEFAULT 3)``
```

2 番目のフォームを使用して、リクエストがセッションのデフォルト設定から継承する個々のリクエストのリダイレクトの最大数を変更します。

Parameters

r

r は HTTP 要求レコードです。

max_redirects

max_redirects は、許可されるリダイレクトの最大数です。0 に設定すると、リダイレクトが無効になります。デフォルトは 3 です。

3.21.22 SET_HEADER

SET_HEADER プロシージャは、HTTP 要求ヘッダーを設定します。署名は次のとおりです。

```
SET_HEADER(<r> IN OUT UTL_HTTP.REQ, <name> IN VARCHAR2, <value>  
IN VARCHAR2 DEFAULT NULL)
```

Parameters

r

r は HTTP 要求レコードです。

name

name はリクエストヘッダーの名前です。

value

value はリクエストヘッダーの値です。デフォルトは `null` です。

3.21.23 SET_RESPONSE_ERROR_CHECK

SET_RESPONSE_ERROR_CHECK プロシージャは、GET_RESPONSE 関数によって返された HTTP4xx および 5xx ステータスコードをエラーとして解釈するかどうかを決定します。署名は次のとおりです。

```
SET_RESPONSE_ERROR_CHECK(<enable> IN BOOLEAN DEFAULT FALSE)
```

Parameters

enable

HTTP4xx および 5xx ステータスコードをエラーとして扱う場合は enable を TRUE に設定し、そうでない場合は FALSE に設定します。デフォルトは FALSE です。

3.21.24 SET_TRANSFER_TIMEOUT

SET_TRANSFER_TIMEOUT プロシージャは、HTTP 要求からの応答を待機するためのデフォルトの転送タイムアウト設定を設定します。この手順には 2 つのシグネチャがあります。

```
SET_TRANSFER_TIMEOUT(<timeout> IN INTEGER DEFAULT 60)
```

そして

```
SET_TRANSFER_TIMEOUT(<r> IN OUT UTL_HTTP.REQ, <timeout> IN  
INTEGER DEFAULT 60)
```

2 番目のフォームを使用して、リクエストがセッションのデフォルト設定から継承する個々のリクエストの転送タイムアウト設定を変更します。

Parameters

r

r は HTTP 要求レコードです。

timeout

timeout は、HTTP 要求の秒単位の転送タイムアウト設定です。デフォルトは 60 秒です。

3.21.25 WRITE_LINE

WRITE_LINE プロシージャは、データをテキスト形式で HTTP リクエスト本文に書き込みます。テキストは CRLF 文字ペアで終了します。署名は次のとおりです。

```
WRITE_LINE(<r> IN OUT UTL_HTTP.REQ, <data> IN VARCHAR2)
```

Parameters

r

r は HTTP 要求レコードです。

data

data は TEXT 形式のリクエスト本文です。

Example

次の例では、HTTP (1) メソッドを使用して、送信するデータ (Account balance \$500.00) をテキスト形式で要求本文に書き込みます。データは、データを受け入れて処理する仮定の Web アプリケーション (post.php) に送信されます。

```
DECLARE
    v_req          UTL_HTTP.REQ;
    v_resp         UTL_HTTP.RESP;
BEGIN
    v_req := UTL_HTTP.BEGIN_REQUEST('http://www.example.com/post.php',
    'POST');
    UTL_HTTP.SET_HEADER(v_req, 'Content-Length', '23');
    UTL_HTTP.WRITE_LINE(v_req, 'Account balance $500.00');
    v_resp := UTL_HTTP.GET_RESPONSE(v_req);
    DBMS_OUTPUT.PUT_LINE('Status Code: ' || v_resp.status_code);
    DBMS_OUTPUT.PUT_LINE('Reason Phrase: ' || v_resp.reason_phrase);
    UTL_HTTP.END_RESPONSE(v_resp);
END;
```

Web アプリケーションが POST メソッドを正常に処理すると仮定すると、次の出力が表示されます。

```
Status Code: 200
Reason Phrase: OK
```

3.21.26 WRITE_RAW

WRITE_RAW プロシージャは、データをバイナリ形式で HTTP リクエスト本文に書き込みます。署名は次のとおりです。

```
WRITE_RAW(<r> IN OUT UTL_HTTP.REQ, <data> IN RAW)
```

Parameters

r

r は HTTP 要求レコードです。

data

data はバイナリ形式のリクエスト本文です。

Example

次の例では、データをバイナリ形式でリクエスト本文に書き込み、HTTP POST メソッドを使用して、そのようなデータを受け入れて処理する仮想 Web アプリケーションに送信します。

```
DECLARE
    v_req          UTL_HTTP.REQ;
    v_resp         UTL_HTTP.RESP;
BEGIN
    v_req := UTL_HTTP.BEGIN_REQUEST('http://www.example.com/post.php',
    'POST');
    UTL_HTTP.SET_HEADER(v_req, 'Content-Length', '23');
    UTL_HTTP.WRITE_RAW(v_req, HEXTORAW
('54657374696e6720504f5354206d6574686f6420696e20485454502072657175657374'));
    v_resp := UTL_HTTP.GET_RESPONSE(v_req);
    DBMS_OUTPUT.PUT_LINE('Status Code: ' || v_resp.status_code);
    DBMS_OUTPUT.PUT_LINE('Reason Phrase: ' || v_resp.reason_phrase);
    UTL_HTTP.END_RESPONSE(v_resp);
END;
```

HEXTORAW 関数に表示されるテキスト文字列は、テキスト Testing POST method in HTTP request の 16 進変換です。

Web アプリケーションが POST メソッドを正常に処理すると仮定すると、次の出力が表示されます。

```
Status Code: 200
Reason Phrase: OK
```

3.21.27 WRITE_TEXT

WRITE_TEXT プロシージャは、データをテキスト形式で HTTP リクエスト本文に書き込みます。署名は次のとおりです。

```
WRITE_TEXT(<r> IN OUT UTL_HTTP.REQ, <data> IN VARCHAR2)
```

Parameters

r

r は HTTP 要求レコードです。

data

data はテキスト形式のリクエスト本文です。

Example

次の例では、HTTP (1) メソッドを使用して、送信するデータ (Account balance \$500.00) をテキスト形式で要求本文に書き込みます。データは、データを受け入れて処理する仮定の Web アプリケーション (post.php) に送信されます。

