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

スキーマ移行調査編

製作者
所属企業：TIS 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

はじめに

■本資料の目的

本資料は、異種 DBMS から PostgreSQL へスキーマ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 へスキーマを移行する際に DDL 仕様の相違等により書き換えが必要となる箇所についてスキーマの種別毎に記載します。

また、移行ツール Ora2Pg の利用を前提とした場合の具体的な変換方法についても記載します。

■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、ツール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	9.2.0
Oracle Database	11gR2 11.2.0.2.0

表 3: 本書で扱うツール

ツール名称	バージョン	ライセンス	入手元	概要
Ora2Pg	9.0	GNU General Public License v3	http://ora2pg.darold.net/	Oracle のスキーマ、データを PostgreSQL に移行するツール

目次

1. 組み込みデータ型

1.1. 組み込みデータ型の違い

Oracle と PostgreSQL では対応している組み込みデータ型に違いがあり、移行元のデータベースで使われているデータ型が移行先のデータベースに存在しないことがあります。移行元のデータ型が移行先に存在しない場合は、移行先のデータベースにおいて対応するデータ型に修正する必要があります。

Oracle と PostgreSQL における組み込みデータ型の対応関係は、別紙「組み込みデータ型対応表（Oracle-PostgreSQL）」を参照してください。

1.2. 変換方針

1.1 に記載した相違点に基づき、以下のような方針で組み込みデータ型を変換します。

1.2.1. 文字型

Oracle のデータ型に対応する PostgreSQL のデータ型に置き換えます。
ただし、可変長文字列はサイズ指定がバイト数/文字数のどちらで指定されているか注意が必要です。

1.2.2. 数値型

Oracle の NUMBER 型に完全に一致するデータ型は PostgreSQL に存在しません。
そのため、精度とスケールに応じて、PostgreSQL の数値データ型から置換える対象を選択します。
なお、Oracle の FLOAT 型は PostgreSQL にも存在するため変換は不要ですが、保有可能な精度が異なる（Oracle 2進数 126 桁、PostgreSQL 2 進数 53 桁）点に注意する必要があります。

1.2.3. 日付/時刻データ型

Oracle のデータ型に対応する PostgreSQL のデータ型に置き換えます。
ただし、DATE 型は Oracle と PostgreSQL で保有する制度が異なるため、注意が必要です。

1.2.4. バイナリ型

PostgreSQL では原則として bytea 型に置き換えます。
ただし、bytea 型で保有できる上限の 1GB を超える場合は、ラージオブジェクトを使用するか、外部ファイルへ別途保存する必要があります。

1.2.5. その他

対応するデータ型が PostgreSQL に存在しないため、機械的な変換は困難です。

2. テーブル

2.1. DDL の違い

Oracle と PostgreSQL における CREATE TABLE 文の違いを比較します。
【Oracle の CREATE TABLE 文】[1]

```
CREATE [ GLOBAL TEMPORARY ] TABLE 表名
 [ (
  { 列名 データ型 [ SORT ][ ENCRYPT 暗号化方式 ]
```

1 Oracle Database SQL 言語リファレンス 11g リリース 2（http://docs.oracle.com/cd/E16338_01/server.112/b56299/statements_7002.htm#i2095331）より引用

```
    [ ( { 列制約 }... ) | 外部参照制約 ]
  | 仮想列名 [データ型] [GENERATED ALWAYS] AS (列値導出式) [VIRTUAL]
    [ 列制約 ... ]
  | 表制約
  } [, ... ]
 ) ]
 [ ON COMMIT { DELETE | PRESERVE } ROWS ]
 [TABLESPACE 表領域名]
 [PCTFREE 空き領域割合]
 [PCTUSED 使用領域割合]
 [STRAGE
  (
   [INITIAL　初期エクステントサイズ]
   [NEXT　増分エクステントサイズ]
   [MINEXTENTS　最小エクステント数]
   [MAXEXTENTS　最大エクステント数|UNLIMITED]
   [PCTINCREASE　エクステントサイズ拡大率]
   [BUFFER_POOL　{DEFAULT | KEEP | RECYCLE}]
   [FREELISTS　空きリスト数]
  )
 ]
 [ {LOGGING | NOLOGGING |  FILESYSTEM_LIKE_LOGGING} ]
 [ {COMPRESS | NOCOMPRESS} ]
 [ 列の記憶域属性 ]
 [ PARTITION BY パーティション定義 ]
 [ CACHE | NOCACHE ]
 [ RESULT_CACHE ( MODE {DEFAULT | FORCE } ) ]
 [ { NOPARALLEL | PARALLEL 並列度 } ]
 [ ROWDEPENDENCIES | NOROWDEPENDENCIES ]
 [ 制約の使用可否 ]...
 [ { ENABLE | DISABLE } ROW MOVEMENT ]
 [ FLASHBACK ARCHIVE [flashback_archive] | NO FLASHBACK ARCHIVE ]
 [ AS 副問い合わせ ]
```

【PostgreSQLの CREATE TABLE 文】[2]

