PGECons
PostgreSQL Enterprise Consortium

PostgreSQL エンタープライズ・コンソーシアム 技術部会 WG#2

ストアドプロシージャ移行調査編

製作者
担当企業名
株式会社 インフォメーション クリエーティブ
クオリカ株式会社

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
2.1	2017/06/26	・「4.2. トランザクション制御」の記述の文言を一部修正。

はじめに

■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。

■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。

■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 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。

■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 2: 本書で扱う DBMS

DBMS 名称	バージョン
PostgreSQL	9.2.1
Oracle Database	11gR2 11.2.0.2.0
IBM DB2	8.2
Microsoft SQLServer	2005 Express

目次

1. 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。

1.1. 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。

1.2. 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。

2. Oracle から PostgreSQL への移行（定義関連）

2.1. CREATE FUNCTION 文

CREATE FUNCTION 文の比較

Oracle	PostgreSQL
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;	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;

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
に書き換える必要があります。

2.2. 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。

2.3. CREATE PACKAGE 文

POCEDURE 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。

2.4. 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。

2.5. DROP FUNCTION 文

DROP FUNCTION 文の比較

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

PostgreSQL では、引き渡しパラメータも含めて指定する必要があります。

3. 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。

3.1. 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

3.2. 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。
```

3.3. 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

3.4. 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。

3.5. 制御構造

3.5.1. 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。

3.5.2. 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。

3.5.3. 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。

3.5.4. 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。

3.5.5. 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。

3.5.6. IF 命令

IF については注意が必要です。

Oracle	PostgreSQL
IF *条件式*　THEN *分岐処理* 　ELSIF *条件式* THEN *分岐処理* 　ELSE *分岐処理* END IF;	IF *条件式*　THEN *分岐処理* 　ELSEIF *条件式* THEN *分岐処理* 　ELSE *分岐処理* END IF;

IF のあとの比較条件式に対して真もしくは偽を判断して THEN もしくは ELSE の後に記述された命令が実行されます。
Oracle では「ELSIF」の記述が PostgreSQL では「ELSEIF」に変わります。
この部分以外での相違はありません。

3.5.7. 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

3.5.8. 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。

3.6. カーソル

3.6.1. 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。

3.6.2. 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。

3.6.3. 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。

3.6.4. 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;	NOTFOUND;

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。

3.6.5. 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。

3.6.6. 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。

3.7. エラーハンドリング

3.7.1. 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
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。
