Workshop

Oracle to Postgres Migration

Part 1 - migrating the database

2016-05-30 @IDM

Chris Mair

http://www.pgtraining.com

The Workshop

- part 1
  - Oracle Database ™
  - how an application interacts with an RDBMS
  - the ora2pg tool
- part 2
  - PostgreSQL features for DBAs and developers

Important Note

- Postgres is an advanced Open Source RDBMS with roots in the '80s at UC Berkeley. It has been developed as a comunity project since 1996

- the project's goal is to develop the most advanced Open Source enterprise-class RDBMS. Notably, its SQL implementation strongly conforms to the ANSI-SQL:2008 standard

- Postgres has never been, is not, and never will be some sort of drop-in replacement for Oracle Database ™ - if you need such a thing, you need to look elsewhere

Getting to know Oracle Database ™

Get your own Instance

- the two quickest ways I know to get your own instance running are:
  - Oracle Database **Express Edition** ™ (running on your own Linux server → "Oracle XE"
  - Oracle Database **Standard Edition** One ™ as a service on AWS → "Oracle SE"
- other approaches need more investment in time (the installation takes some care) or money (licensing)

Oracle XE on own Server

- Oracle XE is **gratis**-ware (read the exact license terms, though)
- product is **limited** to using 1 core, 1 GB RAM and up to **11 GB** user data - the machine can (and should) be larger, however
- you need a (free) OTN account to download it
- I recommend running it on CentOS
- download version 11.2 (~ 300 MB .rpm) **here**

Oracle SE One on AWS

- many RDBMS can be run as a managed service on Amazon Web Services (AWS) - among others also Postgres and Oracle Database ™

- Oracle SE One 11.2 or 12.1 can be run in a very interesting "license included model": the license cost is included in the AWS cost and **charged by the hour** used (depending on machine size, the extra cost starts at ~ 2 cent / h more than for a free RDBMS)

- this product is still limited by the SE One license in socket count, so AWS can offer it only with at most 16 logical cores per instance, but there is no other (low) limit in RAM or disk size

- see the AWS website for **details**

Manuals

- Oracle offers two "easy" manuals **here** for Oracle XE that can be a starting point:
  - 2 Day DBA
  - 2 Day Developer's Guide
- depending on you background be prepared to do some reading...
- generally speaking, being a DBA in the Oracle world can be challenging - fortunately you don't need the full DBA-foo if you're just interested in supporting migrations and/or are using XE or SE as a service

Tools

- Oracles traditional command line tool is called **SQL*Plus**, it comes bundled with Oracle XE or can be downloaded with the so called "instant client" package (more later)

- the "official" GUI tool is **Oracle SQL Developer**: this is a JAVA application that can be downloaded **here** (~ 300 MB)

- both are gratis-ware and are available for Linux (and other OSes too)

Getting to know Postgres

Get your own Instance 1/2

- Postgres is in the standard repo of all major Linux distribution, so that's just one of the usual commands away:

```
yum install postgresql postgresql-server
```

```
apt-get install postgresql-client postgresql
```

- however, the major Linux distributions often carry pretty outdated versions (I'm looking at you, Centos 7, carrying version 9.2, that is **three major versions behind** the current 9.5) or package the product in unusual ways (I'm looking at you, Debian)

Get your own Instance 2/2

- I recommend getting more recent releases from the official **download page**

- the project maintains its own repositories for all major Linux distributions, so you can still use yum, apt, etc. to install and update everything - note that Postgres issues a major version about once a year and supports it for 5 years

- it is pretty much possible to compile from source too (I even prefer to do so, although I'm not recommending this to new users)

- as mentioned, Postgres is also available as a service on AWS

Manuals

- the official (huge) manual is available **here** for reading online or downloading as PDF
- many good books are available - please pay attention to the publication date, you probably should pick something from 2012 or newer (covering version 9.1 or newer)
- the second part of this workshop will deal with Postgres, attending it is a very good way to getting started :)

