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

組み込み関数移行調査編

製作者
担当企業名 TIS 株式会社
NEC ソリューションイノベータ株式会社
富士通株式会社
株式会社富士通ソーシアルサイエンスラボラトリ
三菱電機株式会社

改訂履歴

版	改訂日	変更内容
1.0	2013/04/22	新規作成
2.0	2016/04/13	2015 年度 活動内容の反映 ・最新バージョンへの対応 ・3～5 章の追加
2.1	2016/05/24	「5. 移行が難しいと判断される関数」より TO_MULTI_BYTE, TO_SINGLE_BYTE の記述を削除(orafce 3.2.1 で実装済のため)

はじめに

■本資料の目的

本資料は、異種 DBMS から PostgreSQL へ組み込み関数を移行する作業の難易度、ボリュームを事前に判断するための参考資料として利用す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 を想定し、Oracle Database から PostgreSQL へ SQL を移行する際に組み込み関数の対応状況や仕様の相違により書き換えが必要となる箇所について記載します。

■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 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。

■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。2015 年度の調査では対象の PostgreSQL、orafce のバージョンを最新に変更しています。新しいバージョンでのみ該当する内容については文書内で明示します。

表 2: 本書で扱う DBMS

DBMS 名称	バージョン	
	2012 年度(1 版)	2015 年度(2 版)
PostgreSQL	9.2.0	9.5.0
Oracle Database	11gR2 11.2.0.2.0	11gR2 11.2.0.2.0
orafce	3.0.3	3.2.1

目次

1. 組み込み関数の対応状況

1.1. OracleとPostgreSQLにおける対応状況の違い

OracleとPostgreSQLでは対応している組み込み関数に違いがあるため、移行元のOracleで使える組み込み関数が移行先のPostgreSQLに存在しないことがあります。移行元の組み込み関数が移行先に存在しない場合は、同等の機能を実現する独自のファンクションを定義する等の対応が必要となります。

OracleとPostgreSQLにおける組み込み関数の対応状況は、別紙「組み込み関数対応表(Oracle-PostgreSQL)」を参照してください。

1.2. Oracle互換関数 orafce について

orafce[1]はOracleと互換性のある関数をPostgreSQLに実装することを目的としたプロジェクトです。orafceを導入してPostgreSQLで対応する組み込み関数を追加することで、Oracleからの移行の省力化が期待できます。別紙「組み込み関数対応表(Oracle-PostgreSQL)」において、orafceを導入することで追加される組み込み関数には「orafceでの対応可否」列に「○」を付与しています。

orafceは一般的なPostgreSQLのcontribモジュールと同様に、PostgreSQLのEXTENSIONとしてインストールすることができます。

【Linux環境におけるorafceのインストール手順】

```
※orafceプロジェクトからダウンロードしたソースコードを展開
# cp orafce-3.2.1.tar.gz /usr/local/src/pgsql/contrib
# cd /usr/local/src/pgsql/contrib
# tar zxvf orafce-3.2.1.tar.gz
# cd orafce-3.2.1
※make, make install を実行し、ライブラリをインストール
# make
# make install
※データベースに関数を登録するCREATE EXTENSIONを実行
# su - postgres
$ psql -d postgres -c "create extension orafce"
※next_day関数の実行例
$ psql -d postgres
psql (9.5.0)
Type "help" for help.

postgres=# SELECT next_day(current_date,'saturday');
  next_day
------------
 2016-03-05
(1 row)
```

1 https://github.com/orafce/orafce

2. 組み込み関数の移行方法

本章では Oracle と PostgreSQL の対応状況の違いから、そのままでは移行できない組み込み関数への対応方法を紹介します。なお、ここで紹介するのは代表的な関数のみで、全ての対応パターンは網羅していません。

2.1. 数値ファンクション

2.1.1. BITAND

Oracle の BITAND は2つの引数をビット単位で AND 計算する関数です。

【Oracle の BITAND の使用例】

```
SQL> SELECT BITAND(6,3) FROM DUAL;

BITAND(6,3)
-----------
          2     ※数値 6(バイナリ 1,1,0)と数値 3(バイナリ 0,1,1)の AND は数値 2 (0,1,0)
```

PostgreSQL には BITAND 関数が存在しませんが、＆演算子で同様の演算が可能です。

【PostgreSQL での＆演算子の使用例】

```
postgres=# SELECT integer '6' & integer '3';
6 & 3
-----
    2
(1 行)
```

なお、orafce には BITAND 関数が実装されていますので、orafce を導入した PostgreSQL では関数を変更せずに移行することができます。

2.1.2. COSH

Oracle の COSH は引数の双曲線コサインを計算する関数です。

【Oracle の COSH の使用例】

```
SQL> SELECT COSH(0) "Hyperbolic cosine of 0" FROM DUAL;

Hyperbolic cosine of 0
----------------------
                     1
```

PostgreSQL には COSH 関数が存在しませんが、EXP 関数を使って、「(exp(n) + exp(-n)) / 2」という式に置き換えることで同様の演算が可能です。

なお、orafce には COSH 関数が実装されていますので、orafce を導入した PostgreSQL では関数を変更せずに移行することができます。

2.1.3. SINH

Oracle の SINH は引数の双曲線サインを計算する関数です。PostgreSQL にはSINH関数が存在しませんが、2.1.2 と同様に EXP 関数を使って、「(exp(n)-exp(-n))/2」という式に置き換えることで同様の演算が可能です。

また、orafce には SINH 関数が実装されていますので、orafce を導入した PostgreSQL では関数を変更せずに移行することができます。

2.1.4. TANH

Oracle の TANH は引数の双曲線タンジェントを計算する関数です。PostgreSQL には TANH 関数が存在しませんが、2.1.2 と同様に EXP 関数を使って、「(exp(n)-exp(-n))/(exp(n)+exp(-n))」という式に置き換えることで同様の演算が可能です。

また、orafce には TANH 関数が実装されていますので、orafce を導入した PostgreSQL では関数を変更せずに移行することができます。

2.1.5. NANVL

Oracle の NANVL は引数として n1,n2 をとり、n2 が非数値の場合は代替値 n1 を、数値の場合は n2 を戻す関数です。一方、PostgreSQL には NANVL 関数が存在しませんが、orafce に NANVL 関数が実装されていますので、orafce を導入した PostgreSQL では関数を変更せずに移行することができます。

2.1.6. REMAINDER

Oracle の REMAINDER は引数として n1,n2 をとり、n1 を n2 で割った余りを求める関数です。

PostgreSQL、orafce ともに REMAINDER 関数が存在しません。また、余りを求める点では MOD と類似した関数ですが、内部の計算方法が異なる[2]ため、REMAINDER を MOD で置換することはできません。同様の演算を実現するには「n1 － n2 ＊ ROUND (n1 / n2) 」という式に置き換える必要があります。

2.2. 文字値を戻す文字ファンクション

2.2.1. SUBSTR

Oracle の SUBSTR は引数として char, position, substring_length を取り、char の position 番目から substring_length 文字分の文字列を抜き出して戻す関数です。PostgreSQL にも SUBSTR 関数が存在するため、特に変更するなく利用できます。ただし、第 2 引数の position が負値の場合の挙動が異なる点に注意が必要です。

