

WG2活動報告書
ストアドプロシージャ移行調査編

目次

目次	2
1. 改訂履歴	4
2. ライセンス	5
3. はじめに	6
3.1. 本資料の目的	6
3.2. 本資料で記載する範囲	6
3.3. 本資料で扱う用語の定義	6
3.4. 本資料で扱うDBMSおよびツール	6
4. PostgreSQLのストアードプロシージャについて	7
4.1. PostgreSQLにおけるストアードプロシージャ	7
4.2. PL/pgSQLについて	7
5. OracleからPostgreSQLへの移行(定義関連)	8
5.1. CREATE FUNCTION文	8
5.2. CREATE PROCEDURE文	8
5.3. CREATE PACKAGE文	8
5.4. ALTER FUNCTION文	9
5.5. DROP FUNCTION文	9
6. OracleからPostgreSQLへの移行(標準手続き言語関連)	10
6.1. 構造	10
6.2. コメント	10
6.3. 引数	10
6.4. データ型	11
6.5. 変数の宣言	11
6.6. 制御構造	11
6.6.1. LOOP命令	11
6.6.2. WHILE命令	12
6.6.3. FOR命令	12
6.6.4. EXIT命令	12
6.6.5. CONTINUE命令	13
6.6.6. IF命令	13
6.6.7. CASE命令	13
6.6.8. GOTO命令	13
6.7. カーソル	13
6.7.1. カーソルの宣言	13
6.7.2. カーソルのOPEN	14
6.7.3. カーソルのFETCH	14
6.7.4. カーソルの終了判定	14
6.7.5. カーソルの更新	14
6.7.6. カーソルのCLOSE	14
6.7.7. REFCURSOR	15
6.8. エラーハンドリング	15
6.8.1. EXCEPTION文	15
6.8.2. RAISE文	16
7. OracleからPostgreSQLへの移行(その他)	17
7.1. 起動方法	17
7.2. 呼出方法	17
7.3. トランザクション制御	17
7.4. シーケンス	17
7.5. 組み込み関数	17
7.6. DUAL	18
7.7. パッケージ変数代替	18
8. SQL ServerからPostgreSQLへの移行	20
9. DB2からPostgreSQLへの移行	21
10. 異種DBMSからPostgreSQLへの移行に関するまとめ	22
10.1. Oracleのユーティリティパッケージについて	22
11. 著者	23

1. 改訂履歴

版	改訂日	変更内容
1.0	2013/03/25	新規作成
2.0	2014/03/26	2013 年度活動成果の追加
3.0	2018/03/16	<ul style="list-style-type: none">● PostgreSQLの対象バージョンを10.3に更新● 「5.5. DROP FUNCTION文」の記述を変更● 「6.3. 引数」を追加● 「6.6.6. IF 命令」の記述を変更● 「6.7.7. REFCURSOR」を追加● 「6.8.1. EXCEPTION文」にNO_DATA_FOUNDに関する注意点を追加● 「7.2. 呼出方法」を追加● 「7.4. シーケンス」を追加● 「7.5. 組み込み関数」を追加● 「7.6. DUAL」を追加● 「7.7. パッケージ変数代替」を追加

2. ライセンス

本作品はCC-BYライセンスによって許諾されています。ライセンスの内容を知りたい方は [こちら](#) でご確認ください。文書の内容、表記に関する誤り、ご要望、感想等につきましては、[PGEGConsのサイト](#) を通じてお寄せいただきますようお願いいたします。

- Eclipseは、Eclipse Foundation Incの米国、およびその他の国における商標もしくは登録商標です。
- IBMおよびDB2は、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。
- Intel、インテルおよびXeonは、米国およびその他の国におけるIntel Corporationの商標です。
- Javaは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。
- Linux は、Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。
- Red HatおよびShadowman logoは、米国およびその他の国におけるRed Hat,Inc.の商標または登録商標です。
- Microsoft、Windows Server、SQL Server、米国 Microsoft Corporationの米国及びその他の国における登録商標または商標です。
- MySQLは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。
- Oracleは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。
- PostgreSQLは、PostgreSQL Community Association of Canadaのカナダにおける登録商標およびその他の国における商標です。
- Windows は米国 Microsoft Corporation の米国およびその他の国における登録商標です。
- TPC, TPC Benchmark, TPC-B, TPC-C, TPC-E, tpmC, TPC-H, TPC-DS, QphHは米国Transaction Processing Performance Councilの商標です。
- その他、本資料に記載されている社名及び商品名はそれぞれ各社が 商標または登録商標として使用している場合があります。