Tools

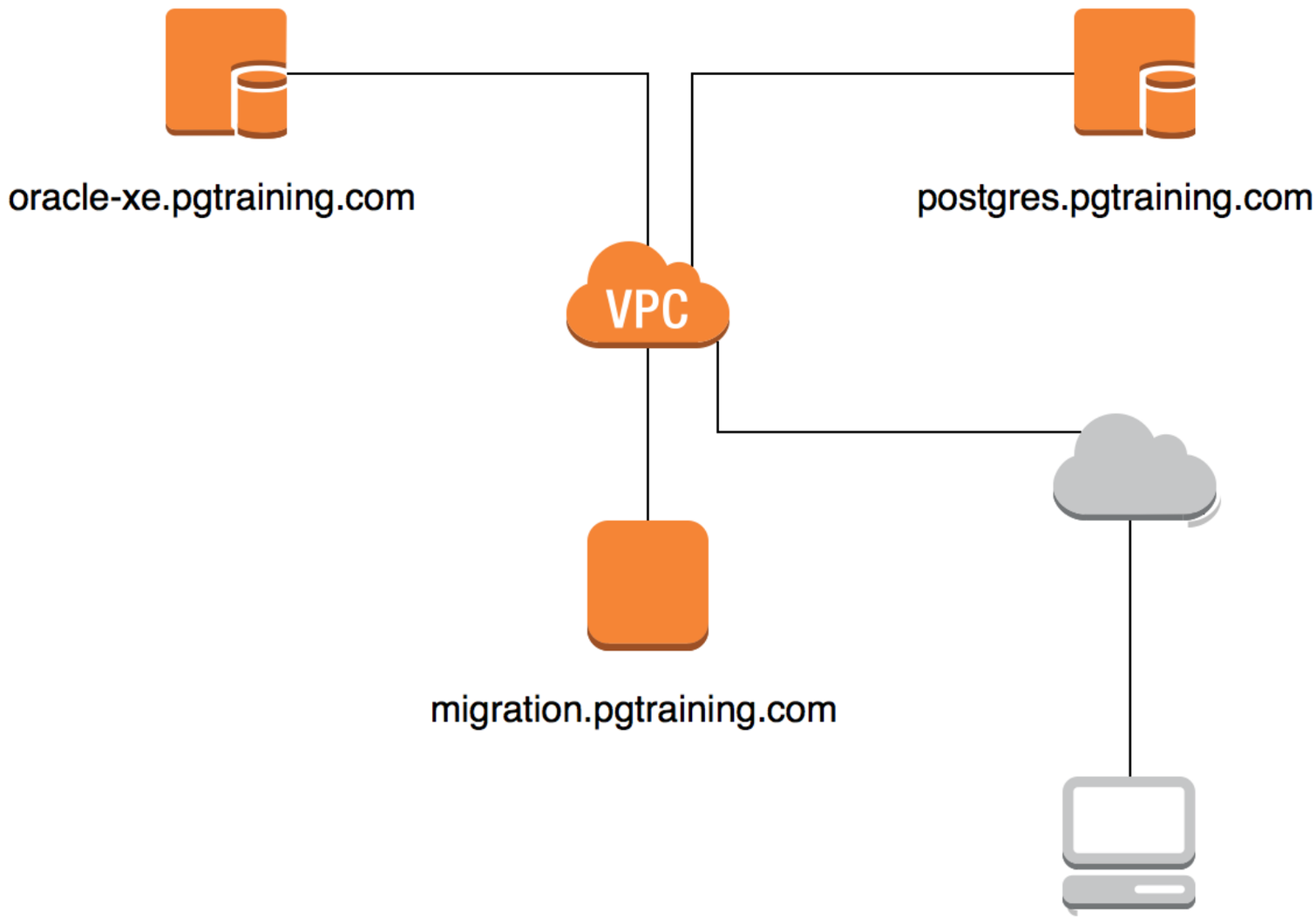
- Postgres' traditional command line tool is called **psql** - again, this can be installed from your Linux distribution's repo or from the Postgres repo

- there is no "official" GUI tool, though!