```
DECLARE
    v_req          UTL_HTTP.REQ;
    v_resp         UTL_HTTP.RESP;
BEGIN
    v_req := UTL_HTTP.BEGIN_REQUEST('http://www.example.com/post.php',
    'POST');
    UTL_HTTP.SET_HEADER(v_req, 'Content-Length', '23');
    UTL_HTTP.WRITE_TEXT(v_req, 'Account balance $500.00');
    v_resp := UTL_HTTP.GET_RESPONSE(v_req);
    DBMS_OUTPUT.PUT_LINE('Status Code: ' || v_resp.status_code);
    DBMS_OUTPUT.PUT_LINE('Reason Phrase: ' || v_resp.reason_phrase);
    UTL_HTTP.END_RESPONSE(v_resp);
END;
```

Web アプリケーションが POST メソッドを正常に処理すると仮定すると、次の出力が表示されます。

```
Status Code: 200
Reason Phrase: OK
```

3.21.28 END_OF_BODY

HTTP 応答本文の最後に到達すると、END_OF_BODY 例外が発生します。

Example

次の例では、例外を処理し、HTTP POST メソッドを使用して送信されるテキスト形式の Exception caught をリクエスト本文に書き込みます。データは、データを受け入れて処理する仮想の Web アプリケーション（post.php）に送信されます。

```
DECLARE
    req UTL_HTTP.REQ;
    resp UTL_HTTP.RESP;
    value VARCHAR2(32768);
BEGIN
    req := UTL_HTTP.BEGIN_REQUEST('https://www.google.com/');
    resp := UTL_HTTP.GET_RESPONSE(req);
LOOP
    UTL_HTTP.READ_LINE(resp, value, TRUE);
END LOOP;
    UTL_HTTP.END_RESPONSE(resp);
EXCEPTION
    WHEN UTL_HTTP.END_OF_BODY THEN
        DBMS_OUTPUT.PUT_LINE('Exception caught');
        UTL_HTTP.END_RESPONSE(resp);
END;
```

Web アプリケーションが POST メソッドを正常に処理したと仮定すると、次の出力が表示されます。

```
Output is:
Exception caught
```

3.22 UTL_MAIL

UTL_MAIL パッケージは、電子メールを管理する機能を提供します。AdvancedServer は、次の手順をサポートしています。

機能/手順	戻りタイプ	説明
<code>SEND(sender, recipients, cc, bcc, subject, message [, mime_type [, priority]])</code>	該当なし	電子メールをパッケージ化し、SMTP サーバーに送信します。
<code>SEND_ATTACH_RAW(sender, recipients, cc, bcc, subject, message, mime_type, priority, attachment [, att_inline [, att_mime_type [, att_filename]]])</code>	該当なし	SEND 手順と同じですが、BYTEA または大きなオブジェクトが添付されています。
<code>SEND_ATTACH_VARCHAR2(sender, recipients, cc, bcc, subject, message, mime_type, priority, attachment [, att_inline [, att_mime_type [, att_filename]]])</code>	該当なし	SEND 手順と同じですが、VARCHAR2 の添付ファイルがあります。

注釈: 管理者は、このパッケージを使用する前に、各ユーザーまたはグループに実行権限を付与する必要があります。

3.22.1 SEND

SEND 手順は、SMTP サーバーに電子メールを送信する機能を提供します。

```
SEND(<sender> VARCHAR2, <recipients> VARCHAR2, <cc> VARCHAR2,  
     <bcc> VARCHAR2, <subject> VARCHAR2, <message> VARCHAR2  
     [, <mime_type> VARCHAR2 [, <priority> PLS_INTEGER ]])
```

Parameters

sender

送信者の電子メールアドレス。

recipients

受信者のコンマ区切りの電子メールアドレス。

cc

コピー受信者のコンマ区切りの電子メールアドレス。

bcc

ブラインドコピー受信者のコンマ区切りの電子メールアドレス。

subject

電子メールの件名。

message

電子メールの本文。

mime_type

メッセージの MIME タイプ。デフォルトは text/plain; charset=us-ascii です。

priority

電子メールの優先度デフォルトは 3 です。

Examples

次の匿名ブロックは、単純な電子メールメッセージを送信します。

```
DECLARE  
    v_sender      VARCHAR2(30);  
    v_recipients  VARCHAR2(60);  
    v_subj        VARCHAR2(20);  
    v_msg         VARCHAR2(200);
```

(次のページに続く)

(前のページからの続き)

```

BEGIN
    v_sender := 'jsmith@enterprisedb.com';
    v_recipients := 'ajones@enterprisedb.com,rrogers@enterprisedb.com';
    v_subj := 'Holiday Party';
    v_msg := 'This year's party is scheduled for Friday, Dec. 21 at ' ||
            '6:00 PM. Please RSVP by Dec. 15th.';
    UTL_MAIL.SEND(v_sender,v_recipients,NULL,NULL,v_subj,v_msg);
END;

```

3.22.2 SEND_ATTACH_RAW

SEND_ATTACH_RAW 手順は、BYTEA データまたはラージオブジェクト（ラージオブジェクトの OID で識別される）のいずれかを含む添付ファイル付きの SMTP サーバーに電子メールを送信する機能を提供します。SEND_ATTACH_RAW の呼び出しは、次の 2 つの方法で記述できます。

```

SEND_ATTACH_RAW(<sender> VARCHAR2, <recipients> VARCHAR2,
    <cc> VARCHAR2, <bcc> VARCHAR2, <subject> VARCHAR2, <message> VARCHAR2,
    <mime_type> VARCHAR2, <priority> PLS_INTEGER,
    <attachment> BYTEA[, <att_inline> BOOLEAN
    [, <att_mime_type> VARCHAR2[, <att_filename> VARCHAR2 ]]])

```

または

```

SEND_ATTACH_RAW(<sender> VARCHAR2, <recipients> VARCHAR2,
    <cc> VARCHAR2, <bcc> VARCHAR2, <subject> VARCHAR2, <message> VARCHAR2,
    <mime_type> VARCHAR2, <priority> PLS_INTEGER, <attachment> OID
    [, <att_inline> BOOLEAN [, <att_mime_type> VARCHAR2
    [, <att_filename> VARCHAR2 ]]])

```

Parameters

sender

送信者の電子メールアドレス。

recipients

受信者のコンマ区切りの電子メールアドレス。

cc

コピー受信者のコンマ区切りの電子メールアドレス。

bcc

ブラインドコピー受信者のコンマ区切りの電子メールアドレス。

subject

電子メールの件名。

message

電子メールの本文。

mime_type

メッセージの **MIME** タイプ。デフォルトは `text/plain; charset=us-ascii` です。

priority

電子メールの優先度。デフォルトは 3 です。

attachment

添付ファイル。

att_inline

TRUE に設定すると、添付ファイルはインラインで表示され、それ以外の場合は FALSE になります。デフォルトは TRUE です。

att_mime_type

添付ファイルの **MIME** タイプ。デフォルトは `application/octet` です。

att_filename

添付ファイルを含むファイル名。デフォルトは NULL です。

3.22.3 SEND_ATTACH_VARCHAR2

SEND_ATTACH_VARCHAR2 手順は、テキスト添付ファイル付きの **SMTP** サーバーに電子メールを送信する機能を提供します。