3. はじめに

3.1. 本資料の目的

本資料は、異種DBMSからPostgreSQLへストアドプロシージャを移行する作業の難易度およびボリュームの事前判断と、実際に書き換えを行う際の参考資料として利用されることを想定しています。

3.2. 本資料で記載する範囲

本資料では、移行元の異種DBMSとしてOracle Database、IBM DB2およびMicrosoft SQLServerを想定し、PostgreSQLへストアドプロシージャを移行する際に書き換えが必要である箇所とその書き換え方針について手続き言語を中心に記載します。スキーマ、SQL、組み込み関数については本資料では取り扱っていません。これらに関しては、それぞれ「スキーマ移行調査編」、「SQL移行調査編」、「組み込み関数移行調査編」を参照してください。

3.3. 本資料で扱う用語の定義

資料で記述する用語について以下に定義します。

表 3.1 用語定義

No.	用語	意味
1	DBMS	データベース管理システムを指します。ここでは、PostgreSQLおよび異種DBMSの総称として利用します。
2	異種DBMS	PostgreSQL ではない、データベース管理システムを指します。本資料では、Oracle Database、IBM DB2 および Microsoft SQLServer が該当します。
3	Oracle	データベース管理システムの Oracle Database を指します。
4	DB2	データベース管理システムの IBM DB2 を指します。
5	SQLServer	データベース管理システムの Microsoft SQLServer を指します。

3.4. 本資料で扱うDBMSおよびツール

本書では以下のDBMSを前提にした調査結果を記載します。

表 3.2 本書で扱うDBMS

DBMS名称	バージョン
PostgreSQL	10.3
Oracle Database	11gR2 11.2.0.2.0
IBM DB2	8.2
Microsoft SQLServer	2005 Express

4. PostgreSQLのストアドプロシージャについて

データベースに対する一連の処理手順をまとめてDBMS内に格納する、「ストアドプロシージャ」についてPostgreSQLにおける特徴を紹介します。

4.1. PostgreSQLにおけるストアドプロシージャ

PostgreSQLではストアドプロシージャはユーザ定義関数(FUNCTION)として定義を行います。実行方法は、関数として実装するため呼び出し方法もSQL文の中で他の関数と同様に利用することになります。処理ロジックの記述には、PostgreSQL専用の手続き言語としてPL/pgSQLが用意されています。上記以外に、CやPerlなどでも処理ロジックを組み込むことも可能です。

4.2. PL/pgSQLについて

PL/pgSQLは、OracleのPL/SQLと同様にSQLに制御構造(条件分岐やLOOP処理)などを組み込んだ、PostgreSQLで標準として実装されている手続き言語です。記述された処理ロジックは、ユーザ定義関数としてデータベースに格納する事が出来ますが、事前にコンパイルはされずに、実行時に解釈され実行されます。

5. OracleからPostgreSQLへの移行(定義関連)

5.1. CREATE FUNCTION文

表 5.1 CREATE FUNCTION文の比較

Oracle	PostgreSQL
<pre>CREATE OR REPLACE FUNCTION ファンクション名 (@引数名 IN データ型) RETURN 戻り値データ型 IS 変数名 データ型; BEGIN 処理内容; END [ファンクション名];</pre>	<pre>CREATE OR REPLACE FUNCTION proc_f (引数名 IN データ型) RETURNS 戻り値データ型 AS \$\$ DECLARE 変数名 データ型; BEGIN 処理内容; END; \$\$ LANGUAGE plpgsql;</pre>