http://www.postgresql.jp/document/9.2/html/errcodes-appendix.html#ERRCODES-TABLE

3.7.2. 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。

4. Oracle から PostgreSQL への移行（その他）

4.1. 起動方法

実行方法については注意が必要です。

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

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。

4.2. トランザクション制御

PostgreSQL のストアドファンクションは、呼び出し元の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すが、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。

5. SQL Server から PostgreSQL への移行(定義関連)

5.1. CREATE FUNCTION 文

CREATE FUNCTION 文の比較

SQL Server	PostgreSQL
CREATE FUNCTION ***ファンクション名*** (***@引数名 データ型***) RETURNS ***戻り値*** AS BEGIN DECLARE ***@変数名 データ型*** ***処理内容*** END	CREATE FUNCTION ***ファンクション名*** (***引数名*** IN ***データ型***) RETURNS ***戻り値*** 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される必要があります。
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(DECLARE ステートメント)の位置が異なりますので、書き換えが必要です。

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. ALTER FUNCTION 文

双方とも、関数名の変更など、CREATE FUNCTION ステートメントを実行して作成された関数定義を変更します。

5.4. DROP FUNCTION 文

DROP FUNCTION 文の比較

SQL Server	PostgreSQL
DROP FUNCTION ***ファンクション名***;	DROP FUNCTION ***ファンクション名*** (***引数名*** IN ***データ型***);

PostgreSQL では、引き渡しパラメータも含めて指定する必要があります。

6. SQL Server から PostgreSQL への移行（標準手続き言語関連）

SQL Server と PostgreSQL にそれぞれ実装されている手続き言語である、Transact-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. BEGIN...END

BEGIN...END には相違があります。Transact-SQL では、複数のステートメントを BEGIN...END で囲むことにより、1つの論理単位を構成するステートメントブロックとして扱うことができます。また、ネストさせることや、同じネストのレベルで複数のステートメントブロックを持つことも可能です。

一方、PL/pgSQL では処理内容を BEGIN...END で囲む（ブロックを定義する）必要があります。ネストさせることも可能ですが、最上レベルで複数のブロックを持つことはできません。

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
        行末までをコメントします。

/* コメント */
        /* 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. 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（SQLServer-PostgreSQL）」を参照してください。

6.4. 変数の宣言

変数宣言の比較

SQL Server	PostgreSQL
DECLARE *@変数名　データ型*	DECLARE *変数名　データ型* ;

双方とも、プログラム内で使用する変数は DECLARE ステートメントにより、宣言する必要があります。
宣言する位置は、Transact-SQL では任意の箇所ですが、PL/pgSQL では BEGIN の直前のみとなります。

但し、PL/pgSQL では、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。
RAISE 文を使ってエラーを発生させます。（RAISE 文については、「6.8.2. RAISERROR 文」を参照ください。）

6.5. 変数への値の代入

Transact-SQL には、変数に値を代入する方法として SELECT ステートメントや SET ステートメントなどを利用できます。
一方、PL/pgSQL では以下のように記述します。

変数名 := 値 ;

6.5.1. SELECT ステートメント

SELECT 命令の比較

SQL Server	PostgreSQL
SELECT ***@変数名*** = ***値***	***変数名*** := ***値***;

PL/pgSQL では SELECT 命令ではなく代入式に書き換えが必要です。

6.5.2. SET ステートメント

SET 命令の比較

SQL Server	PostgreSQL
SET ***@変数名*** = ***値***	***変数名*** := ***値***;

PL/pgSQL では SET 命令は実装されていません。 代入式に書き換えが必要です。

6.6. 制御構造

6.6.1. WHILE 命令

WHILE 命令の比較

SQL Server	PostgreSQL
WHILE (***条件式***) BEGIN ***繰り返し処理*** END	WHILE ***条件式*** LOOP ***繰り返し処理*** ; END LOOP;

PL/pgSQL では、「WHILE」と「LOOP」の間に繰り返す条件式を記述し、これと「END LOOP」の間に繰り返したい処理内容を記述します。

6.6.2. BREAK 命令

BREAK 命令の比較

SQL Server	PostgreSQL
BREAK	EXIT;

PL/pgSQL では BREAK 命令は実装されていません。EXIT 命令に置き換えが必要です。
なお、EXIT 命令では、以下の制御も可能です。

・EXIT ***ラベル*** ;
　　指定されたラベルのループを抜けます。
・EXIT WHEN ***条件式*** ;
　　条件式が真の場合、ループを抜けます。

6.6.3. CONTINUE 命令

CONTINUE の記述には相違ありません。ループの先頭に戻り、処理を続行します。

