PGECons
PostgreSQL Enterprise Consortium

PostgreSQLへのデータベース移行の道標
- 成果物の対応状況と移行のポイント・注意点のご紹介 -

2017年度活動成果報告

PostgreSQLエンタープライズ・コンソーシアム
WG2 (移行WG)

アジェンダ

- PGECons WG2 ～ これまでの活動内容 ～
- 2017年度活動報告
  - 新バージョンに対する成果物の対応状況
  - 移行のポイントや注意点
- おわりに

PGECons WG2

～ これまでの活動内容 ～

ワーキンググループ活動について

- 現在3つのワーキンググループにて活動中

新技術検証WG（WG1）

- 新バージョンの性能や新技術の検証を通じて有用性を明確化
- スケールアップ検証、新機能における性能特性調査等

移行WG（WG2）

- 異種DBMSからの移行をテーマに活動
- 商用DBMSからの移行プロセスに伴う技術調査や検証を実施

課題検討WG（WG3）

- データベース管理者やアプリケーション開発者が抱える現場の課題や困り事に対するテーマを設定
- 可用性・運用性・保守性・セキュリティ・接続性が主な課題領域

移行WG(WG2)活動内容

活動テーマ:異種DBMSからPostgreSQLへの移行

課題認識

- 異種DBMSシステムをPostgreSQLへ移行するプロセスが確立していないことが、普及を妨げる大きな障壁と認識
  - 移行作業をどのように進めればよいかがわからない。
  - 初期段階で移行に必要なトータルコストを算出できない。
  - 過去の経験則や点在するノウハウに依存しているのが現状

活動目標

・異種DBMSからPostgreSQLへの移行を検討する際のガイドラインを提示する。
（難易度判断、留意すべき事項、移行手順）

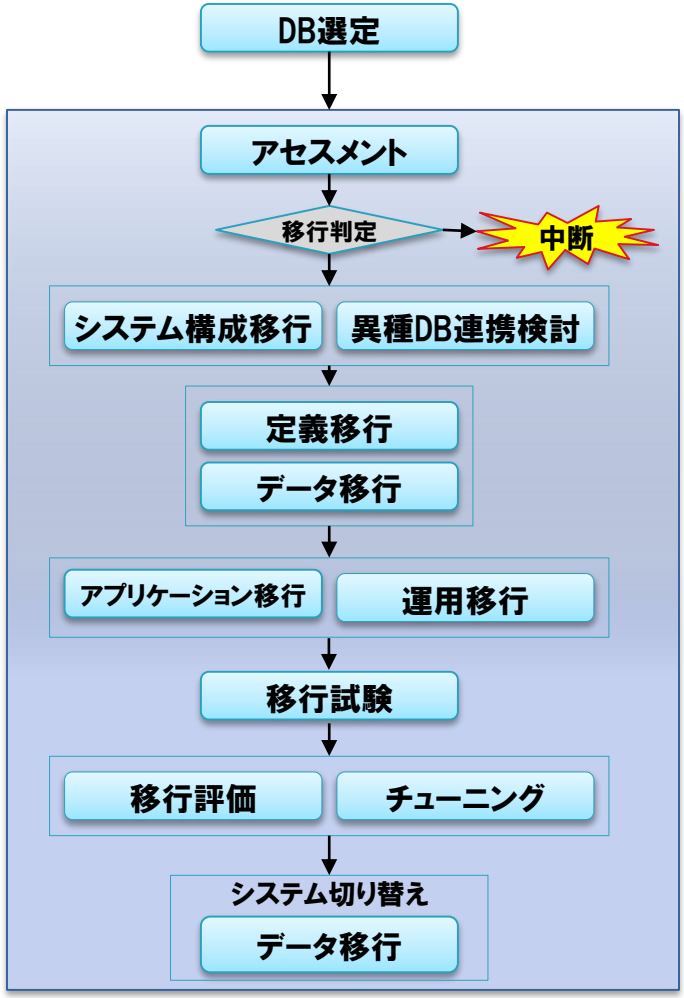
成果物

・「異種DBMSからPostgreSQLへの移行ガイド」を作成

「異種DBMSからPostgreSQLへの移行ガイド」の構成

- 移行作業の全体像を解説
  - DB移行フレームワーク編 （21ページ）
  - DB移行開発見積り編（26ページ）
- 移行作業に含まれる作業内容、手順の調査
  - システム構成調査編（29ページ）
  - 異種DB間連携調査編（18ページ）
  - スキーマ移行調査編（25ページ+別表）
  - データ移行・文字コード変換編（49ページ）
  - ストアドプロシージャ移行調査編（23ページ）
  - アプリケーション移行調査編（10ページ）
  - SQL移行調査編（20ページ+別表）
  - 組み込み関数移行調査編（30ページ+別表）
  - チューニング編（30ページ+別表）
  - バージョンアップ編（39ページ+別表）
  - 試験編（71ページ+別表）
- 移行作業を試行する検証
  - データ移行調査および実践編（60ページ+別表）
  - アプリケーション移行実践編（25ページ+別表）
- DBMSに求められる要件整理
  - DB選定基準編（43ページ+別表）