PostgreSQLでは処理内容の記述部分(変数宣言とBEGINからENDまで)を文字列定数として作成する必要があります。

そのためにドル引用符付け(\$\$)を使って処理記述の範囲を囲います。

単一引用符で範囲を囲む方法も可能ですが、この場合には関数の本体部分で使用される単一引用符(')とバックスラッシュ(\)は二重にする必要があります。

処理内容の記述に使用している言語の指定が必須で、LANGUAGE句で指定します。

変数宣言部にDECLAREが必須ですので追加する必要があります。

引数を持たないFUNCTIONを作成するとき、にはOracleでは"()"を省略できますが、PostgreSQLでは"()"の記述が必須です。

上記以外では

```
RETURN → RETURNS
IS → AS
END [FUNCTION名]; → END;
```

に書き換える必要があります。

PL/SQLではEND部分にFUNCTION名を記載することがありますが、PL/pgSQLでは記載しません。

5.2. CREATE PROCEDURE文

PostgreSQLにはPROCEDUREは実装されていません。

FUNCTIONで代用する事になります。

5.3. CREATE PACKAGE文

PROCEDUREと同様にPACKAGEは実装されていません。

FUNCTIONで代用することになります。

PACKAGEレベルで共通使用する定数などは、一時テーブルに保存するなどの方法を検討する必要があります。

PROCEDUREがPACKAGEに属している構成を元々とっていた場合には、SCHEMAで代替することができます。

表 5.2 PACKAGEとSCHEMAの比較

Oracle	PostgreSQL
<pre>CREATE OR REPLACE PACKAGE パッケージ名 IS PROCEDURE プロシージャ名 ((後略)</pre>	<pre>CREATE SCHEMA IF NOT EXISTS スキーマ名; CREATE OR REPLACE FUNCTION スキーマ名.ファンクション名 ((後略)</pre>

SCHEMAを使用した場合は、FUNCTION名にどのSCHEMAに属しているかを指定する必要があります。ひとつのFUNCTION内で別のFUNCTIONを呼ぶ場合も同様にSCHEMAを指定する必要があります。

またPL/pgSQLでは仕様部と本体に分けず、一つの関数定義は一箇所に記述します。

5.4. ALTER FUNCTION文

OracleとPostgreSQLでは互換性がありません。

Oracleでは再コンパイルに関する命令になります。

PostgreSQLでは関数名の変更、所有者の変更などのFUNCTIONが保持している情報を変更する命令になります。

5.5. DROP FUNCTION文

表 5.3 DROP FUNCTION文の比較

Oracle	PostgreSQL
<pre>DROP FUNCTION ファンクション名;</pre>	<pre>DROP FUNCTION ファンクション名 (引数名 IN データ型);</pre>

PostgreSQLでは、同名の関数が存在している場合、引き渡しパラメータも含めて指定する必要があります。パラメータの指定はデータ型のみでの記載でも問題ありません。

6. OracleからPostgreSQLへの移行(標準手続き言語関連)

OracleとPostgreSQLにそれぞれ実装されている手続き言語である、PL/SQLとPL/pgSQLにおける記述の相違を中心に書換え方法を記述します。

6.1. 構造

構造のステートメントには相違ありません。

```
DECLARE
  変数名 データ型;
BEGIN
  処理内容
END;
```

「DECLARE部」で変数の宣言
 「BEGIN部」で処理内容の記述
 「END」でブロックの終了

6.2. コメント

コメントの記述には相違ありません。

```
-- コメント記述 : 行末までをコメントとします。
/* コメント記述 */ : /* から */ までのブロック(複数行でも可)をコメントとします。
```

6.3. 引数

引数の宣言ではPL/SQLと同じようにPL/pgSQLでもIN引数、OUT引数、INOUT引数を使用することができます。またOUT引数又はINOUT引数を使用した場合はRETURNSの指定が不要になることもPL/SQLと同じです。しかし、以下の事柄について注意が必要です。

同名になってしまうOUT引数付き関数

同名で同引数を持つ関数を複数作成できないことはOracleでもPostgreSQLでも同じですが、PostgreSQLはこの判断にOUT引数を考慮しません。したがって、引数の数や名前が違う場合でもそれらがOUT引数である場合、同名で同引数の関数と判断され作成時にエラーとなります。