【Oracle の SUBSTR の使用例 】

```
SQL> SELECT SUBSTR('ABCDEFG',3,4) "Substring" FROM DUAL;

Substring
---------
CDEF

※ Oracle では第 2 引数が負値の場合、文字列の後ろから開始位置をカウントする。
SQL> SELECT SUBSTR('ABCDEFG',-5,4) "Substring" FROM DUAL;

Substring
---------
CDEF
```

【PostgreSQL の SUBSTR の使用例 】

```
postgres=# select substr('ABCDEFG',3,4) "Substring";
 Substring
-----------
 CDEF
(1 行)

※PostgreSQL では第 2 引数が負値の場合の挙動が Oracle と異なる。
postgres=# select substr('ABCDEFG',-5,4) "Substring";
 Substring
-----------

(1 行)
```

なお、orafce では上記の点について互換性を向上させた SUBSTR 関数が実装されていますので、orafce を導入した PostgreSQL ではそのまま移行することができます。

2.3. 数値を戻す文字ファンクション

2.3.1. INSTR

Oracle の INSTR は引数 string の substring を検索する関数で、見つかった substring の位置を返します。

【Oracle の INSTR の使用例】

2 Oracle の MOD 関数の計算式: n1 － n2 ＊ TRUNC (n1 / n2)、
Oracle の REMAINDER 関数の計算式: n1 － n2 ＊ ROUND (n1 / n2)

```
SQL> SELECT INSTR('abcdefg','b') FROM DUAL;

INSTR('abcdefg','b')
--------------------
                   2
```

一方、PostgreSQL には INSTR 関数が存在しませんが、strpos 関数に置き換えることで同等の機能を実現できます。また、orafce には INSTR 関数が実装されていますので、orafce を導入した PostgreSQL では関数を変更せずに移行することができます。

【PostgreSQL の strpos 関数で置き換えた例】

```
postgres=# select strpos('abcdefg','b');
 strpos
--------
      2
(1 row)
```

2.3.2. NLSSORT

Oracle の NLSSORT は引数として文字列を受け取り、文字列比較やソートに使用するための、辞書順となるようなバイナリ値を返します。また第二引数で照合順序を指定することも可能です。

【Oracle の NLSSORT の使用例 】

```
SQL> SELECT *
  2    FROM test
  3    ORDER BY name;

NAME
------------------------------
Gaardiner
Gaasten
Gaberd

※ デンマークの言語ソート基準でソート、デンマーク語では aa は å(合字)の代用として扱われます。
SQL> SELECT *
  2    FROM test
  3    ORDER BY NLSSORT(name, 'NLS_SORT = XDanish');

NAME
------------------------------
Gaberd
Gaardiner
Gaasten
```

一方、PostgreSQL には NLSSORT 関数は存在しません。NLSSORT 関数を呼び出す SQL を書き換えずに移行するためには、同様の計算を行うファンクションを作成する方法があります。

なお、orafce には NLSSORT 関数が実装されていますので、orafce を導入した PostgreSQL であればファンクションを作成せずとも利用可能です。

ただし、照合順序の指定方法が異なるため、SQL の書き換え[3]が必要となります。

表 2.1: 照合順序指定方法

DBMS 名称	フォーマット	記載例
Oracle Database	NLS_SORT = [言語ソート基準[4]]	NLS_SORT = JAPANESE_M
PostgreSQL	[言語]_[地域].[文字符号化方式][5]	ja_JP.UTF-8

3 PostgreSQL のソート順序はプラットフォーム依存であり、Oracle の照合順序には完全に対応できていないことにご注意下さい。(大文字/小文字/全角/半角で同様のコードポイントを返す指定など)

4 https://docs.oracle.com/cd/E16338_01/server.112/b56307/applocaledata.htm#i637232

5 https://www.postgresql.jp/document/9.5/html/charset.html

SQL の書き換えを行いたくない場合は Oracle 形式の引数を PostgreSQL 形式（本書では Linux を想定しているため、POSIX ロケール名[6]）に変換する処理を追加（ソースに追記かラッパー定義など）することで対処は可能です。

【orafce 導入済の PostgreSQL での NLSSORT の使用例】

```
postgres=# SELECT * FROM test ORDER BY name;
   name
-----------
 Gaardiner
 Gaasten
 Gaberd
(3 rows)

※ デンマークの言語ソート基準でソート
postgres=# SELECT * FROM test ORDER BY nlssort(name, 'da_DK.UTF-8');
   name
-----------
 Gaberd
 Gaardiner
 Gaasten
(3 rows)
```

2.4. 日時ファンクション

2.4.1. ADD_MONTHS

Oracle の ADD_MONTHS は月を演算する関数で、引数の日付に月数を加えて戻します。

【Oracle の ADD_MONTHS の使用例 】

```
SQL> SELECT ADD_MONTHS('2013/3/22',1) FROM DUAL;

ADD_MONT
--------
13-04-22
```

一方、PostgreSQL には ADD_MONTHS 関数が存在しないため、算術演算子を使った書き換えが必要です。

【PostgreSQL で 2013/3/22 の 1 か月後の日付を求める例】

```
postgres=# SELECT date '2013-03-22' + interval '1 months';
  ?column?
---------------------
 2013-04-22 00:00:00
(1 行)
```

なお、orafce には ADD_MONTHS 関数が実装されていますので、orafce を導入した PostgreSQL では SQL を書き換えずに移行することができます。

【orafce 導入済の PostgreSQL での ADD_MONTHS の使用例】

```
postgres=# SELECT add_months(date '2013-03-22',1);
add_months
------------
 2013-04-22
(1 行)
```

2.4.2. CURRENT_DATE

Oracle の CURRENT_DATE は 、データベースサーバー の OS のシステムコールで取得した日付をセッションタイムゾーンに変換した現在日付と時刻（年～秒）を DATE 型で戻します。

【Oracle の CURRENT_DATE の使用例 】

6　OS がサポートしているロケールは「locale -a」コマンドで確認することが出来ます。

```
SQL> select CURRENT_DATE from dual;

CURRENT_DATE
-------------------
2013-04-22 16:28:54
```

一方、PostgreSQL にも現在日付を戻す同名の関数が存在します。ただし、PostgreSQL の CURRENT_DATE で戻される DATE 型は日付までのデータしか保有しない点に注意が必要です。時分秒までのデータが必要な場合は、CURRENT_TIMESTAMP に置き換えてください。

【PostgreSQL の CURRENT_DATE の使用例 】

```
postgres=# SELECT CURRENT_DATE;
CURRENT_DATE
------------
 2013-04-22
(1 行)
```

2.4.3. CURRENT_TIMESTAMP

Oracle の CURRENT_TIMESTAMP は 、データベースサーバー の OS のシステムコールで取得した日付をセッションタイムゾーンに変換した現在日付と時刻(年～秒)を TIMESTAMP WITH TIME ZONE データ型の値で戻します。

【Oracle の CURRENT_TIMESTAMP の使用例 】

```
SQL> select CURRENT_TIMESTAMP from dual;

CURRENT_TIMESTAMP
---------------------------------
13-04-22 16:37:04.281000 +09:00
```