```
CONTINUE;
```

なお、PL/pgSQL では以下の制御が可能です。

・CONTINUE ***ラベル*** ;
　　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。
・CONTINUE　WHEN ***条件式*** ;
　　条件式が真の場合、次の反復に制御を移します。

6.6.4. IF 命令

IF 命令の比較

SQL Server	PostgreSQL
IF ***条件式*** 　***分岐処理*** ELSE 　***分岐処理***	IF ***条件式*** THEN ***分岐処理*** 　ELSIF ***条件式*** THEN ***分岐処理*** 　ELSE ***分岐処理*** END IF;

条件式に対して真もしくは偽を判断し、「THEN」もしくは「ELSE」の後に記述された命令を実行します。

なお、Transact-SQL には ELSEIF 句は存在せず、IF 文を入れ子にして指定することで、ELSIF 句と同等の処理を記述 しています。

6.6.5. CASE 命令

CASE 命令の比較

SQL Server	PostgreSQL
[対応する命令なし]	CASE ***変数*** 　WHEN ***条件値*** THEN 　　***分岐処理*** 　ELSE 　　***分岐処理*** END CASE;

CASE 命令には相違があります。Transact-SQL では関数（CASE 式）としてのみ使用でき、PL/pgSQL ではフロー制御のための命令（CASE 文）としても利用できます。

CASE 式 として利用する場合、式を含めることができる箇所（例えば、SELECT、UPDATE、WHERE 等）であれば使用できます。

6.6.6. GOTO 命令

GOTO 命令の比較

SQL Server	PostgreSQL
GOTO　*ラベル* ;	[対応する命令なし]

PL/pgSQL には GOTO 命令がなく、置換える命令もありません。
（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。）

6.6.7. WAITFOR 命令

WAITFOR 命令の比較

SQL Server	PostgreSQL
WAITFOR {DELAY '*time*' | TIME '*time*' }	[対応する命令なし]

PL/pgSQL には WAITFOR 命令がなく、置換える命令もありません。
（処理を実行する時刻、または実行するまでの待機時間を指定することはできません。）

6.7. カーソル

6.7.1. カーソルの宣言

カーソル宣言の比較

SQL Server	PostgreSQL
DECLARE ***カーソル名*** CURSOR FOR ***クエリー***	***カーソル名*** CURSOR FOR ***クエリー*** ;

Transact-SQL では DELCLARE 文でカーソルの宣言を行いますが、PL/pgSQL では DELCLARE 文は不要です。

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
  [ [ NEXT | PRIOR | FIRST | LAST | ABSOLUTE | RELATIVE ]
    FROM
  ]
 カーソル名
 INTO   取得した値を格納する変数
```

ただし、PL/pgSQL では FETCH の結果行が存在しない場合、空の結果が返され、カーソル位置はそのまま先頭行の前か最終行の後に留まります。

6.7.4. カーソルの終了判定

カーソル終了判定の比較

SQL Server	PostgreSQL
@@FETCH_STATUS ◇ 0	NOT FOUND

Transact-SQL の@@FETCH_STATUS は、最後に実行された FETCH ステートメントのステータスコードを返します。
ステータスコードと、その内容は以下のとおりです。

0 : FETCH ステートメントが成功
-1 : 最後のレコードは読み取り済みか、FETCH ステートメントが失敗した
-2 : 取り出した行がありません

一方、PL/pgSQL では、FOUND 変数で終了判定を行いますが、その内容は以下のとおりです。

TRUE : FETCH ステートメントが成功
FALSE : FETCH ステートメントが失敗

6.7.5. カーソル内のレコード数取得

レコード数取得の比較

SQL Server	PostgreSQL
@@CURSOR_ROWS	[対応する命令なし]

PL/pgSQL には、@@CURSOR_ROWS に該当する機能は実装されていません。
必要に応じて、アプリケーション側で管理する必要があります。

6.7.6. カーソルの更新／削除

カーソルのカレント行に対する更新または削除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。

6.7.7. カーソルの CLOSE

カーソル CLOSE の比較

SQL Server	PostgreSQL
CLOSE *カーソル名* DEALLOCATE *カーソル名*	CLOSE *カーソル名*%;

OPEN したカーソルを閉じます。
Transact-SQL は使用したカーソルの CLOSE 処理とメモリからの開放処理（DEALLOCATE）を実施します。
※CLOSE 命令は、再度オープンできるようにデータ構造をアクセス可能なままにします。

※DEALLOCATE 命令はそのカーソルに関するすべてのデータ構造を開放し、カーソルの定義を削除します。

一方、PL/pgSQL CLOSE 命令はすべてのデータ構造をクローズおよび開放します。

6.8. エラーハンドリング

6.8.1. TRY…CATCH 文

エラー補足の比較

SQL Server	PostgreSQL
BEGIN TRY ***処理内容*** END TRY BEGIN CATCH ***エラー処理内容*** END CATCH	BEGIN ***処理内容*** EXCEPTION WHEN ***エラーコード(もしくは例外名)*** THEN ***エラー処理内容1*** WHEN OTHERS THEN ***エラー処理内容2*** END;