```
SEND_ATTACH_VARCHAR2(<sender> VARCHAR2, <recipients> VARCHAR2, <cc>
VARCHAR2, <bcc> VARCHAR2, <subject> VARCHAR2, <message> VARCHAR2,
<mime_type> VARCHAR2, <priority> PLS_INTEGER, <attachment> VARCHAR2 [,
<att_inline> BOOLEAN [, <att_mime_type> VARCHAR2 [, <att_filename> VARCHAR2
]]])
```

Parameters

sender

送信者の電子メールアドレス。

recipients

受信者のコンマ区切りの電子メールアドレス。

cc

コピー受信者のコンマ区切りの電子メールアドレス。

bcc

ブラインドコピー受信者のコンマ区切りの電子メールアドレス。

subject

電子メールの件名。

message

電子メールの本文。

mime_type

メッセージの **MIME** タイプ。デフォルトは `text/plain; charset=us-ascii` です。

priority

電子メールの優先度デフォルトは 3. です

attachment

VARCHAR2 添付ファイル。

att_inline

TRUE に設定すると、添付ファイルはインラインで表示され、それ以外の場合は FALSE になります。デフォルトは TRUE です。

att_mime_type

添付ファイルの **MIME** タイプ。デフォルトは `text/plain; charset=us-ascii` です。

att_filename

添付ファイルを含むファイル名。デフォルトは NULL です。

3.23 UTL_RAW

UTL_RAW パッケージを使用すると、生データ型の長さを操作または取得できます。

注釈: 管理者は、このパッケージを使用する前に、各ユーザーまたはグループに実行権限を付与する必要があります。

機能/手順	機能または手順	戻りタイプ	説明
CAST_TO_RAW(c IN VARCHAR2)	機能	RAW	VARCHAR2 文字列を RAW 値に変換します。
CAST_TO_VARCHAR2(r IN RAW)	機能	VARCHAR2	RAW 値を VARCHAR2 文字列に変換します。
CONCAT(r1 IN RAW, r2 IN RAW, r3 IN RAW, ...)	機能	RAW	複数の RAW 値を単一の RAW 値に連結します。
CONVERT(r IN RAW, to_charset IN VARCHAR2, from_charset IN VARCHAR2)	機能	RAW	エンコードされたデータのあるエンコードから別のエンコードに変換し、結果を RAW 値として返します。
LENGTH(r IN RAW)	機能	NUMBER	RAW 値の長さを返します。
SUBSTR(r IN RAW, pos IN INTEGER, len IN INTEGER)	機能	RAW	RAW 値の一部を返します。

AdvancedServer の UTL_RAW の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表にリストされている関数と手順のみがサポートされています。

3.23.1 CAST_TO_RAW

CAST_TO_RAW 関数は VARCHAR2 文字列を RAW 値に変換します。署名は次のとおりです。

```
CAST_TO_RAW(<c> VARCHAR2)
```

NULL 以外の値を渡すと、関数は RAW 値を返します。NULL 値を渡すと、関数は NULL を返します。

Parameters

c

RAW に変換される VARCHAR2 値。

Example

次の例では、CAST_TO_RAW 関数を使用して VARCHAR2 文字列を RAW 値に変換します。

```
DECLARE
  v VARCHAR2;
  r RAW;
BEGIN
  v := 'Accounts';
  dbms_output.put_line(v);
  r := UTL_RAW.CAST_TO_RAW(v);
  dbms_output.put_line(r);
END;
```

結果セットには、元の文字列のコンテンツと変換された RAW 値が含まれます。

```
Accounts
\x4163636f756e7473
```

3.23.2 CAST_TO_VARCHAR2

CAST_TO_VARCHAR2 関数は RAW データを VARCHAR2 データに変換します。署名は次のとおりです。

```
CAST_TO_VARCHAR2(<r> RAW)
```

NULL 以外の値を渡すと、関数は VARCHAR2 値を返します。NULL 値を渡すと、関数は NULL を返します。

Parameters

r

VARCHAR2 値に変換される RAW 値。

Example

次の例では、CAST_TO_VARCHAR2 関数を使用して RAW 値を VARCHAR2 文字列に変換します。

```
DECLARE
  r RAW;
  v VARCHAR2;
BEGIN
  r := '\x4163636f756e7473'
  dbms_output.put_line(v);
  v := UTL_RAW.CAST_TO_VARCHAR2(r);
  dbms_output.put_line(r);
END;
```

結果セットには、元の文字列のコンテンツと変換された RAW 値が含まれます。

```
\x4163636f756e7473
Accounts
```

3.23.3 CONCAT

CONCAT 関数は、複数の RAW 値を単一の RAW 値に連結します。署名は次のとおりです。

```
CONCAT(<r1> RAW, <r2> RAW, <r3> RAW, ...)
```

関数は RAW 値を返します。Oracle の実装とは異なり、AdvancedServer の実装は可変機能であり、連結できる値の数の制限はありません。

Parameters

r1, r2, r3, ...

CONCAT が連結する RAW 値。

Example

次の例では、CONCAT 関数を使用して、複数の RAW 値を単一の RAW 値に連結します。

```
SELECT UTL_RAW.CAST_TO_VARCHAR2(UTL_RAW.CONCAT('\x61', '\x62', '\x63')) FROM
DUAL;
  concat
-----
  abc
(1 row)
```

結果（連結された値）は、CAST_TO_VARCHAR2 関数によって VARCHAR2 形式に変換されます。

3.23.4 CONVERT

CONVERT 関数は、あるエンコーディングから別のエンコーディングに文字列を変換し、結果を RAW 値として返します。署名は次のとおりです。

```
CONVERT(<r> RAW, <to_charset> VARCHAR2, <from_charset> VARCHAR2)
```

関数は RAW 値を返します。

Parameters

r

変換される RAW 値。

to_charset

r が変換されるエンコーディングの名前。

from_charset

r の変換元となるエンコーディングの名前。

Example

次の例では、UTL_RAW.CAST_TO_RAW 関数を使用して VARCHAR2 文字列（Accounts）を生の値に変換し、値を UTF8 から LATIN7 に変換し、次に LATIN7 から UTF8 に変換します：

```
DECLARE
    r RAW;
    v VARCHAR2;
BEGIN
    v:= 'Accounts';
    dbms_output.put_line(v);
    r:= UTL_RAW.CAST_TO_RAW(v);
    dbms_output.put_line(r);
    r:= UTL_RAW.CONVERT(r, 'UTF8', 'LATIN7');
    dbms_output.put_line(r);
    r:= UTL_RAW.CONVERT(r, 'LATIN7', 'UTF8');
    dbms_output.put_line(r);
```

この例は、VARCHAR2 値、RAW 値、および変換された値を返します。