PostgreSQL の CURRENT_TIMESTAMP も Oracle と同じ TIMESTAMP WITH TIMEZONE 型を戻すので、そのまま移行することができます。

【PostgreSQL の CURRENT_TIMESTAMP の使用例 】

```
postgres=# SELECT CURRENT_TIMESTAMP;
CURRENT_TIMESTAMP
------------
2013-04-22 14:39:53.662522+09
(1 行)
```

ただし、Oracle と PostgreSQL でデフォルトの出力フォーマットは異なるため、to_char() 関数等を使って出力フォーマットは揃える必要があります。

2.4.4. SYSDATE

Oracle の SYSDATE は、データベースサーバーが稼動する OS のシステムコールから取得した日付と時刻(年～秒)を元に現在日付を DATE 型で戻します。

【Oracle の SYSDATE の使用例 】

```
SQL> select SYSDATE from dual;

SYSDATE
-------------------
2013-04-22 16:07:15
```

一方、PostgreSQL には SYSDATE 関数が存在しないため、前述の CURRENT_DATE、CURRENT_TIMESTAMP として置き換えることを検討します。なお、PostgreSQL の CURRENT_DATE、CURRENT_TIMESTAMP は「現在のトランザクションの開始時刻に基づいた値」を返す仕様のため、実際の現在時刻を取得したい場合は CLOCK_TIMESTAMP を利用します。

また、orafce には同様の値を timestamp 型として取得できる ORACLE.SYSDATE 関数[7]が実装されていますので、orafce を導入した PostgreSQL では ORACLE.SYSDATE に書き換えることで移行することができます。

【orafce 導入済の PostgreSQL での ORACLE.SYSDATE の使用例】

```
postgres=# SELECT oracle.sysdate();
       sysdate
---------------------
 2016-02-27 09:35:28
(1 行)
```

2.4.5. SYSTIMESTAMP

Oracle の SYSTIMESTAMP は、データベースサーバーが稼動する OS のシステムコールから取得した秒の小数部(ミリ秒～ナノ秒)と「タイムゾーンを含む」日付と時間を戻します。

PostgreSQL には SYSTIMESTAMP 関数が存在しないため、SYSDATE と同様に CURRENT_TIMESTAMP もしくは CLOCK_TIMESTAMP を利用します。

2.4.6. LAST_DAY

Oracle の LAST_DAY は、引数の日付の月末を求める関数です。

【Oracle の LAST_DAY の使用例 】

```
SQL> SELECT LAST_DAY('2013/4/22') FROM DUAL;

LAST_DAY
--------
13-04-30
```

一方、PostgreSQL には LAST_DAY 関数が存在しません。LAST_DAY 関数を呼び出す SQL を書き換えずに移行するためには、同様の計算を行うファンクションを作成する方法があります。

なお、orafce には LAST_DAY 関数が実装されていますので、orafce を導入した PostgreSQL では SQL を書き換えずに移行することができます。

【orafce 導入済の PostgreSQL での LAST_DAY の使用例】

```
postgres=# SELECT last_day(date '2013-04-22');
last_day
------------
 2013-04-30
(1 行)
```

2.4.7. NEXT_DAY

Oracle の NEXT_DAY は、指定した曜日で引数の日付より後の最初の日付を求める関数です。

【Oracle の NEXT_DAY の使用例 】

```
SQL> SELECT NEXT_DAY('2013/4/22','SUNDAY') FROM DUAL;

NEXT DAY
----------
2013-04-28
```

一方、PostgreSQL には NEXT_DAY 関数が存在しません。NEXT_DAY 関数を呼び出す SQL を書き換えずに移行するためには、同様の計算を行うファンクションを作成する方法があります。

なお、orafce には NEXT_DAY 関数が実装されていますので、orafce を導入した PostgreSQL では SQL を書き換えずに移行することができます。

【orafce 導入済の PostgreSQL での NEXT_DAY の使用例】

```
postgres=# SELECT next_day(date '2013-04-22', 'sunday');
next_day
------------
```

7 orafce 3.2.1 から導入された関数です。なお、この関数は、orafce.timezone パラメータで指定したタイムゾーンの時刻を戻します。orafce.timezone パラメータが設定されていない場合はタイムゾーンには GMT が使用されます。

```
 2013-04-28
(1 行)
```

2.4.8. MONTHS_BETWEEN

Oracle の MONTHS_BETWEEN は、2 番目の引数から 1 番目の引数までの月数を求める関数です。

【Oracle の MONTHS_BETWEEN の使用例 】

```
SQL> SELECT MONTHS_BETWEEN('2013/03/15','2012/02/20') FROM DUAL;

MONTHS_BETWEEN('2013/03/15','2012/02/20')
-----------------------------------------
                                12.8387097
```

一方、PostgreSQL には MONTHS_BETWEEN 関数が存在しません。MONTHS_BETWEEN 関数を呼び出す SQL を書き換えずに移行するためには、同様の計算を行うファンクションを作成する方法があります。

なお、orafce には MONTHS_BETWEEN 関数が実装されていますので、orafce を導入した PostgreSQL では SQL を書き換えずに移行することができます。

【orafce 導入済の PostgreSQL での MONTHS_BETWEEN の使用例】

```
postgres=# SELECT months_between(date '2013-03-15', '2012-02-20');
months_between
------------
 12.8387097
(1 行)
```

2.4.9. ROUND

Oracle の ROUND は、第 1 引数で指定した日時を、第 2 引数で指定した書式[8]の単位に丸めた結果を求める関数です。

【Oracle の ROUND の使用例 】

```
SQL> SELECT ROUND (TO_DATE ('27-OCT-12'),'YEAR') "New Year" FROM DUAL;

New Year
---------
01-JAN-13
```

一方、PostgreSQL には日付を引数に取る ROUND 関数は存在しません[9]。ROUND 関数を呼び出す SQL を書き換えずに移行するためには、同様の計算を行うファンクションを作成する方法があります。

なお、orafce には ROUND 関数が実装されていますので、orafce を導入した PostgreSQL では SQL を書き換えずに移行することができます。

【orafce 導入済の PostgreSQL での ROUND の使用例】

```
postgres=# SELECT round(date '2012-07-12', 'yyyy');
round
----------
2013-01-01
(1 行)
```

2.4.10. TRUNC

Oracle の TRUNC は、引数の日付を書式モデルで指定した単位まで切り捨てた結果を求める関数です。

【Oracle のTRUNCの使用例 】

```
SQL> SELECT TRUNC(TO_DATE('27-OCT-12','DD-MON-YY'), 'YEAR') "New Year" FROM DUAL;
```

8 具体的な書式については、https://docs.oracle.com/cd/E16338_01/server.112/b56299/sql_elements004.htm を参照して下さい。

9 引数に数値を取り、指定した小数点位置で丸める ROUND 関数は存在します。

```
New Year
---------
01-JAN-12
```

一方、PostgreSQL には日付を引数に取る TRUNC 関数は存在しません[10]。TRUNC 関数を呼び出す SQL を書き換えずに移行するためには、同様の計算を行うファンクションを作成する方法があります。