本編のみでも500ページ越え



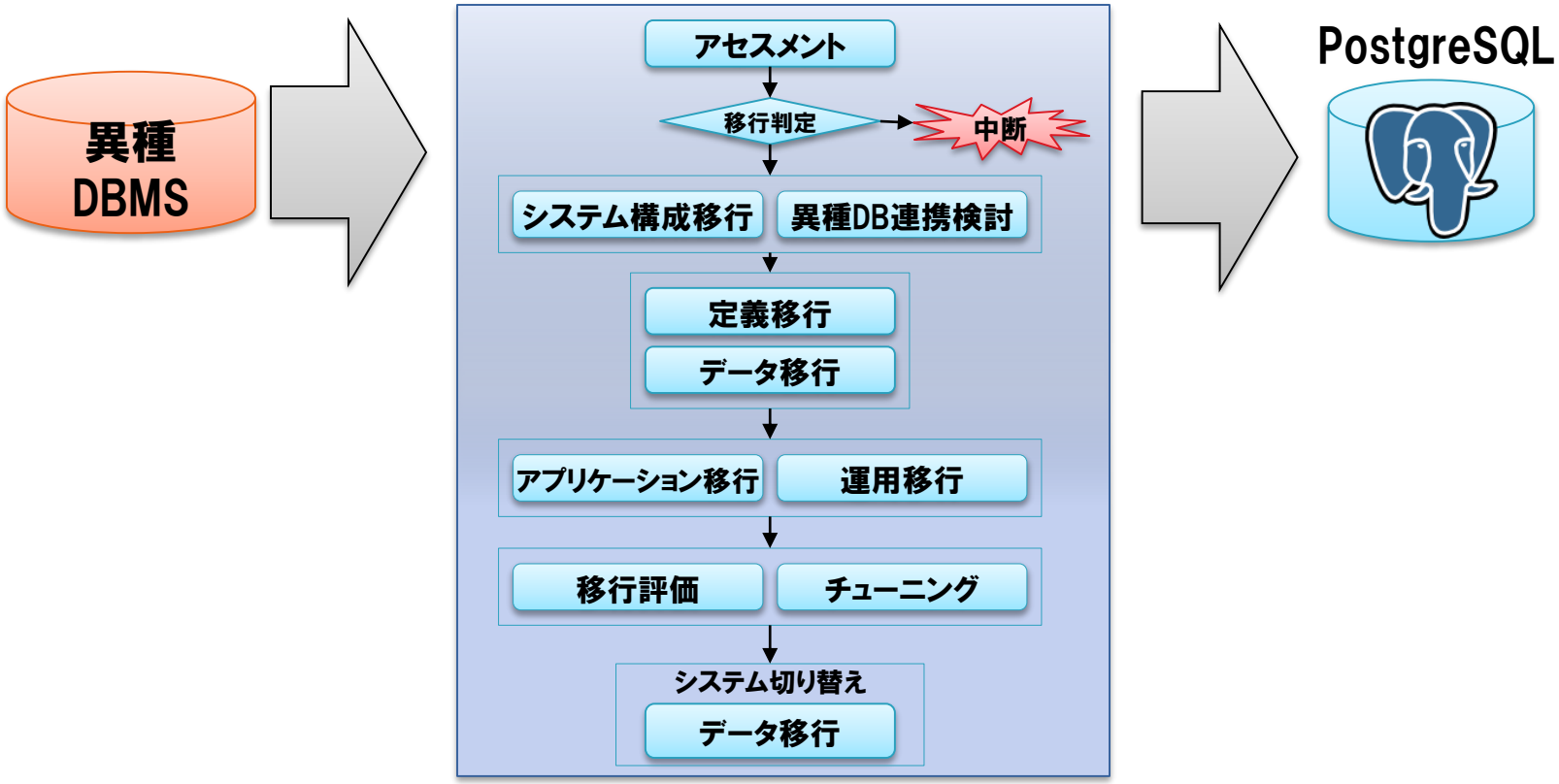
移行プロセス全体像

「異種DBMSからPostgreSQLへの移行ガイド」の文書構成

No.	作成文書（作成年度）	記載概要
1	DB移行フレームワーク編 （2012、2013年）	移行作業全体の解説
2	システム構成調査編（2012年）	DBMSの一般的なシステム構成とPostgreSQLの構成
3	異種DB間連携調査編（2012年）	異種DBMSとPostgreSQLのデータ連携方法
4	スキーマ移行調査編（2012年）	異種DBMSとPostgreSQLのスキーマの違いと書き換え方針
5	データ移行調査および実践編（2012年） データ移行・文字コード変換編（2013年）	異種DBMSからPostgreSQLへのデータ移行時の注意点と実施結果
6	ストアドプロシージャ移行調査編 （2012、2013年）	異種DBMSのストアドプロシージャをPostgreSQLに移行する際の書き換え方針
7	アプリケーション移行調査編（2012年）	APIやトランザクションの差異と対処方法
8	アプリケーション移行実践編（2012年）	異種DBMSからPostgreSQLに移行した際のアプリケーションの移行結果
9	SQL移行調査編（2012年）	異種DBMSとPostgreSQLのSQL互換性調査結果と書き換え方針
10	組み込み関数移行調査編 （2012、2015年）	Oracle, PostgreSQLの組み込み関数互換性調査結果と書き換え方針
11	チューニング（2013年）	PostgreSQLのチューニング手法を記載
12	バージョンアップ編（2013年）	PostgreSQLのバージョンアップ手法と検証結果
13	試験編（2014年）	移行したDBやアプリケーション等の移行結果の妥当性を確認する試験
14	DB選定基準編（2015年）	PostgreSQLをDBMSとして選定する際の基準となる情報
15	DB移行開発見積もり編（2016年）	PostgreSQLへ移行する際の移行作業コストを把握するための情報

DB移行フレームワーク

- PostgreSQLに移行するにはどんな作業が必要か？
  - 「移行プロセスに対する認識を共有する」のが目的
  - DB移行作業の主な流れ



スキーマ

- テーブル、インデックス、シーケンス等のオブジェクトを移行する → DDL文の変換
  - 組み込みデータ型のマッピング
    例：NUMBER（Oracle）→ NUEMRIC（PostgreSQL）
  - 存在しないオブジェクトへの対応（SYNONYM等）
  - 機能の置き換え
    - パーティション構文
    - オプション構文（同様の機能がある場合）
      例：PCTFREE（Oracle）→ FILLFACTOR（PostgreSQL）

