PostgreSQLエンタープライズ・コンソーシアム 技術部会 WG#2

WG2活動報告書
DB移行開発見積り編

目次

1. 改訂履歴

版	改訂日	変更内容
1.0	2017/04/27	新規作成

2. ライセンス

3. はじめに

3.1. 本資料の目的

本資料は、異種DBMSからPostgreSQLへ移行する際の移行作業コストを把握をするための情報提供を目的としています。今回は、「データベースオブジェクトの移行」をターゲットに整理しています。

3.2. 本資料で記載する範囲

本資料では、移行元の異種DBMSとしてOracle Databaseを想定し、「OracleからPostgreSQLへのデータベースオブジェクト移行」の範囲で記載しています。今回は、作業コストを把握しやすいように、データベースオブジェクトから主要なものに限定し取り扱っています。

また、アーキテクチャ、SQL、組み込み関数などの相違については本資料では取り扱っていません。これらに関しては、それぞれ「スキーマ移行調査編」、「SQL移行調査編」、「組み込み関数移行調査編」などを参照してください。

3.3. 本資料で扱う用語の定義

資料で記述する用語について以下に定義します。

表 3.1 用語定義

No.	用語	意味
1	DBMS	データベース管理システムを指します。ここでは、PostgreSQLおよび異種DBMSの総称として利用します。
2	異種DBMS	PostgreSQL以外のデータベース管理システムを指します。
3	Oracle	データベース管理システムのOracle Databaseを指します。

3.4. 本資料で扱うDBMSおよびツール

本書では以下のDBMS、ツールを前提にした調査結果を記載します。

表 3.2 本書で扱うDBMS

DBMS名称	バージョン
PostgreSQL	9.6.0
Oracle Database	11gR2 11.2.0.2.0

表 3.3 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	17.6	GNU General Public License v3	http://ora2pg.darold.net/	OracleのスキーマとデータをPostgreSQLに移行するツール

4. オブジェクト移行コストの算出

4.1. 移行作業の視点

データベースオブジェクト移行にあたり、異種DBMSとPostgreSQLの相違を理解する必要があります。まずはオブジェクトの移行可否を把握し、移行に必要な作業量を測る視点と移行手段を整理します。

1. 移行対象の視点

Oracleが持つオブジェクトをPostgreSQLに移行可能かを識別する視点
移行後も引き続き使用するかもあわせて判断
（テーブル、インデックス、ビュー、シーケンス、シノニムなど）

2. 影響範囲の視点

PostgreSQL移行がオブジェクトの定義(DDL)やアプリケーションへ影響する範囲の視点
（オブジェクトの共通要素、個別要素の影響範囲）

3. 移行手段の視点

移行ツールがカバーする範囲の視点
（カバーされない範囲は自作ツールの開発や手作業の検討）

4.2. 移行作業コストの概算

次章以降では、上記の視点に基づき、データベースオブジェクト移行の共通要素と個別要素についてまとめています。

移行元の異種DBMSで扱っているオブジェクトの棚卸しを行い、各視点における影響範囲に関わる要素がどの程度 あるのかを、オブジェクト定義文やDBMSのディクショナリの検索、目視により調査します。

影響度合いは移行元DBMSにより異なるので、影響する作業量から影響度(高/中/低)などのように分類します。それから、オブジェクト単位（共通要素も含む）に影響本数と影響度に仕訳して、概算工数を算出します。

5. オブジェクト移行の共通要素

データベースオブジェクト移行にあたり、オブジェクト全体に共通した要素への影響を把握する必要があります。

[対象範囲]

- オブジェクトの命名ルール
- 予約語
- データ型

PostgreSQLの命名ルールに従っていない場合や予約語と同じ名前のオブジェクトは修正が必要です。また、データ型の種類や精度の相違があるため、アプリケーション（業務）影響を考慮した対応付けが必要です。

5.1. オブジェクトの命名ルール、予約語

OracleとPostgreSQLではオブジェクトの命名ルール、予約語に違いがあります。

それぞれの仕様を確認し、そのまま使用できない場合は変更をする必要があります。

PostgreSQLの命名ルールに従わない場合や予約語を識別子として使用する場合は、識別子を二重引用符（"）で囲む必要があります。また、アプリケーションのSQLについても識別子を二重引用符で囲む必要があります。

5.1.1. オブジェクトの命名ルール

PostgreSQLではオブジェクトの命名ルールとして以下があります。

- 文字（a～zおよび発音区別符号付き文字と非Latin文字）、アンダースコア（_）で始め、
  2文字目以降には文字、アンダースコア、数字（0～9）あるいはドル記号($)を使用することが可能
- 長さはNAMEDATALEN-1バイト
  （NAMEDATALEN は、src/include/pg_config_manual.hに指定されている。デフォルトは64）

デフォルト設定の場合、PostgreSQLの方が長い識別子を許容するため長さの制限は問題となりませんが、Oracleでオブジェクトの名前にシャープ記号(#)がついている場合は注意が必要です。

5.1.1.1. 移行ツールによる作業量の軽減

Oracleでオブジェクトの名前にシャープ記号を使用している場合、Ora2Pgで出力されるSQLには二重引用符付きで出力されます。

Ora2Pgで出力されたSQLのまま定義した場合、該当オブジェクトを指定する時はSQL内で必ず二重引用符をつけて指定する必要があります。

5.1.1.2. DBMSの相違による手作業の作業量の把握

シャープ記号を使わないオブジェクト名に変更する場合の作業量は、シャープ記号を使用している箇所数に依存します。

また、オブジェクト名を変更する方法、二重引用符で囲む方法のどちらの対処を選択した場合でもアプリケーションのSQLに修正が必要です。

5.1.2. 予約語

Oracleでは予約語ではありませんが、PostgreSQLでは予約語である単語があります。

予約語と同じ名前のオブジェクトや変数を使用している場合は、名前を変更する必要があります。

以下にPostgreSQLでは予約語となる単語の一例を記載します。

- BINARY
- COLLATE
- FREEZE
- OFFSET
- WINDOW

5.1.2.1. 移行ツールによる作業量の軽減

Ora2Pgで出力されるSQLには、PostgreSQLでは定義できない名前でもそのまま出力されます。

そのため、出力された後のSQLに対して名前の変更作業または二重引用符の付加を行う必要があります。さらに、定義の修正以外にもアプリケーションの修正が必要となります。