PL/pgSQL には TRY…CATCH は実装されていません。EXCEPTION 句に書き換える必要があります。

6.8.2. RAISERROR 文

Transact-SQL の RAISERROR 命令は、PL/pgSQL では RAISE 命令に置き換えが必要です。

RAISERROR 文の比較

SQL Server	PostgreSQL
RAISERROR(***メッセージ番号 (もしくはエラーメッセージ)***, ***重大度レベル*** , ***状態***);	RAISE ***レベル*** ***フォーマット*** ;

Transact-SQL で設定する「メッセージ番号(もしくはエラーメッセージ)」は、PL/pgSQL では「フォーマット」で設定します。
双方とも、オプションで変換指定文字を埋め込み、変換させることも可能です。ただし、Transact-SQL では型指定などが可能(たとえば、文字列を指定する場合は、"%s")ですが、PL/pgSQL では指定することはできません("%"で指定)。

Transact-SQL では、オプション「重要度レベル」を指定することが可能です。通常のアプリが指定可能な範囲は 0 から 18 までであり、11～18 を指定した場合、TRY…CATCH 構造の CATCH ブロックに移動します。
これは、PL/pgSQL のオプション「レベル」で"EXCEPTION"を指定し、SQLSTATE もしくは状況名を指定することで置換えが可能です。

Transact-SQL では、「状態」を指定することができます。これには 1～127 の任意の整数を指定することができます。しかし、PL/pgSQL には該当する機能は実装されていません。

7. SQL Server から PostgreSQL への移行（その他）

7.1. 起動方法

実行方法の比較

SQL Server	PostgreSQL
EXECUTE ***プロシージャ名*** [***引数*** , ・・・]	SELECT ***ファンクション名***([***引数*** , ・・・]);

実行方法については、書き換えが必要です。
PL/pgSQL 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。

7.2. トランザクション制御

トランザクションの比較

SQL Server	PostgreSQL
BEGIN TRY BEGIN TRANSACTION ***処理内容*** COMMIT TRANSACTION END TRY BEGIN CATCH ***エラー処理内容*** ROLLBACK TRANSACTION END CATCH	[対応する命令なし]

PL/pgSQL 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。
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。

8. DB2 から PostgreSQL への移行（定義関連）

8.1. CREATE FUNCTION（SQL スカラー関数）文

CREATE FUNCTION 文の比較

DB2	PostgreSQL
CREATE OR REPLACE FUNCTION ***ファンクション名*** (IN ***引数名　データ型***) RETURNS ***戻り値*** LANGUAGE SQL BEGIN 　DECLARE　***変数名　データ型***; 　***処理内容***; END	CREATE FUNCTION ***ファンクション名*** (***引数名*** IN ***データ型***) RETURNS ***戻り値*** 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宣言(LANGUAGE 文)を記述する場所が違い処理記述の後に行います。

引数指定部では入出力指定と変数名の記述順を入れ替える必要があります。

DB2：　入出力指定＋　変数名＋　データ型
PostgreSQL：　変数名＋　入出力指定＋　データ型

引数を持たない FUNCTION を作成するとき、には DB2 では"()"を省略できますが、PostgreSQL では"()"の記述が必須です。

処理記述の開始前には AS の記述が必須です。

8.2. 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。

8.3. ALTER FUNCTION 文

DB2 と PostgreSQL では互換性がありません。

DB2 では関数のプロパティーの変更を行う命令になります。
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。

8.4. DROP FUNCTION 文

DROP FUNCTION 文の比較

DB2	PostgreSQL
DROP FUNCTION ***ファンクション名***;	DROP FUNCTION ファンクション***名*** (***引数名*** IN ***データ型***);