OUT引数関数の呼び出し方

OUT引数(INOUT引数)を持つ関数を呼び出す場合、OUT引数は呼出引数に含めず返り値を引数に代入するように記述する必要があります。また、複数OUT引数がある場合は一旦RECORD型に代入する必要があります。

```
<複数のOUT引数を持つ関数>
CREATE OR REPLACE ファンクション名 (
  引数1 IN データ型,
  引数2 OUT データ型,
  引数3 OUT データ型
) LANGUAGE plpgsql ...

<呼び出し元の関数宣言部に以下を追加>
変数1 RECORD;

<呼び出し部分>
変数1 := ファンクション名 (引数1);
```

また上記以外にも細かな違いとして引数に対してデフォルトの値を与える際にも := ではなく = を使うというものがあります。

Oracle	PostgreSQL
<pre>PROCEDURE プロシージャ名 (変数名 データ型 := デフォルト値 ... </pre>	<pre>CREATE OR REPLACE FUNCTION ファンクション名 (引数1 IN データ型 = デフォルト値 ... </pre>

6.4. データ型

PostgreSQLで使用可能なデータ型はPL/pgSQLで使用できます。

データ型の変換については別ドキュメント「[組み込みデータ型対応表\(Oracle-PostgreSQL\)](#)」を参照してください。

同様に%ROWTYPE型や%TYPEはそのまま使用できます。

RECORD型については注意が必要です。

Oracle	PostgreSQL
<pre>type 変数名 is RECORD (変数名 データ型);</pre>	<pre>変数名 RECORD;</pre>

PL/pgSQLではRECORD型の宣言時にはレコードの内容は記述しません。

レコードの内容は直接SELECT文を記述したり、カーソルのFETCHで使用されると定義が確定されます。

例1. SELECTの結果をレコード型にストアする

```
rec_name IN SELECT C1, C2 FROM tb1
```

例2. カーソルcuの結果をレコード型にストアする

```
fetch cu into rec_name
```

データ型のキャストには組み込み関数を使用することも可能ですが、PostgreSQLでは伝統的に「::」を使用してキャストを行います。

Oracleで用意されている型キャストの関数の中にはPostgreSQLでは存在しないものもあります。

表 6.1 型キャストの比較

Oracle	PostgreSQL
<pre>変数 := TO_NUMBER(値); 変数 := TO_CHAR(値);</pre>	<pre>変数 := 値::numeric; 変数 := 値::text;</pre>

NULL についてもOracleとPostgreSQLでは違いがありますので注意が必要になります。NULLについては別ドキュメント「[SQL移行調査編](#)」を参照してください。

テーブル型はOracleでは宣言する必要がありますが、PostgreSQLではテーブルを定義した時点でそのテーブル名と同名のものが利用できるようになるため、宣言する必要がありません。

6.5. 変数の宣言

プログラム内で使用する変数は必ず宣言部に記述して宣言を行う必要があります。

但し、例外としてFORループで使用するループ変数はこの限りではありません。

例外の名前の宣言はPL/pgSQLでは宣言する事が出来ません。

RAISE文を使ってエラーを発生させます。

6.6. 制御構造

6.6.1. LOOP命令

LOOPの記述には相違ありません。

```
LOOP
  繰り返し処理;
  EXIT WHEN 条件式;
END LOOP;
```

「LOOP」と「END LOOP」の間に記述された命令を繰り返し実行します。
 LOOPを抜けるためにはEXITを使用します。
 EXITに続けてLOOPを抜ける条件式を記述します。
 EXITのみでは無条件でLOOPから抜けます。

6.6.2. WHILE命令

WHILEの記述には相違ありません。

```
WHILE 条件式 LOOP
  繰り返し処理;
END LOOP;
```

「WHILE」と「LOOP」の間に繰り返しの条件式を記述し、
 「END LOOP」の間に繰り返す命令を記述します。
 条件式を満たす前にLOOPを抜けるためにはEXITを使用します。

6.6.3. FOR命令

FORの記述には相違ありません。

```
FOR 変数名 IN 1 .. 10 LOOP
  繰り返し処理;
END LOOP;
```

INの後に記述した最小値から最大値までの間、「LOOP」から「END LOOP」に記述された命令を繰り返し実行します。

但し、「REVERSE」を使って値を最大値から最小値までを行う場合には書換えが必要です。

Oracle	PostgreSQL
<pre>FOR 変数名 IN REVERSE 1 .. 10 LOOP 繰り返し処理; END LOOP;</pre>	<pre>FOR 変数名 IN REVERSE 10 .. 1 LOOP 繰り返し処理; END LOOP;</pre>

