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

アプリケーション移行調査編

製作者
担当企業名　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 へアプリケーション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 へアプリケーションを移行する代表的なパターンについて紹介します。また、JDBC を用いた Java アプリケーションを Oracle から PostgreSQL へ移行する際に考慮が必要となる箇所について記載します。

■本資料で扱う用語の定義

資料で記述する用語について以下に定義します。

表 1: 用語定義

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

■本資料で扱う DBMS およびツール

本書では以下の DBMS を前提にした調査結果を記載します。

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

また、本書では以下の JDBC ドライバを前提にした調査結果を記載します。

表 3: 本書で扱うJDBCドライバ

JDBC ドライバ	JDBC ドライバのバージョン
PostgreSQL	9.2 Build 1002 (JDBC4)
Oracle Database	11.2.0.3

目次

1. Oracle アプリケーションからの移行パターン

Oracle のアプリケーションを PostgreSQL に移行する場合、DBMS への接続方法、使用言語によって移行方法が異なります。以下に移行方法のパターンを整理します。

表 1: Oracle から PostgreSQL へのアプリケーション移行パターン

Oracle		PostgreSQL	
言語	**接続方式**	**言語**	**接続方式**
Java	JDBC (Type4)	Java	JDBC (Type4)
Java	JDBC (Type2)	Java	JDBC (Type4)
Java	SQLJ	-	-
C	Oracle Call Interface (OCI)	C	libpq
C	Pro*C/C++	C	ECPG
PHP	php-oci8、 PHP Data Objects (PDO)	PHP	php-pgsql、 PHP Data Objects (PDO)
Perl	DBI、DBD::Oracle	Perl	DBI、DBD::Pg
Ruby	DBI、ruby-oci8	Ruby	DBI、ruby-pg
シェルスクリプト	SQL*Plus	シェルスクリプト	psql
シェルスクリプト	SQL*Loader	シェルスクリプト	COPY

1.1. Java

Java のアプリケーションから Oracle へ接続する方式として、Java Database Connectivity(JDBC)を利用する方式と、SQLJ を利用する方式があります。

JDBC は Java でリレーショナルデータベース（RDBMS）に接続する機能を標準化した API です。JDBC の標準仕様に準拠した JDBC ドライバを利用することで、異なる RDBMS に対する操作が抽象化され、アプリケーションが RDBMS の機能に依存せずに実装することができる利点があります。

Oracle も PostgreSQL もそれぞれ JDBC ドライバが提供されていますので、JDBC の標準機能で実装された Java アプリケーションであれば、基本的に Oracle から PostgreSQL へは JDBC ドライバを変更することで移行ですますが、細かな注意点については、2 で記載します。

SQLJ は Java のプログラムに SQL 文を埋め込む方法（埋め込み SQL）を定めた ISO 標準[1]です。API である JDBC とは違い、Java 言語を拡張したものですので、SQLJ プログラムを実行するには Java プログラムをコンパイルする前に SQLJ トランスレータと呼ばれるプリプロセッサで変換する必要があります。

なお、PostgreSQL は SQLJ に対応しておらず、SQLJ を前提に書かれたアプリケーションをそのまま PostgeSQL に移行することはできません。そのため、JDBC の API を利用する形にアプリケーションを書き換える必要があります。

1.2. C

Cのアプリケーションから Oracle へ接続する方法として、Oracle Call Interface(OCI)を利用する方式と、Pro*C/C++ を利用する方式があります。

OCI はC言語で Oracle に接続する機能を提供する API です。OCI ライブラリをリンクし、ヘッダファイルをインクルードすることで、C 言語のアプリケーションから Oracle に対してデータベース操作を行うことができます。

OCIと同様にC言語のアプリケーションから PostgreSQL に接続する API として libpq がありますが、JDBC と違って API の仕様が標準化されていないため、OCI を使ったアプリケーションを PostgreSQL に移行する場合は同様の機能を実現する API に置き換える作業が必要になります。

Pro*C/C++は C 言語もしくは C++言語のプログラムに SQL 文を埋め込む方式で、Pro*C 独自のソースコードをプリプロセッサで C 言語のソースに変換してから、実行可能なプログラムにコンパイルする方式です。PostgreSQL には同様の方式として ECPG[2]があるため、ECPG の仕様に合わせて書き換えることを検討します。

