

スキーマ移行調査編

製作者
所属企業: TIS 株式会社、
富士通エフ・アイ・ピー株式会社

改訂履歴

版	改訂日	変更内容
1.0	2013/04/22	新規作成
2.0	2018/04/20	共通 各 DDL の違いの PostgreSQL の CREATE XXX 文の PostgreSQL 文書の抜粋は 9.6 版に変更 2.2.2 PCTFREE、PCTUSED 誤記訂正 2.2.5 NOLOGGING の追加 2.3.1.Ora2Pg で対応していないデータ型 表内のその他 (ROWID,BFILE を削除) 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 Ora2Pg 18.2 の自動変換結果に更新 3.2.1..Ora2Pg による移行例の追加 5.2.1 更新可能なビューの移行方針 の更新 5.3 DDL の移行手順 の更新 5.3.1 Ora2Pg による移行例 ora2pg 移行例 ora2pg18.2 の結果に変更、with read option 句の対応例を追加 6.2.3 CACHE の追加 7.3. DDL の移行手順の更新 7.3.1. Ora2Pg による移行例の追加 9 マテリアライズド・ビュー 更新

ライセンス



本作品は CC-BY ライセンスによって許諾されています。
 ライセンスの内容を知りたい方は <http://creativecommons.org/licenses/by/2.1/jp/> でご確認ください。
 文書の内容、表記に関する誤り、ご要望、感想等につきましては、PGECcons のサイトを通じてお寄せいただきますようお願いいたします。
 サイト URL <https://www.pgecons.org/contact/>

Oracle は、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。
 PostgreSQL は、PostgreSQL Community Association of Canada のカナダにおける登録商標およびその他の国における商標です。
 その他、本資料に記載されている社名及び商品名はそれぞれ各社が商標または登録商標として使用している場合があります。

はじめに

■本資料の目的

本資料は、異種 DBMS から PostgreSQL ヘ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 ヘ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	9.6.5
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	18.2	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.....	5
1.1. 組み込みデータ型の違い.....	5
1.2. 変換方針.....	5
2. テーブル.....	5
2.1. DDL の違い.....	5
2.2. 変換方針.....	6
2.3. DDL の移行手順.....	7
3. パーティショニングテーブル.....	10
3.1. DDL の違い、変換方針.....	10
3.2. DDL の移行手順.....	10
4. 索引.....	12
4.1. DDL の違い.....	12
4.2. 変換方針.....	12
4.3. DDL の移行手順.....	13
5. ビュー.....	15
5.1. DDL の違い.....	15
5.2. 変換方針.....	15
5.3. DDL の移行手順.....	15
6. シーケンス.....	17
6.1. DDL の違い.....	17
6.2. 変換方針.....	17
6.3. DDL の移行手順.....	17
Oracle と同様の採番を行う場合は上記の CACHE 20 を 1 に変更する必要があります。.....	18
7. シノニム.....	19
7.1. DDL の違い.....	19
7.2. 変換方針.....	19
7.3. DDL の移行手順.....	19
8. 制約.....	20
8.1. 利用できる制約の種類.....	20
8.2. 変換方針.....	20
8.3. DDL の移行手順.....	20
9. マテリアライズド・ビュー.....	23
9.1. DDL の違い.....	23
9.2. 変換方針.....	23
9.3. DDL の移行手順.....	23
10. 別紙一覧.....	24

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 ]
    [ ( { 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 ]
  )
]
```

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#2095331) より引用



```
[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 ] table_name
  OF type_name [ (
    { column_name WITH OPTIONS [ column_constraint [ ... ] ]
      table_constraint }
    [ ... ]
  ) ]
[ WITH ( storage_parameter [= value] [ ... ] ) | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace_name ]
```

ここで column_constraint には、次の構文が入ります。

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) [ NO INHERIT ] |
  DEFAULT default_expr |
  UNIQUE index_parameters |
  PRIMARY KEY index_parameters |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
  [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

また、table_constraint には、次の構文が入ります。

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) [ NO INHERIT ] |
  UNIQUE ( column_name [ ... ] ) index_parameters |
  PRIMARY KEY ( column_name [ ... ] ) index_parameters |
  EXCLUDE [ USING index_method ] ( exclude_element WITH operator [ ... ] ) index_parameters [ WHERE ( predicate ) ] |
  FOREIGN KEY ( column_name [ ... ] ) REFERENCES reftable [ ( refcolumn [ ... ] ) ]
  [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

また like_option は、以下の通りです。

```
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | ALL }
```

UNIQUE、PRIMARY KEY および EXCLUDE 制約内の index_parameters は以下の通りです。

```
[ WITH ( storage_parameter [= value] [ ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

