Database Migration: Challenges of Migration from Oracle to Open Source

European Bioinformatics Institute
Maurizio De Giorgi, Database Team
www.ebi.ac.uk
Agenda

Introduction
The project
Technical challenges and adopted solutions
Tools
Lesson learned
Conclusions
Q&A
What is EMBL-EBI?

- Europe’s home for biological data services, research and training
- Part of the European Molecular Biology Laboratory, an intergovernmental research organization, non-profit
- Second largest of the six EMBL sites
- International: 600 members of staff from 57 nations
- Home of the ELIXIR Hub - a research infrastructure for life science
EMBL-EBI Databases Team

- Technical Services Cluster provides central IT support to over 50 “Customer” Teams at EMBL-EBI
  - ~ 750 database instances
  - ~ 800 TB of data
  - Commercial: Oracle, MS SQL Server, Vertica
  - Open Source: MySQL, PostgreSQL, MongoDB, Graph
  - Concerns over large exposure to a feature rich but expensive commercial database
The project

Goal: Reduce the overall Oracle footprint
Target: ~30 Oracle instances
Timeline: 1st phase Jan 2016 - July 2017
Resource: 100% FTE

but also provide:

- technical understanding of porting challenges
- methodology for future porting
- lessons learned (incl. best choice of DBMS)
The Oracle Usage Survey 2014

Identify and map:

- Teams, Users and Services using Oracle
- Number, size and importance of each database
- What features were in use and how critical they were
  - Teams/DB, DB/Features, DB/Users matrices
- Complexity, language and size of the code base
- Relationships among databases and external parties
- Release process, deployment model
- Level of activity, criticality
- Issues
Migrating ground & low hanging fruits

- evaluate: features, returns, activity and criticality, effort (complexity, code base, links, data size, dest. tech.)
- increase success rate, reduce potential damage & stress
- build-up momentum and experience incrementally
- maximize results, minimize effort

<table>
<thead>
<tr>
<th>DB</th>
<th>Returns</th>
<th>Features</th>
<th>Critical</th>
<th>Activity</th>
<th>Willingness</th>
<th>Effort</th>
<th>Code</th>
<th>Size GB</th>
<th>Complexity</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>1</td>
<td>H/H</td>
<td>High</td>
<td>Low</td>
<td>Medium</td>
<td>Small</td>
<td>41</td>
<td>Medium</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>4</td>
<td>M/H</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Small</td>
<td>12560</td>
<td>Low</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>1</td>
<td>L/H</td>
<td>High</td>
<td>High</td>
<td>Large</td>
<td>Small</td>
<td>100</td>
<td>Low</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>2</td>
<td>L/M</td>
<td>Large</td>
<td>Medium</td>
<td>Medium</td>
<td>Large</td>
<td>8368</td>
<td>Low</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>3</td>
<td>L/H</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>Small</td>
<td>295</td>
<td>Low</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
<td>2</td>
<td>M/H</td>
<td>Medium</td>
<td>High</td>
<td>Large</td>
<td>Small</td>
<td>8</td>
<td>Low</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>4</td>
<td>H/H</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
<td>Small</td>
<td>105</td>
<td>Medium</td>
<td>3</td>
</tr>
<tr>
<td>H</td>
<td>4</td>
<td>1</td>
<td>M/M</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
<td>Large</td>
<td>10</td>
<td>Low</td>
<td>4</td>
</tr>
<tr>
<td>I</td>
<td>6</td>
<td>3</td>
<td>L/M</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
<td>143</td>
<td>Medium</td>
<td>2</td>
</tr>
</tbody>
</table>
Access to/from Oracle in PostgreSQL

• Access to Oracle from Pg - **oracle_fdw** [Laurenz Albe]
  [https://github.com/laurenz/oracle_fdw/issues/99](https://github.com/laurenz/oracle_fdw/issues/99)
  new options (prefetch '1-10240', sample_percent '1-100')
  • elapsed time reduced by 50-60% in test cases
  • **CTAS performance ~comparable with oracle to oracle**
  • caveat: **variable push down/cross joins are poor** (so far)
  • workaround: push/get data into pre-allocated tables via fdw