なお、orafce には TRUNC 関数が実装されていますので、orafce を導入した PostgreSQL では SQL を書き換えずに移行することができます。

【orafce 導入済の PostgreSQL での TRUNC の使用例】

```
postgres=# SELECT trunc(date '2012-07-12', 'yyyy');
round
-----------
2012-01-01
(1 行)
```

2.4.11. DBTIMEZONE

Oracle の DBTIMEZONE は、データベースに設定されているタイムゾーンを取得する関数です[11]。

【Oracle の DBTIMEZONE の使用例 】

```
SQL> SELECT DBTIMEZONE FROM DUAL;

DBTIMEZONE
------------
+00:00
```

一方、PostgreSQL にはデータベースのタイムゾーンを取得する DBTIMEZONE 関数は存在しませんので、システムカタログ pg_settings から、'TimeZone'の reset_val の値を取得する必要があります。

【PostgreSQL での pg_settings の取得例】

```
ppostgres=# SELECT reset_val FROM pg_settings WHERE name = 'TimeZone';
 reset_val
------------
 Asia/Tokyo
(1 行)
```

また、orafce には同様の値を取得できる ORACLE.DBTIMEZONE 関数[12]が実装されていますので、orafce を導入した PostgreSQL では ORACLE.DBTIMEZONE に書き換えることで移行することができます。

【orafce 導入済の PostgreSQL での ORACLE.DBTIMEZONE の使用例】

```
postgres=# SELECT oracle.dbtimezone();
 dbtimezone
------------
 GMT
(1 行)
```

2.4.12. FROM_TZ

Oracle の FROM_TZ は、(タイムゾーン無し)TIMESTAMP データ型の値を、TIMESTAMP WITH TIME ZONE データ型に変換する関数です。

【Oracle の FROM_TZ の使用例 】

```
SQL> SELECT FROM_TZ(TIMESTAMP '2016-01-01 00:00:00', 'UTC') FROM DUAL;

FROM_TZ(TIMESTAMP'2016-01-0100:00:00','UTC')
---------------------------------------------------------------------------
```

10 引数に数値を取り、指定した小数点位置で切り捨てる TRUNC 関数は存在します。

11 オフセット値で戻されるか、タイムゾーン名で戻されるかは TIME_ZONE パラメータの設定に依存します。

12 orafce 3.2.1 から導入された関数です。なお、この関数は、orafce.timezone パラメータで指定したタイムゾーンを戻します。orafce.timezone パラメータが設定されていない場合は GMT を戻します。

```
16-01-01 00:00:00.000000000 UTC

SQL> SELECT FROM_TZ(TIMESTAMP '2016-01-01 00:00:00', 'ASIA/TOKYO') FROM DUAL;

FROM_TZ(TIMESTAMP'2016-01-0100:00:00','ASIA/TOKYO')
---------------------------------------------------------------------------
16-01-01 00:00:00.000000000 ASIA/TOKYO
```

一方、PostgreSQL には FROM_TZ 関数は存在しません。しかし、同等の関数として TIMEZONE 関数を使用できますので、TIMEZONE 関数[13]による置き換えを行います。なお、FROM_TZ 関数と TIMEZONE 関数とでは、引数の順番が逆になります。

【PostgreSQL での TIMEZONE の使用例】

```
postgres=# SELECT timezone('UTC', TIMESTAMP'2016-01-01 00:00:00');
        timezone
------------------------
 2016-01-01 09:00:00+09
(1 行)


postgres=# SELECT timezone('ASIA/TOKYO', TIMESTAMP'2016-01-01 00:00:00');
        timezone
------------------------
 2016-01-01 00:00:00+09
(1 行)
```

2.4.13. NEW_TIME

Oracle の NEW_TIME は、指定したタイムゾーンの日付と時刻を、別のタイムゾーンの日付と時刻に変換する関数です。

【Oracle の NEW_TIME の使用例 】

```
SQL> SELECT NEW_TIME(DATE'2016-01-01', 'GMT', 'PST') FROM DUAL;

NEW_TIME
--------
15-12-31

SQL> SELECT TO_CHAR(NEW_TIME(DATE'2016-01-01', 'GMT', 'PST'), 'YYYY-MM-DD HH24:MI:SS') FROM DUAL;

TO_CHAR(NEW_TIME(DATE'2016-01-01','GMT
--------------------------------------
2015-12-31 16:00:00
```

一方、PostgreSQL には NEW_TIME 関数は存在しませんので、TIMEZONE 関数を組み合わせて使用する必要があります。なお、Oracle と PostgreSQL とでは DATE 型に違いがあり、PostgreSQL では TIMESTAMP 型が引数となることに注意が必要です。

【PostgreSQL での TIMEZONE の使用例 】

```
postgres=# SELECT timezone('PST', timezone('GMT', TIMESTAMP WITHOUT TIME ZONE '2016-01-01'));
      timezone
---------------------
 2015-12-31 16:00:00
(1 行)
```

2.4.14. NUMTODSINTERVAL

Oracle の NUMTODSINTERVAL は、指定した値を INTERVAL DAY TO SECOND リテラルに変換する関数です。

【Oracle の NUMTODSINTERVAL の使用例 】

```
SQL> SELECT NUMTODSINTERVAL(10, 'SECOND') FROM DUAL;
```

13 TIMEZONE 関数では、引数に TIMESTAMP WITH TIME ZONE データ型を取ることも可能です。

```
NUMTODSINTERVAL(10,'SECOND')
-------------------------------------------------------------------------------
+000000000 00:00:10.000000000
```

一方、PostgreSQL には、NUMTODSINTERVAL 関数は存在しませんので、MAKE_INTERVAL 関数[14]を使用する必要があります。MAKE_INTERVAL 関数は INTERVAL 型の値を戻しますので、必要に応じて INTERVAL DAY TO SECOND 型に明示的にキャストします。

【PostgreSQL での MAKE_INTERVAL の使用例】

```
postgres=# SELECT MAKE_INTERVAL(SECS:=10);
 make_interval
---------------
 00:00:10
(1 行)

postgres=# SELECT CAST(MAKE_INTERVAL(SECS:=10) AS INTERVAL DAY TO SECOND);
 make_interval
---------------
 00:00:10
(1 行)
```

2.4.15. NUMTOYMINTERVAL

Oracle の NUMTODSINTERVAL は、指定した値を INTERVAL YEAR TO MONTH リテラルに変換する関数です。

【Oracle の NUMTODSINTERVAL の使用例 】

```
SQL> SELECT NUMTOYMINTERVAL(10, 'MONTH') FROM DUAL;

NUMTOYMINTERVAL(10,'MONTH')
-------------------------------------------------------------------------------
+000000000-10

SQL> SELECT NUMTOYMINTERVAL(10, 'YEAR') FROM DUAL;

NUMTOYMINTERVAL(10,'YEAR')
-------------------------------------------------------------------------------
+000000010-00
```