EXCLUDE 制約内の exclude_element は以下の通りです。

```
{ column_name | ( expression ) } [ opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ]
```

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 PostgreSQL 9.6 付属ドキュメント (<http://www.postgresql.jp/document/9.6/html/sql-createtable.html>) より引用

2.2.2. PCTFREE、PCTUSED

PCTFREE で指定する更新用に確保する空き領域割合は、PostgreSQL では WITH 句の fillfactor パラメータで指定します。

ただし、fillfactor は空き領域割合ではなく、挿入で使用可能な領域割合を指定するため、設定値の意味合いとしては下記の式のイメージになります。

$$\text{fillfactor} = 100 - \text{PCTFREE}$$

また、PCTFREE のデフォルトは 10 ですが、fillfactor のデフォルトはテーブルが 100、インデックスが 90 となっています。

PCTUSED に相当するパラメータは、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。

2.2.5. NOLOGGING

PostgreSQL では UNLOGGED テーブルに相当します。Oracle が REDO ログの生成を抑止するのと同様に PostgreSQL では WAL の生成を抑止します。Oracle との相違は、PostgreSQL でインスタンス・リカバリが発生するとデータベースクラスタ内の全ての UNLOGGED テーブルが切り捨てられます。これは immediate オプションのデータベースクラスタの停止も含まれます。

ora2pg では NOLOGGING オプションを検知すると UNLOGGED テーブルに変換しますので、インスタンス・リカバリでデータのロスが許容できない場合は、UNLOGGED を削除します。

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
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	
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]	
その他	BFILE BLOB		bytea	最大長は 1GB のため、それを超えるデータが存在する可能性がある場合はラージオブジェクトでの代替、ファイルパスのみでの管理等を検討する必要がある。
	CLOB NCLOB		text	
	ROWID		oid	移行先のデータ型で要件を満たせるかを確認する必要がある
	XMLType		xml	

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(10),
  char_long text,
  char_clob text,
  char_nclob text,
  num_number1 integer,
  num_number2 bigint,
  num_number3 bigint,
  num_number4 double precision,
  num_number5 double precision,
  date_date timestamp,
  date_timestamp timestamp,
  date_timestamp_timezone timestamp with time zone,
  date_timestamp_local timestamp with time zone
);
```

→ 手作業で timestamp(6) に書き換える
 → 手作業で timestamp(6) with time zone に書き換える
 → 手作業で timestamp(6) with time zone に書き換える

【例 2 移行元の Oracle の CREATE TABLE 文】

INTERVAL 型の列の Ora2Pg で自動変換例。

```
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】

```
CREATE TABLE data_type_test3 (
  id varchar(5) NOT NULL,
  date_interval_year_month INTERVAL YEAR TO MONTH,
  date_interval_day_second INTERVAL DAY TO SECOND(6)
);
```

【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
);
```

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

Oracle の LOB 型、ROWID、XMLType の列の Ora2Pg での自動変換例。

```
CREATE TABLE DATA_TYPE (
  C0 ROWID,
  C1 BLOB,
  C2 CLOB,
  C3 NCLOB,
  C4 BFILE,
  C5 XMLType
);
```

↓ Ora2Pg で変換

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

```
CREATE TABLE data_type (
  c0 oid,
  c1 bytea,
  c2 text,
  c3 text,
  c4 bytea,
  c5 xml
);
```

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 はリストおよびレンジパーティショニングテーブルのみ対応しています。それ以外のパーティショニング方式など、手動での変換が必要になる場合、3.2.2 以降の手順を手作業で行います。

3.2.1. Ora2Pg による移行例

【例 1 移行元の Oracle の CREATE TABLE 文 (パーティション)】

```
CREATE TABLE P_TEST
(
  SALES_YMD          DATE NOT NULL,
  ITEM_CD           VARCHAR2 (6) NOT NULL,
  SU                NUMBER (9, 0),
  KINGAKU           NUMBER (12, 0)
)
PARTITION BY RANGE (SALES_YMD)
(
  PARTITION P_TEST_2016 VALUES LESS THAN (TO_DATE ('2017/01/01', 'YYYY/MM/DD')),
  PARTITION P_TEST_2017 VALUES LESS THAN (TO_DATE ('2018/01/01', 'YYYY/MM/DD')),
  PARTITION P_TEST_2018 VALUES LESS THAN (TO_DATE ('2019/01/01', 'YYYY/MM/DD'))
)
```

↓ Ora2Pg で変換

【例 1 Ora2Pg で変換した PostgreSQL 用の CREATE TABLE 文 (親テーブル)】

```
CREATE TABLE p_test (
  sales_ymd timestamp NOT NULL,
  item_cd varchar (6) NOT NULL,
  su integer,
  kingaku bigint
);
```

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

【例 1 Ora2Pg で変換した PostgreSQL 用の CREATE TABLE 文(子テーブル)】