Oracle→PostgreSQL移行ツール：Ora2Pg
<http://ora2pg.darold.net/>

- Oracle上に定義されているオブジェクト、データをPostgreSQLに移行するツール

スキーマ移行調査

- 異種DBMSとPostgreSQLのスキーマの違いは？
  - Oracle, PostgreSQLのDDL互換性を調査
  - 各種DDLの相違点をもとにした移行方針を解説
  - 組み込みデータ型対応表

属性	Oracle			PostgreSQL				備考
	データ型	単位	備考	データ型	サイズ	単位	備考	
文字	VARCHAR2	byte	4000 バイト	varchar(n)			上限付き可変長	PostgreSQLは文字数指定だが、Oracle はバイト数指定、文字数指定をセマンティクスで区別するので注意が必要（デフォルトはバイト数指定）
		char	4000 バイト	varchar(n)			上限付き可変長	
	NVARCHAR2	char	4000 バイト	varchar(n)			上限付き可変長	8iはDBのキャラクターセットでバイト数か文字数かが異なる。（9iは文字数指定）
	CHAR	byte	2000 バイト	char(n)			空白で埋められた固定長	PostgreSQLは文字数指定だが、Oracle はバイト数指定、文字数指定をセマンティクスで区別するので注意が必要（デフォルトはバイト数指定）
		char	2000 バイト	char(n)			空白で埋められた固定長	
	NCHAR	char	2000 バイト	char(n)			空白で埋められた固定長	8iはDBのキャラクターセットでバイト数か文字数かが異なる。（9iは文字数指定）
	LONG （下位互換）		2G-1 バイト	text			可変長(最大1GB)	1GBを超える場合は、ラージオブジェクトを使用するか、外部ファイルに保存する。
	CLOB		4G-1 バイト	text			可変長(最大1GB)	
	NCLOB		4G-1 バイト	text			可変長(最大1GB)	

組み込み関数

PostgreSQLに互換性のある関数が存在するか

同機能の関数が存在する場合

- 関数名、引数、戻り値等に差分がある場合は書き換えを行う

同機能の関数が存在しない場合

- 複数の関数や演算の組み合わせなどによる代替
- 必要に応じてファンクションの実装やアプリケーション側で対応

Oracle互換モジュール：orafce <https://github.com/orafce/orafce>

- Oracle互換の関数群をPostgreSQLに提供
- UTL_FILE、DBMS_OUTPUTなど、良く使用される一部のユーティリティパッケージにも対応

組み込み関数対応表

- Oracle、PostgreSQLの組み込み関数互換性を調査
- orafceによる対応状況も記載

Oracle関数	説明	PostgreSQL本体での対応可否(注1)	対応するPostgreSQL関数もしくはPostgreSQLでの対応方法	orafceでの対応可否(注2)
日時ファンクション				
ADD_MONTHS (date, integer)	日付dateに月数integerを加えて戻す。	○	+演算子を使って書き換え可能 例: select date '2013-03-22' + interval '1 months'	○
CURRENT_DATE	セッション・タイムゾーンの現在の日付を戻す。	○	current_date、current_timestamp	
CURRENT_TIMESTAMP	セッション・タイムゾーンの現在の日付および時刻をTIMESTAMP WITH TIME ZONEデータ型の値で戻す。	○	current_timestamp	
DBTIMEZONE	データベースのタイムゾーンの値を戻す。	○	システムカタログpg_settingsから、'TimeZone'のreset_valの値を取得する。 SELECT reset_val FROM pg_settings WHERE name = 'TimeZone';	○
EXTRACT (element FROM date)	日時式または期間式から指定された日時フィールドの値を抽出して戻す。	○	extract(field from timestamp)	
FROM_TZ(timestamp, time_zone_value)	タイムスタンプ値およびタイムゾーンをTIMESTAMP WITH TIME ZONE値に変換する。	○	timezone(time_zone_value, timestamp);	
LAST_DAY(d)	d を含む月の最後の日付を戻す。	○	last_date(date)	○
LOCALTIMESTAMP	セッション・タイムゾーンの現在の日付および時刻をTIMESTAMPデータ型の値で戻す。	○	localtimestamp	
MONTHS_BETWEEN(d1, d2)	d1 とd2 の間の月数を戻す。	○	months_between(d1, d2)	○
NEW_TIME(d, z1, z2)	時間帯z1の日時がdの時点の時間帯z2の日時を戻す。	○	timezone(z2, timezone(z1, TIMESTAMP WITHOUT TIME ZONE d))	
NEXT_DAY(d, char)	charで指定した曜日で日付d以降の最初の日付を戻す。	○	next_day(date, text)	○
NUMTODSINTERVAL(n, 'char_expr')	n をINTERVAL DAY TO SECOND リテラルに変換する。	○	MAKE_INTERVAL(expr:=n)	
NUMTOYMINTERVAL(n, 'char_expr')	n をINTERVAL YEAR TO MONTH リテラルに変換する。	○	MAKE_INTERVAL(expr:=n)	
ROUND(d, fmt)	d を書式 fmt で指定した単位に丸めた結果を戻す。	○	round(date, text)	○

PostgreSQL本体での対応可否と、対応する組み込み関数名/式を記載

orafceを導入することで同名の組み込み関数が利用可能な場合は「○」

2017年度活動報告

- 新バージョンに対する成果物の対応状況
- 移行のポイントや注意点

活動メンバー

- 2017年度は下記6社にて活動しました
  - SRA OSS, Inc. 日本支社
  - 日本電気株式会社（主査）
  - 日本電子計算株式会社
  - 富士通株式会社
  - 富士通エフ・アイ・ピー株式会社
  - 三菱電機株式会社