- some people use pgAdmin (Open Source), some people use propretiary multi-product tools, some people just use psql

- I recommend sticking with psql

Hands-on Setup



Talking with a RDBMS 1/3

- when using a RDBMS you don't deal with files, but rather use a client or client application that comunicates with the RDBMS over a network socket

- each product has its own native networking protocol

- Oracle uses the proprietary "TNS" protocol with the default TCP port 1521 and the OCI library as native call interface

- Postgres uses its own "frontend/backend" protocol with the default TCP port 5432 and the libpq library as native call interface

- these two sets of protocols / libraries have nothing whatsoever in common

Talking with a RDBMS 2/3

- the structured query language (SQL) is used to "instruct" a RDBMS
- it is used to define the database schema (DDL statements), to manipulate the data (DML statements) and to query the data
- information about the database schema is stored as data in the "catalog", so SQL can be used to query the catalog too
- there is an ANSI SQL standard that covers many, but not all aspects

Talking with a RDBMS 3/3

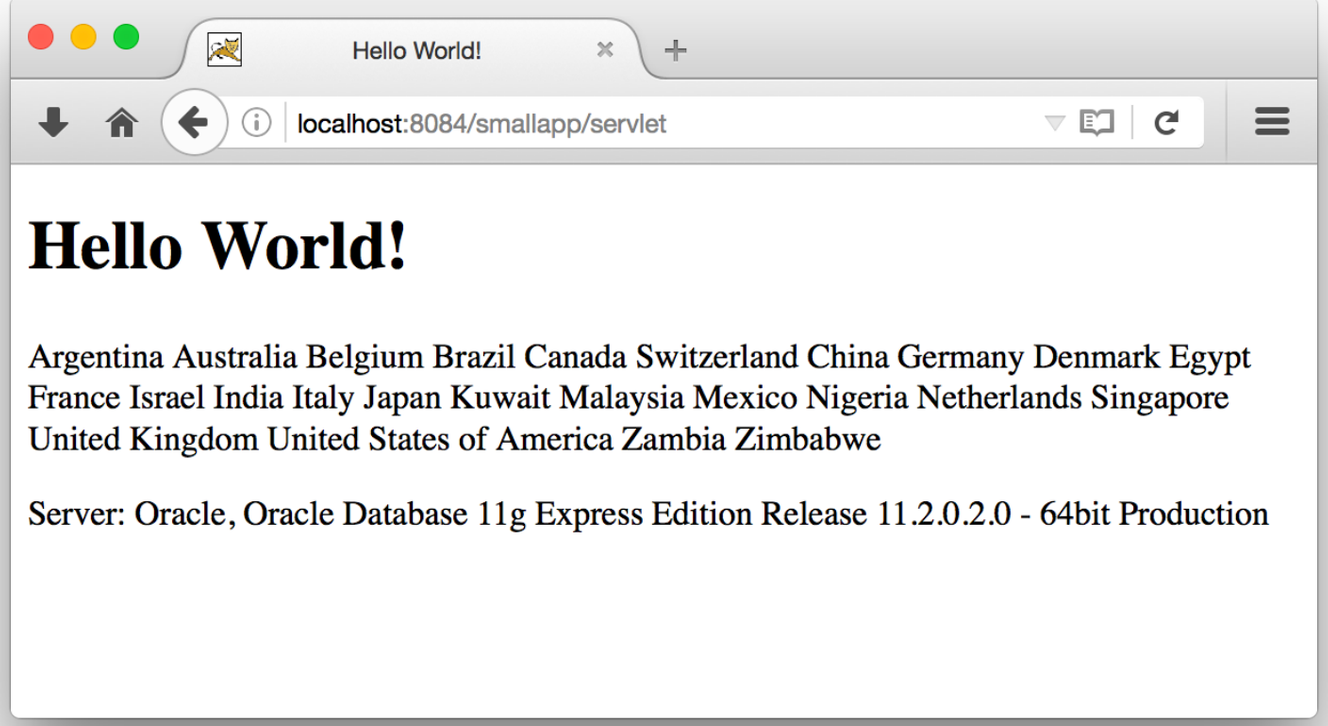
- fortunately most applications written for Oracle Database ™ don't use TNS/OCI natively but send SQL statements over a higher level API such as ODBC (Win/C++), JDBC (Java), the .NET data provider API (C#), etc.