5.1.2.2. DBMSの相違による手作業の作業量の把握

予約語をオブジェクトや変数の名前に使用している箇所数に依存します。

5.2. データ型

OracleからPostgreSQLへ移行する際、組み込みデータ型はPostgreSQLの適切なデータ型に マッピングをして置き換えを行う必要があります。

表 5.1 データ型の移行可否

型分類	移行可否	備考
文字列型	○	用途に合わせてchar(固定長)、varchar(上限付き可変長)、text(可変長)に移行。
数値型	○	NUMBER型、浮動小数点型について精度やスケールに合わせて適切な型を選択して移行。
日付時刻型	○	PostgreSQLのdate型は時刻情報を保持しないため、timestamp系のデータ型に移行。
バイナリ型	○	バイナリ系のデータ型はbytea型に移行。
その他のデータ型	△	ROWID、XMLType等のPostgreSQLに移行先の型はあるが機能的には十分でないもの、PostgreSQLに対応するデータ型がないもの、ユーザ定義型などについては、代替案やPostgreSQL上での実装を検討。

5.2.1. 移行ツールによる作業量の軽減

基本的なデータ型については、Ora2Pgにより適切なPostgreSQLのデータ型への自動変換が可能です。移行先の型をパラメータで制御したり、データ型毎に任意の変換先のデータ型を指定することも可能です。

5.2.1.1. 文字列型

Oracleの各種文字列型からPostgreSQLの文字列型に置き換えます。文字列型についてはchar、varchar、textから適切な型を自動選択して変換が可能です。

5.2.1.2. 数値型

OracleのNUMBER型や浮動小数点型からPostgreSQLの適切なデータ型に置き換えます。PostgreSQLのnumeric型は演算処理の負荷が高いため、Ora2Pgでは下記のようにnumeric型を極力選択しない方向で変換を行います。

- 整数は桁に合わせてsmallint, integer, bigintから適切な型を自動選択。
- 小数点ありの場合、精度とスケールに応じてreal、floatに変換。
- それ以外の場合はnumericに変換。

5.2.1.3. 日付時刻型

Oracleの日付時刻系のデータ型から適切なPostgreSQLのデータ型に置き換えます。DATE型はPostgreSQLにも同名の型が存在しますが、PostgreSQLでは日付情報のみを保持するため、基本的にはtimestamp系のデータ型に変換します。

- タイムゾーン情報の有無に合わせて、timestamp系のデータ型(timestamp、timestamp with time zone)に変換します。

5.2.1.4. バイナリ型

OracleのRAWやBLOB等のバイナリ系のデータ型は、PostgreSQLのbytea型に変換します。

5.2.2. DBMSの相違による手作業の作業量の把握

データ型の変換後の仕様差異による注意点や、ツールでは自動変換できないケースへの対応について記載します。

5.2.2.1. 文字列型

- PostgreSQLのtext型の最大長は1GBであるため、それを超えるサイズの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等を検討する必要があります。
- 文字列型に指定する長さは、Oracleでは文字数もしくはバイト数として扱うことが可能ですが、PostgreSQLは文字数固定となるため注意が必要です。

5.2.2.2. 数値型

- FLOAT型はPostgreSQLにも存在していますが、Oracleの方が最大精度が大きいため注意が必要です。

5.2.2.3. 日付時刻型

- TIMESTAMP系のデータ型の秒未満の精度情報は維持されません。精度指定なしの場合はPostgreSQLにおける精度は6桁になるため、必要に応じて追記をします。また、Oracleにおける最大精度は9桁であるため注意が必要です。

5.2.2.4. バイナリ型

- PostgreSQLのbytea型の最大長は1GBであるため、それを超えるサイズの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ります。

5.2.2.5. その他の型

- ROWIDやBFILE、XMLType等はOra2Pgで自動変換が可能ですが、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ります。
- UROWID型など、Ora2Pgで自動変換ができないデータ型についてはそのまま出力されるため、別途対応方針を検討する必要があります。
- ユーザ定義のデータ型を使用している場合、個別にPostgreSQL上での実装や代替方法の検討が必要です。

サンプルとして、Oracleにて下記テーブルが定義されているとします。

```
CREATE TABLE DATA_TYPE (
        C1 ROWID,
        C2 BFILE,
        C3 XMLType,
        C4 UROWID
);
```

Ora2Pgを使用すると、デフォルトでは下記のように変換されます。

```
CREATE TABLE data_type (
        c1 oid,
        c2 bytea,
        c3 xml,
        c4 UROWID  -- 自動変換不可のためそのまま出力
) ;
```

また、移行先のデータ型は任意に指定することが可能です。

例えば、ROWID型をoid型に変換することを避ける場合、設定ファイルのDATA_TYPEに下記のように設定することで、int型に変換されます。

```
DATA_TYPE    ROWID:int
```

下記は本設定を行いサンプルのテーブルの変換した場合の出力です。

```
CREATE TABLE data_type (
        c1 int,
        c2 bytea,
        c3 xml,
        c4 UROWID
) ;
```

6. スキーマオブジェクトの移行

スキーマオブジェクトは、ユーザによってスキーマ内に作成される論理構造です。表や索引などのいくつかのオブジェクトはデータを保持します。ビューやシノニムなどのその他のオブジェクトは定義のみで構成されます。

[対象範囲]

- テーブル
- 制約
- 索引(インデックス)
- ビュー
- マテリアライズドビュー
- トリガ
- シーケンス
- シノニム
- データベースリンク

6.1. テーブル

6.1.1. 移行対象の選定作業

データベースの根幹をなすオブジェクトであるテーブルについては、テーブルの種別や構成する列の定義に対して移行可否を検討します。

表 6.1 テーブル種別による移行可否

型分類	移行可否	備考
一時テーブル	△	PostgreSQLにも存在するが、セッションごとに定義する必要がある。
パーティションテーブル	○	同等の構文は存在しないが、機能の組み合わせによって代替可能。

表 6.2 列定義による移行可否

型分類	移行可否	備考
データ型	○	文字列、数値、日付時刻、バイナリ等、基本的なデータ型については移行可能。
DEFAULT句	△	DEFAULT句はPostgreSQLでも定義可能。定義内でOracle固有の関数やユーザ定義関数を使用している場合、代替が可能かを検討。