PostgreSQL では、引き渡しパラメータも含めて指定する必要があります。

9. DB2 から PostgreSQL への移行（標準手続き言語関連）

DB2 と PostgreSQL にそれぞれ実装されている手続き言語である、SQL PL 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。

9.1. 構造

変数の宣言を行う場所が相違しているので注意が必要です。
構造部分の比較

DB2	PostgreSQL
BEGIN DECLARE　***変数名　データ型***; ***処理内容***; END;	DECLARE ***変数名　データ型***; BEGIN ***処理内容***; END;

「BEGIN 部」で処理内容の記述
DB2 では BEGIN 部で DECLARE 文を使って変数の宣言ができますが、PL/pgSQL では出来ません。
BEGIN 部の外側で DECLARE 部を使って事前に宣言してください。
「END」でブロックの終了

9.2. 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。

9.3. 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（DB2-PostgreSQL）」を参照して変換してください。

9.4. 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。

9.5. 変数への値の代入

値の代入には書換が必要です。
代入命令の比較

DB2	PostgreSQL
SET ***変数名*** := ***値***; SET ***変数名*** = (***SELECT ステートメント***);	***変数名*** := ***値***; ***変数名*** := (***SELECT ステートメント***);

以下の点に関して書換が必要です。
・SET 命令が不要
・等号(=)の前にコロン(:=)を付加

SELECT ステートメントの例） SELECT COUNT(*) FROM foo

9.6. 制御構造

9.6.1. LOOP 命令

LOOP 命令の比較

DB2	PostgreSQL
ラベル名:LOOP 　***繰り返し処理***; 　IF ***条件式*** THEN LEAVE ***ラベル名***; END LOOP ***ラベル名***;	<<***ラベル名***>> LOOP 　***繰り返し処理***; 　IF ***条件式*** THEN EXIT ***ラベル名***; END LOOP ***ラベル名*** ;

繰り返し処理である LOOP 命令そのものは同じですが、LOOP の反復制御の命令が違います。
「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 LEAVE ではなく EXIT を使用します。
同様に LOOP の先頭に戻る ITERATE は CONTINUE に置き換えます。

9.6.2. WHILE 命令

WHILE 命令の比較

DB2	PostgreSQL
WHILE ***条件式*** DO 　***繰り返し処理***; END WHILE;	WHILE ***条件式*** LOOP 　***繰り返し処理***; END LOOP;

WHILE の記述には注意が必要です。
「DO」を「LOOP」に書き換えて「WHILE」との間に繰り返しの条件式を記述します。
「END WHILE」を「END LOOP」に書き換えて繰り返す命令を「LOOP」との間に記述します。
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。

9.6.3. REPEAT 命令

REPEAT 命令の比較

DB2	PostgreSQL
REPEAT 　***繰り返し処理***; UNTIL ***条件式*** END REPEAT;	［対応する命令なし］

PostgreSQL には REPEAT 命令は実装されていません。
WHILE 命令等で書き換える必要があります。

9.6.4. FOR 命令

同じ FOR 命令がありますが処理内容が違います。
DB2 の FOR 命令は指定した SELECT ステートメントから戻された結果セットを 1 行づつ処理するために使用します。
PostgreSQL では LOOP 命令や WHILE 命令と同等で処理ルーチンを繰り返すことを目的としています。
したがって、書き換えるためには SELECT ステートメントをカーソル化して繰り返し処理するように記述する必要

があります。

9.6.5. LEAVE 命令

LEAVE 命令の比較

DB2	PostgreSQL
LEAVE ***ラベル名***;	EXIT ***ラベル名***;

LEAVE 命令は EXIT 命令に書き換える必要があります。
指定されたラベルの繰り返し処理を抜けます。
PL/pgSQL の EXIT 命令は WHEN に続いて条件式を記述することが出来ます。
条件式が真になると EXIT を実行することが出来ます。

