PostgreSQLエンタープライズ・コンソーシアム 技術部会 WG#2

WG2活動報告書
ストアドプロシージャ移行調査編
(PostgreSQL 11版)

目次

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	• 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
4.0	2020/03/17	PostgreSQLの対象バージョンを11.6に更新 • 「3. はじめに」からDB2、SQL Serverの記述を削除 • 「4. PostgreSQLのストアドプロシージャについて」の記述を変更 • 「5.1. CREATE FUNCTION文」を削除 • 「5.1. CREATE PROCEDURE文」を追加 • 「5.2. CREATE PACKAGE文」の記述を変更 • 「5.3. ALTER FUNCTION文」を削除 • 「5.3. ALTER PROCEDURE文」を追加 • 「5.4. DROP FUNCTION文」を削除 • 「5.4. DROP PROCEDURE文」を追加 • 「6.3. 引数」の記述を変更 • 「6.5. 変数の宣言」の記述を変更 • 「6.7.7. REFCURSOR」の記述を変更 • 「6.8.1. EXCEPTION文」のリンクを変更 • 「7.1. 起動方法」の記述を変更 • 「7.2. 呼出方法」を「7.2. プロシージャからの呼び出し方法」に変更 • 「7.2. プロシージャからの呼び出し方法」の記述を変更 • 「7.3. トランザクション制御」の記述を変更 • 「8. 異種DBMSからPostgreSQLへの移行に関するまとめ」の記述を変更

2. ライセンス

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 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。

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	11.6
Oracle Database	11gR2 11.2.0.2.0

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 11からは、Oracleと同様にプロシージャが利用可能です。
プロシージャの実行はEXECUTE文ではなく、CALL文を使用します。
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などで処理ロジックを組み込むことも可能です。

本資料では、PL/pgSQLで記述されるプロシージャに移行する場合について記載しています。

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、ユーザ定義関数やプロシージャ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 PROCEDURE文

表 5.1 CREATE PROCEDURE文の比較

Oracle	PostgreSQL

CREATE OR REPLACE PROCEDURE
　プロシージャ名（引数名　IN　データ型)
IS
　変数名 データ型;
BEGIN
　処理内容;
END [プロシージャ名];
	CREATE OR REPLACE PROCEDURE
	　プロシージャ名 (引数名　IN　データ型)
	AS $$
	　DECLARE
	　　変数名 データ型;
	　BEGIN
	　　処理内容;
	　END;
	$$
	　LANGUAGE plpgsql;

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、この場合にはプロシージャ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PROCEDUREを作成するときにはOracleでは”()”を省略しますが、PostgreSQLでは”()”の記述が必須です。

上記以外では

IS → AS
OUT → INOUT
END [プロシージャ名]; → END;

に書き換える必要があります。
PL/SQLではEND部分にPROCEDURE名を記述することがありますが、PL/pgSQLでは記述できません。

5.2. CREATE PACKAGE文

PACKAGEは実装されていません。
FUNCTIONまたはPROCEDURE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
CREATE OR REPLACE PACKAGE パッケージ名 IS 　PROCEDURE プロシージャ名 ((後略)	CREATE SCHEMA IF NOT EXISTS スキーマ名; CREATE OR REPLACE PROCEDURE スキーマ名.プロシージャ名 ((後略)

SCHEMAを使用した場合は、PROCEDURE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PROCEDURE内で別のPROCEDURE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、一つのプロシージャ定義は一箇所に記述します。

5.3. ALTER PROCEDURE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ではプロシージャ名の変更や所有者の変更などの、PROCEDUREが保持している情報を変更する命令になります。

PostgreSQLのPL/pgSQLは事前にコンパイルしておくことはできないため、代替機能はありません。

5.4. DROP PROCEDURE文

表 5.3 DROP PROCEDURE文の比較

Oracle	PostgreSQL
DROP PROCEDURE プロシージャ名;	DROP PROCEDURE プロシージャ名 (引数名 IN データ型);

PostgreSQLでは、同名のプロシージャ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引数名とデータ型の両方、もしくはデータ型のみでの記述が可能です。

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

Oracleの引数の宣言ではIN引数、OUT引数、IN OUT引数を使用することができますが、PostgreSQLではIN引数またはINOUT引数のみ使用することができます。
OUT引数を使用していた場合はINOUT引数に書き換える必要があります。