1 ISO/IEC 9075-10

2 http://www.postgresql.jp/document/9.2/html/ecpg.html

1.3. PHP

PHP のアプリケーションから Oracle へ接続する方法として、php-oci8 を利用する方法と、PHP Data Objects (PDO) を利用する方法があります。

php-oci8 は PHP で Oracle に接続する機能を提供するAPIで、内部では OCI を利用して Oracle に対するデータベース操作を行います。

php-oci8 と同様に PHP のアプリケーションから PostgreSQL に接続する API として、php-pgsql があります。php-pgsql も内部では libpq を利用して PostgreSQL に対するデータベース操作を行います。

php-oci8 と php-pgsql の API に互換性はないため、php-oci8 を使ったアプリケーションを PostgreSQL に移行する場合は同様の機能を実現する API に置き換える作業が必要になります。

PDO はデータベースの違いを吸収し、共通の関数でデータベースを操作できるように設計された拡張モジュールです。データベース固有の機能や SQL の違いまでを完全に吸収できるわけではありませんが、PDO の標準 API で実装されたアプリケーションであれば、基本的に Oracle から PostgreSQL へはドライバを変更することで移行できます

1.4. Perl

Perl のアプリケーションからデータベースへ接続する場合、DBI および DBD モジュールを利用する方法が一般的です。DBI は Perl のためのデータベースアクセス用モジュールで、データベースに依存しない API を提供します。一方 DBD モジュールは DBI モジュールのためのデータベース接続用ドライバで、DBI モジュールと一緒に機能することで、データベースに対する操作を実現しています。

Oracle には DBD::Oracle、PostgreSQL には DBD::Pg という DBD モジュールが用意されていますので、基本的に Oracle から PostgreSQL へは DBD モジュールを変更することで移行できます。

1.5. Ruby

Ruby のアプリケーションから Oracle へ接続する場合、DBI および DBD モジュールを利用する方法が一般的です。DBI は Ruby のためのデータベースアクセス用モジュールで、データベースに依存しない API を提供します。一方 DBD モジュールは DBI モジュールのためのデータベース接続用ドライバで、DBI モジュールと一緒に機能することで、データベースに対する操作を実現しています。

Oracle には ruby-oci8、PostgreSQL には ruby-pg という DBD モジュールが用意されていますので、基本的に Oracle から PostgreSQL へは DBD モジュールを変更することで移行できます。

1.6. シェルスクリプト

簡単な SQL 実行を行うバッチアプリケーション等ではシェルスクリプトから Oracle のインタフェースユーティリティである SQL*Plus を直接呼び出す形で実装されるケースがあります。このケースでは、シェルスクリプトで SQL*Plus を呼び出している部分を PostgreSQL のインタフェースユーティリティである psql に置き換えることになります。

また、データベースに大量のデータを投入するバッチアプリケーション等では、シェルスクリプトから Oracle のユーティリティである SQL*Loader を直接呼び出す形で実装されるケースがあります。

このケースでは、シェルスクリプトで SQL*Loader を呼び出している部分を PostgreSQL の COPY コマンドに置き換えることになります。

2. JDBC を用いた Java アプリケーションの移行

ここでは JDBC を用いて Oracle に接続している Java アプリケーションを、PostgreSQL に移行する際に注意が必要となる点について記載します。

2.1. JDBC ドライバの種類

Oracle は「Oracle JDBC OCI Driver(Type2)」と「Oracle JDBC Thin Driver(Type4)」の 2 種類の JDBC ドライバが提供されています。

■ Oracle Call Interface(OCI)ドライバ（Type2）

OCI ドライバは Oracle Call Interface（OCI）を使用して Oracle データベースに接続するタイプの JDBC ドライバです。実行時に OCI のライブラリ（インストールディレクトリの「lib」以下）が必要で、実質的に Oracle Client がインストールさ