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] 表名
 [ (
    { 列名 データ型 [ COLLATE 照合順 ] [ 列制約 [ ... ] ]
     | 表制約
     | LIKE コピー元表名 [ like_option ... ] }
     [, ... ]
 ) ]
[ INHERITS ( 親テーブル [, ... ] ) ]
[ WITH
   (
    [fillfactor = 使用領域割合]
        [autovacuum_enabled = 自動 VACUUM 有効/無効]
    [, ... ]
   )
 | WITH OIDS | WITHOUT OIDS
]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE テーブル空間名 ]
[ AS 副問い合わせ ]
```

2　PostgreSQL 9.2.0 付属ドキュメント（http://www.postgresql.jp/document/9.2/html/sql-createtable.html）より引用

2.2. 変換方針

2.1 に記載した CREATE TABLE 文の相違点から、以下のような方針でDDLを修正します。

2.2.1. 列定義

Oracle の仮想列は PostgreSQL に存在しないため、別途ビューの作成を検討してください。
組み込みデータ型自体の変換方法は、1 を参照してください。
列制約、表制約の違い、変換方法については、8 を参照してください。

2.2.2. PCTFREE、PCTUSED

PCTUSED で指定する使用領域割合は、PostgreSQL では WITH 句の fillfactor パラメータで指定します。
PCTFREE で指定する空き領域割合を指定するパラメータは、PostgreSQL に存在しないため削除します。

2.2.3. STORAGE, LOGGING, COMPRESS, CACHE, RESULT_CACHE, PARALLEL, ROWDEPENDENCIES, ENABLE ROW MOVEMENT, FLASHBACK ARCHIVE

PostgreSQL には存在しないパラメータのため、削除します。

2.2.4. PARTITION BY

PostgreSQL では「テーブルの継承」「CHECK 制約」「トリガ」の仕組みを使ってパーティション表を実現します。
パーティションテーブルの違い、変換方法については、3 を参照してください。

2.3. DDL の移行手順

移行ツールの Ora2Pg を利用すると Oracle のテーブルから PostgreSQL の CREATE TABLE 文を生成することができます。ただし、一部のデータ型は Ora2Pg での変換がうまく行われず、手作業で修正する必要があります。

2.3.1. Ora2Pg で対応していないデータ型

以下のデータ型は Ora2Pg で変換が行えないため、手作業で移行します。

表 2.1: Ora2Pg での自動変換に対応していないデータ型

	Oracle		PostgreSQL
	データ型	単位	データ型
日時	INTERVAL YEAR TO MONTH	年、月	interval [fields] [(p)]
	INTERVAL DAY TO SECOND	日～秒	interval [fields] [(p)]
その他	BFILE		対応なし
	ROWID		
	UROWID		
	ユーザ定義型		
	任意型		
	空間型		
	メディア型		
	Expression Filter 型		

2.3.2. Ora2Pg で自動変換されるが、PostgreSQL では適切ではないデータ型

以下のデータ型は Ora2Pg で自動変換されますが、適切なデータ型が選択されていない可能性があるため、自動変換された DDL を手作業で修正します。

表 2.2: Ora2Pg で自動変換後、手修正すべきデータ型

	Oracle		PostgreSQL	移行時の修正方法
	データ型	単位	データ型	
文字	NCHAR	char	char(n)	エクスポート前の文字数を確認して、文字数を指定する。 例）NCHAR(10) → char(10)
真数	NUMBER		decimal	精度とスケールに応じて PostgreSQL のデータ型を指定する。
			numeric	
			integer	
			bigint	
日時	TIMESTAMP		timestamp [(p)] [without time zone]	TIMESTAMP 型のミリ秒以下のスケールを指定する。 タイムゾーンを持つ場合は、データ型にタイムゾーンを指定する。
	TIMESTAMP WITH TIMEZONE		timestamp [(p)] with time zone	
	TIMESTAMP WITH LOCAL TIMEZONE		timestamp [(p)] [without time zone]	

2.3.3. Ora2Pg による移行例

【例 1 移行元の Oracle の CREATE TABLE 文 】

```
CREATE TABLE DATA_TYPE_TEST1 (
    ID                          VARCHAR2(5)     NOT NULL,
    CHAR_VARCHAR2_BYTE          VARCHAR2(20),
    CHAR_VARCHAR2_CHAR          VARCHAR2(20 CHAR),
    CHAR_NVARCHAR2              NVARCHAR2(20),
    CHAR_CHAR_BYTE              CHAR(10),
    CHAR_CHAR_CHAR              CHAR(10 CHAR),
    CHAR_NCHAR                  NCHAR(10),
    CHAR_LONG                   LONG,
    CHAR_CLOB                   CLOB,
    CHAR_NCLOB                  NCLOB,
    NUM_NUMBER1                 NUMBER(5),
    NUM_NUMBER2                 NUMBER(10),
    NUM_NUMBER3                 NUMBER(11),
    NUM_NUMBER4                 NUMBER(10,2),
    NUM_NUMBER5                 NUMBER(15,5),
    DATE_DATE                   DATE,
    DATE_TIMESTAMP              TIMESTAMP(6),
    DATE_TIMESTAMP_TIMEZONE     TIMESTAMP(6) WITH TIME ZONE,
    DATE_TIMESTAMP_LOCAL        TIMESTAMP(6) WITH LOCAL TIME ZONE
)
```

↓Ora2Pg で変換

【例 1 Ora2Pg で変換したPostgreSQL用の CREATE TABLE 文】

```
CREATE TABLE "data_type_test1" (
 "id" varchar(5) NOT NULL,
 "char_varchar2_byte" varchar(20),
 "char_varchar2_char" varchar(20),
 "char_nvarchar2" varchar(20),
 "char_char_byte" char(10),
 "char_char_char" char(10),
 "char_nchar" char,                    → 手作業で char(10) に書き換える
 "char_long" text,
 "char_clob" text,
 "char_nclob" text,
 "num_number1" numeric(5),             → 手作業で int に書き換える
 "num_number2" numeric(10),            → 手作業で bigint に書き換える
 "num_number3" numeric(11),            → 手作業で bigint に書き換える
 "num_number4" decimal(10,2),
 "num_number5" decimal(15,5),
 "date_date" timestamp,
 "date_timestamp" timestamp,           → 手作業で timestamp(6) に書き換える
 "date_timestamp_timezone" timestamp,  → 手作業で timestamp(6) with time zone に書き換える
 "date_timestamp_local" timestamp      → 手作業で timestamp(6) に書き換える
);
```

【例 2 移行元の Oracle の CREATE TABLE 文 】