INOUT引数プロシージャの呼び出し方

```
<INOUT引数を持つプロシージャ>
  CREATE OR REPLACE PROCEDURE プロシージャ名(変数名 INOUT データ型)
    …

<呼び出し元の宣言部>
  変数  データ型;

<呼び出し部分>
  CALL プロシージャ名(変数);
```

また、上記以外の細かな違いとして、引数に対してデフォルトの値を設定する際に := ではなく = を使うというものがあります。

Oracle	PostgreSQL

CREATE OR REPLACE PROCEDURE プロシージャ名 (
  変数名 データ型 := デフォルト値
…
	CREATE OR REPLACE PROCEDURE プロシージャ名 (
	  変数名 IN データ型 = デフォルト値
	…

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
type 変数名 is RECORD (変数名 データ型);	変数名 RECORD;

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
変数 := TO_NUMBER(値); 変数 := TO_CHAR(値);	変数 := 値::numeric; 変数 := 値::text;

NULL に関しても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。
また、例外の名前の宣言はPL/pgSQLでは宣言する事が出来ません。

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

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

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行づ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と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。オープンの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

プロシージャの引数や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 PROCEDURE プロシージャ名 ()
AS $$
DECLARE
    カーソル名 refcursor;
BEGIN
    (処理)
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ましたので、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/11/html/errcodes-appendix.html

*1 NO_DATA_FOUND に関して注意すべき点があります。
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
EXECUTE プロシージャ名	CALL プロシージャ名();

PostgreSQLではCALL文を使用してプロシージャを呼び出します。
また、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. プロシージャからの呼出方法

プロシージャの中で別のプロシージャを実行する場合、以下のように呼び出します。

```
(text型のINOUT引数を持つプロシージャを呼ぶ場合)
CREATE OR REPLACE PROCEDURE プロシージャ名()
AS $$
 DECLARE
   変数 text;
 BEGIN
   CALL 呼び出すプロシージャ名(変数);
 END;
$$
LANGUAGE plpgsql;
```

7.3. トランザクション制御

PostgreSQLのストアドプロシージャでもトランザクション制御が可能です。
ただし、例外ハンドラを伴うブロック内では使用できないなど、Oracleに比べて使用できる場所に制限があります。

例えば、以下のような場合はCOMMIT部分でエラーになります。

```
CREATE OR REPLACE PROCEDURE プロシージャ名(...)
AS $$
 BEGIN
   処理1
   COMMIT;    /* エラー */
   処理2
 EXCEPTION
   ROLLBACK;    /* 正常実行可能 */
   例外処理
 END;
$$
LANGUAGE plpgsql;
```

また、他にも以下のような違いがあります。

- プロシージャ内でエラーが発生した場合、Oracleの場合はエラーより前の処理を例外処理部でコミットすることができますが、PostgreSQLではエラーの発生したプロシージャ内で、コミットされていない処理は全てロールバックされます。
  PostgreSQLで例外処理部(EXCEPTION句)の中でコミットした場合、EXCEPTION句内でコミットより前に記述した処理と、プロシージャの呼び出し元で実行された処理をコミットします。
- プロシージャが入れ子になっていて、親プロシージャにEXCEPTION句がある場合は、トランザクション制御を含む子プロシージャを実行部(BEGINブロック)から呼ぶことはできません。

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移行開発見積り編」も参照ください。

7.5. 組み込み関数

組み込み関数に関し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もしくはプロシージャ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 $$
DECLARE
    取得する一時テーブル名 TEXT := '一時テーブル名';
    取得した値用変数 TEXT := NULL;
BEGIN
    PERORM スキーマ名.初期化用ファンクション名 ();
    EXECUTE 'SELECT val FROM ' || quote_ident(取得する一時テーブル名) || ' WHERE key = $1' INTO 取得した値用変数 USING 値を取
得したい変数;
    RETURN 取得した値用変数;
END;
$$;
```

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の場合は関数内で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. 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では呼び出し方法が変わる
- 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
- トランザクション制御の機能に制限がある

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。

8.1. Oracleのユーティリティーパッケージについて

Oracleのストア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Orafceではユーティリティーパッケージの一部の実装を実現しています。
但し、仕様的に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
Orafce　送信都度

9. 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ーティブ株式会社	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
ストアドプロシージャ移行調査編 第4版 （2019年度 WG2）	富士通株式会社	ソフトウェア事業本部	佐藤 光洋
	富士通株式会社	ソフトウェア事業本部	西垣 雅樹
	富士通株式会社	ソフトウェア事業本部	豊島 良美