れているマシンでしか使用することができませんが、後述の Thin ドライバではサポートされない機能がいくつか利用できる高機能なドライバです。

■ Thin ドライバ（Type4）

Thin ドライバは Oracle のクライアントライブラリを使用せず、Net8 プロトコルを用いて直接データベースに接続するタイプの JDBC ドライバです。クラスライブラリだけで動作する事ができるので、Oracle Client がインストールされていないマシンでも動作させる事が可能ですが、前述の OCI ドライバでサポートされる機能で使えないものがあります。

一方で、PostgreSQL で提供される JDBC ドライバは Type4 のみですので、Oracle で OCI ドライバを利用している場合でも、PostgreSQL では Type4 のドライバを利用することになります。

Oracle、PostgreSQL の JDBC ドライバはともに JDBC4.0 の標準機能をサポートしていますので、JDBC の標準機能を使って実装されているアプリケーションであれば、ほとんど書き換えることなく移行することができます。

2.2. Oracle 拡張機能

Oracle の JDBC ドライバには JDBC 標準の実装を拡張する Java クラスとインタフェースが用意されており、JDBC 標準では実現されていない Oracle 拡張機能を使用することができますが、PostgreSQL に移行する場合は Oracle 拡張機能が使えなくなります。具体的には、Oracle 独自の SQL データ形式を表現するクラス群の oracle.sql パッケージ、または java.sql のインタフェースに対する Oracle 拡張機能を提供するクラス群の oracle.jdbc パッケージを直接利用しているアプリケーションは、PostgreSQL では利用できないため、書き換えが必要になります。

なお、Oracle の JDBC ドライバによる Oracle 拡張機能としては以下の様なものがあります。[3]

2.2.1. JDBC を使用したデータベース管理

JDBC メソッド startup および shutdown(oracle.jdbc.OracleConnection インタフェース内)を使用すると、Oracle Database インスタンスの起動と停止を実行できます。

2.2.2. Oracle データ型のサポート

現在のバージョン（11g リリース 2）では標準の JDBC で定義されたデータ型の利用が推奨されていますが、標準の JDBC で定義されない Oracle 独自のデータ型を使う場合、oracle.sql パッケージのクラスを使用します。

2.2.3. Oracle オブジェクトのサポート

Oracle データベース内の構造化オブジェクトの使用をサポートしています。

2.2.4. スキーマの命名サポート

Oracle オブジェクト・データ型クラスには、完全修飾スキーマ名の受入れおよび復帰を行う機能があります。

2.2.5. DML RETURNING

データ操作言語(DML)文に RETURNING 句を使用でき、2 つの SQL 文を 1 文に結合できます。

2.2.6. PL/SQL 索引付き表へのアクセス

JDBC アプリケーションで、索引付き表パラメータを使用して PL/SQL をコールできます。

2.2.7. OCI 接続プーリング

OCI ドライバでは接続プーリングの機能を提供します。この機能によって、アプリケーションは少数の物理接続を使用して、複数の論理接続を保持できます。PostgreSQL の JDBC ドライバには接続プーリングの機能はありませんが、Tomcat, JBoss といったアプリケーションサーバや pgpool-II[4]といったミドルウェアで接続プーリングの機能

3 Oracle Database JDBC 開発者ガイド
（http://docs.oracle.com/cd/E16338_01/java.112/b56281/oraint.htm#g1090378、
http://docs.oracle.com/cd/E16338_01/java.112/b56281/instclnt.htm#CHDCJEDD）より引用

4 http://www.pgpool.net/docs/latest/pgpool-ja.html

は実現可能です。

2.2.8. クライアント結果キャッシュ

OCI ドライバでは SQL 問合せ結果セットをクライアント側のメモリーへキャッシュすることが可能となり、繰返しの問合せでのレスポンス時間を改善することができます。PostgreSQL の JDBC ドライバにはクライアント結果キャッシュの機能はありませんが、pgpool-II のようなミドルウェアを組合せることで同様のクエリキャッシュ機能が実現可能です。

2.2.9. 透過的アプリケーション・フェイルオーバー

OCI ドライバでは接続先のデータベース・インスタンスがダウンした場合でも、データベースに自動的に再接続できる機能が提供されます。PostgreSQL の JDBC ドライバにはアプリケーション・フェイルオーバーの機能はありませんが、pgpool-II のようなミドルウェアを組合せることで同様のフェイルオーバー機能が実現可能です。