```
CREATE TABLE p_test_2016 ( CHECK (
  SALES_YMD < to_date(' 2017-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS')
) ) INHERITS (P_TEST);
CREATE TABLE p_test_2017 ( CHECK (
  SALES_YMD >= to_date(' 2017-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS')
  AND SALES_YMD < to_date(' 2018-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS')
) ) INHERITS (P_TEST);
CREATE TABLE p_test_2018 ( CHECK (
  SALES_YMD >= to_date(' 2018-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS')
  AND SALES_YMD < to_date(' 2019-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS')
) ) INHERITS (P_TEST);
```

【例 1 Ora2Pg で変換した PostgreSQL 用の CREATE FUNCTION 文と CREATE TRIGGER 文】

```
CREATE OR REPLACE FUNCTION P_TEST_insert_trigger()
RETURNS TRIGGER AS $$
BEGIN
  IF ( NEW.SALES_YMD < to_date(' 2017-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS') )
    THEN INSERT INTO P_TEST_2016 VALUES (NEW.*);
  ELSIF ( NEW.SALES_YMD < to_date(' 2018-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS') )
    THEN INSERT INTO P_TEST_2017 VALUES (NEW.*);
  ELSIF ( NEW.SALES_YMD < to_date(' 2019-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS') )
    THEN INSERT INTO P_TEST_2018 VALUES (NEW.*);
  ELSE
    -- Raise an exception
    RAISE EXCEPTION 'Value out of range in partition. Fix the P_TEST_insert_trigger() function!';
  END IF;
  RETURN NULL;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER P_TEST_trigger_insert
BEFORE INSERT ON P_TEST
FOR EACH ROW EXECUTE PROCEDURE p_test_insert_trigger();
```

3.2.2. 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。

3.2.3. 子テーブルの作成

3.2.2 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。

3.2.4. 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り込みが自動的に行われるようになります。

3.2.5. 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。

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트리初期数]
[STORAGE
(
  [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 ] [ [ IF NOT EXISTS ] name ] ON table_name [ USING method ]
( { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [ ... ] )
[ WITH ( storage_parameter = value [ , ... ] ) ]
[ TABLESPACE tablespace_name ]
[ WHERE predicate ]
```

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

索引の種類	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。

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 9.6. 付属ドキュメント (<http://www.postgresql.jp/document/9.6/html/sql-createindex.html>) より引用

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

索引の種類	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ないので、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 ] [ RECURSIVE ] VIEW name [ ( column_name [, ... ] ) ]
[ WITH ( view_option_name [= view_option_value] [, ... ] ) ]
AS query
[ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

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

PostgreSQL 9.3 以降では更新可能ビューがサポートされました。以下の条件を満たす場合、自動的に更新可能ビューになります。

- ビューの FROM リストには正確に1つだけの項目を持たなければならない、それはテーブルまたは他の更新可能ビューでなければなりません。
- ビューの定義の最上位レベルにおいて WITH、DISTINCT、GROUP BY、HAVING、LIMIT、OFFSET を含めてはなりません。
- ビューの定義の最上位レベルにおいて集合操作 (UNION、INTERSECT、EXCEPT) を含めてはなりません。
- ビューの選択リストに、集約関数、ウィンドウ関数、集合を返す関数を含めてはなりません。

Oracle でも同様な条件を満たす場合は更新可能 View となりますが、参照専用のビューとしたい場合は [WITH READ ONLY] オプションを付与することで参照専用にできます。PostgreSQL では参照専用のオプションは存在しないため、運用の要件を考慮し対応を検討します。

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。

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, DATE_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
(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_date, date_timestamp, date_timestamp_timezone, date_timestamp_local)
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_DATE, DATE_TIMESTAMP, DATE_TIMESTAMP_TIMEZONE, DATE_TIMESTAMP_LOCAL
FROM DATA_TYPE_TEST1;
```

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 9.6. 付属ドキュメント (<http://www.postgresql.jp/document/9.6/html/sql-createview.html>) より引用



【例2 移行元の Oracle の CREATE VIEW 文】

```
CREATE OR REPLACE VIEW VIEW_TEST1
AS
select ID, CHAR_VARCHAR2_BYTE 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 (id, char_varchar2_byte)
AS SELECT ID, CHAR_VARCHAR2_BYTE FROM data_type_test1;
```

※末尾の with read onry 句は PostgreSQL でサポートされないため削除されます。参照専用とする場合は更新可能ビューの更新条件を満たさないように変更する必要があります

↓ 書き換え例

【例2 Ora2Pg で変換した PostgreSQL 用の CREATE VIEW 文を参照専用に変更した例】