- drivers for these APIs are (of course) available for Postgres too

- hence, **ideally**, migrating an application that talks SQL over - say - JDBC is not too hard ... ideally...

Example App

- a super simple web app written in Java: it displays the countries from the "hr"-sample database



Example App - Code

```
final String driver = "oracle.jdbc.OracleDriver";
final String db_url = "jdbc:oracle:thin:hr/hr@oracle-xe.pgtraining.com:1521/xe";

try {
    Class.forName(driver);
} catch (ClassNotFoundException ex) {
    System.err.println("Error: unable to load driver class!");
}

ArrayList<String> list = new ArrayList<>();
String ver = "";
try {
    Connection conn = DriverManager.getConnection(db_url);
    ver = conn.getMetaData().getDatabaseProductName() + ", " +
          conn.getMetaData().getDatabaseProductVersion();
    Statement st = conn.createStatement();
    ResultSet res = st.executeQuery("select country_name from countries");
    while (res.next()) {
        list.add(res.getString(1));
    }
    res.close();
    st.close();
    conn.close();
} catch (SQLException e) {
    System.err.println("Error: unable to run query");
}
```

JDBC-API calls

SQL

JDBC driver and connection info

Can this run on Postgres?

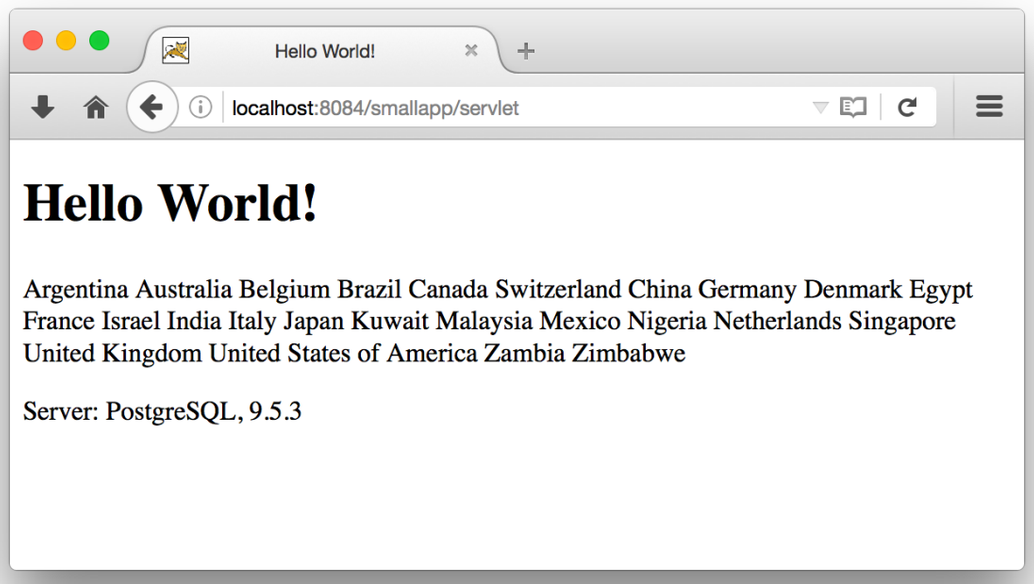
- yes! we need to:
  - create the schema (table countries) in Postgres
  - copy over the data
  - change the JDBC driver and connection info
- let's do that (manually exporting the table, using SQL Developer)...

With some work...

- the app was easy:

```
final String driver = "org.postgresql.Driver";
final String db_url = "jdbc:postgresql://postgres.pgtraining.com/hr?user=hr&password=hr";
```

- the data too, but the schema was not (data types are not really portable: varchar2 vs varchar, case sensitivity of quoted identifiers, ...)