• Access to Pg from Oracle - **odbc** (last resort solution)
  configuration, troubleshooting & performance not exceptional

• Substitute for Oracle Export (ad hoc) [Boris Bursteinas]
  generate DDL/CTL, CSV (java API copy manager) -> sql loader
Dealing with 3rd party DB

- **Jira**: Migrating JIRA's data to a different type of database server
- **FishEye**: Migrating to an external database, Migrating to PostgreSQL
- **RT4**: `rt-validator --check` && `rt-serializer`, `rt-importer`

All of the above with some effort worked well enough, RT4 required more effort and specific initial loading pg conf (wal minimum)

- **Confluence**: vendor procedure Migrating to Another Database has documented **limitations** that made it unsuitable for our case (size >500MB, unsupported character set), used `ora2pg`, export to file, encoding conversion, import to pg
Loading files: POC

CSV file ~2.4M rec. load table with 4 varchar columns (50-255) pgloader --root-dir .../reports/ --logfile pgloader.log ...

1. drop/create table in BEFORE LOAD => 12.192s
   a. no indexes/constraints exist
2. drop/create table in BEFORE LOAD => 28.417s
   a. indexes/constraints creation in AFTER LOAD
3. truncate table (indexes/constraints in place) => 57.497s
   a. default indexes/constraints maintenance during copy
4. truncate+drop indexes => 46.208s
   a. indexes/constraints dropped in BEFORE LOAD
   b. indexes/constraints parallel creation in AFTER LOAD
Loading files: multiple files in parallel

- Load ~6000 CSV files in parallel
  LOAD CSV FROM all filenames matching ~<(.*).csv>
- Tuning params to balance performance/resource consumption can require some time/effort
  WITH truncate,
    batch rows = 500, batch size = 32MB, prefetch rows = 500,
    workers = 2, concurrency = 1
  **Total import 16529101 rows in 39m41.322s**
- When hitting memory limits: rebuild from source
  - [http://pgloader.io/download.html](http://pgloader.io/download.html)
  - simple when using bootstrap script
  - make DYNSIZE=8192 pgloader
- disable/enable autovacuum on table before/after load
Porting PL/SQL to PL/pgSQL: differences

1. **No concept of package ⇒ no globals**, in oracle packages are widely used and often a good practice.

2. **A procedure is part of an “outer” transaction**.

3. As a consequence **no embedded commit is allowed** while in oracle you can do as you need.

4. **DDL statements are transactional** instead are preceded/followed by an implicit commit in Oracle.

5. **Embedded SQL statements are not “visible” at run time** like normal SQL statements are.

6. **Incompatible syntax** (some SQL + oracle supplied packages and functions).
Porting PL/SQL to PL/pgSQL: an example

-- PL/SQL
prepare_releases(p_release_type);
FOR v_load IN c_load LOOP
  move_staging_data(p_in_dbid => v_load.dbid);
  load_release(p_in_dbid => v_load.dbid, p_in_load_release => v_load.id);
END LOOP;

verify_xref_id_not_null(); -- just keep in mind this function for later

-- PL/pgSQL
perform rnc_update.prepare_releases(p_release_type);
FOR v_load IN c_load LOOP
  perform rnc_update.move_staging_data(p_in_dbid => v_load.dbid);
  perform rnc_update.load_release(p_in_dbid => v_load.dbid, p_in_load_release => v_load.id);
END LOOP;
Porting PL/SQL to PL/pgSQL: issues

Initial troubleshooting of errors, or mis-behaving queries, happening deep in the calls stack of a large, long running transaction (perhaps containing DDL) is difficult because:

• transaction is rolled back after the error
• actual data/structures needed are not there
• embedded SQL statements execution is not shown
• auto_explain, pg_stat_statement are quite useful but only for statements that managed to complete!
Porting PL/SQL to PL/pgSQL: syntax

-- PL/SQL
MERGE INTO rnc_ref_map t1 USING (SELECT t3.acc,t3.div,t4.id FROM load_rnc_refs t3, rnc_refs t4 WHERE t3.md5 = t4.md5) t2
  ON (t1.acc = t2.acc AND t1.ref_id = t2.id)
WHEN MATCHED THEN UPDATE SET t1.div=t2.div
WHEN NOT MATCHED THEN INSERT ( t1.acc,t1.div,t1.ref_id) VALUES ( t2.acc,t2.div,t2.id);