```
Accounts
\x4163636f756e7473
\x4163636f756e7473
\x4163636f756e7473
```

3.23.5 LENGTH

LENGTH 関数は、RAW 値の長さを返します。署名は次のとおりです。

```
LENGTH (<r> RAW)
```

関数は RAW 値を返します。

Parameters

r

LENGTH が評価する RAW 値。

Example

次の例では、LENGTH 関数を使用して RAW 値の長さを返します。

```
SELECT UTL_RAW.LENGTH (UTL_RAW.CAST_TO_RAW ('Accounts')) FROM DUAL;
length
-----
      8
(1 row)
```

次の例では、LENGTH 関数を使用して、マルチバイト文字を含む RAW 値の長さを返します。

```
SELECT UTL_RAW.LENGTH (UTL_RAW.CAST_TO_RAW ('hello'));
length
-----
     12
(1 row)
```

3.23.6 SUBSTR

SUBSTR 関数は、RAW 値の部分文字列を返します。署名は次のとおりです。

```
SUBSTR (<r> RAW, <pos> INTEGER, <len> INTEGER)
```

この関数は RAW 値を返します。

Parameters

r

部分文字列が返される RAW 値。

pos

返された部分文字列の最初のバイトの RAW 値内の位置。

- pos が 0 または 1 の場合、サブストリングは RAW 値の最初のバイトから始まります。
- pos が 1 より大きい場合、サブストリングは pos で指定された最初のバイトから始まります。たとえば、pos が 3 の場合、サブストリングは値の 3 バイト目から始まります。
- pos が負の場合、部分文字列はソース値の末尾から pos バイトで始まります。たとえば、pos が -3 の場合、部分文字列は値の末尾から 3 バイト目から始まります。

len

返される最大バイト数。

Example

次の例では、SUBSTR 関数を使用して、RAW 値の先頭から 3 バイトで始まる部分文字列を選択します。

```
SELECT UTL_RAW.SUBSTR(UTL_RAW.CAST_TO_RAW('Accounts'), 3, 5) FROM DUAL;
 substr
-----
count
(1 row)
```

次の例では、SUBSTR 関数を使用して、RAW 値の末尾から 5 バイトで始まる部分文字列を選択します。

```
SELECT UTL_RAW.SUBSTR(UTL_RAW.CAST_TO_RAW('Accounts'), -5 , 3) FROM DUAL;
 substr
-----
oun
(1 row)
```

3.24 UTL_SMTP

UTL_SMTP パッケージは、SMTP（SimpleMailTransferProtocol）を介して電子メールを送信する機能を提供します。

注釈: 管理者は、このパッケージを使用する前に、各ユーザーまたはグループに実行権限を付与する必要があります。

機能/手順	機能または手順	戻りタイプ	説明
CLOSE_DATA(c IN OUT)	手続き	該当なし	電子メールメッセージを終了します。
COMMAND(c IN OUT, cmd [, arg])	両方	REPLY	SMTP コマンドを実行します。
COMMAND_REPLIES(c IN OUT, cmd [, arg])	機能	REPLIES	複数の応答行が予想される場所で SMTP コマンドを実行します。
DATA(c IN OUT, body VARCHAR2)	手続き	該当なし	電子メールメッセージの本文を指定します。
EHLO(c IN OUT, domain)	手続き	該当なし	SMTP サーバーとの最初のハンドシェイクを実行し、拡張情報を返します。
HELO(c IN OUT, domain)	手続き	該当なし	SMTP サーバーで初期ハンドシェイクを実行する
HELP(c IN OUT [, command])	機能	REPLIES	HELP コマンドを送信します。
MAIL(c IN OUT, sender [, parameters])	手続き	該当なし	メールトランザクションを開始します。
NOOP(c IN OUT)	両方	REPLY	null コマンドを送信します。
OPEN_CONNECTION(host [, port [, tx_timeout]])	機能	CONNECTION	接続を開きます。
OPEN_DATA(c IN OUT)	両方	REPLY	DATA コマンドを送信します。
QUIT(c IN OUT)	手続き	該当なし	SMTP セッションを終了して切断します。
RCPT(c IN OUT, recipient [, parameters])	手続き	該当なし	電子メールメッセージの受信者を指定します。
RSET(c IN OUT)	手続き	該当なし	現在のメールトランザクションを終了します。
VRFY(c IN OUT, recipient)	機能	REPLY	電子メールアドレスを検証します。
WRITE_DATA(c IN OUT, data)	手続き	該当なし	電子メールメッセージの一部を書きます。

AdvancedServer の UTL_SMTP の実装は、Oracle のバージョンと比較すると部分的な実装です。上記の表にリストされている関数と手順のみがサポートされています。

次の表に、UTL_SMTP パッケージで使用可能なパブリック変数を示します。

パブリック変数	データ型	価値	説明
connection	RECORD		SMTP 接続の説明。
reply	RECORD		SMTP 返信行。

3.24.1 CONNECTION

CONNECTION レコードタイプは、SMTP 接続の説明を提供します。

```
TYPE connection IS RECORD (
    host          VARCHAR2(255),
    port          PLS_INTEGER,
    tx_timeout    PLS_INTEGER
);
```

3.24.2 REPLY/REPLIES

REPLY レコードタイプは、SMTP 応答行の説明を提供します。REPLIES は、複数の SMTP 応答行のテーブルです。

```
TYPE reply IS RECORD (
    code          INTEGER,
    text          VARCHAR2(508)
);
TYPE replies IS TABLE OF reply INDEX BY BINARY_INTEGER;
```

3.24.3 CLOSE_DATA

CLOSE_DATA プロシージャは、次のシーケンスを送信して電子メールメッセージを終了します。

```
<CR><LF>.<CR><LF>
```

これは、行の先頭にある単一の期間です。

```
CLOSE_DATA(<c> IN OUT CONNECTION)
```

Parameters

c

閉じられる SMTP 接続。

3.24.4 COMMAND

COMMAND プロシージャは、SMTP コマンドを実行する機能を提供します。複数の返信行が必要な場合は、COMMAND_REPLIES を使用します。

```
<reply> REPLY COMMAND(<c> IN OUT CONNECTION, <cmd> VARCHAR2  
    [, <arg> VARCHAR2 ])  
  
COMMAND(<c> IN OUT CONNECTION, <cmd> VARCHAR2 [, <arg> VARCHAR2 ])
```

Parameters

c

コマンドの送信先の SMTP 接続。

cmd

処理される SMTP コマンド。

arg

SMTP コマンドへの引数。デフォルトは null です。

reply

コマンドへの SMTP 応答。SMTP が複数の返信を返す場合、最後の返信のみが reply で返されます。

REPLY と REPLIES の説明については、[Reply/Replies](#) を参照してください。

3.24.5 COMMAND_REPLIES

COMMAND_REPLIES 関数は、複数の応答行を返す SMTP コマンドを処理します。応答行が 1 行のみである場合は、COMMAND を使用します。