最大値と最小値の値の指定が逆になります。

6.6.4. EXIT命令

EXITの記述には相違ありません。

```
EXIT;
EXIT [ ラベル名 ] ;
EXIT WHEN A1 > 10;
```

ラベルが指定されない場合には最も内側のLOOPを終わらせます。
 ラベルの指定がある場合には指定されたラベルのループを抜けます。
 WHENが指定された場合には、条件式を満たしていればEXITを実行します。

6.6.5. CONTINUE命令

CONTINUEの記述には相違ありません。

```
CONTINUE;
CONTINUE [ ラベル 名 ] ;
CONTINUE WHEN 条件式 ;
```

ラベルが指定されない場合には実行しているLOOPの先頭に戻り次の反復に制御を移します。
ラベルの指定がある場合には指定されたラベルの先頭に戻り次の反復に制御を移します。
WHENが指定された場合には、条件式を満たしていればCONTINUEを実行します。

6.6.6. IF命令

IF文については、Oracleの記述と相違ありません。

6.6.7. CASE命令

CASEの記述には相違ありません。

```
CASE 変数
  WHEN 条件値 THEN
    分岐処理
  ELSE
    分岐処理
END CASE;
```

WHEN句内の値と比較を行い一致すれば指定された命令が実行されます。
全てのWHENを順番に評価した後一致するものがない場合、ELSEの命令を実行します。
一致するWHENがなくELSEの記述が無い場合には、CASE_NOT_FOUND例外が発生します

6.6.8. GOTO命令

PostgreSQLにはGOTO命令がありません。

Oracle	PostgreSQL
GOTO ラベル ;	[対応する命令なし]

置換える命令がありません。
無条件に指定したラベルに制御を移すことは出来ません。

6.7. カーソル

6.7.1. カーソルの宣言

カーソルの宣言については注意が必要です。

Oracle	PostgreSQL
CURSOR カーソル名 IS クエリー;	カーソル名 CURSOR FOR クエリー;

どちらも宣言はDECLARE部で行いますが、文法が違います。
FORの部分はISで記述されていても文法エラーにはなりません。

また引数を宣言する際にINというキーワードがPL/pgSQLでは不要になります。

Oracle	PostgreSQL
CURSOR カーソル名 (引数 IN データ型) IS ...	カーソル名 CURSOR (引数 データ型) FOR ...

6.7.2. カーソルのOPEN

カーソルのOPENの記述には相違ありません。

```
OPEN カーソル名 ;
```

宣言をしたカーソルから行を取り出すために、OPENによりカーソルを開きます。

6.7.3. カーソルのFETCH

カーソルのFETCHの記述には相違ありません。

```
FETCH カーソル名 INTO 取得した値を格納する変数 ;
```

カーソルから行を1行ずつ取り出して変数に格納します。

6.7.4. カーソルの終了判定

カーソルをすべてFETCHしたときの判定方法は注意が必要です。

Oracle	PostgreSQL
カーソル名 %NOTFOUND ;	NOT FOUND ;

Oracleでは、カーソル名を明示して終了判定 (NOTFOUND) しますが、PostgreSQLではカーソル名の指定はできません。

6.7.5. カーソルの更新

カーソルのカレント行に対する更新の記述には相違ありません。

```
<更新>
UPDATE テーブル名 SET 更新内容 WHERE CURRENT OF カーソル名 ;

<削除>
DELETE FROM テーブル名 WHERE CURRENT OF カーソル名 ;
```

カーソルの宣言時にFOR UPDATEを使って作成したカーソルの現在行に対して項目の値の変更およびレコードの削除を行います。

6.7.6. カーソルのCLOSE

カーソルのCLOSEの記述には相違ありません。

```
CLOSE カーソル名 ;
```

OPENしたカーソルを閉じます。

PL/pgSQLには%ISOPENが存在していません。PL/SQLではカーソルの閉じ忘れ防止としても使用していましたが、PL/pgSQLではそれができません。クローズを忘れないようにすれば問題ありませんが、%ISOPENの代用としては以下の方法があります。

```
BEGIN
  CLOSE カーソル名;
EXCEPTION
  WHEN invalid_cursor_name THEN NULL;
END;
```