-- PL/pgSQL
insert into rnc_ref_map as t1 (acc, div, ref_id)
select t3.acc, t3.div, t4.id from load_rnc_refs t3 join rnc_refs t4 on (t3.md5 = t4.md5)
on conflict (acc, ref_id)
do update set div=excluded.div;

Plenty of info sources to deal with incompatible syntax. MERGE statements can be converted to INSERT ON CONFLICT but some effort and tuning might be needed to refactor some complex query.
Porting PL/SQL to PL/pgSQL: update not HOT

-- PL/SQL originally in `verify_xref_id_not_null()`
UPDATE xref SET id = XREF_PK_SEQ.nextval WHERE id IS NULL;

-- PL/pgSQL as resulting after conversion
UPDATE xref SET id = nextval('xref_pk_seq') WHERE id IS NULL;

The Heap Only Tuple (HOT) feature eliminates redundant index entries and allows the re-use of space taken by DELETEd or obsoleted UPDATEd tuples without performing a table-wide vacuum. It does this by allowing single-page vacuuming, also called "defragmentation".

-- to monitor HOT updates vs total updates
select n_tup_upd, n_tup_hot_upd from pg_stat_user_tables;
Partitioning & PEL

- ora2pg can translate list/range partitioning well enough
  - initial issue 18.1 list/list subpartitions wrongly translated
    - fixed: https://github.com/darold/ora2pg/issues/334
  - always aim at using the latest ora2pg version
  - prepare alternative machine to install/test ora2pg fixes
- using inheritance, check constraints, triggers (9.5.X)
  - it is limited but not all bad, some interesting aspects
- extensions: pg_partman time/serial based partitioning
- pg 10 declarative partitioning (reduce manual steps)

- PEL need a substantial rewrite
  execute 'alter table xref_p' || p_in_db_id || '_old no inherit xref';
  execute 'alter table xref_p' || p_in_db_id || ' inherit xref';
- testing and troubleshooting need time/effort
- do not forget: SET constraint_exclusion = on;
Porting PL/SQL to PL/pgSQL: solutions?

1) `package⇒schema: pkg.proc⇒schema.proc`
   user defined custom variables (defaults in postgresql.conf) or temp. tables for globals

2) rewrite as **1 proc ⇔ 1 trans when possible**, and/or
   invoke or combine the procedures with launcher or
   wrapper in python/other language

3) insert **commits where required in the wrapper**

4) [to replicate oracle behavior commit before/after DDL]

5) extend **code instrumentation**

6) many examples on the web, use orafce or other
   commercially available compatibility modules
InterPro DW: JSON to load MySQL and Elasticsearch

Python/Django/cx_Oracle
Dynamic queries (2 DBs)
Range based batches
One output file per batch
Unload/load in parallel

NB: Example code used for loading in pg/cstore during initial tests

[Author: Gustavo Salazar]
Sequence Version Archive (ENA): Archiving in MongoDB

- Document model seems a good match
  - Searchable metadata + compressed sequence ‘files’
  - SVA & ENA Browser pipelines/services can be merged into one
  - Search/Retrieve/DML of entire doc./history, consistency is ok
- GridFS/object store for (only larger?) compressed files?
- Version tracking in multiple ways (oplog tailing, kafka?)
- Built-in HA (3-member replica set across multiple DCs)
- Built-in scalability based on sharding and compression
- Kafka to deal with long migration elapsed time and new/old system coexistence?
Tools: ora2pg

Ora2pg - [http://ora2pg.darold.net/](http://ora2pg.darold.net/) - perl based

- Moves Oracle and MySQL to PostgreSQL
- Mature, actively maintained [Gilles Darold]
  - 2001 05 09:v1.0 - 2017 09 01:v18.2
- Report: useful for surveying and estimates
- Very flexible, highly configurable, good defaults
- Automation: export_schema.sh, import_all.sh
- Parallelism: table, jobs, indexes
- Character set conversion
- LOB support: NO_LOB_LOCATOR|LONGREADLEN+DATA_LIMIT
Tools: outside the db