成果物の現状

- 移行WGの活動も6年目に突入
  - 活動初期のPostgreSQLのバージョンは9.2
    - 最新の10までバージョンを5つ経ている
  - 成果物で紹介している移行関連ツール(Ora2Pg、orafce等)も継続的に開発が行われている
    - 新しくできることも増えてきている

成果物のメンテナンスが課題

「異種DBMSからPostgreSQLへの移行ガイド」の文書構成

No.	作成文書（作成年度）	記載概要
1	DB移行フレームワーク編 （2012、2013年）	移行作業全体の解説
2	システム構成調査編（2012年）	DBMSの一般的なシステム構成とPostgreSQLの構成
3	異種DB間連携調査編（2012年）	異種DBMSとPostgreSQLのデータ連携方法
4	スキーマ移行調査編（2012年）	異種DBMSとPostgreSQLのスキーマの違いと書き換え方針
5	データ移行調査および実践編（2012年） データ移行・文字コード変換編（2013年）	異種DBMSからPostgreSQLへのデータ移行時の注意点と実施結果
6	ストアドプロシージャ移行調査編 （2012、2013年）	異種 る際の書き換え方針
7	アプリケーション移行調査編（2012年）	API
8	アプリケーション移行実践編（2012年）	異種 ンの移行結果
9	SQL移行調査編（2012年）	異種 換え方針
10	組み込み関数移行調査編 （2012、2015年）	Oracle, PostgreSQLの組み込み関数互換性調査結果と書き換え方針
11	チューニング（2013年）	PostgreSQLのチューニング手法を記載
12	バージョンアップ編（2013年）	PostgreSQLのバージョンアップ手法と検証結果
13	試験編（2014年）	移行したDBやアプリケーション等の移行結果の妥当性を確認する試験
14	DB選定基準編（2015年）	PostgreSQLをDBMSとして選定する際の基準となる情報
15	DB移行開発見積もり編（2016年）	PostgreSQLへ移行する際の移行作業コストを把握するための情報

2012、2013年度の
成果物が多め

PostgreSQLの進化

■ バージョンごとの主な機能追加内容

PostgreSQL バージョン	主な追加機能
9.3	マテリアライズド・ビュー、更新可能ビュー、外部データラッパによる書き込み、ページチェックサム
9.4	JSONB型、ロジカルデコーディング、バックグラウンドワーカー
9.5	INSERT ～ ON CONFLICT構文、BRINインデックス、行単位セキュリティ
9.6	パラレルクエリ、複数同期スタンバイ
10	宣言的パーティショニング、ロジカルレプリケーション

移行に関連する項目も含まれている

マテリアライズド・ビュー（PostgreSQL 9.3）

- マテリアライズド・ビューの機能を実現する方法

旧　別途実テーブルを用意してトリガやバッチ処理で同期

新　マテリアライズド・ビューの定義が可能

- 注意事項
  - 読み取り専用
  - リフレッシュは明示的に実行する必要あり（自動不可）
  - リフレッシュはデータの再作成を行う
- Ora2Pgによる移行可

更新可能ビュー（PostgreSQL 9.3）

- ビューに対する更新を可能とする方法

旧　ビューに対してルールやトリガを定義して代替

新　ビューに対して更新が可能

- 条件を満たす必要あり
  - 単一のテーブルが対象、集約関数の結果を返さない、等
  - 明示的に参照専用とすることはできない
  - 更新可否の確認方法（information_schema参照）

```
=> select table_name,is_updatable,is_insertable_into from
information_schema.views where table_name = 'v_emp';
 table_name | is_updatable | is_insertable_into
------------+--------------+--------------------
 v_emp      | YES          | YES
```

INSERT ON CONFLICT（PostgreSQL 9.5）

- **OracleのMERGE構文相当の機能**

旧　WITH句の中でUPDATE文を用いることで実現

新　INSERT ～ ON CONFLICT構文で実現

- **複雑なSQLは不要に**

```
INSERT INTO master
  VALUES (diff.id,diff.val)
  ON CONFLICT (id)
  DO UPDATE
  SET master.val = master.val + diff.val
```

id列が重複するレコードが存在しない場合INSERT

id列が重複するレコードが存在する場合UPDATE

宣言的パーティショニング（PostgreSQL 10）

- パーティションの実装方法

旧 テーブルの継承やトリガなどの機能を組み合わせて実現

新 CREATE TABLEにPARTITION用の専用構文が追加

- パーティションの定義がシンプルに

```
CREATE TABLE colors (name text, data text) PARTITION BY LIST (name);
CREATE TABLE tbl_red PARTITION OF colors FOR VALUES ('red');
CREATE TABLE tbl_green PARTITION OF colors FOR VALUES ('green');
CREATE TABLE tbl_blue PARTITION OF colors FOR VALUES ('blue');
```