一方、PostgreSQL には、NUMTOYMINTERVAL 関数は存在しませんので、NUMTODSINTERVAL と同様に MAKE_INTERVAL 関数を使用する必要があります。MAKE_INTERVAL 関数は INTERVAL 型の値を戻しますので、必要に応じて INTERVAL YEAR TO MONTH 型に明示的にキャストします。

【PostgreSQL での MAKE_INTERVAL の使用例】

```
postgres=# SELECT MAKE_INTERVAL(MONTHS:=10);
 make_interval
---------------
 10 mons
(1 行)

postgres=# SELECT MAKE_INTERVAL(YEARS:=10);
 make_interval
---------------
 10 years
(1 行)
```

2.4.16. SESSIONTTIMEZONE

Oracle の SESSIONTIMEZONE は、現在のセッションのタイムゾーンを取得する関数です。

【Oracle の SESSIONTIMEZONE の使用例 】

```
SQL> SELECT SESSIONTIMEZONE FROM DUAL;
```

14 PostgreSQL 9.4 から導入された関数です。

```
SESSIONTIMEZONE
--------------------------------------------------------------------------------
+09:00
```

一方、PostgreSQL には、セッションのタイムゾーンを取得する SESSIONTTIMEZONE 関数は存在しませんので、CURRENT_SETTING 関数を使用して、現在の'TimeZone'の値を取得する必要があります。

【PostgreSQL での CURRENT_SETTING の使用例】

```
postgres=# SELECT current_setting('TimeZone');
 current_setting
-----------------
 Asia/Tokyo
(1 行)
```

また、orafce には同様の値を取得できる ORACLE.SESSIONTIMEZONE 関数[15]が実装されていますので、orafce を導入した PostgreSQL では ORACLE.SESSIONTIMEZONE に書き換えることで移行することができます。

【orafce 導入済の PostgreSQL での ORACLE.SESSIONTIMEZONE の使用例】

```
postgres=# SELECT oracle.sessiontimezone();
 sessiontimezone
-----------------
 Asia/Tokyo
(1 行)
```

2.4.17. SYS_EXTRACT_UTC

Oracle の SYS_EXTRACT_UTC は、タイムスタンプ値を協定世界時(UTC)に変換した値を、タイムゾーン無しのタイムスタンプ値として戻す関数です。タイムゾーンの指定がない場合は、セッションのタイムゾーンが使用されます。

【Oracle の SYS_EXTRACT_UTC の使用例 】

```
SQL> SELECT SYS_EXTRACT_UTC(TIMESTAMP '2016-01-01 00:00:00 + 9:00') FROM DUAL;

SYS_EXTRACT_UTC(TIMESTAMP'2016-01-0100:00:00+9:00')
---------------------------------------------------------------------------
15-12-31 15:00:00.000000000

SQL> SELECT SYS_EXTRACT_UTC(TIMESTAMP '2016-01-01 00:00:00') FROM DUAL;

SYS_EXTRACT_UTC(TIMESTAMP'2016-01-0100:00:00')
---------------------------------------------------------------------------
15-12-31 15:00:00.000000000
```

一方、PostgreSQL には、SYS_EXTRACT_UTC 関数は存在しませんので、timestamp 関数を使用する必要があります。タイムゾーン付タイムスタンプ値を変換する場合は、引数のタイムゾーンに'UTC'を指定します。タイムゾーン無しのタイムスタンプ値の場合、timezone 関数がタイムゾーン付タイムスタンプ値を戻しますので、さらに timezone 関数を適用し、タイムゾーン無しのタイムスタンプ値に変換する必要があります。

【PostgreSQL での timestamp の使用例】

```
postgres=# SELECT timezone('UTC', TIMESTAMP WITH TIME ZONE '2016-01-01 00:00:00 + 9:00');
      timezone
---------------------
 2015-12-31 15:00:00
(1 行)

postgres=# SELECT timezone('UTC', TIMESTAMP '2016-01-01 00:00:00');
        timezone
------------------------
 2016-01-01 09:00:00+09
(1 行)
```

15 orafce 3.2.1 から導入された関数です。

```
postgres=# SELECT timezone('UTC', timezone('UTC', TIMESTAMP '2016-01-01 00:00:00'));
      timezone
---------------------
 2016-01-01 00:00:00
(1 行)
```

2.4.18. TZ_OFFSET

Oracle の TZ_OFFSET は、指定したタイムゾーンのオフセット値を取得する関数です。

【Oracle の TZ_OFFSET の使用例 】

```
SQL> SELECT TZ_OFFSET('ASIA/TOKYO') FROM DUAL;

TZ_OFFSET('ASI
--------------
+09:00
```

一方、PostgreSQL には、タイムゾーンのオフセット値を取得する TZ_OFFSET 関数は存在しませんので、システムカタログ pg_timezone_names や pg_timezone_abbrevs から、タイムゾーンのオフセット値を取得する必要があります。TZ_OFFSET 関数を呼び出す SQL を書き換えずに移行するためには、同様の値を取得するファンクションを作成する方法があります。

【PostgreSQL での pg_timezone_names と pg_timezone_abbrevs の取得例】

```
postgres=# SELECT utc_offset FROM pg_timezone_names WHERE name = 'Asia/Tokyo';
 utc_offset
------------
 09:00:00
(1 行)

postgres=# SELECT utc_offset FROM pg_timezone_abbrevs WHERE abbrev = 'JST';
 utc_offset
------------
 09:00:00
(1 行)
```

2.5. 変換ファンクション

2.5.1. CONVERT

Oracle の CONVERT は、引数として char,dest_char_set,source_char_set を取り、変換対象文字列 char を変換前のキャラクタセット source_char_set から、変換後のキャラクタセット dest_char_set に変換した文字列を戻す関数です。

【Oracle の CONVERT の使用例 】

```
※Latin-1(WE8ISO8859P1)文字列を ASCII(US7ASCII)に変換する
SQL> SELECT CONVERT('Ａ Ｅ Ｉ Ｏ Ｏ Ａ Ｂ Ｃ Ｄ Ｅ ', 'US7ASCII', 'WE8ISO8859P1') FROM DUAL;
```

一方、PostgreSQL にも CONVERT 関数が存在しますが、変換前の符号化方式と変換後の符号化方式を引数で与える順序が逆になる点に注意が必要です。

【PostgreSQL の CONVERT の使用例】

```
※UTF-8 で登録されているデータ EUC-JP に変換して表示する
postgres=# SELECT convert(kanjiitem,'UTF-8','EUC-JP') FROM abctbl;
```

また、Oracle で指定できるキャラクタセットと PostgreSQL で指定できる符号化方式は名称が異なるため、引数の書き換えが必要となります。詳細は『Oracle Database グローバリゼーション・サポート・ガイド[16]』および『PostgreSQL 9.2.0 文書[17]』を参照してください。

16 http://docs.oracle.com/cd/E16338_01/server.112/b56307/applocaledata.htm#i635016

17 http://www.postgresql.jp/document/9.2/html/functions-string.html#CONVERSION-NAMES

2.5.2. UNISTR

Oracle の UNISTR は、引数で指定した UNICODE エンコード値を各国語文字列に変換して返却する関数です。

【Oracle の UNISTR の使用例】

```
SQL> SELECT UNISTR('¥0192') FROM dual;
```

PostgreSQL では、組み込み関数ではありませんが、引用符付き識別子の変異体の記述方法(U&"¥xxxx")を利用することで同様に各国語文字列を返却することが可能です。