6.1.2. 移行ツールによる作業量の軽減

基本的なテーブルの構文はOra2Pgで変換が可能です。PostgreSQLが対応していないオプションについては変換時に削除されます。

6.1.2.1. 一時テーブル

Ora2Pgではデータベース上の定義から一時テーブルの変換をすることができません。DDL文を記載したファイルを入力とすることで、PostgreSQL向けに変換することは可能です。

6.1.2.2. パーティションテーブル

PostgreSQLにはPARTITION BY構文は存在しないため、PostgreSQLのテーブルの継承機能、CHECK制約、トリガ等を組み合わせてパーティションを実現しています。

Ora2Pgを使用する場合、方式によって自動変換の可否が異なります。

- リスト、レンジパーティションの場合
  親テーブル、子テーブル、子テーブルの分割キー列のインデックス、CHECK制約、INSERTトリガ等の構成要素に変換されます。
- ハッシュ、コンポジットパーティションの場合
  Ora2Pgでは対応していないため、単体のテーブルとして変換されます。

6.1.3. DBMSの相違による手作業の作業量の把握

6.1.3.1. 列定義

Ora2Pgで移行した結果、列に関連する定義について、対応が必要なポイントを整理します。

- Ora2Pgでは変換できないデータ型やユーザ定義型などについて、移行方法の検討をします。

6.1.3.2. オプション

Ora2Pgによる変換時にオプション構文はほぼ削除されることになりますが、PostgreSQL に同じような機能を持つオプションが存在する場合、追記することを検討します。

表 6.3 テーブルオプション

オプション名	説明	PostgreSQLでの対応
PCTFREE	ブロック内で更新用に確保する領域の割合を指定	FILLFACTORにより代用が可能です。FILLFACTORでは挿入で使用可能な領域の割合を指定するため、設定値としては逆の意味合いを持ちます。またPCTFREEのデフォルト値は10ですが、FILLFACTORのデフォルト値は100であるため、PostgreSQLのデフォルトでは更新用の領域は確保をしないことになります。
NOLOGGING	一部処理でトランザクションログの出力をスキップする	PostgreSQLにはUNLOGGED TABLEというトランザクションログを出力しないテーブルが存在します。ただし、トランザクションログが必要な状況に陥った場合、テーブルが壊れた状態になってしまうため、単純な代替としては使用できません。

6.1.3.3. 一時テーブル

PostgreSQLの一時テーブルはセッション終了時に削除されてしまうため、アプリケーション等からセッションごとに都度定義をする必要があります。

Ora2Pgで構文を変換した場合はON COMMIT句が削除されてしまうため、PostgreSQLのデフォルトである「PRESERVE ROWS」以外の挙動をさせる場合、定義を追記する必要があります。

また、Oracleのデフォルトの挙動は「DELETE ROWS」である点も注意が必要です。PostgreSQLは自動コミットで動作するため、Oracleに合わせて「DELETE ROWS」に 設定した場合は、明示的にトランザクションに入って処理をしないと、即座に データが削除されてしまいます。

6.1.3.4. パーティションテーブル

ハッシュやコンポジット等のOra2Pgが対応していない方式のパーティションの場合、パーティションの振り分けに使用する関数を作り込む必要があります。

また、性能面で問題がないかについては、事前に検討をしておく必要があります。PostgreSQLのパーティションでは、パーティション数の目安は100くらいまでとされており、それ以上の場合オーバヘッドが懸念されるため、パーティションの単位を変更することも 検討する必要があります。

6.2. 制約

Oracleの制約については、表制約、列制約ともにPostgreSQLでも基本的な構文はそのまま使用可能です。

表 6.4 制約の移行可否

種別	移行可否	備考
主キー制約 (表、列)	○	
外部キー制約 (表、列)	○	
一意性制約 (表、列)	○	
NOT NULL制約 (列)	○	
検査制約 (表、列)	△	関数や演算子を使用している場合、別途それらの移行が必要になる可能性がある。

制約の状態に関するオプションについては、一部使用できないものがあります。

表 6.5 制約のオプションの移行可否

オプション名	説明	移行可否	備考
[NOT] DIFERRABLE	トランザクション完了まで制約のチェックを遅延する	○	遅延可能な制約には制限あり。
INITIALLY [IMMEDIATE | DIFERRED]	制約チェックのタイミングを指定する	○	
[ENABLE | DISABLE]	制約の使用可否を指定する	×	PostgreSQLには機能が存在しない。
[VALIDATE | NOVALIDATE]	制約使用時に既存のデータに適用するかを指定する	×	PostgreSQLには機能が存在しない。

また、外部キー制約の参照先のデータが削除された場合の挙動に関するOracleのオプションについては、PostgreSQLでも使用可能です。

表 6.6 外部キー制約のオプションの移行可否

オプション名	説明	移行可否	備考
指定なし	参照先の行は削除不可	○	
ON DELETE CASCADE	参照先の行削除時に参照元の行も削除	○	
ON DELETE SET NULL	参照先の行削除時に参照元の列にNULLを格納	○	

6.2.1. 移行ツールによる作業量の軽減

基本的な構文に違いはありませんが、PostgreSQLでは使用できないオプションについては 削除する必要があります。

Ora2Pgを使用して移行した場合、不要なオプションについては削除が可能です。また、制約の定義については主にALTER TABLE構文として出力されますが、制約によってはCREATE TABLE文に含めるかALTER TABLE構文として外出しするかを 制御可能です。他に、制約の名称に関する制御を行うオプション等も存在します。

6.2.2. DBMSの相違による手作業の作業量の把握

6.2.2.1. 検査制約

制約に関する差異ではありませんが、検査制約においてOracle固有の関数や演算子を使用している場合、その関数の代替を検討する必要があります。

6.2.2.2. 外部キー制約のON UPDATE句

PostgreSQLでは外部キー制約の参照先に対する更新(UPDATE)時の挙動を制御するオプション (ON UPDATE 〜)が使用可能であるため、OracleでON UPDATE句相当の機能を別の手段を用いて実現していた場合には、制約の定義に置き換えることが可能です。

6.3. 索引(インデックス)