```
CREATE TABLE DATA_TYPE_TEST3 (
    ID                          VARCHAR2(5)     NOT NULL,
    DATE_INTERVAL_YEAR_MONTH    INTERVAL YEAR(2) TO MONTH,
    DATE_INTERVAL_DAY_SECOND    INTERVAL DAY(2) TO SECOND(6)
)
```

↓Ora2Pg で変換

【例 2 Ora2Pg で変換したPostgreSQL用の CREATE TABLE 文】
INTERVAL 型の列を Ora2Pg で自動変換できないため、手動で DDL を作成する必要がある。

【例 3 移行元の Oracle の CREATE TABLE 文 】

```
CREATE TABLE DATA_TYPE_TEST4 (
    ID                          VARCHAR2(5)     NOT NULL,
    BIN_RAW                     RAW(100),
    BIN_BLOB                    BLOB
)
```

↓Ora2Pg で変換

【例 3 Ora2Pg で変換したPostgreSQL用の CREATE TABLE 文】

```
CREATE TABLE "data_type_test4" (
 "id" varchar(5) NOT NULL,
 "bin_raw" bytea,
 "bin_blob" bytea
);
```

3. パーティショニングテーブル

3.1. DDLの違い、変換方針

Oracleではテーブルのパーティショニングは組み込みの機能として実装されており、CREATE TABLE文のPARTITION BY句を定義することで、パーティショニングテーブルを作成できます。

一方、PostgreSQLはパーティショニングを組み込みの機能としては持たないため、「テーブルの継承」「CHECK制約」「トリガ」の仕組みを使って、パーティショニングテーブルを実現することになります。以下にそれらの構成要素について簡単に紹介します。[3]

3.1.1. 継承

オブジェクト関係データベース管理システム (ORDBMS) であるPostgreSQLはテーブルの『継承』という機能を持っています。テーブルに親子関係を定義し、子は親の列構成を引き継ぎます（列の追加もできます）。親テーブルへのクエリは子テーブルも含むように自動的に展開されます。オブジェクト指向を取り入れたスキーマ設計のための機能ですが、最近ではもっぱらテーブル・パーティショニングのために利用されているようです。

3.1.2. CHECK制約

パーティションに含まれるデータの範囲の定義には『CHECK制約』を使用します。PostgreSQLでは分割方法について、特に「レンジ」「リスト」「ハッシュ」などの区別をせず、単純にWHERE句の条件が制約に適合するかのみで判定します。柔軟性はあるものの、判定が順番に行われるためパーティション数に比例する処理コストがかかるのが難点です。分割数は100個に留めたほうが良いでしょう。

3.1.3. トリガ

PostgreSQL 8.2以降であれば、親テーブルに対するSELECT, UPDATE, DELETEは自動的に子テーブルを展開し、絞込みを行ってくれます。ただしINSERTについては何もしないので、親テーブルに『トリガ』を定義し、パーティション・キーの値に基づいて挿入先を振り分ける必要があります。

3.2. DDLの移行手順

移行ツールのOra2Pgはパーティショニングテーブルに対応していないため、以下の様な手順を手作業で行います。

3.2.1. 親テーブルの作成

Oracleのテーブル定義を2.2の方針でPostgreSQLのCREATE TABLE文に変換し、PostgreSQLのデータベースに作成します。このテーブルはパーティションテーブルではなく、通常のテーブルとして作成されます。

3.2.2. 子テーブルの作成

3.2.1で作成したテーブルを親テーブルとして継承する子テーブルを作成します。この時、子テーブルはOracleのパーティション・テーブルにおける各パーティションに相当するため、パーティション数分の子テーブルを作成することになります。なお、継承する親テーブルは子テーブルのCREATE TABLE文のINHERITS句に指定します。

3.2.3. CHECK制約の作成

Oracleのパーティション定義に合わせて、各パーティション（子テーブル）に含まれるデータの範囲をCHECK制約で定義します。このCHECK制約により、SELECT、UPDATE、DELETE文の実行時に走査するデータ範囲の絞込みが自動的に行われるようになります。

3.2.4. INSERTトリガの定義

親テーブルへのINSERT文を実行した時、適切なパーティション（子テーブル）に自動的にSQLを振り分けたい場合は、親テーブルにINSERTトリガを定義します。

3 Lets' Postgres（http://lets.postgresql.jp/documents/technical/partitioning/2)より引用

4. 索引

4.1. DDLの違い

OracleとPostgreSQLにおけるCREATE INDEX文の違いを比較します。

【Oracle の CREATE INDEX 文】[4]

```
CREATE [ UNIQUE | BITMAP ] INDEX インデックス名
  ON { 表名 [ 相関名(別名) ]
       (列名 [ ASC | DESC ] ...) [ index_attributes ]
     | CLUSTER クラスタ名 [ index_attributes ]
     }
  [ UNUSABLE ] ;

※ index_attributes 部分には以下のパラメータを指定可能
-----------------------------------------------------
[PCTFREE 空き領域割合]
[PCTUSED 使用領域割合]
[INITRANS 同時実行トランザクションエントリ初期数]
[STRAGE
  (
    [INITIAL  初期エクステントサイズ]
    [NEXT  増分エクステントサイズ]
    [MINEXTENTS  最小エクステント数]
    [MAXEXTENTS  最大エクステント数|UNLIMITED]
    [PCTINCREASE  エクステントサイズ拡大率]
    [BUFFER_POOL  {DEFAULT | KEEP | RECYCLE}]
    [FREELISTS  空きリスト数]
  )
]
[ONLINE]
[TABLESPACE { テーブル空間名 | DEFAULT ]
[COMPRESS 接頭辞長 | NOCOMPRESS]
[SORT | NOSORT]
[REVERSE]
[VISIBLE | INVISIBLE]
[NOPARALLEL | PARALLEL 並列度 ]
-----------------------------------------------------
```

【PostgreSQLの CREATE INDEX 文】[5]