【PostgreSQL での引用符付き識別子の変異体の記述方法の使用例】

```
postgres=# SELECT u&'¥0192';
```

引用符付き識別子の変異体の記述方法の詳細は、『PostgtreSQL 9.5.0 文書[18]』を参照してください。

2.5.3. HEXTORAW, RAWTOHEX

Oracle の HEXTORAW は、引数で指定した文字型の HEX 値に対応する RAW 値を返却する関数です。逆に、RAWTOHEX は、引数で指定した RAW 値を HEX 値を文字型として返却する関数です。

PostgreSQL には RAW 型が存在しません。
「スキーマ移行調査編 別紙1」に沿って RAW 型を bytea 型への対応付けが可能な場合はそれぞれ decode、encode 関数で代用することが可能です。
また、「3.組み込み関数の移行方法(問い合わせ言語関数)」を使用して、HEXTORAW、RAWTOHEX という名前の関数を定義して利用することも可能です、詳細は「3.1.2HEXTORAW, RAWTOHEX」を参照してください。

【PostgreSQL での decode 関数での代用例】

```
※bytea 型の列を持つテーブルを定義
postgres=# create table rawhex_test(id int, data bytea);
CREATE TABLE
postgres=# insert into rawhex_test VALUES (2, decode('63636363', 'hex'));
INSERT 0 1
※INSERT 文で挿入されたデータを確認
postgres=# select encode(data, 'escape') from rawhex_test where id=2;
 encode
----------
 cccc
(1 row)
```

【PostgreSQL での encode 関数での代用例】

```
※bytea 型の列を持つテーブルを定義
postgres=# create table rawhex_test(id int, data bytea);
CREATE TABLE
postgres=# insert into rawhex_test VALUES (1,'aaaa'::bytea);
INSERT 0 1
※INSERT 文挿入したデータを hex 値として取り出し
postgres=# select encode(data, 'hex') from rawhex_test where id=1;
 encode
----------
 61616161
(1 row)
```

2.6. エンコーディング・ファンクションおよびデコーディング・ファンクション

2.6.1. DECODE

Oracle の DECODE は、引数として expr,search,result を取り、expr が search と等しい場合に対応する result を返す関数です。

【Oracle の DECODE の使用例 】

18 https://www.postgresql.jp/document/9.5/html/sql-syntax-lexical.html

```
※ warehouse_idが1の場合「Southlake」、2の場合「San Francisco」、3の場合「New Jersey」、
   4の場合「Seattle」、1、2、3、4のいずれでもない場合、「Non domestic」を戻す。
SQL> SELECT product_id,DECODE (warehouse_id, 1, 'Southlake',
                               2, 'San Francisco',
                               3, 'New Jersey',
                               4, 'Seattle',
                                  'Non domestic') "Location"
  FROM inventories  WHERE product_id < 1775  ORDER BY product_id;
```

PostgreSQL には DECODE 関数が存在しませんが、CASE 式に置き換えることで同等の機能を実現できます。

【Oracle の DECODE 使用例を CASE 式で置換えた例】

```
postgres=# SELECT product_id,
  CASE warehouse_id
    WHEN 1 THEN 'Southlake'
    WHEN 2 THEN 'San Francisco'
    WHEN 3 THEN 'New Jersey'
    WHEN 4 THEN 'Seattle'
    ELSE 'Non domestic'
  FROM inventories  WHERE product_id < 1775  ORDER BY product_id;
```

なお、orafce には DECODE 関数が実装されていますので、orafce を導入した PostgreSQL では SQL を書き換えずに移行することができます。

2.6.2. DUMP

Oracle の DUMP は、引数として expr を取り、expr のデータ型コード、長さ(バイト単位)および内部表現を含む文字列を返す関数です。

【Oracle の DUMP の使用例 】

```
※ 第二引数に数値を指定した場合、内部表記を8進数/10進数/16進数等で表示可能
SQL> SELECT DUMP('abc', 16) FROM DUAL;

DUMP('ABC',16)
-----------------------------------------
Typ=96 Len=3: 61,62,63
```

一方、PostgreSQL には DUMP 関数は存在しません。DUMP 関数を呼び出す SQL を書き換えずに移行するためには、同様の計算を行うファンクションを作成する方法があります。

なお、orafce には DUMP 関数が実装されていますので、orafce を導入した PostgreSQL では SQL を書き換えずに移行することができます。ただし、PostgreSQL と Oracle ではデータ型の定義が異なるため、返ってくる値については同一にはなりません。

【orafce 導入済の PostgreSQL での DUMP の使用例】

```
postgres=# SELECT DUMP('abc', 16) FROM DUAL;
              dump
---------------------------------
 Typ=25 Len=7: 1c,0,0,0,61,62,63
```

2.7. NULL 関連ファンクション

2.7.1. NVL

Oracle の NVL は、引数として expr1,expr2 を取り、expr1 が NULL の場合 expr2 を、NULL でない場合 expr1 を返す関数です。

【Oracle の NVL の使用例 】

```
SQL> SELECT last_name, NVL(TO_CHAR(commission_pct), 'Not Applicable') commission FROM employees;
```

PostgreSQL には NVL 関数が存在しませんが、COALESCE 関数に置き換えることで同等の機能を実現できます。

【Oracle の NVL 使用例を COALESCE 関数で置換えた例】

```
postgres=# SELECT last_name, COALESCE(TO_CHAR(commission_pct), 'Not Applicable') commission
  FROM employees;
```

なお、orafce には NVL 関数が実装されていますので、orafce を導入した PostgreSQL では SQL を書き換えずに移行することができます。

3. 組み込み関数の移行方法(問い合わせ言語関数)

本章では既存の機能を使用するだけでは対応は難しいものの、該当する処理を問い合わせ言語関数として定義することで代替可能とする、組み込み関数への対応方法を紹介します。
問い合わせ言語関数では、条件分岐やループを含むロジックを作り込むことが難しいため、引数の型や取り得る値の範囲のチェックを必要とする場合は、PL/pqSQL などの手続き型言語関数や C 言語関数によるチェックロジックの作り込みを検討してください。

3.1. 変換ファンクション

3.1.1. BIN_TO_NUM

Oracle の BIN_TO_NUM は、1 または 0 で構成されるビット列を可変長引数として取り、10 進数の数値に変換し NUMBER 型として返却します。

【Oracle の BIN_TO_NUM の使用例】

```
SQL> SELECT BIN_TO_NUM(1,1,1,1,1,1,1,0) FROM DUAL
```

PostgreSQL では、代替の関数を作成して対処することが可能です。以下に、integer 型を返却する BIN_TO_NUM のファンクション定義例を示します。

【PostgreSQL のファンクションによる BIN_TO_NUM の使用例】

```
※BIN_TO_NUM 関数の作成例(引数は可変長引数の int、戻り値は int 型とする)
CREATE OR REPLACE FUNCTION BIN_TO_NUM(VARIADIC arr int[])
RETURNS integer AS $$
SELECT lpad(array_to_string($1, ''), 32, '0')::bit(32)::int;
$$ LANGUAGE SQL IMMUTABLE;

※使用例
functest=# select bin_to_num(1,1,1,1,1,1,1,0);
 bin_to_num
------------
        254
(1 row)
```

