Database Migration: Challenges of Migration from Oracle to Open Source

European Bioinformatics Institute

Maurizio De Giorgi, Database Team

www.ebi.ac.uk

EMBL-EBI

Agenda

- Introduction
- The project
- Use cases: 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

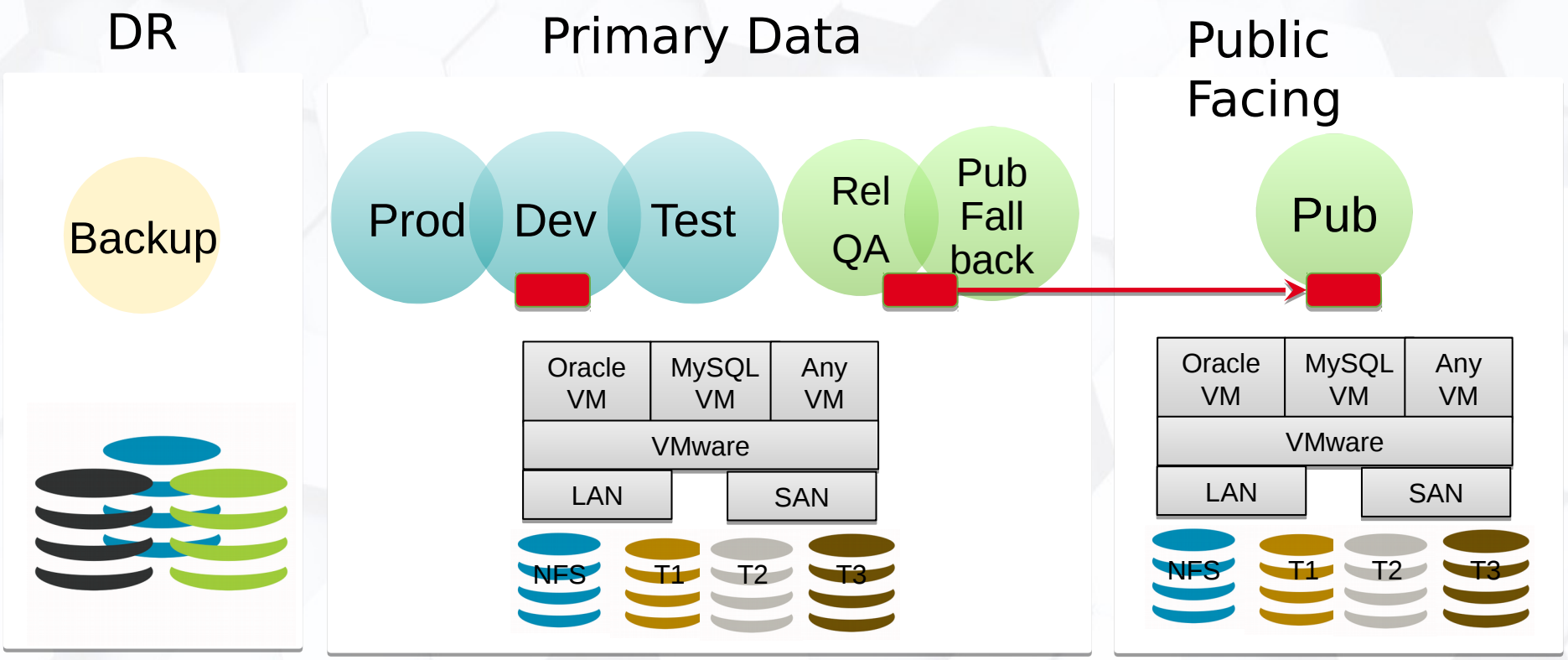
EMBL-EBI Databases Team

- 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
- ...and therefore scope is to provide:
  - technical understanding of the porting challenges
  - a methodology for any porting
  - lessons learned (incl. best choice of DBMS)
- Target: ~30 Oracle instances
- Timeline: 1st phase Jan 2016 - July 2017
- 100% FTE

An aspect of the EMBL-EBI IT architecture



Data Copy Management

T1 SSD/Flash
T2 Hybrid Flash/Conventional Disks
T3 Conventional Disks

The Oracle Usage Survey 2014

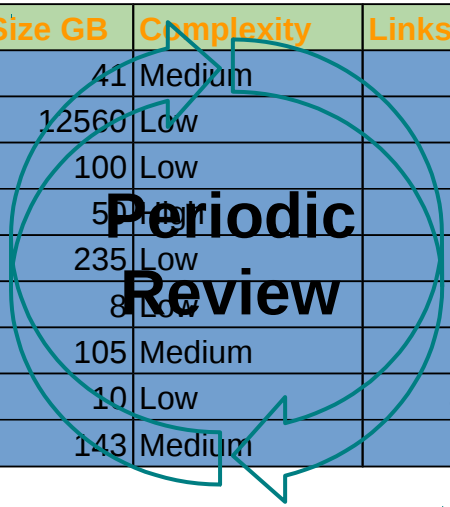
Identify and map

- **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 with 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**