- Ora2Pgの対応状況
  - いずれの形式でも出力可能
    （宣言的パーティショニング対応済み）

更新対象文書の選定

- 下記点を考慮して優先度を決定

ダウンロード数

- 良く参照されている文書（需要がある）

作成年度

- 移行WGの活動初期に作成され更新されていない文書

バージョン差分

- バージョンを経ることで差分が発生しやすい文書

ノウハウ

- 追加したいノウハウがある文書

「異種DBMSからPostgreSQLへの移行ガイド」の文書構成

No.	作成文書（作成年度）	記載概要
1	DB移行フレームワーク編 （2012、2013年）	移行作業全体の解説
2	システム構成調査編（2012年）	DBMSの一般的なシステム構成とPostgreSQLの構成
3	異種DB間連携調査編（2012年）	異種DBMSとPostgreSQLのデータ連携方法
4	スキーマ移行調査編（2012年）	異種DBMSとPostgreSQLのスキーマの違いと書き換え方針
5	データ移行調査および実践編（2012年） データ移行・文字コード変換編（2013年）	異種DBMSからPostgreSQLへのデータ移行時の注意点と実施結果
6	ストアドプロシージャ移行調査編 （2012、2013年）	異種DBMSのストアドプロシージャをPostgreSQLに移行する際の書き換え方針
7	アプリケーション移行調査編（2012年）	APIやトランザクションの差異と対処方法
8	アプリケーション移行実践編（2012年）	異種DBMSからPostgreSQLに移行した際のアプリケーションの移行結果
9	SQL移行調査編（2012年）	異種DBMSとPostgreSQLのSQL互換性調査結果と書き換え方針
10	組み込み関数移行調査編 （2012、2015年）	Oracle[illegible]greSQLの組み込み関数互換性調査結果と書き換え方針
11	チューニング（2013年）	
12	バージョンアップ編（2013年）	
13	試験編（2014年）	
14	DB選定基準編（2015年）	PostgreSQLをDBMSとして選定する際の基準となる情報
15	DB移行開発見積もり編（2016年）	PostgreSQLへ移行する際の移行作業コストを把握するための情報

本年度はこれらの文書を更新

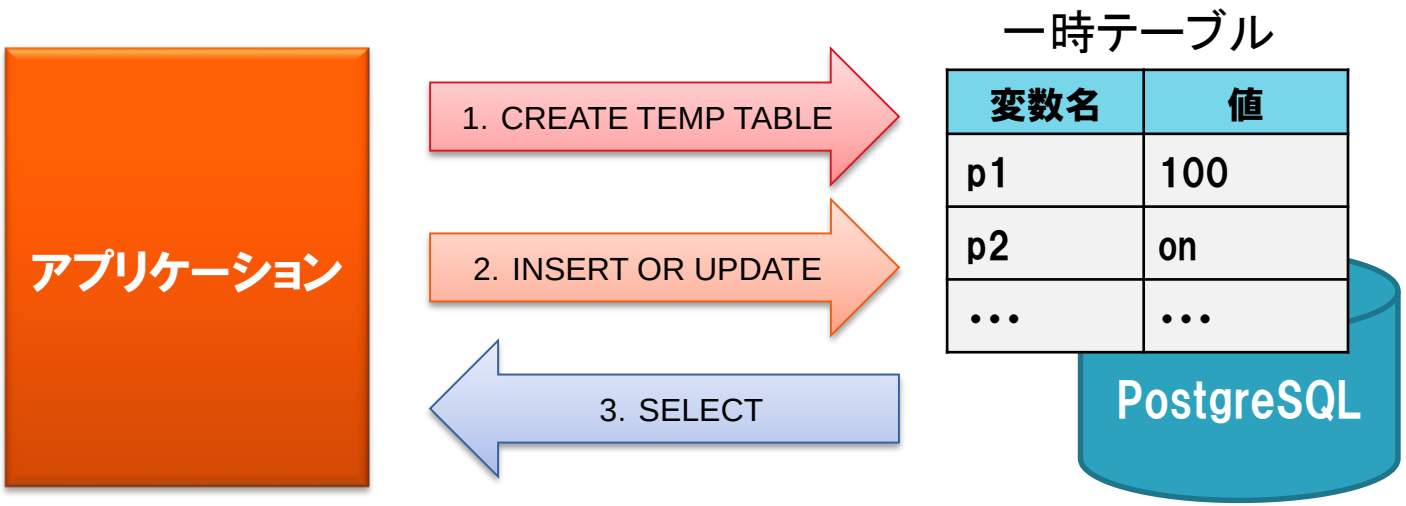
更新内容(ストアドプロシージャ移行調査編)

- PostgreSQL 10ベースで更新
- 文書のHTML化
- 書き換えの具体例の追加
- 新規ノウハウの追加
  - パッケージ変数代替
  - NO_DATA_FOUNDの取り扱い
  - 引数の取り扱い

パッケージ変数の代替

- PostgreSQLにはパッケージ変数は存在しない
  - ファンクションを跨いだ共通の変数は使用できない
- 一時テーブルを利用して代替
  1. 一時テーブルの作成（セッションごとに作成が必要）
  2. 設定値の登録・変更
  3. 設定値の取得

- セッションごとなので競合はしない
- セッション終了時にテーブル削除



NO_DATA_FOUNDの取り扱い

- 例外名称、意味は同様
  - SELECTの結果が0件

- 実際にSELECTの結果が0件になった場合の挙動
  - Oracle(PL/SQL)の場合
    - 例外
  - PostgreSQL(PL/pgSQL)の場合
    - 通常は例外にはならない
    - 例外にするには・・・
      - SELECT INTOにSTRICTオプションを付与して例外にする
      - 代入先の変数の状態(NULLかどうか)を確認して明示的に例外とする

更新内容(スキーマ移行調査編)

- PostgreSQL 9.6ベースで更新
- バージョン差分への対応
  - マテリアライズド・ビューに関する記述の変更
  - 更新可能ビューに関する記述の変更
  - 宣言的パーティショニングについては今後更新予定
- 移行ツール(Ora2Pg)に関する情報更新
  - 現在のバージョンの挙動に合わせて記載内容見直し、移行例の追記

更新内容(SQL移行調査編)

- PostgreSQL 9.6ベースで更新
- バージョン差分への対応
  - MERGE構文に対してINSERT ～ ON CONFLICTでの書き換え例を追記
- 異種DBMSとの仕様差異の追記
  - 演算の挙動
  - トランザクションの挙動

成果物の公開

- 成果物は無償でダウンロード可能
  - https://pgecons-sec-tech.github.io/tech-report/
- 年度別に公開されていた成果物を集約