引数のチェックが必要な場合には、PL/pqSQL での記述を検討してください。

3.1.2. HEXTORAW, RAWTOHEX

「2.5.3HEXTORAW, RAWTOHEX」にも記載の通り、PostgreSQL には RAW 型が存在しないため、対応する関数がありません。
「スキーマ移行調査編 別紙1」に沿って RAW 型を bytea 型への対応付けが可能な場合はそれぞれ decode、encode 関数で代用することが可能です。[19]
ここでは、decode 関数、encode 関数を利用して HEXTORAW 関数、RAWTOHEX 関数として定義する例を記載します。これにより、既存アプリケーションの修正箇所を減らすことができます。

【PostgreSQL のファンクションによる HEXTORAW,の定義例および使用例】

```
※decode 関数を利用して HEXTORAW 関数を作成する例(引数は text 型、戻り値は bytea 型とする)
CREATE OR REPLACE FUNCTION HEXTORAW(text)
RETURNS bytea AS $$
SELECT decode($1, 'hex');
$$ LANGUAGE SQL IMMUTABLE;

※使用例(「2.5.3 HEXTORAW, RAWTOHEX」に記載の例と同様)
postgres=# insert into rawhex_test VALUES (2, hextoraw('63636363', 'hex'));
INSERT 0 1
※INSERT 文で挿入されたデータを確認
postgres=# select encode(data, 'escape') from rawhex_test where id=2;
 encode
```

19 encode 関数、decode 関数の方が機能範囲が広いため、第 2 引数を'hex'で固定して実装します。

```
----------
 cccc
(1 row)
```

【PostgreSQL のファンクションによる RAWTOHEX の定義例および使用例】

```
※encode 関数を利用して RAWTOHEX 関数を作成する例(引数は bytea 型、戻り値は text 型とする)
CREATE OR REPLACE FUNCTION RAWTOHEX(bytea)
RETURNS text AS $$
SELECT encode($1, 'hex');
$$ LANGUAGE SQL IMMUTABLE;

※使用例(「2.5.3 HEXTORAW, RAWTOHEX」に記載の例と同様)
postgres=# insert into rawhex_test VALUES (1,'aaaa'::bytea);
INSERT 0 1
※INSERT 文挿入したデータを hex 値として取り出し
postgres=# select rawtohex(data, 'hex') from rawhex_test where id=1;
 rawtohex
----------
 61616161
(1 row)
```

4. 組み込み関数の移行方法(C 言語関数)

本章では C 言語関数として定義することで代替可能とする、組み込み関数への対応方法を紹介します。PostgreSQL における C 言語関数の実装方法の詳細については、『PostgtreSQL 9.5.0 文書[20]』を参照してください。

4.1. 数値を戻す文字ファンクション

4.1.1. REGEXP_INSTR

Oracle の REGEXP_INSTR は、検索文字列のパターンに正規表現を使用できるように、INSTR を拡張した関数です。引数 string の substring を検索する関数で、見つかった substring の位置を返します。

【Oracle の REGEXP_INSTR の使用例】

```
-- 数字が 3 個続いている部分文字列の開始位置を返却
SQL> SELECT REGEXP_INSTR('12abc345def','[0-9]{3}') FROM DUAL;

REGEXP_INSTR('12ABC345DEF','[0-9]{3}')
--------------------------------------
                                     6
```

PostgreSQL には REGEXP_INSTR 関数が存在しませんが、正規表現を使用する類似関数として regexp_matches が存在します。regexp_matches は Oracle の REGEXP_SUBSTR 関数の代替として使用可能な PostgreSQL の組み込み関数であり、引数の文字列から正規表現にマッチする部分文字列を返却します。REGEXP_SUBSTR と REGEXP_INSTR は返却する値は検索した文字列自体と出現位置で異なっていますが、正規表現を使用して文字列を検索するという点においては一致しています。そのため、regexp_matches の機能を流用することで、比較的簡単に regexp_instr が実装可能です。以降、その実装方法の例を記載しています。

まず、regexp_matches がどのように実装されているか確認してみます。PostgreSQL のソース上で、実際にどの関数名で実装されているかを確認するには、pg_proc システムカタログの prosrc 列を参照します。

```
postgres=# select proname, prosrc from pg_proc where proname = 'regexp_matches';
    proname     |         prosrc
----------------+------------------------
 regexp_matches | regexp_matches_no_flags
 regexp_matches | regexp_matches
(2 rows)
```

上記の結果から、regexp_matches もしくは regexp_matches_no_flags という名前で定義されていることが確認できます。実際にソースを検索すると、これらの関数は regexp.c[21]というファイルに記述されています。
regexp_matches_no_flags 関数はオプションフラグ用の引数を省略するためのラッパーであり、処理の実体は regexp_matches 関数に記述されています。

regexp_matches 関数の中身を見ると、正規表現の処理自体は setup_regexp_matches 関数で実行しています。用途によって引数を変更するため、引数の意味を理解しておく必要はありますが、関数自体はそのまま流用が可能です。

```
regexp_matches_ctx *matchctx;
matchctx = setup_regexp_matches(PG_GETARG_TEXT_P_COPY(0), pattern,
                    flags,
                    PG_GET_COLLATION(),
                    false, true, false);
```

関数の戻り値の regexp_matches_ctx 構造体には、パターンにマッチした文字列の開始位置、終了位置や、パターンにマッチした回数などの情報が格納されています。この戻り値を build_regexp_matches_result 関数に渡し、その開始位置と終了位置の情報を使用して部分文字列を切り出した結果を返却をしています。

20 https://www.postgresql.jp/document/9.5/html/xfunc-c.html

21 src/backend/utils/adt/regexp.c

```
static ArrayType *
build_regexp_matches_result(regexp_matches_ctx *matchctx)
{
(略)
     for (i = 0; i < matchctx->npatterns; i++)
     {
          int            so = matchctx->match_locs[loc++];  // 開始位置
          int            eo = matchctx->match_locs[loc++];  // 終了位置

          if (so < 0 || eo < 0)
          {
               elems[i] = (Datum) 0;
               nulls[i] = true;
          }
          else
          {
               elems[i] = DirectFunctionCall3(text_substr,
                                PointerGetDatum(matchctx->orig_str),
                                  Int32GetDatum(so + 1),
                                  Int32GetDatum(eo - so));  // 部分文字列の取得
               nulls[i] = false;
          }
     }

 (略)
}
```

matchctx->match_locs 配列には開始位置、終了位置が順に(複数マッチしている場合は繰り返し)格納されています。regexp_instr は文字列の開始位置を返却すればいいため、単純に開始位置の情報をを元に、下記の値を結果として返却するようにします。

```
matchctx->match_locs[0] + 1;  // 開始位置、0オリジンのため+1
```