6.3.1. 移行対象の選定作業

Oracleの索引タイプに着目し、PostgreSQLへの移行可否を検討します。

- 移行可能・・・移行対象。索引定義の対象数を把握します。
- 移行不可・・・移行対象外。事前に性能面での影響確認の必要があります。

表 6.7 索引タイプの種類による移行可否

索引タイプ	移行可否
通常索引	○
逆キー索引	×
ビットマップ	×
ドメイン	×
関数ー通常索引	○
関数ー逆索引	×
関数ービットマップ	×
関数ードメイン	×
クラスタ	×
索引構成表	×
パーティション索引	○

6.3.2. 移行ツールによる作業量の軽減

索引定義(DDL)をPostgreSQL用の定義に変換する作業が必要です。
Ora2Pgで自動変換することで手作業量を軽減することが出来ます。

6.3.2.1. 通常索引

Ora2Pgでの構文変換は、以下の範囲では100%の変換率です。

- 索引定義に物理要素のオプションが指定がある場合は、「ソート順」]以外は除去されます。
- 物理要素は移行元とは異なる物理配置の検討が必要です。

表 6.8 利用別の変換内容

用途	変換内容
主キー制約	構文変換
外部キー制約	構文変換
ユニーク制約	構文変換
インデックス	構文変換

6.3.2.2. 関数索引

Ora2Pgでの構文変換は、ユーザ定義関数・演算式では100%の変換率です。

- ユーザ定義関数と演算子の記述箇所は変換されません。
- 組み込み関数は、PostgreSQL対応の異なる関数名に変換されます。
- PostgreSQLにない組み込み関数は事前に用意が必要です。

表 6.9 関数の種類による変換内容

関数の種類	変換内容
ユーザ定義関数	構文変換
組み込み関数	構文変換、PostgreSQL用関数に変換
演算式	構文変換

6.3.3. DBMSの相違による手作業の作業量の把握

DBMSの構造や仕様などの主要な相違点や対処方法を整理しました。
利用頻度の高いもの（★印の項目）に関して、手作業時の作業量を計る視点ををを記述しています。

6.3.3.1. 通常索引

[1]索引の利用用途からの把握

表 6.10 索引の利用用途からの把握

用途	移行可否	作業内容
外部キー制約	○	
ユニーク制約(★)	△	指定するカラムがNotNull制約でない場合、データ構造やアプリケーションの影響範囲を検討します。
インデックス	○	

[2]索引の物理属性(使用頻度の高いもの)からの把握

表 6.11 索引の物理属性(使用頻度の高いもの)からの把握

オプション名	説明	移行可否	作業内容
PCTFREE(★)	同一ブロック内の未使用領域の割合	△	FILLFACTOR という使用領域の割合で代替します
INITRANS	データ・ブロック内で保持する初期トランザクション数	×	オプションが存在しないため、考慮不要です
TABLESPACE	データの論理構造名称	×	テーブルスペースの概念が異なります

[3]デフォルト値で設定されているもの(DDLに記述がない）による把握

表 6.12 Oracle側のデフォルト値(DDLに記述がない）による把握

Oracle構文			PostgreSQL構文		
オプション名	説明	デフォルト値	オプション名	デフォルト値	作業内容
SORT	索引を昇順ソートする	ASC	ASC/DESC	ASC	無し
PCTFREE	同一ブロック内の未使用領域の割合	10	FILLFACTOR	90	無し
INITRANS	データ・ブロック内で保持する初期トランザクション数	2	--	--	無し

表 6.13 PostgreSQL側のデフォルト値（DDLに記述が無い）による把握

オプション名	説明	デフォルト値	作業内容
COLLATION	照合順序	指定した列、対象式の照合順序	列や式の照合順に起因します
ASC/DESC	索引を昇順ソートする	ASC	影響しません
NULLS FIRST(★)	NULLを非NULLより前にソートする	DESCが指定された場合	物理的な格納順で利用上の問題はありません
NULLS LAST(★)	NULLを非NULLより後にソートする	DESCが指定されない場合	性能の影響確認を検討します
FILLFACTOR(★)	同一ブロック内の使用領域の割合	B-treeは90%	性能の影響確認を検討します
TABLESPACE_NAME	テーブルスペース名	恒久テーブル・・・default_tablespace 一時テーブル・・・temp_tablespaces	PostgreSQLのアーキテクチャ設計に起因します

6.3.3.2. 関数－通常索引

[1]関数の種類による把握

表 6.14 関数の種類による把握

関数の種類	作業内容
ユーザ定義関数	なし
組み込み関数(★)	DBMSの仕様相違で関数が不変オブジェクトでない場合や関数が存在しない場合は、関数を作成する必要があります。Oracle互換関数として、orafceで多数の関数が提供されているので活用することを検討します。
演算式	なし

6.4. ビュー

使用される頻度が高いビューの移行について説明します。

ビューに指定された副問合せの移行作業については触れていませんので、別途考慮する必要があります。

Oracleで利用可能なエディショニング・ビュー、オブジェクト・ビュー、XMLTypeビューについても触れませんので、利用している場合は別途考慮する必要があります。

6.4.1. 移行対象の選定作業

Oracleで使用頻度が高いと思われるビューの定義に対する、PostgreSQL移行時の移行可否は以下のとおりです。

表 6.15 ビュー移行の可否

オプション	移行可否	備考
WITH CHECK OPTION	○	そのまま利用可能。
WITH READ ONLY	×	移行不可。削除が必要。
(更新可能ビュー)	△	移行可能だが手作業での修正が必要な場合あり。

6.4.2. 移行ツールによる作業量の軽減

上記項目に対する移行ツールの変換対応は以下のとおりです。

6.4.2.1. WITH READ ONLY

Ora2Pgを使用した場合、削除した状態でSQLが出力されます。

もしビューに対して同等の設定を行いたい場合は手作業で対応する必要があります。

6.4.2.2. 更新可能ビュー

Ora2Pgでは対処しません。

PostgreSQLでビューに対して更新を行いたい場合、手作業での対応が必要な場合があります。

6.4.3. DBMSの相違による手作業の作業量の把握

上記項目で手作業が必要なものに対する作業内容は以下のとおりです。

表 6.16 書き換え作業内容

オプション	作業内容
WITH READ ONLY	SQL実行ユーザに対して更新権限を制限することで実現します。 WITH READ ONLYにする必要があるビューに対してINSERT権限、UPDATE権限、DELETE権限を取り消すREVOKE文を実行し対処します。
(更新可能ビュー)	PostgreSQL文書に記載された更新可能ビューの条件に合うビューについては更新に対応しています。更新可能ビューはPostgreSQL 9.3から実装しています。 PostgreSQL 9.3以前のバージョンや、PostgreSQL文書に記載された条件を満たさないビューについては別途トリガやルールを作成して対処します。 更新可能かどうかは情報スキーマ(information_schema)のviewsを参照することで確認可能です。