```
CREATE [UNIQUE] INDEX [CONCURRENTLY] インデックス名]
  ON 表名 [USING {btree | hash | gist | spgist | gin}]
    (列名 [ COLLATE 照合順 ] [ 演算子クラス ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] ,...)
    [ WITH
      (
        [fillfactor = 使用領域割合]
      )
    ]
    [ TABLESPACE テーブル空間名 ]
    [ WHERE 部分インデックス用の制約式 ]
```

4 Oracle Database SQL 言語リファレンス 11g リリース２（http://docs.oracle.com/cd/E16338_01/server.112/b56299/statements_5012.htm#i2062403）より引用

5 PostgreSQL 9.2.0 付属ドキュメント（http://www.postgresql.jp/document/9.2/html/sql-createindex.html）より引用

4.2. 変換方針

4.1 に記載した CREATE INDEX 文の相違点から、以下のような方針でDDLを修正します。

4.2.1. index_attributes 部分のパラメータ

PCTUSED で指定する使用領域割合のみ、PostgreSQL では WITH 句の fillfactor パラメータに指定できます。それ以外のパラメータは PostgreSQL に存在しないため、削除します。

4.2.2. 利用できる索引の種類

Oracle と PostgreSQL でそれぞれ利用できる索引の種類を以下にまとめます。

表 4.1: Oracle で利用できる索引

索引の種別	説明
B-tree 索引	デフォルトで選択される索引
ファンクション索引	Oracle 関数やユーザ定義関数を利用した索引
ビットマップ索引	カーディナリティが低いデータに適した索引
逆キー索引	索引列のデータをビット単位で反転させ、その反転させたデータをソートして作成する索引。 データ挿入時に特定の索引ブロックにアクセスが集中する場合に有効だが、検索処理では索引が有効に働かないパターンがあり注意が必要。
ドメイン索引	ユーザが定義した索引タイプ(INDEXTYPE)を使用する索引
パーティション索引	パーティションテーブルに対する索引
B-tree クラスタ索引	クラスタで使用される索引
ハッシュ・クラスタ索引	クラスタで使用される索引

表 4.2: PostgreSQL で利用できる索引

索引の種別	説明
B-tree 索引	デフォルトで選択される索引
ファンクション索引	PostgreSQL 関数やユーザ定義関数を利用した索引
R-tree 索引	特に二次元の空間的なデータに対する問い合わせに適した索引。WAL に書き込まれないため、DB クラッシュ後に索引の再構築が必要。
ハッシュ索引	単純な等価性比較のみを扱うことができる索引。WAL に書き込まれないため、DB クラッシュ後に索引の再構築が必要。
GiST 索引	多くの異なるインデックス戦略を実装。主に幾何情報に対して有効。例えば、垂直であるか並行であるかといった情報をもとに索引を作成できる。
GIN 索引	PostgreSQL の配列などの集合に対する索引

Oracle の B-tree 索引は PostgreSQL でも実装されているため、そのまま移行することができます。また、Oracle のファンクション索引は同様の索引が PostgreSQL でも実装されていますが、索引対象とするデータ型に制約があること、ファンクション自体が PostgreSQL に用意されている必要があることに注意が必要です。

上記以外の索引は PostgreSQL に対応する索引が実装されていませんので、B-tree 索引や GiST 索引で代用する等の設計変更を検討することになります。

4.3. DDLの移行手順

移行ツールのOra2Pgを利用するとOracleのB-tree索引とファンクション索引の一部をPostgreSQLのCREATE INDEX文を生成することができます。それ以外の索引は手作業でDDLを作成します。

4.3.1. Ora2Pgによる移行例

【例1 移行元のOracleのCREATE INDEX文】

```
CREATE UNIQUE INDEX INDEX_TEST1
    ON DATA_TYPE_TEST1 (ID, CHAR_VARCHAR2_BYTE)
    COMPRESS 1
```

↓Ora2Pgで変換

【例1 Ora2Pgで変換したPostgreSQL用のCREATE INDEX文】

```
CREATE UNIQUE INDEX index_test1
  ON data_type_test1
  USING btree
  (id COLLATE pg_catalog."default" , char_varchar2_byte COLLATE pg_catalog."default" );
```

【例2 移行元のOracleのCREATE INDEX文】

```
CREATE INDEX INDEX_TEST3
    ON DATA_TYPE_TEST1 ("TO_CHAR("DATE_TIMESTAMP",'YYYYMMDD')")
```

↓Ora2Pgで変換

【例2 Ora2Pgで変換したPostgreSQL用のCREATE INDEX文】

```
CREATE INDEX index_test3 ON "data_type_test1" (to_char("date_timestamp",'yyyymmdd'));
********** エラー **********

ERROR: functions in index expression must be marked IMMUTABLE
SQLステート:42P17
```

※PostgreSQL では出力形式が変更可能で値が随時変わってしまうデータ型にファンクション索引を作成できない。

【例3 移行元のOracleのCREATE INDEX文】

```
CREATE INDEX INDEX_TEST4
    ON DATA_TYPE_TEST1 ("SUBSTR("CHAR_VARCHAR2_BYTE",0,2)")
```

↓Ora2Pgで変換

【例3 Ora2Pgで変換したPostgreSQL用のCREATE INDEX文】

```
CREATE INDEX index_test4 ON "data_type_test1" (substring("char_varchar2_byte" from 0 for 2));
```

5. ビュー

5.1. DDLの違い

Oracle と PostgreSQL における CREATE VIEW 文の違いを比較します。

【Oracle の CREATE VIEW 文】[6]

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW ビュー名
  AS SELECT 文 [WITH READ ONLY] [WITH CHECK OPTION]
```