流用箇所から呼び出している一部の関数について、改造は不要であるものの PostgreSQL のソースから直接 include して使用することができず、一部コードについてはコピーする必要がありましたが、既存コードの流用により大きなコード追加・修正を行うことなく regexp_instr を作成することができました。

【PostgreSQL の C 言語関数で実装した regexp_instr の例】

```
postgres=# create or replace function public.regexp_instr(text, text)
postgres-#  returns setof integer
postgres-# as '$libdir/regexp.so', 'regexp_instr_no_flags'
postgres-# language c immutable strict;
CREATE FUNCTION
-- 数字が3個続いている部分文字列の開始位置を返却
postgres=# select regexp_instr('12abc345def','[0-9]{3}');
 regexp_instr
--------------
            6
(1 row)
```

4.1.2. REGEXP_COUNT

Oracle の REGEXP_COUNT は、検索文字列の正規表現パターンの出現回数を取得する関数です。

【Oracle の REGEXP_COUNT の使用例】

```
-- 数字が2個続いている部分文字列の出現回数
SQL> SELECT REGEXP_COUNT('12abc345def','[0-9]{2}') FROM DUAL;

REGEXP_COUNT('12ABC345DEF','[0-9]{2}')
---------------------------------------
                                     2
```

PostgreSQL には REGEXP_COUNT 関数が存在しませんが、REGEXP_INSTR 関数と同じ要領で、C 言語関数による実装をすることが可能です。

setup_regexp_matches の実行により、戻り値には指定したパターンにマッチした回数が格納されているため、その情報を結果として返却するようにします。

```
matchctx->nmatches;  // パターンにマッチした回数
```

【PostgreSQL の C 言語関数で実装した regexp_count の例】

```
-- 数字が 2 個続いている部分文字列の出現回数
postgres=# select regexp_count('12abc345def','[0-9]{2}');
 regexp_count
--------------
            2
(1 row)
```

4.2. エンコーディング・ファンクションおよびデコーディング・ファンクション

4.2.1. VSIZE

Oracle の VSIZE は、expr の内部表現でのバイト数を返却します。

【Oracle の VSIZE の使用例】

```
SQL> SELECT VSIZE('abc') FROM DUAL;

VSIZE('ABC')
------------
           3
```

PostgreSQL では、代替の関数を作成して対処することが可能です。

関数内で行う処理としては、まず引数の型の OID を求め、OID から pg_type カタログに保持されているデータ型情報を参照し、バイト数(typlen カラム)を求めます。また、typlen が負の値をとる可変長データにも対応するため、datumGetSize 関数(datum.c)を使用し実サイズを求めます。

以下コードの抜粋になります。

```
PG_FUNCTION_INFO_V1(vsize);

Datum
vsize(PG_FUNCTION_ARGS)
{
    Oid             valtype = get_fn_expr_argtype(fcinfo->flinfo, 0); /* 要素の型の OID を取得 */
    List    *args;
    int16   typlen;
    bool    typbyval;
    Size    length;
    Datum   value;
(省略)
    value = PG_GETARG_DATUM(0); /* 要素を取得 */
    get_typlenbyval(valtype, &typlen, &typbyval); /* OID から typlen(固定長型であればバイト数)を取得 */
    length = datumGetSize(value, typbyval, typlen); /* 要素の実サイズを求める関数 */
(省略)
    PG_RETURN_INT32(length); /* 戻り値は 4byte 整数型 */
}
```

上記処理は orafce の DUMP 関数の中で Len の値を求める際にも行われているため、orafce の"others.c"に記載されている DUMP 関数の箇所を参照することにより容易に実装可能です。

【PostgreSQL の C 言語関数で実装した VSIZE の使用例】

```
postgres=# SELECT VSIZE('abc');
 vsize
-------
   7
(1 row)
```

なお orafce を導入している場合は、C 言語関数を定義せずとも DUMP 関数の戻り値の当該箇所のみ抽出するファンクションを作成することでも実現可能と思われます。

5. 移行が難しいと判断される関数

本章では移行方法について調査したものの、実現するのは難しいと判断した組み込み関数について紹介します。

5.1. 対応するデータ型が存在しない場合

5.1.1. CHARTOROWID, ROWIDTOCHAR, ROWIDTONCHAR

これらの Oracle 関数は、CHAR 型、NCHAR 型と ROWID 型とを相互に変換する関数です。
PostgreSQL では、変換先、変換元の ROWID 型とデータ型のマッピングができず、移行が難しくなります。

5.2. 構文を変更する必要がある場合

5.2.1. FIRST、LAST

単純に関数単体で実行するような関数ではなく、WINDOW 関数のように他の構文と合わせて使用する関数である場合、関数の実装だけではなく構文レベルでの修正が必要です。
FIRST や LAST のような、WINDOW 関数の構文に組み込まれているような関数では、PostgreSQL 本体の構文解析等にも修正が必要になるため、移行が難しくなります。

5.3. 文字の変換が入る場合

5.3.1. COMPOSE, DECOMPOSE

Oracle での COMPOSE, DECOMPOSE の使用例を以下に示します。

【Oracle の COMPOSE, DECOMPOSE の使用例】

```
※「o」とウムラウトを結合する
SQL> SELECT COMPOSE( 'o' || UNISTR('¥0308') )
 FROM DUAL;

CO
--
ö

※「Châteaux」の「â」を「a」と「^」に分割する
SQL> SELECT DECOMPOSE ('Châteaux')
 FROM DUAL;

DECOMPOSE
---------
Cha^teaux
```

UNICODE 文字列に対する分解および結合の対応関係を実装する必要があるため、移行が難しくなります。

5.4. 各国語を取り扱う場合

5.4.1. NCHR

指定された引数と同等のバイナリを持つ各国語キャラクタセット[22]での文字を返す関数になります。
現状 PostgreSQL は各国語キャラクタセットに対応していないため、独自に実装を追加する必要があり、移行が難しくなります。

5.5. 内部処理が不明である場合

5.5.1. ORA_HASH

32bit 長のハッシュ値を生成し NUMBER 型で返す関数です。
同様の形式（符号なし 32bit 数値型）のハッシュ値を返す関数であれば作成可能ですが、アルゴリズムが不明であるため同様のハッシュ値を生成することが難しいです。

22 各国語キャラクタセットとは Oracle の各国語文字列を格納する際に使用されるキャラクタセットです。
http://docs.oracle.com/cd/E16338_01/server.112/b56307/ch2charset.htm#i1007017

6. 別紙一覧

- 別紙: 組み込み関数対応表(Oracle-PostgreSQL)

著者

版	所属企業・団体名	部署名	氏名
組み込み関数移行調査編 1.0 （2012 年度 WG2）	TIS 株式会社	戦略技術センター	中西 剛紀
組み込み関数移行調査編 2.0 （2015 年度 WG2）	NEC ソリューションイノベータ株式会社	第四 PF ソフトウェア事業部	黒澤 彰
	富士通株式会社	ミドルウェア事業本部	山本 明範 山本 貢嗣 榎本 友理枝
	株式会社富士通ソーシアルサイエンスラボラトリ	公共ビジネス本部第三システム部	高橋 勝平
	三菱電機株式会社	情報技術総合研究所	田中 覚