We need a Tool!

- **ora2pg** by Gilles Darold comes to the rescue
- ora2og connects to Oracle and dumps schema and data in a Postgres-compatible format; it is highly configurable and can even connect to Postgres and migrate everything on the fly
- ora2pg reads Oracle's catalog and knows how to create the equivalent Postgres objects (tables, views, sequences, indexes), with unique, primary, foreign key and check constraints without syntactic pitfalls
- let's install ora2pg on our third machine ("migration")

Installing ora2pg 1/4

- we need to install the Oracle instant client rpms from **here** - get these three files:

```
oracle-instantclient12.1-basic-12.1.0.2.0-1.x86_64.rpm
oracle-instantclient12.1-devel-12.1.0.2.0-1.x86_64.rpm
oracle-instantclient12.1-sqlplus-12.1.0.2.0-1.x86_64.rpm
```

- and set the environment:

```
export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib:$LD_LIBRARY_PATH
export PATH=/usr/lib/oracle/12.1/client64/bin:$PATH
export ORACLE_HOME=/usr/lib/oracle/12.1/client64
```

- let's try connecting to the Oracle server:

```
sqlplus hr/hr@oracle-xe.pgtraining.com
```

✓

Installing ora2pg 2/4

- analogously, we want to install the Postgres client:

```
yum install postgresql
```

- and test it by connecting to the Postgres server:

```
psql -h postgres.pgtraining.com -U hr hr
```

✓

(note all these machines are firewalled, you do not want to leave ports 1521 and 5432 open to the world, especially not with such silly passwords)

Installing ora2pg 3/4

- then we need to install a few Perl modules, among which is DBD::Oracle - as it's not distributed by Red Hat, we need to download it from CPAN and build it:

```
yum install perl-DBI perl-DBD-Pg perl-ExtUtils-MakeMaker gcc
wget http://search.cpan.org/CPAN/authors/id/P/PY/PYTHIAN/DBD-Oracle-1.74.tar.gz
tar xf DBD-Oracle-1.74.tar.gz
cd DBD-Oracle-1.74
perl Makefile.PL -l    # watch out for missing Perl modules
make
make install
```

Installing ora2pg 4/4

- finally we can install ora2pg:

```
wget https://github.com/darold/ora2pg/archive/v17.4.tar.gz
tar xf v17.4.tar.gz
cd ora2pg-17.4
perl Makefile.PL      # watch out for missing Perl modules
make
make install
```

- and verify that it is installed:

```
/usr/local/bin/ora2pg -v
```

Configuration

- a template configuration file is installed in

  `/etc/ora2pg/ora2pg.conf.dist`

- there are **way** to many parameters to list here, see the **ora2pg documentation**...

```
ORACLE_HOME     /usr/lib/oracle/12.1/client64
ORACLE_DSN      dbi:Oracle:host=oracle-xe.pgtraining.com;sid=xe
ORACLE_USER     hr
ORACLE_PWD      hr
USER_GRANTS     1
SCHEMA          hr
TYPE            TABLE,VIEW,PROCEDURE,TRIGGER
DROP_FKEY       1
```

Run 1

- ora2pg-schema-to-file.conf → dump schema to output.sql

```
/usr/local/bin/ora2pg -c ora2pg-schema-to-file.conf
[=========================>] 7/7 tables (100.0%) end of scanning.
[=========================>] 7/7 tables (100.0%) end of table export.
[=========================>] 1/1 views (100.0%) end of output.
[=========================>] 2/2 procedures (100.0%) end of output.
[=========================>] 1/1 triggers (100.0%) end of output.

psql -h postgres.pgtraining.com -U hr hr
Password for user hr:
psql (9.2.15, server 9.5.3)
WARNING: psql version 9.2, server version 9.5.
         Some psql features might not work.
SSL connection (cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256)
Type "help" for help.
hr=> \i output.sql
```