6.5. マテリアライズドビュー

6.5.1. 移行対象の選定作業

マテリアライズドビューの種類に着目し、PostgreSQLへの移行可否を検討します。

- 移行可能 ・・・ 移行対象。対象数を把握する。
- 移行不可 ・・・ 移行対象外。影響確認の必要がある。

表 6.17 マテリアライズドビューの種類による移行可否

マテリアライズドビューの種類	移行可否
読取専用	○
更新可能	×
書込み可能	×

6.5.2. 移行ツールによる作業量の軽減

マテリアライズドビュー定義(DDL)をPostgreSQL用の定義に変換する作業が必要です。
Ora2Pgで自動変換することで手作業量を軽減することができます。

Ora2Pgでの構文変換は、以下の範囲では100%の変換率です。

- 物理領域のオプションは除去されます。
- リフレッシュのオプションは除去されます。(※PostgreSQLに存在しない)
- SELECT句は変換されません。

以下の表では、マテリアライズドビューで想定される使用目的に応じて、移行の方針立ての参考にするために利用用途別にツールでの変換内容をまとめています。

表 6.18 利用用途の種類による変換内容

利用用途	変換内容
リモートデータのスナップショット	構文変換
集計や結合処理の結果の高速化	構文変換

6.5.3. DBMSの相違による手作業の作業量の把握

DBMSの構造や使用などの主要な相違点や対処方法を整理しました。

リフレッシュの設定はPostgreSQLでは手動で設定する必要があります。

- REFRESH MATERIALIZED VIEW コマンドを定期的に実行する。

[2] デフォルト値で設定されているもの(DDLに記述がない)による把握

表 6.19 Oracle側のデフォルト値(DDLに記述がない)による把握

Oracle構文			PostgreSQL構文		
オプション名	説明	デフォルト値	オプション名	デフォルト値	作業内容
PCTFREE	同一ブロック内の未使用領域の割合	10	FILLFACTOR	100	性能の影響確認を検討します
IMMEDIATE / DEFERRED	マテリアライズドビュー作成時にデータを投入する	IMMEDIATE	WITH [NO] DATA	WITH DATA	なし
リフレッシュ方法	リフレッシュ要求時のリフレッシュ方法の指定	FORCE	--	--	手動実行を検討します
リフレッシュタイミング	リフレッシュの実行タイミングの指定	On demand	--	--	手動実行を検討します
リフレッシュ方法	クエリーリライトでのマテリアライズドビューの使用可否の指定	無効	--	--	手動実行を検討します

表 6.20 PostgreSQL側のデフォルト値(DDLに記述がない)による把握

オプション名	説明	デフォルト値	作業内容
FILLFACTOR	同一ブロック内の使用領域	テーブルは100%	性能の影響確認を検討します
TABLESPACE_NAME	テーブルスペース名	default_tablespace	PostgreSQLのアーキテクチャ設計に起因する
WITH [NO] DATA	マテリアライズドビューのデータ投入タイミング	作成時に投入	影響しません

6.6. トリガ

6.6.1. 移行対象の選定作業

使用頻度が高いと思われる、DMLに対するトリガはOracle、PostgreSQLともに定義が可能です。

CREATE文やALTER文などのDDLに対するトリガは、PostgreSQL 9.3以降であればイベントトリガとして定義することが可能です。

Oracleで定義が可能なログオン、ログオフに対するトリガはPostgreSQLでは対応していません。

表 6.21 Oracleのトリガへの対応

Oracle	PostgreSQL
BEFORE/AFTER INSERT/UPDATE/DELETE	PostgreSQLにてトリガとして作成可能
BEFORE/AFTER CREATE/ALTER/DROP (CREATEについてはAFTERのみ)	PostgreSQLではイベントトリガとして作成可能
BEFORE LOGOFF / AFTER LOGON	作成不可

DMLとDDLに対するトリガを整理し、移行する対象を選定します。

6.6.2. 移行ツールによる作業量の削減

Ora2Pgにおいて、移行対象のオブジェクト種別に"TRIGGER"を指定することで、DMLに対するトリガを作成するCREATE TRIGGER文が出力されます。

また、OracleのCREATE TRIGGER文の中に書かれていた処理内容にもとづき、PostgreSQLのトリガに使用するためのファンクションを作成するCREATE FUNCTION文も同時に出力されます。※

> ※ OracleとPostgreSQLとでは、OracleがCREATE TRIGGER文の中にトリガ処理内容を直接記述できるのに対して、PostgreSQLではファンクションとして与える必要がある、という違いがあります。

しかし、OracleとPostgreSQLではファンクションの定義に使用する言語や、組み込みファンクションに違いがあるため、そのままでは動作しない場合があります。

作成されたファンクションについては、内容を確認をする必要があります。

6.6.3. DBMSの相違による手作業の作業量

OracleではCREATE TRIGGER文の中にトリガ処理内容を直接記述できるのに対して、PostgreSQLではファンクションとして与える必要があります。

しかし、前述の通りPostgreSQLとOracleとでは、ファンクションに使用する言語や、組み込みファンクションに違いがあります。

そのため、各ファンクションの中身の確認や、処理が異なる場合の修正が手作業として必要になります。

6.7. シーケンス

6.7.1. 移行対象の選定作業

シーケンス(順序)についてはOracleとPostgreSQLで大幅な機能差異はなく、比較的単純に移行が可能です。Oracleのシーケンスの定義構文で指定可能なオプションについて、PostgreSQLへの移行可否を整理します。

表 6.22 シーケンスのオプションの移行可否

オプション名	説明	移行可否	備考
INCREMENT BY	シーケンスの増分値を指定	○	
START WITH	シーケンスの初期値を指定	○	
MAXVALUE | NOMAXVALUE	シーケンスの最大値を指定	○	
MINVALUE | NOMINVALUE	シーケンスの最小値を指定	○	
CYCLE | NOCYCLE	シーケンスの循環可否を指定	○	
CACHE | NOCACHE	シーケンスのメモリへの事前割り当て数を指定	○	デフォルト値が異なる。
ORDER | NOORDER	シーケンスの採番順を維持するかどうかを指定	×	PostgreSQLには存在しないが、主にOracle RAC構成で必要となるパラメータであるため影響なし。