DB	Retuns	Features	Critical	Activity	Willingness	Effort	Code	Size GB	Complexity	Links
A	5	1	H/H	High	Low	Medium	Small	41	Medium	1
B	1	4	M/H	Medium	Medium	Medium	Small	12560	Low	3
C	2	1	L/H	High	High	Large	Small	100	Low	5
D	2	2	L/M	Large	Medium	Medium	Large	50	High	0
E	4	3	L/H	Medium	Low	Medium	Small	235	Low	2
F	3	2	M/H	Medium	High	Large	Small	8	Low	1
G	6	4	H/H	High	Medium	High	Small	105	Medium	3
H	4	1	M/M	Low	Medium	High	Large	10	Low	4
I	6	3	L/M	Low	High	High	Medium	143	Medium	2



Some use cases: current status

DB	Oracle	Net GB	Gross GB	PostgreSQL	MySQL	MongoDB	Dest GB	Notes
Proteome	11gR2	40[1,4]	128	9.5.8			~33->64[6]	ora2pg
Confluence	11gR2	18[2,4]	84	9.5.8			3.4	ora2pg
Jira/FishEye	11gR2	12[2,4]	59	9.5.8			1	ora2pg
Metabolights	11gR2	10[1,4]	100	9.5.6			1	ora2pg
Expr. Atlas	11gR2	95[1,4]	232	9.5.7			48	ora2pg
RT4	11gR2	76[2,4]	130	9.5.7			28	ora2pg
UniRule	11gR2	8[2,4]	333[1]	9.5.8			5	ora2pg
RNA Central	11gR2	106[3,4]	405	9.5.7			86	ora2pg
GVA	11gR1	493	1041	9.5.8			N/A	ora2pg
InterPro DW	11gR2	~400	1519		5.6.24		~116[5]	Refactoring
SVA	11gR1	~18000[1]	~20000	9.5.8?		3.4.7	~2048	Refactoring

[1] deprecated|obsolete data|objects removal
[2] BLOB -> bytea|Attachments on FS
[3] NVARCHAR2->VARCHAR|TEXT, 2->1 byte
[4] CLOB -> TEXT (TOASTed)
[5] +Elasticsearch
[6] almost doubled in size in few months

Use cases:
technical challenges and adopted solutions

- Access to/from Oracle
- Dealing with 3rd party DB
- Loading files (POC & Multiple files)
- Porting PL/SQL to PL/pgSQL
- Partitioning & PEL
- *InterPro DW: JSON to load MySQL and Elasticsearch*
- *Sequence Version Archive (ENA): Archiving in MongoDB*

Access to/from Oracle

- Access to Oracle from Pg - **oracle_fdw** [Laurenz Albe]
  - 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
- 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

Confluence: Character set conversion, LOBs

- Character set unsupported by Atlassian `(US7ASCII)`
  - data with mixed encoding: in situ conversion last resort
  - export table, assessment, conversion, checking, import
  - file --mime..., iconv -f ${incs} -t ${outcs}..., python/bash
- LOBs cardinality/size large enough to:
  - impact significantly on elapsed time (one-by-one processing)
  - cause out of memory errors (batch processing)
- Assess max/avg/tot LOBs size and cardinality in Oracle
  - “batch” mode for high cardinality/small LOBs ⇑ memory usage
  - one-by-one for low cardinality/Very Large LOBs ⇑ elaps. time

Confluence - Assessing LOBs

max_len, tot_data, cardinality

```
-- query to generate actual query to run (tested with Oracle 11gr1, 11gr2)
select -- lob_num, lob_count, rownum,
     sql_text || case when lob_num = lob_count
                 then ' order by lob_data_tot desc nulls last, row_count desc;'
                 else ' union ' end sql_script
  from (select 'SELECT ''' || owner || '.' || table_name || '.' || column_name || ''' lob_col,' ||
               ' max (dbms_lob.getlength('||column_name||')) max_lob_len, '||
               ' sum (dbms_lob.getlength('||column_name||')) lob_data_tot, '||
               ' count (*) row_count ' ||
               ' FROM ' || owner ||'.'|| table_name sql_text,
               row_number () over (order by owner, table_name, column_name) lob_num,
               count (*) over () lob_count
          from dba_lobs
         where owner = '&&OWNER'
         order by owner, table_name, column_name);
```