移行WG（WG2）

「異種DBMSからPostgreSQLへの移行」をテーマとして調査・検証を行い、収集した技術ノウハウを成果として取り纏めた資料を公開しています。

成果物一覧

文書名	概要	リンク(HTML・PDF版)	リンク(圧縮版)	最終更新日
異種DBMSからPostgreSQLへの移行ガイド	各成果物の活用場面をイメージして頂くために、一般的なシステム移行手順を提示した上で、各タスクとWG2成果物の関係を表現しています。	・本文(HTML) ・本文(PDF)		2017/06/22
DB移行フレームワーク編	異種DBMSからの移行とは具体的に何を行うのかを紹介します。 DBMSの移行作業において一般的に発生すると考えられる作業工程を定義し、各工程における検討結果をベースとして移行可否判断の手がかりとなる情報を	・本文(PDF)	・2013年度WG2成果物(zip) ・2013年度WG2成果物	2014/04/16
異種DB間連携調査編	異種 DBMSで稼動する既存システムとの連携を想定し、異種DBMSとPostgreSQL の連携について、実現方法や移行前後における機能差などを紹介します。	・本文(PDF)	・2013年度WG2成果物(zip) ・2013年度WG2成果物(tar.bz2)	2014/04/16
スキーマ移行調査編	PostgreSQLへスキーマを移行する際に注意すべき点を調査し、異種DBMSとPostgreSQL間におけるDDL仕様の相違点や書き換えが必要なDDLの変換方法を紹介します。	・本文(PDF) ・別紙1(PDF)		2018/04/20
データ移行調査および実践編	異種DBMSからPostgreSQLへデータの移行するために必要となるデータ抽出(Extract)、変換(Transform)、およびPostgreSQLへの書き出し(Load)を中心に紹介します。また、本文書には実際にDB移行作業を実施したレポートが含まれます。	・本文(PDF) ・別紙1(PDF) ・別紙2(PDF) ・別紙3(PDF)	・2013年度WG2成果物(zip) ・2013年度WG2成果物(tar.bz2)	2014/04/16

2017年度活動報告

- 新バージョンに対する成果物の対応状況
- 移行のポイントや注意点

移行のポイントや注意点

- 実際に移行作業を経験してみると(まず感想)
- 移行作業の全体像
- 商用データベースの機能レベル差
- 商用データベースの仕様差
- PostgreSQLへ移行しやすい条件
- まとめ