```
<replies> REPLIES COMMAND(<c> IN OUT CONNECTION, <cmd> VARCHAR2  
    [, <arg> VARCHAR2 ])
```

Parameters

c

コマンドの送信先の SMTP 接続。

cmd

処理される SMTP コマンド。

arg

SMTP コマンドへの引数。デフォルトは `null` です。

`replies`

コマンドに対する SMTP 応答行。REPLY と REPLIES の説明については [Reply/Replies](#) を参照してください。

3.24.6 DATA

DATA プロシージャは、電子メールメッセージの本文を指定する機能を提供します。メッセージは `<CR><LF>`、`<CR><LF>` シーケンスで終了します。

```
DATA(<c> IN OUT CONNECTION, <body> VARCHAR2)
```

Parameters

`c`

コマンドの送信先の SMTP 接続。

`body`

送信する電子メールメッセージの本文。

3.24.7 EHLO

EHLO の手順は、接続の確立後に SMTP サーバーとの最初のハンドシェイクを実行します。EHLO 手順により、クライアントは RFC821 に従って SMTP サーバーに対して自分自身を識別できます。RFC1869 は、サーバーの応答で返される情報の形式を指定します。*HELO* プロシージャは同等の機能を実行しますが、サーバーに関する情報を返しません。

```
EHLO(<c> IN OUT CONNECTION, <domain> VARCHAR2)
```

Parameters

`c`

ハンドシェイクを実行する SMTP サーバーへの接続。

`domain`

送信ホストのドメイン名。

3.24.8 HELO

HELO の手順は、接続の確立後に SMTP サーバーとの最初のハンドシェイクを実行します。HELO プロシージャを使用すると、クライアントは RFC821 に従って SMTP サーバーに対して自身を識別できます。*EHLO* プロシージャは同等の機能を実行しますが、サーバーに関する詳細情報を返します。

```
HELO (<c> IN OUT, <domain*> VARCHAR2)
```

Parameters

c

ハンドシェイクを実行する SMTP サーバーへの接続。

domain

送信ホストのドメイン名。

3.24.9 HELP

HELP 機能は、HELP コマンドを SMTP サーバーに送信する機能を提供します。

```
<replies> REPLIES HELP(<c> IN OUT CONNECTION [, <command> VARCHAR2 ])
```

Parameters

c

コマンドの送信先の SMTP 接続。

command

ヘルプが要求されるコマンド。

replies

コマンドに対する SMTP 応答行。REPLY と REPLIES の説明については [Reply/Replies](#) を参照してください。

3.24.10 MAIL

MAIL プロシージャは、メールトランザクションを開始します。

```
MAIL(<c> IN OUT CONNECTION, <sender> VARCHAR2
[, <parameters> VARCHAR2 ])
```

Parameters

c

メールトランザクションを開始する SMTP サーバーへの接続。

sender

送信者の電子メールアドレス。

parameters

RFC1869 で定義されている key=value 形式のメールコマンドパラメーター。

3.24.11 NOOP

NOOP 関数/手順は、null コマンドを SMTP サーバーに送信します。NOOP は、成功した応答を取得することを除いて、サーバーに影響しません。

```
<reply> REPLY NOOP(<c> IN OUT CONNECTION)

NOOP(<c> IN OUT CONNECTION)
```

Parameters

c

コマンドを送信する SMTP 接続。

reply

コマンドへの SMTP 応答。SMTP が複数の返信を返す場合、最後の返信のみが reply で返されます。REPLY と REPLIES の説明については [Reply/Replies](#) を参照してください。

3.24.12 OPEN_CONNECTION

OPEN_CONNECTION 関数は、SMTP サーバーへの接続を開きます。

```
<c> CONNECTION OPEN_CONNECTION(<host> VARCHAR2 [, <port>
PLS_INTEGER [, <tx_timeout> PLS_INTEGER DEFAULT NULL])
```

Parameters

host

SMTP サーバーの名前。

port

SMTP サーバーがリッスンしているポート番号。デフォルトは 25 です。

tx_timeout

秒単位のタイムアウト値。待機しないは、0 を指定することによって示されます。無期限に待機することは、タイムアウトをヌルに設定することによって示されます。デフォルトは null です。

c

SMTP サーバーから返された接続ハンドル。

3.24.13 OPEN_DATA

OPEN_DATA プロシージャは、DATA コマンドを SMTP サーバーに送信します。

```
OPEN_DATA(<c> IN OUT CONNECTION)
```

Parameters

c

コマンドを送信する SMTP 接続。

3.24.14 QUIT

QUIT プロシージャは、SMTP サーバーとのセッションを閉じます。

```
QUIT(<c> IN OUT CONNECTION)
```

Parameters

c

終了する SMTP 接続。

3.24.15 RCPT

RCPT 手順は、受信者の電子メールアドレスを提供します。複数の受信者をスケジュールするには、RCPT を複数回呼び出します。

```
RCPT(<c> IN OUT CONNECTION, <recipient> VARCHAR2
[, <parameters> VARCHAR2 ])
```

Parameters

c

受信者を追加する SMTP サーバーへの接続。

recipient

受信者の電子メールアドレス。

parameters

RFC1869 で定義されている key=value 形式のメールコマンドパラメーター。

3.24.16 RSET

RSET プロシージャは、現在のメールトランザクションを終了する機能を提供します。

```
RSET(<c> IN OUT CONNECTION)
```

Parameters

c

メールトランザクションをキャンセルする SMTP 接続。

3.24.17 VRFY

VRFY 機能は、受信者の電子メールアドレスを検証および検証する機能を提供します。有効な場合、受信者の氏名と完全修飾メールボックスが返されます。

```
<reply> REPLY VRFY(<c> IN OUT CONNECTION, <recipient> VARCHAR2)
```

Parameters

c

電子メールアドレスを確認する SMTP 接続。

recipient

確認する受信者の電子メールアドレス。

reply

コマンドへの SMTP 応答。SMTP が複数の返信を返す場合、最後の返信のみが reply で返されます。REPLY と REPLIES の説明については [Reply/Replies](#) を参照してください。

3.24.18 WRITE_DATA

WRITE_DATA 手順は、VARCHAR2 データを電子メールメッセージに追加する機能を提供します。WRITE_DATA プロシージャは、データを追加するために繰り返し呼び出される場合があります。

```
WRITE_DATA(<c> IN OUT CONNECTION, <data> VARCHAR2)
```

Parameters

c

データを追加する SMTP 接続。

data

電子メールメッセージに追加するデータ。データは RFC822 仕様に準拠する必要があります。

3.24.19 包括的な例

次の手順では、UTL_SMTP パッケージを使用してテキスト電子メールメッセージを作成して送信します。