すでにクローズされたカーソルをクローズしようとするエラーが発生しますが、それを例外として拾いそこでは何もしないという処理をしています。オープン状態であればクローズし、クローズされていれば何もしません。

6.7.7. REFCURSOR

関数の引数や返り値、変数としてカーソルを使用する場合は、refcursor型として宣言します。PL/SQLではSYS_REFCURSORと宣言されていたものです。

```
CREATE OR REPLACE FUNCTION ファンクション名 ()
  RETURNS refcursor LANGUAGE plpgsql
AS $$
DECLARE
  カーソル名 refcursor;
BEGIN
  (中略)
  RETURN カーソル名;
END;
```

6.8. エラーハンドリング

6.8.1. EXCEPTION文

EXCEPTIONの記述には相違ありません。

```
EXCEPTION
  WHEN エラーコード(もしくは例外名) 1 THEN エラー処理内容1
  WHEN エラーコード(もしくは例外名) 2 THEN エラー処理内容2
  WHEN OTHERS THEN エラー処理内容3
END;
```

WHENの後に記述された例外の内容と合致したときにTHENの後に記述された処理を行います。指定された例外以外が発生したときは、呼び出し元にエラー情報が伝搬します。

例外に設定されている名前に相違があるものは個別に書換えが必要です。以下は例外の一部についての対比をまとめたので、参考にしてください。

Oracleの例外名	PostgreSQLの例外名	相違
CASE_NOT_FOUND	CASE_NOT_FOUND	同じ
INVALID_CURSOR	INVALID_CURSOR_STATE	書換え必要
NO_DATA_FOUND	NO_DATA_FOUND	同じ*1
STORAGE_ERROR	OUT_OF_MEMORY	書換え必要
TOO_MANY_ROWS	TOO_MANY_ROWS	同じ
ZERO_DIVIDE	DIVISION_BY_ZERO	書換え必要

なお、PostgreSQLのエラーコードに対する例外名はマニュアルの付録に記載があるので参考にしてください。
<https://www.postgresql.jp/document/10/html/errcodes-appendix.html#errcodes-table>

*1 NO_DATA_FOUND に関して注意すべき点あります。

OracleではSELECTの結果が0であった場合にこの例外に該当しますが、PostgreSQLでは明示的にハンドリングしなければSELECTの結果が0行であっても例外として判

断されません。

SELECT INTO 文にSTRICTを加えるかもしくは代入先の変数がNULLであるかを確認して例外を投げる必要があります。

6.8.2. RAISE文

RAISEを使った例外を発生させる記述には相違ありません。

```
RAISE exception;
```

事前定義の例外を明示的に呼び出します。

但し、Oracleでは宣言部で例外の名前を宣言して、RAISEで例外を呼び出せますが、PostgreSQLでは宣言部での名前の宣言が出来ないので、RAISE文で例外の詳細を記述する事になります。

代替として、任意のSQLSTATE(5文字の状態コード)を使用することができます。

PL/pgSQL関連の例外はP0から始まるのが作法ですが、最低限マニュアルにも記載されている規定のSQLSTATEと重複しなければ問題ありません。

7. OracleからPostgreSQLへの移行(その他)

7.1. 起動方法

実行方法については注意が必要です。

Oracle	PostgreSQL
<pre>BEGIN EXECUTE プロシージャ名 END;</pre>	<pre>SELECT ファンクション名 ();</pre>

PostgreSQLでは、ストアドファンクション(関数)として登録していますのでSELECT文を使って呼び出します。Oracleでは引数がない場合には括弧は不要ですが、PostgreSQLでは括弧が必要です。

7.2. 呼出方法

関数の中で別の関数を実行する場合、基本的には呼び出し先の関数が返す値に合わせたデータ型の変数を宣言し、それに代入するような形で記載します。

```
(TEXT型を返す関数を呼ぶ場合)
DECLARE
  変数 TEXT;
BEGIN
  変数 := ファンクション名 ();
```