6.7.2. 移行ツールによる作業量の軽減

Oracleの構文で指定可能なオプションについては、ほぼ同等の内容をPostgreSQLでも指定可能ですが、一部書き換えを行う必要があります。Ora2Pgを使用することで、下記のような差分については自動変換が可能です。

- PostgreSQLではNOCACHEを指定できません。キャッシュを無効にするにはCACHEを省略、もしくは1に設定します。
- PostgreSQLにはORDER/NOORDERオプションは存在しないため、指定されている場合はDDLから削除する必要があります。
- NOで始まるキーワード(NOMAXVALUE、NOMINVALUE、NOCYCLE)は、PostgreSQLではNOの後にスペースを入れる(NO MAXVALUE、NO MINVALUE、NO CYCLE)必要があります。

6.7.3. DBMSの相違による手作業の作業量の把握

OracleとPostgreSQLのシーケンスに関する主な仕様差異を説明します。必要に応じて手作業での修正が必要になります。

6.7.3.1. キャッシュの仕様

各オプション構文を省略した場合のデフォルト値は、各DBMSで概ね同様の値が設定されますが、CACHEについてはOracleが20、PostgreSQLが1(キャッシュ無効)と異なっています。また、Ora2Pgを使用した場合は設定値が正しく引き継がれますが、そもそもキャッシュの仕様に 差異があるため注意が必要です。

Oracleではインスタンス単位でキャッシュに指定された数のシーケンス値が確保され、そこから各セッションが値を順次取得します。PostgreSQLではこのキャッシュの処理をセッションごとに行うため、キャッシュに指定された数のシーケンス値を各セッションが確保します。そのため、シーケンス値に抜け番号が発生したり、採番された値が全体では時系列で並ばなくなることがあります。これを避けるには、CACHEを1に設定してキャッシュを無効にすることで、各セッションからシーケンス値が1つずつ確保されるようにします。

6.7.3.2. 諸元値

Oracleのシーケンスの取りうる値の範囲は「-(10^27-1) 〜 10^28-1」、PostgreSQLでは「-(2^63-1) 〜 2^63-1」であるため、値の範囲はPostgreSQLの方が狭くなります。既にPostgreSQLの諸元範囲外の値を使用している場合は、格納値の調整を行う必要が出てきます。

ただし、PostgreSQLの範囲でも現実的に値を使い切ることは考えにくく、影響があるケースはほぼないと思われます。

6.7.3.3. 呼び出し方式

シーケンスから値を取得したり現在の値を確認する際は、Oracleの場合は疑似列にアクセスする形を取りますが、PostgreSQLでは関数呼び出しを行う必要があります。アプリケーションから明示的に発行をしている場合、書き換える必要があります。

- Oracleの例
  ```
  INSERT INTO TBL1 VALUES (SEQ1.NEXTVAL);
  SELECT SEQ1.CURRVAL FROM DUAL;
  ```
- PostgreSQLの例
  ```
  INSERT INTO tbl1 VALUES (nextval('seq1'));
  SELECT currval('seq1');
  ```

6.7.3.4. 自動採番

Oracleでは特定の列のデフォルト値にシーケンスを設定することができない(12c以降は可能)ため、トリガを使用して疑似的に自動採番を実装している場合があります。
PostgreSQLでは列のデフォルト値に設定して採番することが可能であるため、トリガを廃止して列の定義にDEFAULT句を追加することを検討します。

6.8. シノニム

6.8.1. 移行対象の選定作業

PostgreSQLでは、シノニムをサポートしてませんが、 以下のような目的で使用している一部のシノニムについては、ビューなどで代替することが可能です。

Oracleで使用していたシノニムの内、代替手段を使用して移行するシノニムを整理します。

表 6.23 代替手段により移行が可能なシノニム

	目的	対象	移行方法
1	スキーマ名による修飾の省略	ファンクション,テーブル	検索パスを設定する。
2	テーブルに別名をつける	テーブル	Oracleのシノニムと同じ名前でビューを作成する。ビューに対し権限を設定する。

6.8.2. 移行ツールによる作業量の削減

Ora2Pgにおいて、移行対象のオブジェクト種別に"SYNONYM"を指定することで上記の代替手段の内、2のビューを使用した移行について実施するこが可能です。

- Ora2Pgを使用することで、シノニムの対象となっているテーブルに対するビューが作成されます。
- また、作成されたビューに対する所有者の設定と権限の設定が行われます。
- Ora2Pgで作成されたビューの名前については、適宜修正する必要があります。

6.8.3. DBMSの相違による手作業の作業量

代替手段の内、1の検索パスを設定する方法については、手作業により移行することが必要となります。

6.9. データベースリンク

6.9.1. 移行対象の選定作業

PostgreSQLにはOracleのデータベースリンクに相当するオブジェクトは存在しませんが、代替案として 他のデータベースにアクセスする仕組である **FDW、dblink** の2種類が用意されています。

表 6.24 データベースリンク代替案

代替案	利用目的	概要
FDW	アプリケーションからの問い合わせ	リモートDBのテーブルを外部表と定義してテーブルと同様に扱います。
dblink	運用時の一時的な問い合わせ	クエリにdblink関数（引数：DB接続情報,SELECT文）を使用します。

FDWは標準的なSQLで操作ができ、トランザクション管理や問い合わせの最適化などの機能面でdblinkより優れています。
また、様々なデータベースやデータファイルとの連携も可能です。
そのために、Oracleデータベースリンクの代替としては、FDWの方が適しています。

[Oracle構文]

データベースリンクを通して、他のOracleデータベースのテーブルにアクセスします。

```
-- データベースリンクの作成
CREATE DATABASE LINK dblink1
CONNECT TO user1 IDENTIFIED BY password1  USING'remotedb'

-- リモートDBの問い合わせ
SELECT * FROM remote_tbl@dblink1
```

[FDW構文]