```
CREATE OR REPLACE VIEW view_test1 (id, char_varchar2_byte)
AS SELECT ID, CHAR_VARCHAR2_BYTE FROM data_type_test1
JOIN (SELECT) dummy ON (true);
```

※例では SELECT だけの行も列も生成せず、結合条件は常に真にするようなダミーの結合を行い更新可能ビューにならないようにしています。ただし、この方法は現時点での更新可能ビューの条件を満たさないだけであり、将来的にも保証されるものではありません。

【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, DATE_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
(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_date, date_timestamp, date_timestamp_timezone, date_timestamp_local)
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_DATE, DATE_TIMESTAMP, DATE_TIMESTAMP_TIMEZONE, DATE_TIMESTAMP_LOCAL FROM data_type_test1 where id < 3
with CHECK OPTION;
```


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】⁸

```
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】⁹

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE [ IF NOT EXISTS ] name [ INCREMENT [ BY ] increment ]
[ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]
[ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE ]
[ OWNED BY { table_name.column_name | 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。

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: $9999999999999999999999999999999999 (10^{28}-1)$

負の数: $-9999999999999999999999999999999999 (-(10^{27}-1))$

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。

6.2.3. CACHE

Oracle と PostgreSQL ではキャッシュの仕様が異なります。(「DB 移行開発見積り編 5.7.3.1.キャッシュの仕様」参照) Oracle と同様なキャッシュの採番を行うためには CACHE に 1 を設定し、キャッシュを無効にします。

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

8 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

9 PostgreSQL 9.6. 付属ドキュメント (<http://www.postgresql.jp/document/9.6/html/sql-createsequence.html>) より引用



【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
```

Oracle と同様の採番を行う場合は上記の CACHE 20 を 1 に変更する必要があります。

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よう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】¹⁰

```
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 では単体のビューへの変換が可能です。

7.3.1. Ora2Pg による移行例

【例 1 移行元の Oracle の CREATE SYNONYM 文】

```
CREATE SYNONYM test.data_type_test5
FOR test2.data_type_test5
```

↓ Ora2Pg で変換

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

```
CREATE VIEW test.data_type_test5 AS SELECT * FROM test2.data_type_test5;
ALTER VIEW test.data_type_test5 OWNER TO test2;
GRANT ALL ON test.data_type_test5 TO test;
```

10 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 3 Ora2Pg 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 4 Ora2Pg 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】¹¹

```
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】¹²

```
CREATE MATERIALIZED VIEW [ IF NOT EXISTS ] table_name
  [ (column_name [, ...] ) ]
  [ WITH ( storage_parameter [= value] [, ...] ) ]
  [ TABLESPACE tablespace_name ]
  AS query
  [ WITH [ NO ] DATA ]
```

9.2. 変換方針

PostgreSQL では読取専用マテリアライズド・ビューをサポートしますが更新可能または書込可能マテリアライズド・ビューはサポートされておりません。また、リフレッシュはコマンドによる全体リフレッシュしかありません。Oracle のようにマテリアライズド・ビューログを利用しての高速リフレッシュの機能はないため、システム要件として高速同期を満たす必要がある場合は、マテリアライズド・ビューに保持する検索結果を別テーブルとして用意し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、トリガーなどで実装する必要があります。

9.3. DDLの移行手順

読取専用かつ定期的な全件リフレッシュで良い場合は ora2pg により変換で対応します。ただし、物理領域のオプションは削除されるため、必要に応じて手動で設定する必要があります。(「DB 移行開発見積り編 5.5 マテリアライズド・ビュー」参照)

上記以外の条件の場合は、手作業でマテリアライズド・ビューと同等のテーブルと更新するための仕組みを作成します。

9.3.1. Ora2Pg による移行例

【例 1 移行元の Oracle の CREATE MATERIALIZED VIEW 文】

```
CREATE MATERIALIZED VIEW MV_EMP
PCTFREE 10
PCTUSED 40
TABLESPACE USERS
REFRESH FORCE ON DEMAND
FOR UPDATE
AS SELECT EMP_ID, EMP_NAME FROM EMP
```

↓ Ora2Pg で変換

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

```
CREATE MATERIALIZED VIEW mv_emp AS
SELECT EMP.EMP_ID EMP_ID, EMP.EMP_NAME EMP_NAME FROM EMP EMP;
```

ora2pg では領域やリフレッシュ等のオプションは削除されます。

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_6002.htm#i2063793) より引用

12 PostgreSQL 9.6. 付属ドキュメント (<http://www.postgresql.jp/document/9.6/html/sql-createsequence.html>) より引用



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
スキーマ移行調査編 2.0 (2017年度 WG2)	富士通エフ・アイ・ピー株式会社	システム技術推進部	多田 明弘