• **DBeaver CE**: Free Universal SQL Client [https://dbeaver.jkiss.org/](https://dbeaver.jkiss.org/)
  IDE, sessions/locks checking, Eclipse/Plugins architecture

• **SQL Workbench/J**: free, DBMS-independent, cross-platform SQL query tool (java) [http://www.sql-workbench.net/](http://www.sql-workbench.net/)
  IDE, **console mode**, **batch mode**, CL commands (DataPumper, WbCopy, WbExport, WbImport, WbSchemaDiff, WbDataDiff, WbGenerate*, WbList*, WbGrep*)

• **pgAdmin 4**: popular Open Source administration and development platform for PostgreSQL (python) [https://www.pgadmin.org/](https://www.pgadmin.org/)
  IDE, administration, sessions/locks/performance dashboard

• Monitoring and troubleshooting performance
  Nagios & MNTOS (Multi Nagios Tactical Overview System)
  VMware tools (vSphere, vRops), Linux tools, OEM, Sql Developer
Tools: inside pg

**Statistic Collector**: collection and reporting of server activity information
[https://www.postgresql.org/docs/9.5/static/monitoring-stats.html](https://www.postgresql.org/docs/9.5/static/monitoring-stats.html)

- Views: current state (pg_stat_activity, ...) and collected statistics
  (pg_stat_all_tables, pg_stat_all_indexes, pg_stat_user_functions, ...)

  ```sql
  -- Identify Idle in transaction/Waiting sessions
  SELECT * FROM pg_stat_activity
  WHERE (state_change < CURRENT_TIMESTAMP - INTERVAL '30' MINUTE
  AND state = 'idle in transaction') OR waiting = 't';
  ```

- Functions:

  ```sql
  -- Showing PIDs and current queries of all backends
  SELECT pg_stat_get_backend_pid(s.backendid) AS pid,
    pg_stat_get_backend_activity(s.backendid) AS query
  FROM (SELECT pg_stat_get_backend_idset() AS backendid) AS s;
  ```

**pg_stat_statement**: execution statistics, top consumers analysis
[https://www.postgresql.org/docs/9.5/static/pgstatstatements.html](https://www.postgresql.org/docs/9.5/static/pgstatstatements.html)

**auto_explain**: logging execution plans of *slow* statements automatically
[https://www.postgresql.org/docs/9.5/static/auto-explain.html](https://www.postgresql.org/docs/9.5/static/auto-explain.html)
Tools: inside pg

Error Reporting and Logging:
https://www.postgresql.org/docs/9.5/static/runtime-config-logging.html

- `log_min_messages` (WARNING|ERROR)
- `log_min_duration_statement` (0=all, int=ms)

Locks:
https://www.postgresql.org/docs/9.5/static/view-pg-locks.html

Starting from pg 9.6 also: select pg_blocking_pids ();
NB: IS NOT DISTINCT FROM essentially treat NULL as if it was a known value
Lesson Learned

• POC and testing are essential to choose the right tool
• Plan for finishing testing as closer as possible to switch
• Avoid last minute/untested changes before switch
• Timely detection/sharing of anomalies is crucial
• Allow plenty of logging capacity during first days/weeks
• Collaboration & communications are valuable and critical
• SMART goals, tracking, periodic review are a must
Conclusions

• Year 1 concluded: 38 instances successfully migrated
• Lots of challenges/solutions, a lot of work!
  • PL/SQL has been the main hurdle so far in terms of effort
  • Still looking at how improve monitoring of wait events and embedded SQL in PL/pgSQL
• PostgreSQL has proven reliable, performant, well supported and documented: looking forward to pg 10!
• The same for ora2pg, pgloader, oracle_fdw and the IDEs
• Interesting opportunity with MySQL/MongoDB for specific use cases when refactoring
Q & A

• Further info about EMBL-EBI: www.ebi.ac.uk

• Please get in touch:
  • [maurizio] at (ebi.ac.uk) - Myself
  • [systems-dba] at (ebi.ac.uk) - DB team*

• Questions?

*DB Team contributors (Alessio, Andy, Jorge, Luis, Younes)
*Manuela Menchi (DB Team Coordinator - Sys. Apps)