Confluence - Assessing LOBs

max_len, tot_data, cardinality

```
-- example query generated
SELECT '<OWNER>.<TABLE>.<LOB_COL>' lob_col, max (dbms_lob.getlength(<LOB_COL>)) max_lob_len,
       sum (dbms_lob.getlength(<LOB_COL>)) lob_data_tot, count (*) row_count
  FROM <OWNER>.<TABLE>
 union ...
 order by lob_data_tot desc nulls last, row_count desc;

-- example results obtained
LOB_COL|MAX_LOB_LEN|LOB_DATA_TOT|ROW_COUNT
<OWNER>.<TABLE>.<LOB_COL>|3899|2255591728|1292287
...
```

Loading files: POC

CSV file ~2.4M rec. load table with 4 varchar columns (50-255)

pgloader --root-dir .../reports/ --logfile pgloader.log .../cmdfile

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 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
  - simple when using bootstrap script
  - make DYNSIZE=8192 pgloader
- Concatenate files makes tuning easier and more performing
- Disable/Enable autovacuum on table before/after load

Porting PL/SQL to PL/pgSQL: challenges

Significant differences to address:

1. **No concept of package ⇒ no globals**
2. **A procedure is part of an “outer” transaction** and every procedure called from another one is part of the caller’s transaction
3. as a consequence **no embedded commit** is allowed
4. **DDL are transactional** while encapsulated within implicit commits in Oracle
5. **Embedded SQL is not “visible” at run time** in pg_stat_activity
6. **Incompatible syntax**

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();

-- 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: issue

Initial troubleshooting of errors, or mis-behaving queries, happening deep in the call 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 online 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.

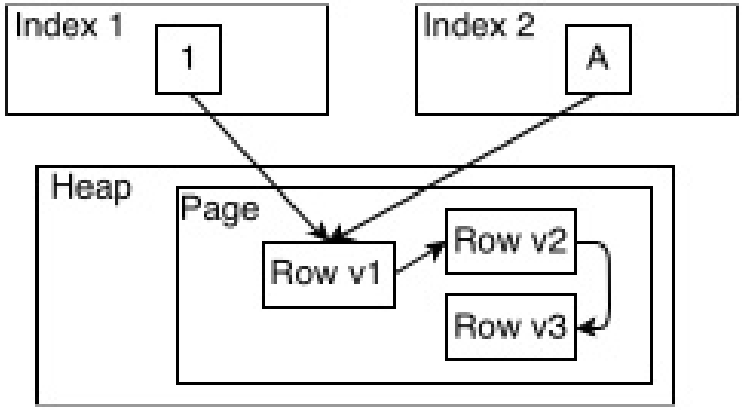
Porting PL/SQL to PL/pgSQL: update is not HOT!

-- 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 migration

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, (index/constr. maintenance “lighter”)

-- to monitor HOT updates vs total updates

select n_tup_upd, n_tup_hot_upd from pg_stat_user_tables;

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, invoke or combine the procedure call with launcher or **wrapper in python/other language with commits where required**
3. same as above
4. same as above
5. extend **code instrumentation**
6. many examples online, use **orafce or other** commercially available **compatibility modules**

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
    - aim at using the latest ora2pg version
    - 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
  - check if extensions can suit your case
  - pg 10 introduce some declarative partitioning
- PEL needed a substantial rewrite

  ...and testing and troubleshooting needed time/effort
- do not forget: SET constraint_exclusion = on;

Proteome - Managing post-migration

- Minor issues to fix immediately
  - data types, suboptimal query performance
  - failing query (oracle syntax, huge statements)
- Tuning
  - autovacuum 20->5%, memory 4->8GB, work_mem
  - locks, “*idle in transaction*”/waiting sessions
- **Extensive logging, monitoring, alerts**
- Trust-based relationship and **quick channels of communication**

InterPro DW: JSON to load MySQL and Elasticsearch

```
def oracle2json(interpro_db, protein_db, block_size, match_pos, compare_sym, end):
    ipro = DATABASES['interpro_ro']
    print('connecting to '+ipro['NAME'])
    con = cx_Oracle.connect(
        ipro['USER'], ipro['PASSWORD'],
        cx_Oracle.makedsn(ipro['HOST'], ipro['PORT'], ipro['NAME']) if ipro['PORT'] else ipro['NAME']
    )
    is_for_interpro_entries = interpro_db == "interpro"
    where = []
    if protein_db == "reviewed":
        where.append("p.DBCODE='S'")
    elif protein_db == "unreviewed":
        where.append("p.DBCODE='T'")
    if not is_for_interpro_entries:
        where.append("pe.DBCODE='{}'".format(dbcode[interpro_db]))
    if match_pos is not None:
        where.append("pe.POS_FROM {} {}".format(compare_sym, match_pos))

    part = 0
    for chunk in chunks(get_from_db(con, end, " AND ".join(where), is_for_interpro_
        f = open("{}/ipro_tst_{}_{}_{:06}{}.json".format(
            interpro_db,  # to save in folder
            interpro_db,
            protein_db,
            part,
            '' if match_pos is None else '_' + str(match_pos)
        ), "w")
        f.write(json.dumps(list(chunk)))
        f.close()
        part += 1
```