返り値がない関数を実行する場合には、PL/SQLとは違いPL/pgSQLではPERFORM命令が必要になります。

```
PERFORM ファンクション名 ();
```

7.3. トランザクション制御

PostgreSQLのストアドファンクションは、外部トランザクションの一部として実行されますので、処理中にCOMMITを実行できません。

Oracleでは「PRAGMA AUTONOMOUS_TRANSACTION」を使って呼び出し元とトランザクションを分離する事が出来ませんが、PostgreSQLにはこのような機能はありません。

EXCEPTIONで例外の発生が判断された時は、BEGIN以降のすべてのデータベースに対する更新処理が自動的にロールバックします。

7.4. シーケンス

PostgreSQLとOracleでは、シーケンスから値を取り出す構文が異なります。

以下がPostgreSQLでのシーケンス値の取り出し方です。

```
nextval ('シーケンス名') -- 次の値を取り出す
setval ('シーケンス名', 値) -- 値をセットする
currval ('シーケンス名') -- 現在値を再度取り出す
```

その他シーケンス移植時の情報は別ドキュメント「[DB移行開発見知り編](#)」も参照ください。

7.5. 組み込み関数

組み込み関数についても書き換えが必要になる部分があります。
これに関しては別ドキュメント「[組み込み関数移行調査編](#)」を参照ください。

7.6. DUAL

PostgreSQLではOracleのようにDUALテーブルは用意されていません。
対応方法に関しては別ドキュメント「[SQL移行調査編](#)」を参照ください。

7.7. パッケージ変数代替

PostgreSQLではパッケージという概念が無いためPL/SQLのパッケージ変数をそのまま移植することができません。PL/pgSQLでそれを実現させるためにはいくつか方法があるかもしれませんが、ここでは一時テーブルを使用したものを紹介します。PostgreSQLの一時テーブルは接続ごとに独立して作成され、接続が切断されるとテーブル定義はそのデータと共に消えます。

これを実現させるためにはその一時テーブルの作成とテーブルへのデータ挿入、更新そしてデータの取得を行うための関数をパッケージごとに作成する必要があります。
PostgreSQLではパッケージという概念がないためスキーマを代わりに使用している前提となります。

(初期化用関数例)

```
CREATE OR REPLACE FUNCTION スキーマ名.初期化用ファンクション名 ()
    RETURNS void LANGUAGE plpgsql
    AS $$
    DECLARE
        変数 INTEGER := 0;
    BEGIN
        SELECT INTO 変数 count(*) FROM information_schema.tables WHERE table_name = lower('一時テーブル名');
        IF 変数 = 0 THEN
            CREATE TEMP TABLE 一時テーブル名 (key TEXT, val TEXT);
            INSERT INTO 一時テーブル名 VALUES
                ('パッケージ変数名1', '値1'),
                ('パッケージ変数名2', '値2'),
                ...;
        END IF;
    END;
    $$;
```

すでに一時テーブルが存在していないことを確認し、一時テーブルを作成します。テーブル内の列は2つでkeyに変数名を格納し、valにその変数の値を格納します。変数の値は一旦文字列として保存し取り出す際にあるべきデータ型にキャストすることになります。

(登録用関数)

```
CREATE OR REPLACE FUNCTION スキーマ名.登録用ファンクション名 (登録する変数 TEXT, 登録する値 TEXT)
    RETURNS void LANGUAGE plpgsql
    AS $$
    DECLARE
        更新する一時テーブル名 TEXT := '一時テーブル名';
        変数 INTEGER := 0;
    BEGIN
        PERFORM スキーマ名.初期化用ファンクション名 ();
        EXECUTE 'SELECT count(*) FROM ' || quote_ident(更新する一時テーブル名) || ' WHERE key = $1' INTO 変数
    USING 登録する変数;
        IF 変数 = 0 THEN
            EXECUTE 'INSERT INTO ' || quote_ident(更新する一時テーブル名) || ' VALUES ($1, $2)' USING 登録する変数, 登録する値;
        ELSE
            EXECUTE 'UPDATE ' || quote_ident(更新する一時テーブル名) || ' SET val = $1 WHERE key = $2' USING
            登録する値, 登録する変数;
        END IF;
    END;
    $$;
```