9.6.6. ITERATE 命令

ITERATE 命令の比較

DB2	PostgreSQL
ITERATE ***ラベル名***;	CONTINUE ***ラベル名***;

ITERATE 命令は CONTINUE 命令に書き換える必要があります。
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。
PL/pgSQL の CONTINUE 命令は WHEN に続いて条件式を記述することが出来ます。
条件式が真になると CONTINUE を実行することが出来ます。

9.6.7. IF 命令

IF の記述には相違ありません。

```
IF 条件式   THEN  分岐処理
   ELSEIF  条件式  THEN  分岐処理
   ELSEIF  分岐処理
END IF;
```

IF のあとの比較条件式に対して真もしくは偽を判断して THEN もしくは ELSE の後に記述された命令が実行されます。

9.6.8. 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。

9.6.9. GOTO 命令

GOTO 命令の比較

DB2	PostgreSQL
GOTO ***ラベル名***;	[対応する命令なし]

PL/pgSQL には GOTO 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。

9.7. カーソル

9.7.1. カーソルの宣言

カーソルの宣言方法の比較

DB2	PostgreSQL
DECLARE ***カーソル名*** CURSOR FOR ***クエリー***;	***カーソル名*** CURSOR FOR ***クエリー***;

カーソルの宣言については注意が必要です。
DB2 では DECLARE 命令で行いますが、PL/pgSQL では DECLARE 部で行います。

9.7.2. 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。

9.7.3. 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。

9.7.4. カーソルの終了判定

カーソルの終了判定方法の比較

DB2	PostgreSQL
・条件ハンドラの設定と条件式の組み合わせで判断 ・SQLCODE か SQLSTATE の値を条件式で判断	EXIT WHEN NOTFOUND;

LOOP 処理でカーソルをすべて FETCH したときの判定方法は注意が必要です。

DB2 で FETCH の終了判定を行うためには「条件ハンドラ」を事前に定義して条件式で判定を行うか、SQLCODE などを条件式に使って FETCH LOOP の終了判定をおこないます。
PL/pgSQL では FETCH 命令の後に「EXIT WHEN NOTFOUND;」と記述することで FETCH LOOP を抜け出すことが出来ます。

9.7.5. 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。

9.7.6. 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。

9.8. エラーハンドリング

9.8.1. DECLARE HANDLER 命令

DECLARE HANDLER 命令の比較

DB2	PostgreSQL
DECLARE CONTINUE HANDLER FOR SQLSTATE '0200' SET I =0;	[対応する命令なし]

PL/pgSQL には DECLARE HANDLER 命令は実装されていません。
DB2 は事前に DECLARE HANDLER 命令で例外が発生した時に呼び出される条件ハンドラを定義して、指定された例外に対する処理内容をおこない、処理の制御を例外発生場所に戻したり、ロールバックすることが出来ます。
PL/pgSQL には同等な命令はありませんが EXCEPTION 命令を使って同等のエラーハンドリングを行います。

9.8.2. SIGNAL 命令

SIGNAL 文の比較

DB2	PostgreSQL
SIGNAL SQLSTATE 条件値 ;	RAISE EXCEPTION SQLSTATE = 条件値;

例外エラーを通知する命令ですが PL/pgSQL には SIGNAL 命令と同等な命令に書換が可能ですが注意が必要です。
PostgreSQL では RAISE を使用します。
設定する SQLSTATE の値は、PostgreSQL 内で使用されていないものであることを確認する必要があります。

10. DB2 から PostgreSQL への移行（その他）

10.1. 起動方法

実行方法の比較

DB2	PostgreSQL
CALL *プロシージャ名*;	SELECT *ファンクション名*();

実行方法については書き換えが必要です。
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。
戻り値は、GET DIAGNOSTICS で取得していた部分を SELECT の INTO に書き換えて取得します。

10.2. 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。
COMMIT は呼び出し元の処理との整合を合わせて、呼び出し元で行う必要があります。

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。

11. 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。

11.1. 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

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