FDWモジュールをPostgreSQLに機能拡張することで様々な外部データにアクセスできます。
外部表を通して、通常の表と同様にSQLで操作します。

```
-- fdwモジュールの拡張定義
CREATE EXTENSION postgres_fdw;

-- 外部サーバの定義
CREATE SERVER dblink1
   FOREIGN DATA WRAPPER postgres_fdw
   OPTIONS(host'remote_db',port'5432',dbname'remotedb');

-- 外部サーバのユーザマップの定義
 CREATE USER MAPPING FOR postgres
   SERVER dblink1
   OPTIONS(user'user1',password'password1');

-- 外部表の定義
CREATE FOREIGN TABLE remote_tbl
(
 remote_id       integer,
 remote_name     text
)
SERVER postgres_server
OPTIONS(schema_name'postgres',table_name'remote_tbl');

-- リモートDBの問い合わせ
SELECT * FROM remote_tbl;
```

[dblink構文]

他のデータベースへの問い合わせを行うための関数群です。
dblinkは、DB接続、切断、カーソルOPEN/CLOSE/FETCH単位に制御する関数が存在しています。

```
-- リモートDBの接続
    SELECT dblink_connect
       ('dblink1',hostaddr=remote_db port=5432
        dbname=remotedb  user=user1 password=password1);

-- リモートDBの問い合わせ
    SELECT * FROM
        dblink('dbname=dblink_name',
               'select remote_id, remote_name from remote_tbl')
        AS t1(remote_id integer, remote_name text);

-- リモートDBの切断
    SELECT dblink_disconnect('dblink1');
```

6.9.2. 移行ツールによる作業量の削減

データベースリンク定義（DDL)からFDW関連定義の雛形が生成されます。

表 6.25 FDWの雛形生成

構文	生成範囲	生成されるDDL
CREATE SERVER	データベースリンク名	CREATE SERVER <データベースリンク名> FOREIGN DATA WRAPPER oracle_fdw OPTION(dbserver '接続子名');
CREATE USER MAPPING	ユーザ名	CREATE USER MAPPING FOR <ユーザ名> SERVER <データベースリンク名> OPTION(user <接続ユーザ> password 'secret');

6.9.3. DBMSの相違による手作業の作業量

FDWを使うためには以下の作業が必要です。

表 6.26 FDW使用のための作業

作業	作業内容	作業単位
fdwモジュールのインストール	DBMSの種類に応じたfdw専用モジュールを入手します。	DBMS単位
PostgreSQLの拡張定義	入手したfdw専用モジュールを拡張定義として登録します。	DBMS単位
外部サーバの定義	連携先のDB接続情報を定義します。	DB単位
接続ユーザの定義	連携元と先のユーザマッピングを定義します。	DB単位
接続テーブルの定義	連携先のテーブルを専用スキーマに外部表として定義します。	テーブル単位
SQLの修正	データベースリンク名が指定されているSQLを外部表の指定に修正します。	SQL単位

※接続先DBがPostgreSQLの場合は、外部表を個別定義せずに、
接続先DBのスキーマのテーブルを外部表としてインポートすることが可能です。

[IMPORT構文]

```
IMPORT FOREIGN SCHEMA <接続先スキーマ名>
  FROM SERVER <外部サーバ名> INTO <内部スキーマ名>;
```

.

7. 非スキーマオブジェクトの移行

非スキーマ・オブジェクトは、スキーマに含まれないデータベースで扱うことのできる構成要素です。代表的なものとして、ユーザや権限があります。

[対象範囲]

- スキーマ
- ユーザ
- ロール
- 権限
- テーブルスペース

7.1. スキーマ

7.1.1. 移行対象の選定作業

スキーマについてはOracleとPostgreSQLで違いがあります。

Oracleではユーザとスキーマが1対1で対応していますが、PostgreSQLではユーザとスキーマが対応づいていません。

また、CREATE SCHEMA文の構文や意味も変わるため注意が必要です。

7.1.1.1. ユーザ名と同一名のスキーマについて

Oracleではユーザ名と同じ名前のスキーマが自動的に作成されます。

PostgreSQLではユーザ名と同じスキーマが自動的に作成されることはありませんが、明示的に定義することは可能です。また、スキーマ指定をしない場合に使用するスキーマを変更することも可能です。

Oracleと同様のスキーマを作成したい場合はこれらの作業を移行計画に含めておく必要があります。

7.1.2. 移行ツールによる作業量の削減

Ora2Pgでスキーマを出力するよう設定すれば、定義されていたスキーマのDDLが出力されます。ただし、移行不要なスキーマが出力される場合がありますので、サンプルのスキーマなど不要なスキーマ定義文がないか確認し、必要に応じて削除を行ってください。

7.1.3. DBMSの相違による手作業の作業量の把握

Ora2Pgでスキーマを出力するよう設定すれば、手作業での移行は不要です。

また、SQLに明示的にスキーマ名を指定しない場合でも、search_pathパラメータを変更していなければ、ユーザ名と同じ名前のスキーマを優先してオブジェクトを検索するので設定変更も不要です。

7.1.3.1. CREATE SCHEMA文について

PostgreSQLにも同様の文がありますが、構文・意味合いが異なります。

OracleのCREATE SCHEMA文と同様の処理を行いたい場合は、以下の手順で同等の処理を実施します。

1. スキーマが存在しない場合はCREATE SCHEMA文でスキーマを作成します。
2. CREATE SCHEMA文に指定していたテーブルやビューを作成する文を、該当するスキーマを指定して個別に実行します。

7.2. ユーザ

7.2.1. 移行対象の選定作業

PostgreSQLでは、Oracleと同様にユーザを作成することが可能です。※

> ※ PostgreSQLにおいては、ユーザはログイン権限を持つロールのことであり、Oracleのユーザとは考え方が異なります。ただし、Oracleと同様にCREATE USER文により、ユーザ(ログイン権限を持つロール)を作成することが可能です。

Oracleで使用しているユーザを整理し、PostgreSQLでも引き続き使用するユーザを選定します。

7.2.2. 移行ツールによる作業量の削減

Ora2Pgにおいて、移行対象のオブジェクト種別に"GRANT"を指定することで、Oracleで使用していたユーザを作成する CREATE USER 文が出力されます。