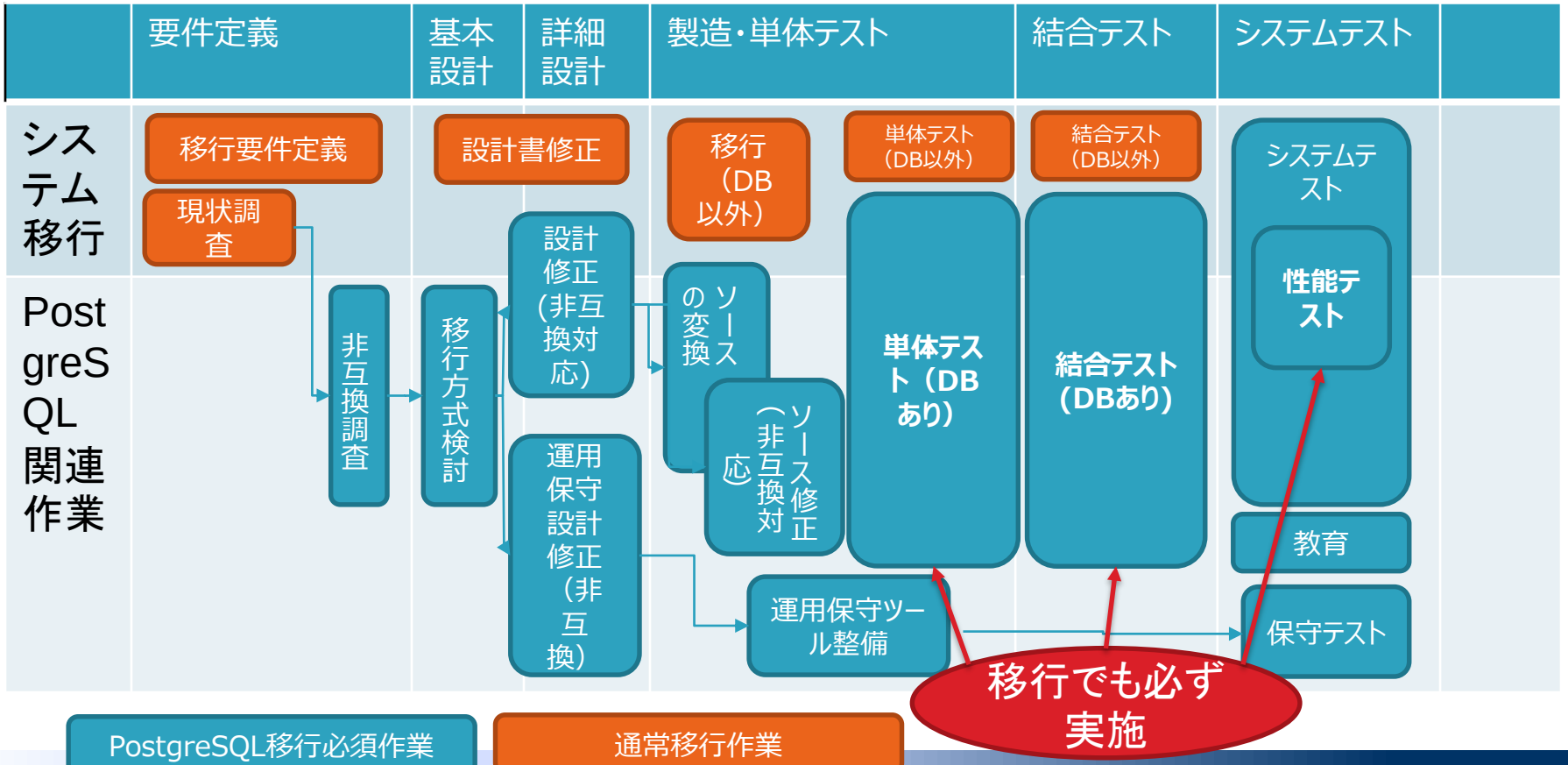
移行のポイントや注意点

- 実際に商用DBからPostgreSQLへの移行作業を経験してみると(まず感想)
  - 感心したところ
    - 意外と軽量トランザクションは商用DB同等に捌く
      (むしろ早い場合もある)
  - 苦労したところ
    - 機能はあっても移行元の商用DB同様に使えるかどうかは別
    - 思いもよらない"基本的な部分での仕様差"がある
    - 性能問題への調査、対応方法の手段が少ない
  - PostgreSQLってどう?
    - 商用DBより劣っている部分はあるが、劣っている部分の影響が小さければコスト、使用条件などで優位となるケースも多い
    - 小規模はPostgreSQLで十分(移行は移行コストによる)

移行のポイントや注意点

■ 移行作業の全体像

DB移行を含むシステム移行では以下PostgreSQL関連作業の全てを見積もる必要があります。特にテスト工程は重要です。



移行のポイントや注意点

■ 移行作業の全体像

□ 事例

事例ではテスト＋エラー修正が大半を占める。

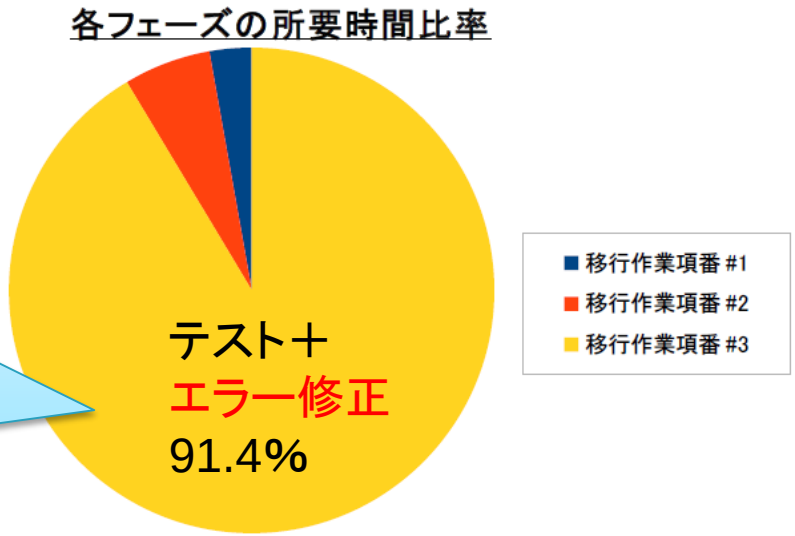
エラーの原因に着目すると

表 5.3: 移行作業の所要時間比率

移行作業項番	移行作業項目	所要時間比率
#1	PostgreSQL 用 JDBC ドライバファイルの追加 DBMS 別分岐処理への PostgreSQL 用処理の追加 PostgreSQL 用設定ファイルの追加	2.8%
#2	SQL 文の修正	5.8%
#3	SQL 文の動作確認 # テスト中に発見したエラーの修正時間も含む	91.4%

【エラーの原因】
- DB非互換（機能、仕様）の調査不足
- DB仕様差(未知）による結果異常
- 製造工程の修正ミス

※赤字は移行特有の原因



PGEConsドキュメント「アプリケーション移行実践編」より

異種DB移行は
- 非互換（機能差、仕様差（既知））を考慮しより正確な移行コストを見積もる
- 単体テストから実施→非互換（調査不足）、仕様差（未知）への対応

移行のポイントや注意点

- 商用データベースとの機能差
  - 機能の有無
    - PostgreSQLの主要機能は商用DB並み
      → 商用DBからの移行先DBの筆頭とされることが多い
  - 機能レベル
    - 個々の機能レベルでは不足を感じる部分も多い
      ただし、バージョン・アップで機能不足感は減少傾向
      ex.パラレルクエリ、パーティショニング・・
      →移行先のPostgreSQLのバージョンに依存
    - 移行では詳細に機能レベルで非互換を検証する必要がある

移行のポイントや注意点

- 商用データベースとの仕様差
  - 基本的な部分で仕様差が存在します
    - 対応にはアプリケーションの構造の見直しが必要な場合もある
    - よくあるところでは暗黙の型変換など　etc
    - 情報のない(未知な)ものもある可能性
  - WG#2の移行関連ドキュメントが参考になります
    - 既知の仕様差は記載

移行のポイントや注意点

■ 商用データベースとの仕様差(例１)

□ 除算を含む計算

例:1÷3×3＝1 、1/3 + 1/3 + 1/3＝1

Oracle	PostgreSQL

SQL> SELECT 1/3*3 結果 FROM dual
結果
--------- 	
1
 	
SQL> SELECT (1/3 + 1/3 +1/3) 結果 FROM dual
結果
--------- 	
1
	test01=# SELECT 1/3*3;
	?column?
	---------- 	
	0
	test01=# select 1.0/3.0*3.0;
	?column?
	------------------------ 	
	0.99999999999999999990
	test01=# SELECT (1/3 + 1/3 + 1/3)
	?column?
	---------- 	
	0
	test01=# select (1.0/3.0 + 1.0/3.0 + 1.0/3.0);
	?column?
	------------------------ 	
	0.9999999999999999999

移行のポイントや注意点

■ 商用データベースとの仕様差(例2)

□ トランザクションの違い(1/3)

OracleもPostgrSQLもトランザクション分離レベルのデフォルトはリードコミッティドになります。
トランザクションが正常に終了した場合であっても以下に示すように実行結果に差異がある場合もあります。