【PostgreSQLの CREATE VIEW 文】[7]

```
CREATE [OR REPLACE] [TEMP | TEMPORARY] VIEW ビュー名 [ ( 列名 [, ...] ) ]
    [ WITH ( security_barrier ) ]
    AS SELECT 文
```

5.2. 変換方針

Oracle から PostgreSQL へビューを移行する際の注意点を記載します。

5.2.1. 更新可能なビューの移行方針

Oracle ではビューに対する更新が可能ですが、PostgreSQL ではビューに対して更新ができません。[8]ただし、PostgreSQL では対象のビューに対する RULE もしくはトリガーを組合せることで、Oracle の更新可能なビューと同等の機能を実現することは可能です。

したがって、Oracle で [WITH READ ONLY] オプションが付与されていないビューを PostgreSQL に移行する場合、更新を行うか、読み取り専用であるかによって以下の様に対応方針を変える必要があります。

表 5.1: Oracle から PostgreSQL へのビュー移行方針

Oracle		PostgreSQL
オプション	読取専用	
指定なし	○	CREATE VIEW
	×	RULE もしくは、トリガーを使用
WITH READ ONLY	○	CREATE VIEW
WITH CHECK OPTION	×	RULE もしくは、トリガーを使用

PostgreSQL の RULE を使って、更新可能なビューを実現する方法を以下に示します。

【VIEW】

```
CREATE VIEW person_detail_job_vw AS
SELECT p.pid, p.pname, j.job FROM person_detail p LEFT JOIN person_job j
ON (j.pid = p.pid);
```

【INSERT RULE】

```
CREATE OR REPLACE RULE person_detail_job_vw_INSERT AS ON INSERT
```

6 Oracle Database SQL 言語リファレンス 11g リリース 2 (http://docs.oracle.com/cd/E16338_01/server.112/b56299/statements_8004.htm#i2065510) より引用

7 PostgreSQL 9.2.0 付属ドキュメント(http://www.postgresql.jp/document/9.2/html/sql-createview.html)より引用

8 PostgreSQL 9.3 では更新可能ビューがサポートされる予定です。 http://lets.postgresql.jp/documents/technical/9.3/updatable_view を参照。

```
TO person_detail_job_vw DO INSTEAD
(
 INSERT INTO person_detail VALUES(NEW.pid, NEW.pname);
 INSERT INTO person_job VALUES (NEW.pid, NEW.job)
);
```

【UPDATE RULE】

```
CREATE OR REPLACE RULE person_detail_job_vw_UPDATE AS ON UPDATE
TO person_detail_job_vw DO INSTEAD
(
  UPDATE person_detail SET pid=NEW.pid, pname=NEW.pname WHERE pid=OLD.pid;
  UPDATE person_job SET pid=NEW.pid, job=NEW.job WHERE pid=OLD.pid
);
```

【DELETE RULE】

```
CREATE OR REPLACE RULE person_detail_job_vw_DELETE AS ON DELETE
TO person_detail_job_vw DO INSTEAD
(
  DELETE FROM person_job WHERE pid=OLD.pid;
  DELETE FROM person_detail WHERE pid=OLD.pid
);
```

また、PostgreSQL のトリガーを使って、更新可能なビューを実現する方法を以下に示します。

【VIEW】

```
CREATE VIEW person_detail_job_vw AS
SELECT p.pid, p.pname, j.job FROM person_detail p LEFT JOIN person_job j
ON (j.pid = p.pid);
```

【トリガーから呼び出されるファンクション】

```
CREATE OR REPLACE FUNCTION person_detail_job_vw_dml()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $function$
   BEGIN
      IF TG_OP = 'INSERT' THEN
        INSERT INTO  person_detail VALUES(NEW.pid,NEW.pname);
        INSERT INTO  person_job VALUES(NEW.pid,NEW.job);
        RETURN NEW;
      ELSIF TG_OP = 'UPDATE' THEN
       UPDATE person_detail SET pid=NEW.pid, pname=NEW.pname WHERE pid=OLD.pid;
       UPDATE person_job SET pid=NEW.pid, job=NEW.job WHERE pid=OLD.pid;
       RETURN NEW;
      ELSIF TG_OP = 'DELETE' THEN
       DELETE FROM person_job WHERE pid=OLD.pid;
       DELETE FROM person_detail WHERE pid=OLD.pid;
       RETURN NULL;
      END IF;
      RETURN NEW;
    END;
$function$;
```

【トリガー】

```
CREATE TRIGGER person_detail_job_vw_dml_trig
    INSTEAD OF INSERT OR UPDATE OR DELETE ON
      person_detail_job_vw FOR EACH ROW EXECUTE PROCEDURE person_detail_job_vw_dml();
```

5.3. DDLの移行手順

移行ツールの Ora2Pg を利用すると Oracle のビューから CREATE VIEW 文を生成することができます。ただし、with read only, with check option 句は手作業で削除する必要があります。

また、更新可能なビューに対する RULE、トリガーは Ora2Pg では作成されないため、5.2 の変換方針に沿って手作業で作成する必要があります。

5.3.1. Ora2Pg による移行例

【例 1 移行元の Oracle の CREATE VIEW 文 】

```
CREATE OR REPLACE VIEW VIEW_TEST3
AS
select
"ID","CHAR_VARCHAR2_BYTE","CHAR_VARCHAR2_CHAR","CHAR_NVARCHAR2","CHAR_CHAR_BYTE","CHAR_CHAR_CHAR","CHAR_NCHAR"
,"CHAR_LONG","CHAR_CLOB","CHAR_NCLOB","NUM_NUMBER1","NUM_NUMBER2","NUM_NUMBER3","NUM_NUMBER4","NUM_NUMBER5","D
ATE_DATE","DATE_TIMESTAMP","DATE_TIMESTAMP_TIMEZONE","DATE_TIMESTAMP_LOCAL" from DATA_TYPE_TEST1
```