```
CREATE OR REPLACE PROCEDURE send_mail (  
    p_sender      VARCHAR2,  
    p_recipient   VARCHAR2,  
    p_subj        VARCHAR2,  
    p_msg         VARCHAR2,  
    p_mailhost    VARCHAR2  
)  
IS  
    v_conn        UTL_SMTP.CONNECTION;  
    v_crlf        CONSTANT VARCHAR2(2) := CHR(13) || CHR(10);  
    v_port        CONSTANT PLS_INTEGER := 25;  
BEGIN
```

(次のページに続く)

(前のページからの続き)

```

v_conn := UTL_SMTP.OPEN_CONNECTION(p_mailhost,v_port);
UTL_SMTP.HELO(v_conn,p_mailhost);
UTL_SMTP.MAIL(v_conn,p_sender);
UTL_SMTP.RCPT(v_conn,p_recipient);
UTL_SMTP.DATA(v_conn, SUBSTR(
    'Date: ' || TO_CHAR(SYSDATE,
    'Dy, DD Mon YYYY HH24:MI:SS') || v_crlf
    || 'From: ' || p_sender || v_crlf
    || 'To: ' || p_recipient || v_crlf
    || 'Subject: ' || p_subj || v_crlf
    || p_msg
    , 1, 32767));
UTL_SMTP.QUIT(v_conn);
END;

EXEC send_mail('asmith@enterprisedb.com','pjones@enterprisedb.com','Holiday
Party','Are you planning to attend?','smtp.enterprisedb.com');

```

次の例では、DATA プロシージャの代わりに OPEN_DATA, WRITE_DATA および CLOSE_DATA プロシージャを使用します。

```

CREATE OR REPLACE PROCEDURE send_mail_2 (
    p_sender      VARCHAR2,
    p_recipient   VARCHAR2,
    p_subj        VARCHAR2,
    p_msg         VARCHAR2,
    p_mailhost    VARCHAR2
)
IS
    v_conn        UTL_SMTP.CONNECTION;
    v_crlf        CONSTANT VARCHAR2(2) := CHR(13) || CHR(10);
    v_port        CONSTANT PLS_INTEGER := 25;
BEGIN
    v_conn := UTL_SMTP.OPEN_CONNECTION(p_mailhost,v_port);
    UTL_SMTP.HELO(v_conn,p_mailhost);
    UTL_SMTP.MAIL(v_conn,p_sender);
    UTL_SMTP.RCPT(v_conn,p_recipient);
    UTL_SMTP.OPEN_DATA(v_conn);
    UTL_SMTP.WRITE_DATA(v_conn,'From: ' || p_sender || v_crlf);
    UTL_SMTP.WRITE_DATA(v_conn,'To: ' || p_recipient || v_crlf);
    UTL_SMTP.WRITE_DATA(v_conn,'Subject: ' || p_subj || v_crlf);
    UTL_SMTP.WRITE_DATA(v_conn,v_crlf || p_msg);
    UTL_SMTP.CLOSE_DATA(v_conn);
    UTL_SMTP.QUIT(v_conn);
END;

EXEC send_mail_2('asmith@enterprisedb.com','pjones@enterprisedb.com','Holiday

```

(次のページに続く)

(前のページからの続き)

```
Party','Are you planning to attend?','smtp.enterprisedb.com');
```

3.25 UTL_URL

UTL_URL パッケージは、URL 内の不正な予約文字をエスケープする方法を提供します。

機能/手順	戻りタイプ	説明
ESCAPE(url, escape_reserved_chars, url_charset)	VARCHAR2	ESCAPE 関数を使用して、URL 内の不正な予約文字をエスケープします。
UNESCAPE(url, url_charset)	VARCHAR2	URL を元の形式に変換する UNESCAPE 関数。

関数の呼び出しに誤った形式の URL が含まれている場合、UTL_URL パッケージは BAD_URL 例外を返します。

3.25.1 ESCAPE

ESCAPE 関数を使用して、URL 内の不正な予約文字をエスケープします。署名は次のとおりです。

```
ESCAPE(<url> VARCHAR2, <escape_reserved_chars> BOOLEAN,  
<url_charset> VARCHAR2)
```

予約文字はパーセント記号に置き換えられ、その後にエスケープ文字の ascii 値の 2 桁の 16 進コードが続きます。

Parameters

url

url は、UTL_URL がエスケープする UniformResourceLocator を指定します。

escape_reserved_chars

escape_reserved_chars は BOOLEAN 値であり、ESCAPE 関数に予約文字と不正文字をエスケープするよう指示します。

- escaped_reserved_chars が FALSE の場合、ESCAPE は指定された URL の不正な文字のみをエスケープします。
- escape_reserved_chars が TRUE の場合、ESCAPE は指定された URL の不正な文字と予約文字の両方をエスケープします。

デフォルトでは、escape_reserved_chars は FALSE です。

URL 内の有効な文字は次のとおりです。

大文字の A から Z	小文字の a から z	0 から 9
アスタリスク (*)	感嘆符 (!)	ハイフン (-)
左括弧 (()	ピリオド (.)	右括弧 ())
単一引用符 (')	チルダ (~)	下線 (_)

一部の文字は URL の一部で有効ですが、他の文字では無効です。違法文字に関する包括的な規則を確認するには、RFC2396 を参照してください。URL の任意の部分で違法と見なされる文字の*例* は次のとおりです。

違法なキャラクター	エスケープシーケンス
空白スペース ()	% 20
中括弧 ({ または })	% 7b および % 7d
ハッシュマーク (#)	% 23

ESCAPE 関数は、次の文字が予約されていると見なし、escape_reserved_chars が TRUE に設定されている場合はエスケープします。

予約キャラクター	エスケープシーケンス
アンパサンド (&)	% 5C
アットマーク (@)	% 25
コロン (:)	% 3a
コンマ (,)	% 2c
ドル記号 (\$)	% 24
等号 (=)	% 3d
プラス記号 (+)	% 2b
疑問符 (?)	% 3f
セミコロン (;)	% 3b
スラッシュ (/)	% 2f

url_charset

url_charset は、特定の文字がエスケープされる前に変換される文字セットを指定します。url_charset が NULL の場合、キャラクターは変換されません。url_charset のデフォルト値は ISO-8859-1 です。

Examples

次の匿名ブロックは、ESCAPE 関数を使用して URL の空白スペースをエスケープします。

```
DECLARE
    result varchar2(400);
```

(次のページに続く)

(前のページからの続き)