時系列	Oracle		PostgreSQL	
	トラン1	トラン2	トラン1	トラン2
1	SQL> select * from test_t; COL1 COL2 ---------- ---------- 1 1 2 1 3 2 4 2 5 3 5行が選択されました。	SQL> select * from test_t; COL1 COL2 ---------- ---------- 1 1 2 1 3 2 4 2 5 3 5行が選択されました。	test01=# select * from test_t; col1 | col2 ------+------ 1 | 1 2 | 1 3 | 2 4 | 2 5 | 3 (5 行)	test01=# select * from test_t; col1 | col2 ------+------ 1 | 1 2 | 1 3 | 2 4 | 2 5 | 3 (5 行)

2	SQL> update test_t
	set col2 = col2 + 1;
	
	5行が更新されました。
			test01=# begin;
			BEGIN
			test01=# update test_t set col2 = col2 + 1;
			UPDATE 5	

移行のポイントや注意点

■ 商用データベースとの仕様差(例2)

□ トランザクションの違い(2/3)

時系列	Oracle		PostgreSQL	
	トラン1	トラン2	トラン1	トラン2

3		SQL> delete from test_t
		where col2 = 3;
				test01=# begin;
				BEGIN
				test01=# delete from test_t
				where col2 = 3;

4	SQL> commit; コミットが完了しました。		test01=# commit; COMMIT	
5		2行が削除されました。 SQL> commit; コミットが完了しました。		DELETE 0 test01=# commit; COMMIT

移行のポイントや注意点

■ 商用データベースとの仕様差(例2)

□ トランザクションの違い(3/3)

正常終了したまったく同じトランザクションの結果が異なります。

時系列	Oracle		PostgreSQL	
	トラン1	トラン2	トラン1	トラン2
6	SQL> select * from test_t; COL1 COL2 ---------- ---------- 1 2 2 2 5 4 3行が選択されました。	SQL> select * from test_t; COL1 COL2 ---------- ---------- 1 2 2 2 5 4 3行が選択されました。	test01=# select * from test_t; col1 | col2 ------+------ 1 | 2 2 | 2 3 | 3 4 | 3 5 | 4 (5 行)	test01=# select * from test_t; col1 | col2 ------+------ 1 | 2 2 | 2 3 | 3 4 | 3 5 | 4 (5 行)

移行のポイントや注意点

■ 商用データベースとの仕様差(例3)

□ 制約チェックのタイミング

OracleとPostgreSQLでは制約のチェックのタイミングが異なります。OracleではDML文の変更実行後に制約のチェックを行いますが、PostgreSQLでは行単位にチェックが行われるため、行の物理的な格納順に依存してSQLが結果が異なるケースがあります。

```
test01=# SELECT * FROM test_tbl;
pkey      | col
--------+---------
        1 | a
        2 | aa
        3 | ABC

※pkeyがプライマリキー

【NGとなるケース】(Oracleではエラーになりません)
test01=# UPDATE test_tbl SET pkey = pkey + 1;
ERROR: duplicate key value violates unique constraint "pkey"
DETAIL: Key (pkey)=(2) already exists

【OKとなるケース】
test01=# UPDATE test_tbl SET pkey = pkey - 1;
UPDATE 3
```

移行のポイントや注意点

- PostgreSQLへ移行しやすい条件
  - 単純なRDBMSの機能だけ利用している
  - ストアドプロシジャ、パッケージを使っていない
  - 保守時間を確保できる（ 24時間運用でない）
  - 性能要件がシビアでない

PostgreSQL移行に向いています

商用データベースである必要性を再確認してみましょう。

移行のポイントや注意点

- PostgreSQLへ移行しやすい条件（詳細）
  - 移行コスト面
  - 性能面
  - 運用面

移行しやすいケース	理由
単純なRDBMSの機能だけ使用しているシステム	非互換部分が減り移行コストが削減できます。
ストアド・プロシジャ、パッケージで記述されたアプリケーションが少ない	基本的にはこの部分は作り直しとなります。このアプリケーションが多ければ移行コストが大になります。
保守時間を確保できる	定期的な索引の再構築、VACUUM FULLなど運用が容易になります。
性能要件がシビアでない	厳密な性能を要求されても、DBの稼動情報レポートなど商用DBに対し不足。またチューニング手段も限られます。シビアな場合、対応コスト増となります。

移行のポイントや注意点

- まとめ
  - 非互換機能・仕様を意識し、正確な移行コストを見積もる
    - WG#2ドキュメントを参考
  - 移行のテスト工程は単体テスト以降を見積もる
  - 移行に向かないケースも多い。以下を考慮すること
    - 移行コスト
    - 運用面
    - 性能面
  - ノウハウのあるベンダーのサポートを活用する
    - PGECons参加企業などの知見を適切に利用

おわりに

2017年度活動を振り返って

- 成果
  - WG活動において近年の大きな課題であった過去成果物の更新について着手し、新しい情報を盛り込みつつ公開することができた
  - 移行プロセスについて、移行する側寄りの視点で、改めて議論をすることができた

- 反省
  - 活動としては移行見積もりに関する検討も継続していたが、成果物としてまとめるところまでは至らず

今後の活動について

- 成果物の更新は今後も継続
  - メンテナンスが必要な点については随時検討を継続し、年度にかかわらず都度公開をできるように計画したい
- 積み残し事項への対応
  - 移行見積もりに関しての議論を継続し、成果として形にしたい
- 移行に対する懸念の解消
  - PostgreSQLの機能面は充実しており、新規開発には十分使えるが、移行となるとちょっと・・・という懸念を少しでも減らせるような活動を目指したい

皆さんもWG活動に参加してみませんか

PGECons

PostgreSQL Enterprise Consortium