↓Ora2Pg で変換

【例 1 Ora2Pg で変換したPostgreSQL用の CREATE VIEW 文】

```
CREATE OR REPLACE VIEW view_test3 AS
 SELECT data_type_test1.id, data_type_test1.char_varchar2_byte, data_type_test1.char_varchar2_char,
data_type_test1.char_nvarchar2, data_type_test1.char_char_byte, data_type_test1.char_char_char,
data_type_test1.char_nchar, data_type_test1.char_long, data_type_test1.char_clob, data_type_test1.char_nclob,
data_type_test1.num_number1, data_type_test1.num_number2, data_type_test1.num_number3,
data_type_test1.num_number4, data_type_test1.num_number5, data_type_test1.date_date,
data_type_test1.date_timestamp, data_type_test1.date_timestamp_timezone, data_type_test1.date_timestamp_local
   FROM data_type_test1;
```

【例 2 移行元の Oracle の CREATE VIEW 文 】

```
CREATE OR REPLACE VIEW VIEW_TEST1
AS
select
"ID","CHAR_VARCHAR2_BYTE","CHAR_VARCHAR2_CHAR","CHAR_NVARCHAR2","CHAR_CHAR_BYTE","CHAR_CHAR_CHAR","CHAR_NCHAR"
,"CHAR_LONG","CHAR_CLOB","CHAR_NCLOB","NUM_NUMBER1","NUM_NUMBER2","NUM_NUMBER3","NUM_NUMBER4","NUM_NUMBER5","D
ATE_DATE","DATE_TIMESTAMP","DATE_TIMESTAMP_TIMEZONE","DATE_TIMESTAMP_LOCAL" from data_type_test1
with read only
```

↓Ora2Pg で変換

【例 2 Ora2Pg で変換したPostgreSQL用の CREATE VIEW 文】

```
CREATE OR REPLACE VIEW "view_test1" AS
SELECT
"id","char_varchar2_byte","char_varchar2_char","char_nvarchar2","char_char_byte","char_char_char","char_nchar"
,"char_
long","char_clob","char_nclob","num_number1","num_number2","num_number3","num_number4","num_number5","date_dat
e","date_t
imestamp","date_timestamp_timezone","date_timestamp_local"  FROM data_type_test1
with read only ;
```

※末尾の with read onry 句は手作業で削除する。

【例 3 移行元の Oracle の CREATE VIEW 文 】

```
CREATE OR REPLACE VIEW VIEW_TEST2
AS
select
```

```
"ID","CHAR_VARCHAR2_BYTE","CHAR_VARCHAR2_CHAR","CHAR_NVARCHAR2","CHAR_CHAR_BYTE","CHAR_CHAR_CHAR","CHAR_NCHAR"
,"CHAR_LONG","CHAR_CLOB","CHAR_NCLOB","NUM_NUMBER1","NUM_NUMBER2","NUM_NUMBER3","NUM_NUMBER4","NUM_NUMBER5","D
ATE_DATE","DATE_TIMESTAMP","DATE_TIMESTAMP_TIMEZONE","DATE_TIMESTAMP_LOCAL" from data_type_test1 where id < 3
with CHECK OPTION
```

↓Ora2Pg で変換

【例 3 Ora2Pg で変換したPostgreSQL用の CREATE VIEW 文】

```
CREATE OR REPLACE VIEW "view_test2" AS
SELECT
"id","char_varchar2_byte","char_varchar2_char","char_nvarchar2","char_char_byte","char_char_char","char_nchar"
,"char_
long","char_clob","char_nclob","num_number1","num_number2","num_number3","num_number4","num_number5","date_dat
e","date_t
imestamp","date_timestamp_timezone","date_timestamp_local"  FROM data_type_test1 where id < 3
with CHECK OPTION ;
```

※末尾の With CHECK OPTION 句は手作業で削除する。

6. シーケンス

6.1. DDL の違い

Oracle と PostgreSQL における CREATE SEQUENCE 文の違いを比較します。

【Oracle の CREATE SEQUENCE 文】[9]

```
CREATE SEQUENCE シーケンス名
  [ START WITH 初期値 ]
  [ INCREMENT BY 増分値 ]
  [ MAXVALUE 最大値 | NOMAXVALUE ]
  [ MINVALUE 最小値 | NOMINVALUE ]
  [ CYCLE | NOCYCLE ]
  [ CACHE キャッシュ数 | NOCYCLE ]
  [ ORDER | NOORDER ]
```

【PostgreSQLの CREATE SEQUENCE 文】[10]