2.2.10. OCI ネイティブ XA

OCI ドライバにはネイティブ XA と呼ばれる機能が用意されています。この機能により、ネイティブ API を使用して XA コマンドを送信できます。

2.3. JDBC ドライバのデータベース接続文字列の違い

JDBC の標準機能を使って実装されているアプリケーションであれば、ほとんど書き換えることなく移行することができますが、JDBC ドライバのクラス名、データベース接続時の URL 文字列は DBMS に依存しますので、変更が必要になります。

【Oracle データベースへの接続方法】

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection conn=DriverManager.getConnection("jdbc:oracle:oci8:@oracle.techscore","scott","tiger")
```

【PostgreSQL データベースへの接続方法】

```
Class.forName("org.postgresql.Driver");
Connection conn = DriverManager.getConnection("jdbc:postgresql://localhost/test?
user=fred&password=secret&ssl=true");
```

ただし、このようなデータベース接続用の情報はアプリケーションのソースコード上に定義されることは稀で、アプリケーションが利用するフレームワークやアプリケーションサーバの設定ファイル上に定義するケースが一般的ですので、この場合は設定ファイルの変更のみで対応できます。

2.4. マッピングされるデータ型の違い

2.4.1. Date 型の取扱い

日付を表す Date 型は、Oracle では日付と時刻を両方保有します。一方で PostgreSQL では日付のみを保有し、時刻は保有しません。[5]この違いが JDBC がデータベースから Java オブジェクトへマッピングする際の挙動の違いとなって現れることがあります。

Oracle では、Date 型で保有している時刻データを失わないため、デフォルトで Date 型は java.sql.Timestamp にマッピングされます。そのため、データベースの Date 型のデータを取り出した後、Java アプリケーションで時刻を保有していることを前提とした実装がされている場合があります。

一方で PostgreSQL はデータベースの Date 型を java.sql.Date にマッピングし、時刻データを持たない前提ですので、Oracle で使用していた「時刻を保有している前提」のアプリケーションはそのまま PostgreSQL で稼動しないことがあり得ます。

5　時刻のみを表現する場合は time 型、日付と時刻を両方表現する場合は timestamp 型を使います。

2.5. トランザクションの取扱い

2.5.1. DDL(Data Difinition Language)の暗黙的コミット

Oracle では CREATE TABLE や DROP TABLE 等の DDL 文を実行した場合、明示的に COMMIT を実行しなくてもトランザクションがコミットされます（暗黙的コミット）が、PostgreSQL では暗黙的コミットが行われません。

そのため、DDL 文を実行するアプリケーションがある場合、PostgreSQL で実行するアプリケーションには明示的に COMMIT を実行する処理を追加する必要があります。

2.5.2. エラーが発生した場合の振る舞い

複数の DML(Data Manipulation Language)文を実行するトランザクションの途中でエラーが発生した場合、Oracle では COMMIT を実行することができ、正常に実行できた DML 文の処理を確定することができます。

一方、PostgreSQL ではトランザクションの途中でエラーが発生すると、それ以降の DML 文は無条件にエラーとなり、トランザクションを ROLLBACK することしかできなくなります。

そのため、PostgreSQL にアクセスする Java アプリケーションでは、DML を実行する際の例外処理を見直す必要があります。

2.6. エラーメッセージ、エラーコードの違い

JDBC を利用した Java アプリケーションでは、データベースで発生した場合のエラーは例外クラス java.sql.SQLException として throw されます。Java アプリケーションではこの SQLException を catch し、内部にセットされたエラーコード、エラーメッセージを取り出して、エラーハンドリングを行います。

ただし、Oracle と PostgreSQL では返されるエラーコード、エラーメッセージが異なるため、Oracle のエラーコード, エラーメッセージを前提に実装されている部分は、PostgreSQL のエラーコード、エラーメッセージに合わせて書き換える必要があります。

著者

版	所属企業・団体名	部署名	氏名
アプリケーション移行調査編 1.0 (2012 年度 WG2)	TIS 株式会社	戦略技術センター	中西 剛紀