しかし、Ora2Pgではパスワードが引き継がれないため、それらの移行を検討する必要があります。

7.2.3. DBMSの相違による手作業の作業量

7.2.3.1. ユーザ名以外の移行

ユーザの移行では、パスワード以外にもCREATE USERのオプションに含まれない、権限の付与なども必要な作業となります。

ユーザの移行については、運用を検討し、権限なども移行する必要があります。

ユーザの移行時に検討・対応が必要となるポイントは以下の通りです。

表 7.1 ユーザ移行時のポイント

項目	Ora2Pgでの変換対応	説明
パスワード有効期限	出力されるCREATE USER文には設定されない	VALID UNTILオプションで指定することが必要となる
権限	対象のオブジェクトにGRANTを指定した場合にGRANT文が作成される(権限のページを参照)	ユーザの作成後に権限の設定が必要となる
テーブルスペース	出力されるCREATE USER文には設定されない	default_tablespaces パラメータや temp_tablespaces パラメータの設定が必要となる

7.3. ロール

7.3.1. 移行対象の選定作業

Oracleで使用しているロールを整理し、PostgreSQLでも引き続き使用するロールを選定します。

7.3.2. 移行ツールによる作業量の削減

Ora2Pgにおいて、移行対象のオブジェクト種別に"GRANT"を指定することでOracleで使用していたロールの移行が可能です。

7.3.3. DBMSの相違による手作業の作業量

Ora2Pgで出力されるSQLの中にある CREATE ROLE文だけでは権限が設定されませんので、同時に出力されるGRANT文を使用する必要があります。

権限については、権限のページを参照して下さい。

7.4. 権限

7.4.1. 移行対象の選定作業

権限には大きく分けるとシステム権限とオブジェクト権限の2種類があります。

PostgreSQLでも同様の権限設定は可能ですが、SELECT権限等の一部を除いてそのまま移行することはできません。

また、PostgreSQLでシステム権限を設定する場合、GRANT文ではなくCREATE ROLE文やCREATE USER文を使用するものがあります。

7.4.2. 移行ツールによる作業量の軽減

Ora2Pgで出力する対象としてGRANTを指定した場合、PostgreSQLでそのまま定義可能な権限についてはGRANT文を出力します。

しかし、SELECT権限等のごく一部であるため、必要な定義が出力されているかの確認を実施する必要があります。

7.4.3. DBMSの相違による手作業の作業量の把握

設定している権限の種類、設定量に応じて作業量を考慮する必要があります。

以下に、定義される頻度が高いと思われる権限の対応例を示します。

表 7.2 システム権限の移行例

Oracleの権限	PostgreSQLでの定義例
CREATE PROCEDURE	作成先のスキーマに対してGRANT文で実行ユーザへのCREATE権限を設定
EXECUTE ANY PROCEDURE	GRANT文で該当関数に対して実行ユーザへのEXECUTE権限を設定
CREATE ROLE CREATE USER	● CREATE ROLE文またはCREATE USER文でユーザ作成時にCREATEROLE権限を設定 ● ALTER ROLE文またはALTER USER文でCREATEROLE権限を設定
CREATE SESSION	● CREATE ROLE文でユーザ作成時にLOGIN権限を設定 ● CREATE USER文でユーザを作成(NOLOGIN権限は設定しない) ● ALTER ROLE文またはALTER USER文でLOGIN権限を設定
CREATE TABLE	GRANT文で作成先のスキーマやテーブル空間に対して実行ユーザへのCREATE権限を設定

表 7.3 オブジェクト権限の移行例

Oracleの権限	PostgreSQLでの定義例
テーブルに対するALTER	テーブルの所有者のみ可能であるため、以下のように設定します。 1. CREATE ROLE文でALTER権限を持たせるためのロールを作成 2. ALTER文で該当テーブルの所有者を作成したロールに変更 3. ALTER権限を持たせたいユーザをロールのメンバに設定

7.5. テーブルスペース

7.5.1. 移行対象の選定作業

PostgreSQLとOracleとでは、データベースを構成するファイル構造が異なるため、I/O 負荷の分散や容量の管理などの運用に差異が発生します。

それらの差異を踏まえて、移行後のテーブルスペースの構成を検討します。

7.5.2. 移行ツールによる作業量の削減

Ora2Pgにおいて、移行対象のオブジェクト種別に"TABLESPACE"を指定することでOracleで使用していたテーブルスペースを作成するCREATE TABLESPACE文と、各テーブルペースにテーブルや索引を割り当てるALTER TABLE文やALTER INDEX文が出力されます。

ただし、CREATE TABLESPACE文で指定されたディレクトリパスは、Oracleにもとづいているため、修正する必要があります。

7.5.3. DBMSの相違による手作業の作業量

前述の通り PostgreSQL と Oracle とでは、データベースを構成するファイル構造が異なります。

そのため、テーブルスペースを単純に移行しても、期待する性能とはならない場合があります。

以下のような観点で、運用を検討し、必要に応じてテーブルスペースの構成を手作業で変更します。

表 7.4 テーブルスペース移行時のポイント

観点	Oracleの特徴	PostgreSQLでの対応
I/O 負荷の分散	Oracleでは、1つのテーブルを別々のディスクに格納された複数のテーブルスペースに割り当てることにより、I/O 分散が可能です。	PostgreSQLでは基本的に1つのテーブルは1つのテーブルスペースに格納する必要があります。同様の負荷分散を行いたい場合は、パーティショニングなどの検討を行います。
容量の追加	Oracleでは、個々のテーブルが使用できるディスク容量が不足する場合、後から追加されたディスクにテーブルスペースを確保し、そのテーブルスペースをテーブルに追加することが可能です。	PostgreSQLでは個々のテーブルに対してテーブルスペースを追加することができません。追加したディスクに新たなテーブルスペースを作成し、テーブルが使用するテーブルスペースを変更する必要があります。そのため、初めから十分な容量を検討しておくことが重要です。

8. 著者

（企業・団体名順）

版	所属企業・団体名	部署名	氏名
第1.0版 (2016年度WG2)	NECソリューションイノベータ株式会社	サポートサービス事業部	黒澤 彰
	NECソリューションイノベータ株式会社	サポートサービス事業部	近藤 太樹
	日本電子計算株式会社	技術本部 技術サービス部	毛塚 賢一
	日本電子計算株式会社	技術本部 技術サービス部	大久保 明彦
	日本電子計算株式会社	技術本部 技術サービス部	伊藤 渉
	富士通株式会社	ミドルウェア事業本部	山本 明範
	富士通株式会社	ミドルウェア事業本部	豊島 良美
	富士通株式会社	ミドルウェア事業本部	伊與田 智也
	三菱電機株式会社	情報技術総合研究所	田中 覚