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE シーケンス名
  [ START [WITH] 初期値 ]
  [ INCREMENT [ BY ] 増分値 ]
  [ MAXVALUE 最大値 | NO MAXVALUE ]
  [ MINVALUE 最小値 | NO MINVALUE ]
  [ CYCLE | NO CYCLE ]
  [ CACHE キャッシュ数 ]
  [ OWNED BY { 表名.列名 | NONE } ]
```

6.2. 変換方針

Oracle から PostgreSQL へシーケンスを移行する際の注意点を記載します。

9 Oracle Database SQL 言語リファレンス 11g リリース 2 (http://docs.oracle.com/cd/E16338_01/server.112/b56299/statements_6015.htm#i2067093)より引用

10 PostgreSQL 9.2.0 付属ドキュメント(http://www.postgresql.jp/document/9.2/html/sql-createsequence.html)より引用

6.2.1. MAXVALUE、MINVALUE

Oracleと PostgreSQL でシーケンスの取りうる値の範囲が異なるため、Oracle のシーケンスの値が PostgreSQL の値の範囲を超える場合はシーケンスの値を初期化したり、別のシーケンスと組合せる等の対処を検討します。

【PostgreSQL の値】
正の数： 9223372036854775807（2^63-1）
負の数： -9223372036854775808（-(2^63-1)）

【Oracle の値】
正の数： 9999999999999999999999999999 (10^28 -1)
負の数： -9999999999999999999999999999 (-(10^27 -1))

ただし、PostgreSQL の最大値でも 1 万/秒でカウントしたとして、約 3000 万年ほどかかる値であるため、通常はシーケンスの最大値が問題になることはありません。

6.2.2. ORDER

Oracle では RAC を使用する場合、要求順序どおりの番号生成を保証するために ORDER 句を指定することがありますが、PostgreSQL では指定できないパラメータですので、移行時には削除します。

6.3. DDL の移行手順

移行ツールの Ora2Pg を利用すると Oracle のシーケンスから CREATE SEQUENCE 文を生成できます。

6.3.1. Ora2Pg による移行例

【例 1 移行元の Oracle の CREATE SEQUENCE 文 】

```
CREATE SEQUENCE SEQ_TEST_01
    INCREMENT BY 1
    START WITH 1
    NOMAXVALUE
    NOMINVALUE
    NOCYCLE
    NOORDER
    NOCACHE
```

↓Ora2Pg で変換

【例 1 Ora2Pg で変換したPostgreSQL用の CREATE SEQUENCE 文】

```
CREATE SEQUENCE ”seq_test_01”
    INCREMENT 1
    MINVALUE 1
    NO MAXVALUE
    START 1
    CACHE 1
```

【例 2 移行元の Oracle の CREATE SEQUENCE 文 】

```
CREATE SEQUENCE SEQ_TEST_02
    INCREMENT BY 1
    START WITH 1
    MAXVALUE 100
    NOMINVALUE
    CACHE 20
    CYCLE
    ORDER
```

↓Ora2Pg で変換

【例 2 Ora2Pg で変換したPostgreSQL用の CREATE SEQUENCE 文】

```
CREATE SEQUENCE "seq_test_02"
    INCREMENT 1
    START 1
    MAXVALUE 100
    MINVALUE 1
    CACHE 20
    CYCLE
```

7. シノニム

Oracle のシノニムはテーブルや索引等のデータベースオブジェクトに対するエイリアス（別名）を定義するもので、実体のオブジェクトへの透過アクセスを可能にします。Oracle のシノニムには以下の様な操作を行うことができます。

DML

- INSERT、SELECT、UPDATE、DELETE
- LOCK TABLE
- EXPLAIN PLAN
- FLASHBACK TABLE

DDL

- GRANT、REVOKE
- COMMENT
- AUDIT、NOAUDIT

7.1. DDL の違い

Oracle と PostgreSQL における CREATE SYNONYM 文の違いを比較します。

【Oracle の CREATE SYNONYM 文】[11]

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM 別名
  FOR スキーマ名.オブジェクト名
```

【PostgreSQLの CREATE SYNONYM 文】

```
シノニムは存在しない
```

7.2. 変換方針

PostgreSQL にはシノニムが存在しないため、Oracle のシノニムを完全に移行することはできません。

ただし、テーブルに対する DML 操作であれば Oracle のシノニムと同様の機能をビューで実現することができます。ビューの実現方針については 5.2 を参照してください。

7.3. DDL の移行手順

Oracle のシノニムは Ora2Pg では移行できないため、手作業で変換します。

11 Oracle Database SQL 言語リファレンス 11g リリース 2 (http://docs.oracle.com/cd/E16338_01/server.112/b56299/statements_7001.htm#CJAJCDDF) より引用

8. 制約

8.1. 利用できる制約の種類

OracleとPostgreSQLでそれぞれ利用できる制約の種類を以下にまとめます。

表 8.1: Oracleで利用できる制約

制約名	列	表	オプション
PRIMARY KEY 制約	○	○	
UNIQUE 制約	○	○	
NOT NULL 制約	○	×	
REFERENCES 制約	○	○	指定なし
			ON DELETE CASCADE
			ON DELETE SET NULL
CHECK 制約	○	○	
DEFAULT 句	○	×	

表 8.2: PostgreSQLで利用できる制約

制約名	列	表	オプション
PRIMARY KEY 制約	○	○	
UNIQUE 制約	○	○	
NOT NULL 制約	○	×	
REFERENCES 制約	○	○	指定なし
			ON DELETE CASCADE
			ON DELETE SET NULL
			ON DELETE RESTRICT
			ON DELETE NO ACTION
			ON DELETE SET DEFAULT
			ON UPDATE RESTRICT
			ON UPDATE NO ACTION
			ON UPDATE CASCADE
			ON UPDATE SET NULL
			ON UPDATE SET DEFAULT
CHECK 制約	○	○	
DEFAULT 句	○	×	
EXCLUDE 制約	×	○	

8.2. 変換方針

Oracleで使用できる制約は全てPostgreSQLでも利用可能ですので、基本的にそのまま移行することが可能です。ただし、CHECK制約のSQLにOracle独自の関数を使用している場合は、移行時にPostgreSQL用のSQLに書き換える必要があります。

8.3. DDL の移行手順

移行ツールの Ora2Pg を利用すると Oracle の制約を移行できます。

8.3.1. Ora2Pg による移行例

【例 1 移行元の Oracle の CREATE TABLE 文 】

```
CREATE TABLE CONST_TEST1 (
    ID                          VARCHAR2(5)     NOT NULL,
    DATA                        VARCHAR2(200),
    FK_ID                       VARCHAR2(5),
    CONSTRAINT CONST_TEST1 PRIMARY KEY(ID) ,
    CONSTRAINT CONST_TEST1_FK_ID FOREIGN KEY(FK_ID)
               REFERENCES DATA_TYPE_TEST1 (ID)
)
```