```
BEGIN
  result := UTL_URL.ESCAPE('http://www.example.com/Using the ESCAPE
function.html');
  DBMS_OUTPUT.PUT_LINE(result);
END;
```

結果の（エスケープされた）URL は次のとおりです。

```
http://www.example.com/Using%20the%20ESCAPE%20function.html
```

関数を呼び出すときに `escape_reserved_chars` パラメーターに `TRUE` の値を含めると:

```
DECLARE
  result varchar2(400);
BEGIN
  result := UTL_URL.ESCAPE('http://www.example.com/Using the ESCAPE
function.html', TRUE);
  DBMS_OUTPUT.PUT_LINE(result);
END;
```

ESCAPE 関数は、URL 内の予約された文字と不正な文字をエスケープします。

```
http%3A%2F%2Fwww.example.com%2FUsing%20the%20ESCAPE%20function.html
```

3.25.2 UNESCAPE

UNESCAPE 関数は、ESCAPE 関数によって URL に追加されたエスケープ文字を削除し、URL を元の形式に変換します。

署名は次のとおりです。

```
UNESCAPE(<url> VARCHAR2, <url_charset> VARCHAR2)
```

Parameters

`url`

`url` は、UTL_URL がエスケープしない `UniformResourceLocator` を指定します。

`url_charset`

文字をエスケープ解除した後、文字は `url_charset` エンコーディングであると想定され、そのエンコーディングからデータベースエンコーディングに変換されてから返されます。 `url_charset` が `NULL` の場合、キャラクターは変換されません。 `url_charset` のデフォルト値は `ISO-8859-1` です。

Examples

次の匿名ブロックは、ESCAPE 関数を使用して URL の空白スペースをエスケープします。

```
DECLARE
    result varchar2(400);
BEGIN
    result := UTL_URL.UNESCAPE('http://www.example.com
Using%20the%20UNESCAPE%20function.html');
    DBMS_OUTPUT.PUT_LINE(result);
END;
```

結果の（エスケープされていない）URL は次のとおりです。