初期化用の関数を実行することですでにテーブルができている状態を確立します。条件分岐ではもし一時テーブル内に登録しようとしている変数が存在していなければ新規登録を行い、すでに存在している場合は更新を行うようになっています。

(取得用関数)

```
CREATE OR REPLACE FUNCTION スキーマ名.取得用ファンクション名 (値を取得したい変数 TEXT)
  RETURNS TEXT LANGUAGE plpgsql
AS $$
DECLARE
  取得する一時テーブル名 TEXT := '一時テーブル名';
  取得した値用変数 TEXT := NULL;
BEGIN
  PERFORM スキーマ名.初期化用ファンクション名 ();
  EXECUTE 'SELECT val FROM ' || quote_ident(取得する一時テーブル名) || ' WHERE key = $1' INTO 取得した値用変数 USING 値を取得したい変数;
  RETURN 取得した値用変数;
END;
$$;
```

上記3つの関数を使用することでDBへの接続ごとに値を保持することが可能になり関数間で使い回すことが可能になります。

ただし、PL/SQLの場合関数内でROLLBACKが実行された場合でもパッケージ変数の値は保たれますが、上記の方法ではそれを実現できていないことに注意してください。

8. SQL ServerからPostgreSQLへの移行

本項目に関しては以下の資料を御参照下さい。

https://www.pgecons.org/wp-content/uploads/PGECons/2013/WG2/06_StoredProcedureMigrationResearch.pdf#page=16

9. DB2からPostgreSQLへの移行

本項目に関しては以下の資料を御参照下さい。

https://www.pgecons.org/wp-content/uploads/PGECons/2013/WG2/06_StoredProcedureMigrationResearch.pdf#page=25

10. 異種DBMSからPostgreSQLへの移行に関するまとめ

SQLレベルであったり手続き言語の構文については、ある程度単純な置換え作業は可能と思われます。しかし業務処理を移行するためには以下の様な問題があります。

- PostgreSQLではファンクション(関数)としてのみしか実装できないので呼び出し手順が変わる
- 異種DBMSの個別機能(例えばOracleのパッケージなど)の対応が複雑もしくは代替手段がない
- 複雑なバッチ処理に必要なトランザクション制御が実装できない

このような状況を考えると、単純に移行が出来る異種DBMSのストアードプロシージャは限られてくるものと思われます。

もう一つPL/pgSQLの特徴として、実行時にソースの解析が行われます。

異種DBMSに実装されている事前コンパイル機能などにより、実行レスポンスを向上させる目的で使用しているのであれば、この部分においては移行前と同等の性能は期待できない可能性があります。

これらを総合すると処理の内容によっては、異種DBMSのストアードプロシージャは、PL/pgSQLに移行するよりも他の言語で実装する方が容易になる可能性があります。

10.1. Oracleのユーティリティパッケージについて

Oracleのストアードプロシージャでは、ユーティリティパッケージ(DBMS_OUTPUTやUTL_FILE)が、よく使用されていますが、これらはOracleが提供しているのでPostgreSQLには実装されていません。

DBMS_OUTPUTは同様の機能としてRAISE NOTICEで代用できるものもありますが、構文が違うので個別での対応が必要と思われます。

参考ですがOracleではユーティリティパッケージの一部の実装を実現しています。

但し、仕様のOracleとの違いがありますので注意が必要です。

例) DBMS_OUTPUTの通知のタイミング

Oracle トランザクションの終了時

Oracle 送信都度

11. 著者

版	所属企業・団体名	部署名	氏名
ストアドプロシージャ移行調査編 第2版 (2013年度 WG2)	クオリカ株式会社	開発センター	坂本 浩行
	インフォメーションクリエイティブ株式会社	ソリューション開発本部	林田 竜一
ストアドプロシージャ移行調査編 第3版 (2017年度 WG2)	SRA OSS, Inc. 日本支社	OSS 事業本部	佐藤 友章
	SRA OSS, Inc. 日本支社	OSS 事業本部	千田 貴大