Run 2

- ora2pg-all-to-file.conf → dump schema **and data** to output.sql

```
/usr/local/bin/ora2pg -c ora2pg-all-to-file.conf
[=========================>] 7/7 tables (100.0%) end of scanning.
[=========================>] 7/7 tables (100.0%) end of table export.
[=========================>] 1/1 views (100.0%) end of output.
[=========================>] 2/2 procedures (100.0%) end of output.
[=========================>] 1/1 triggers (100.0%) end of output.

[=========================>] 25/25 rows (100.0%) Table COUNTRIES (25 recs/sec)
[=========================>] 27/27 rows (100.0%) Table DEPARTMENTS (27 recs/sec)
[=========================>] 107/107 rows (100.0%) Table EMPLOYEES (107 recs/sec)
[=========================>] 19/19 rows (100.0%) Table JOBS (19 recs/sec)
[=========================>] 10/10 rows (100.0%) Table JOB_HISTORY (10 recs/sec)
[=========================>] 23/23 rows (100.0%) Table LOCATIONS (23 recs/sec)
[=========================>] 4/4 rows (100.0%) Table REGIONS (4 recs/sec)
[=========================>] 215/215 rows (100.0%) on total
                            estimated data (1 sec., avg: 215 recs/sec)
```

```
psql -h postgres.pgtraining.com -U hr hr
Password for user hr:
psql (9.2.15, server 9.5.3)
[...]
hr=> \i output.sql
```

Post-Migration Testing...

```
hr=> select * from job_history;
 employee_id |     start_date      |      end_date       |   job_id   | department_id
-------------+---------------------+---------------------+------------+---------------
         102 | 2001-01-13 00:00:00 | 2006-07-24 00:00:00 | IT_PROG    |            60
         [...]
         200 | 2002-07-01 00:00:00 | 2006-12-31 00:00:00 | AC_ACCOUNT |            90
(10 rows)

hr=> update employees set job_id = 'ST_MAN' where employee_id = 100;
UPDATE 1

hr=> select * from job_history;
 employee_id |     start_date      |         end_date          |   job_id   | department_id
-------------+---------------------+---------------------------+------------+---------------
         102 | 2001-01-13 00:00:00 | 2006-07-24 00:00:00       | IT_PROG    |            60
         [...]
         200 | 2002-07-01 00:00:00 | 2006-12-31 00:00:00       | AC_ACCOUNT |            90
         100 | 2003-06-17 00:00:00 | 2016-05-30 03:38:32.7629  | AD_PRES    |            90
(11 rows)
```

PL/SQL vs PL/PgSQL

- ora2pg can automatically translate only very simple procedures or functions (as the trigger examples in the hr demo schema)

- more complicated procedures need to be translated **manually**

- fortunately, PL/PgSQL is easy and rich in features: in my experience typical procedures can be rewritten without too much hassle **most** of the times

- some people suggest getting rid of procedures alltogether when migrating (move them to the middle tier) - I tend to find it easier to just translate them!

Scenario 1: Bad

- application is linked against OCI / uses TNS natively (like the SQL*Plus executable)
- if the source code is not available, a migration is not possible
- if the source code is available and can be changed, at the very least the database access layer needs to be rewritten from scratch (using libpq)

Scenario 2: Bad

- application uses some proprietary software or component such as Oracle Forms ™
- an easy migration is generally not possible

Scenario 3: so-so

- application uses ODBC, JDBC, etc. - **however**, it uses lots of queries with Oracle custom-syntax (such as (+) instead of outer joins, mixes quoted and unquoted identifiers, uses lots of Oracle specific functions etc.) and has long and complex PL/SQL procedures

- if the source code is not available, a migration is not possible

- if the source code is available, and can be changed, queries and procedures need to be translated manually

Scenario 4: good

- application uses ODBC, JDBC, etc. - **also**, it either uses no or very few Oracle custom-syntax and PL/SQL procedures or the vendor has made a version for Postgres backends available

- application that use abstractions layers such as ORM-wrappers usually fall into this category

- a migration is normally not too hard (unless the software is completely locked-down and does not even allow to select the ODBC, JDBC, etc. driver)

What is you scenario?

'k thx bye ;)