↓Ora2Pg で変換

【例 1 Ora2Pg で変換したPostgreSQL用の CREATE TABLE 文】

```
CREATE TABLE const_test1
(
  id character varying(5) NOT NULL,
  data character varying(200),
  fk_id character varying(5),
  CONSTRAINT const_test1_pkey PRIMARY KEY (id ),
  CONSTRAINT const_test1_fk_id FOREIGN KEY (fk_id)
      REFERENCES data_type_test1 (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
```

【例 2 移行元の Oracle の CREATE TABLE 文 】

```
CREATE TABLE CONST_TEST2 (
    ID                          VARCHAR2(5)     NOT NULL,
    DATA                        VARCHAR2(200),
    FK_ID                       VARCHAR2(5),
    CONSTRAINT CONST_TEST2 PRIMARY KEY(ID) ,
    CONSTRAINT CONST_TEST2_FK_ID FOREIGN KEY(FK_ID)
               REFERENCES DATA_TYPE_TEST1 (ID) ON DELETE CASCADE
)
```

↓Ora2Pg で変換

【例 2 Ora2Pg で変換したPostgreSQL用の CREATE TABLE 文】

```
CREATE TABLE const_test2
(
  id character varying(5) NOT NULL,
  data character varying(200),
  fk_id character varying(5),
  CONSTRAINT const_test2_pkey PRIMARY KEY (id ),
  CONSTRAINT const_test2_fk_id FOREIGN KEY (fk_id)
      REFERENCES data_type_test1 (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE CASCADE
)
```

【例３移行元の Oracle の CREATE TABLE 文 】

```
CREATE TABLE CONST_TEST3 (
    ID                          VARCHAR2(5)     NOT NULL,
    DATA                        VARCHAR2(200),
    FK_ID                       VARCHAR2(5),
    CONSTRAINT CONST_TEST3 PRIMARY KEY(ID) ,
    CONSTRAINT CONST_TEST3_FK_ID FOREIGN KEY(FK_ID)
               REFERENCES DATA_TYPE_TEST1 (ID) ON DELETE SET NULL
)
```

↓Ora2Pg で変換

【例３Ora2Pg で変換したPostgreSQL用の CREATE TABLE 文】

```
CREATE TABLE const_test3
(
  id character varying(5) NOT NULL,
  data character varying(200),
  fk_id character varying(5),
  CONSTRAINT const_test3_pkey PRIMARY KEY (id ),
  CONSTRAINT const_test3_fk_id FOREIGN KEY (fk_id)
      REFERENCES data_type_test1 (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE SET NULL
)
```

【例４移行元の Oracle の CREATE TABLE 文 】

```
CREATE TABLE CONST_CHECK_TEST1 (
    ID                          VARCHAR2(5)     NOT NULL,
    NUM                         NUMBER(3),
    CONSTRAINT CONST_CHECK_TEST1 PRIMARY KEY(ID) ,
    CONSTRAINT CONST_CHECK_TEST1_CK CHECK(num<70)
)
```

↓Ora2Pg で変換

【例４Ora2Pg で変換したPostgreSQL用の CREATE TABLE 文】

```
CREATE TABLE const_check_test1
(
  id character varying(5) NOT NULL,
  num numeric(3,0),
  CONSTRAINT const_check_test1_pkey PRIMARY KEY (id ),
  CONSTRAINT const_check_test1_ck CHECK (num < 70::numeric)
)
```

9. マテリアライズド・ビュー

9.1. DDLの違い

OracleとPostgreSQLにおけるCREATE MATERIALIZED VIEW文の違いを比較します。

【Oracle の CREATE MATERIALIZED VIEW 文】[12]

```
CREATE MATERIALIZED VIEW ビュー名
      [REFRESH [FAST | COMPLETE] [NEXT SYSDATE[+NUMTODSINTERVAL(IntegerLiteral),IntervalUnit]]
      AS SELECT文
      [PRIMARY KEY (ColumnName [,…])]
      [UNIQUE HASH ON (HashColumnName [,…])
      PAGES = PrimaryPages]
```

【PostgreSQLの CREATE MATERIALIZED VIEW 文】

```
マテリアライズド・ビューは存在しない
```

9.2. 変換方針

PostgreSQLにはマテリアライズド・ビューが存在しないため[13]、Oracleのマテリアライズド・ビューを完全に移行することはできません。

PostgreSQLでOracleのマテリアライズド・ビューと同様の機能を実現する方法として、Oracleのマテリアライズド・ビューと同じカラム名を持つテーブルをPostgreSQLに作成し、マテリアライズド・ビューに保持する検索結果を実データとして投入しておく方法があります。ただし、この方法では参照元のテーブルにおける最新データと同期させる仕組みを、トリガや定期的にテーブルを作り直すバッチプログラムとして実装する必要があります。

9.3. DDLの移行手順

Oracleのマテリアライズド・ビューはOra2Pgでは移行できないため、手作業で変換します。

12 Oracle Database SQL 言語リファレンス 11g リリース 2 (http://docs.oracle.com/cd/E16338_01/server.112/b56299/statements_6002.htm#i2063793) より引用

13 PostgreSQL 9.3 ではマテリアライズド・ビューがサポートされる予定です。

10. 別紙一覧

- 別紙: 組み込みデータ型対応表(Oracle-PostgreSQL)

著者

版	所属企業・団体名	部署名	氏名
スキーマ移行調査編 1.0 (2012 年度 WG2)	TIS 株式会社	戦略技術センター	中西 剛紀