```
http://www.example.com/Using the UNESCAPE function.html
```

第 4 章

Conclusion

OracleDevelopers ビルトインパッケージガイド向け EDBPostgresAdvancedServer データベース互換性

著作権 ©2007-2020EnterpriseDBCorporation。

無断複写・転載を禁じます。

EnterpriseDBCorporation

34CrosbyDrive、Suite201、Bedford、MA01730、アメリカ

T+17813573390F+19784671307E

info@enterprisedb.com

www.enterprisedb.com

- EnterpriseDB および PostgresEnterpriseManager は、EnterpriseDBCorporation の登録商標です。EDB および EDBPostgres は、EnterpriseDBCorporation の商標です。Oracle は、Oracle、Inc の登録商標です。その他の商標は、それぞれの所有者の商標です。
- EDB は、存在する場合、EDBPostgres 製品のログオン UI の設計、コーディングのベストプラクティスの確立、レビュー、および入力検証の検証を行います。EDB は追加の入力コンポーネントに対して同じアプローチに従いますが、製品の性質上、限定された検証が可能な信頼できるユーザーによって入力および送信されるフリーフォーム SQL、WMI またはその他の文字列を受け入れる必要がある場合があります。このような場合、ユーザーが誤った入力や危険な入力を禁止することはできません。
- EDB は、将来、認証された信頼できるユーザーからのフリーフォーム SQL、WMI またはその他の潜在的に危険な入力を受け入れる製品に機能を追加する権利を留保しますが、そのような機能はすべて、可能な限り最小のリスクを提供するように設計およびテストされます可能であれば、スーパーユーザーまたは同等の特権が必要です。
- EDB は、すべての潜在的な脅威を予測できることを保証しないため、すべての潜在的な脆弱性が対処または検討されたことを完全に保証することはできません。

索引

ALTER_POLICY, 183
 ALTER_QUEUE, 51
 ALTER_QUEUE_TABLE, 52
 ANALYZE_DATABASE, ANALYZE_SCHEMA and ANALYZE
 PART_OBJECT, 252
 APPEND, 82

BASE64_DECODE, 269
 BASE64_ENCODE, 270
 BEGIN_REQUEST, 307
 BIND_VARIABLE_RAW, 230
 BIND_VARIABLE, 228
 BIND_VARIABLE_CHAR, 229
 BROKEN, 73
 Built-In Packages, 23

CANONICALIZE, 254
 CAST_TO_RAW, 331
 CAST_TO_VARCHAR2, 331
 CHANGE, 73
 CHARARR, 102
 CLOSE_CURSOR, 230
 CLOSE_DATA, 339
 COLUMN_VALUE, 231
 COLUMN_VALUE_CHAR, 232
 COLUMN_VALUE_LONG, 233
 COLUMN_VALUE_RAW, 233
 COMMA_TO_TABLE, 256
 COMMAND, 340
 COMMAND_REPLIES, 340
 COMPARE, 82
 CONCAT, 332
 Conclusion, 353
 CONNECTION, 339
 CONVERT, 333
 CONVERTTOBLOB, 83
 CONVERTTOCLOB, 84
 COPY, 85
 CREATE_JOB, 209
 CREATE_PIPE, 113
 CREATE_PROGRAM, 212
 CREATE_QUEUE, 53
 CREATE_QUEUE_TABLE, 54
 CREATE_SCHEDULE, 213
 Creating Packages, 13
 Creating the Package Body, 14
 Creating the Package Specification, 13

DATA, 341
 DB_VERSION, 257
 DBMS_ALERT, 24
 DBMS_ALERT_Comprehensive_example, 30
 DBMS_ALERT_Register, 26
 DBMS_ALERT_Remove, 26
 DBMS_AQ, 34
 DBMS_AQ_Register, 43
 DBMS_AQADM, 48

DBMS_CRYPTO, 61
 DBMS_JOB, 71
 DBMS_JOB_Remove, 75
 DBMS_LOB, 79
 DBMS_LOB_SUBSTR, 88
 DBMS_LOCK, 92
 DBMS_MVIEW, 93
 DBMS_OUTPUT, 101
 DBMS_OUTPUT_DISABLE, 102
 DBMS_OUTPUT_ENABLE, 102
 DBMS_OUTPUT_PUT, 107
 DBMS_OUTPUT_PUT_LINE, 108
 DBMS_PIPE, 111
 DBMS_PIPE_Comprehensive_example, 123
 DBMS_PROFILER, 127
 DBMS_PROFILER - Reference, 141
 DBMS_RANDOM, 155
 DBMS_RANDOM_VALUE_FIRST_FORM, 159
 DBMS_RANDOM_VALUE_SECOND_FORM, 159
 DBMS_REDACT, 161
 DBMS_REDACT_ADD_POLICY, 179
 DBMS_REDACT_DROP_POLICY, 187
 DBMS_REDACT_ENABLE_POLICY, 186
 DBMS_RLS, 192
 DBMS_RLS_DROP_POLICY, 203
 DBMS_RLS_ENABLE_POLICY, 204
 DBMS_RLSADD_POLICY:, 196
 DBMS_SCHEDULER, 206
 DBMS_SCHEDULER_DISABLE, 216
 DBMS_SCHEDULER_ENABLE, 220
 DBMS_SESSION, 224
 DBMS_SQL, 225
 DBMS_UTILITY, 249
 DECRYPT, 63
 DEFINE_COLUMN, 234
 DEFINE_COLUMN_CHAR, 236
 DEFINE_COLUMN_LONG, 237
 DEFINE_COLUMN_RAW, 237
 DEFINE_PROGRAM_ARGUMENT, 215
 DEQUEUE, 40
 DESCRIBE_COLUMNS, 238
 DISABLE_POLICY, 185
 DROP_JOB, 217
 DROP_PROGRAM, 218
 DROP_PROGRAM_ARGUMENT, 218
 DROP_QUEUE, 56
 DROP_QUEUE_TABLE, 57
 DROP_SCHEDULE, 219
 Dropping a Package, 21

EHLO, 341
 ENCRYPT, 65
 END_OF_BODY, 324
 END_REQUEST, 308
 END_RESPONSE, 308
 ENQUEUE, 37
 ERASE, 86

ESCAPE, 349
EVALUATE_CALENDAR_STRING, 221
EXEC_DDL_STATEMENT, 258
EXECUTE, 239
EXECUTE_AND_FETCH, 240

FCLOSE, 284
FCLOSE_ALL, 284
FCOPY, 285
FETCH_ROWS, 242
FFLUSH, 287
FLUSH_DATA, 128
FOPEN, 288
FORMAT_CALL_STACK, 259
FREMOVE, 288
FRENAME, 289

GET_BODY_CHARSET, 309
GET_CPU_TIME, 259
GET_DEPENDENCY, 259
GET_FOLLOW_REDIRECT, 309
GET_HASH_VALUE, 260
GET_HEADER, 310
GET_HEADER_BY_NAME, 311
GET_HEADER_COUNT, 312
GET_LINE, 103, 291
GET_LINES, 105
GET_MV_DEPENDENCIES, 95
GET_PARAMETER_VALUE, 262
GET_RESPONSE, 313
GET_RESPONSE_ERROR_CHECK, 313
GET_STORAGE_LIMIT, 86
GET_TIME, 263
GET_TRANSFER_TIMEOUT, 313
GET_VERSION, 128
GETLENGTH, 87

HASH, 67
HELO, 342
HELP, 342
HTML_PIECES, 307

INITIALIZE, 156
INSTR, 87
INTERNAL_VERSION_CHECK, 128
INTERVAL, 74
Introduction, 1
IS_OPEN, 243, 292

LAST_ERROR_POSITION, 245
LAST_ROW_COUNT, 244
LENGTH, 334
LNAME_ARRAY, 252

MAC, 68
MAIL, 343
MIMEHEADER_DECODE, 271
MIMEHEADER_ENCODE, 271

NAME_TOKENIZE, 263
NEW_LINE, 107, 293
NEXT_DATE, 75
NEXT_ITEM_TYPE, 114
NOOP, 343
NORMAL, 156

OPEN_CONNECTION, 344
OPEN_CURSOR, 246

OPEN_DATA, 344

PACK_MESSAGE, 116
Package Body Syntax, 8
Package Components, 4
Package Specification Syntax, 4
Packages, 3
PARSE, 247
PAUSE_PROFILER, 129
PLSQL_PROFILER_DATA, 142
PLSQL_PROFILER_RAWDATA, 143
PLSQL_PROFILER_RUNS, 141
PLSQL_PROFILER_UNITS, 142
PURGE, 117
PURGE_QUEUE_TABLE, 58
PUTF, 297

QUIT, 344
QUOTED_PRINTABLE_DECODE, 273
QUOTED_PRINTABLE_ENCODE, 274

RANDOM, 157
RANDOMBYTES, 69
RANDOMINTEGER, 69
RANDOMNUMBER, 70
RCPT, 345
READ, 88
READ_LINE, 314
READ_RAW, 315
READ_TEXT, 316
RECEIVE_MESSAGE, 118
Referencing a Package, 16
REFRESH, 95
REFRESH_ALL_MVIEWS, 97
REFRESH_DEPENDENT, 98
REMOVE_PIPE, 119
REMOVEALL, 27
REPLY/REPLIES, 339
REQ, 307
REQUEST, 317
REQUEST_PIECES, 318
RESET_BUFFER, 120
RESP, 307
RESUME_PROFILER, 129
RSET, 345
RUN, 76
RUN_JOB, 222

SEED, 157, 158
SEND, 326
SEND_ATTACH_RAW, 327
SEND_ATTACH_VARCHAR2, 328
SEND_MESSAGE, 121
SERVEROUTPUT, 109
SET_BODY_CHARSET, 318
SET_FOLLOW_REDIRECT, 319
SET_HEADER, 319
SET_JOB_ARGUMENT_VALUE, 223
SET_RESPONSE_ERROR_CHECK, 320
SET_ROLE, 224
SET_TRANSFER_TIMEOUT, 320
Setting File Permissions with utl_file.umask, 283
SIGNAL, 27
SLEEP, 92
START_PROFILER, 129
START_QUEUE, 59
STOP_PROFILER, 130

STOP_QUEUE, 59
STRING, 158
SUBMIT, 76

TABLE_TO_COMMA, 266
TERMINATE, 159
TEXT_DECODE, 275
TEXT_ENCODE, 276
TRIM, 89

UNCL_ARRAY, 252
UNESCAPE, 351
UNIQUE_SESSION_NAME, 122
UNPACK_MESSAGE, 123
UNREGISTER, 45
UPDATE_FULL_REDACTION_VALUES, 188
Using Calendar Syntax to Specify a Repeating Interval, 208
Using DBMS_PROFILER, 131
Using DBMS_REDACT Constants and Function Parameters, 164
Using Packages With User Defined Types, 17
UTL_ENCODE, 268
UTL_FILE, 280
UTL_FILE_PUT, 294
UTL_FILE_PUT_LINE, 296
UTL_HTTP, 300
UTL_MAIL, 325
UTL_RAW, 330
UTL_RAW_SUBSTR, 334
UTL_SMTP, 336
UTL_SMTP_Comprehensive_example, 346
UTL_URL, 349
UUDECODE, 277
UUENCODE, 278

VRFY, 345

WAITANY, 27
WAITONE, 29
WHAT, 78
WRITE, 90
WRITE_DATA, 346
WRITE_LINE, 321
WRITE_RAW, 322
WRITE_TEXT, 323
WRITEAPPEND, 90