[Author: Gustavo Salazar]

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

```
f = open(path, 'r')
cur.copy_expert("COPY {} FROM STDOUT csv quote e'\x01' delimiter e'\x02'".format(temp_table), f)
cur.execute("INSERT INTO {} SELECT p.* FROM {} CROSS JOIN lateral json_populate_recordset(null::{}, doc) as p;".format(
    table, temp_table, table
))
```

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

Sequence Version Archive (ENA): Archiving in MongoDB

- Document model is 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 safe

  GridFS or Object Store as file store (>16MB/always) ?
- 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 temporary coexistence?

Tools: ora2pg

ora2pg - http://ora2pg.darold.net/ - perl/DBI/DBD based

- Moves Oracle and MySQL database to PostgreSQL
- Mature, actively maintained [Gilles Darold]
  - 2001 05 09 - Initial version 1.0
  - 2017 09 01 - v18.2
- Report: very useful for surveying and estimates
- Very flexible, highly configurable, good defaults
- Wrapper scripts: export_schema.sh, import_all.sh
- Parallelism: table, jobs, indexes
- Character set conversion
- LOB support (NO_LOB_LOCATOR|LONGREADLEN+DATA_LIMIT)

Tools: ora2pg example report

Ora2Pg - Database Migration Report

Version Oracle Database 12c Enterprise Edition Release 12.1.0.2.0
Schema HR
Size 9.62 MB

Object	Number	Invalid	Estimated cost	Comments	Details
DATABASE LINK	4	0	12	Database links will be exported as SQL/MED PostgreSQL's Foreign Data Wrapper (FDW) extentions using oracle_fdw.	
FUNCTION	2	0	9	Total size of function code: 421 bytes.	get_tab_ptf: 4 get_tab_tf: 3
INDEX	29	0	4.8	17 index(es) are concerned by the export, others are automatically generated and will do so on PostgreSQL. Bitmap index(es) will be exported as b-tree index(es) if any. Cluster, domain, bitmap join and IOT indexes will not be exported at all. Reverse indexes are not exported too, you may use a trigram-based index (see pg_trgm) or a reverse() function based index and search. Use 'varchar_pattern_ops', 'text_pattern_ops' or 'bpchar_pattern_ops' operators in your indexes to improve search with the LIKE operator respectively into varchar, text or char columns.	5 domain index(es) 1 function based b-tree index(es) 11 b-tree index(es)
JOB	0	0	0	Job are not exported. You may set external cron job with them.	
MATERIALIZED VIEW	2	0	6	All materialized view will be exported as snapshot materialized views, they are only updated when fully refreshed.	
PACKAGE BODY	2	0	44	Total size of package code: 2992 bytes. Number of procedures and functions found inside those packages: 6.	emp_mgmt.create_dept: 3 emp_mgmt.hire: 11 emp_mgmt.increase_comm: 3 emp_mgmt.increase_sal: 3 emp_mgmt.remove_dept: 3 emp_mgmt.remove_emp: 3
PROCEDURE	2	0	8	Total size of procedure code: 772 bytes.	secure_dml: 3 add_job_history: 3
SEQUENCE	4	0	0.4	Sequences are fully supported, but all call to sequence_name.NEXTVAL or sequence_name.CURRVAL will be transformed into NEXTVAL('sequence_name') or CURRVAL('sequence_name').	
SYNONYM	0	0	0	SYNONYMs will be exported as views. SYNONYMs do not exists with PostgreSQL but a common workaround is to use views or set the PostgreSQL search_path in your session to access object outside the current schema.	emp_details_view_v is an alias to HR.EMP_DETAILS_VIEW public.emp_table is a link to hr.employees@curr_user offices is an alias to HR.LOCATIONS
TABLE	36	0	18.2	1 external table(s) will be exported as file_fdw foreign table. See EXTERNAL_TO_FDW configuration directive to export as standard table or use COPY in your code if you just want to load data from external files. 2 check constraint(s).	1 binary columns 5 unknow types Total number of rows: 1552 Top 5 of tables sorted by number of rows: customer_summary has 1154 rows employees has 107 rows user_role has 55 rows t1 has 32 rows departments has 27 rows
TABLE PARTITION	2	0	0.2	Partitions are exported using table inheritance and	1 range partitions

Tools: ora2pg example report

TABLE PARTITION	2	0	0.2	Partitions are exported using table inheritance and check constraint. Hash partitions are not supported by PostgreSQL and will not be exported.	1 range partitions
TABLE SUBPARTITION	2	0	0.4		
TRIGGER	6	1	36	Total size of trigger code: 2120 bytes.	check_raise_on_avg: 18 update_job_history: 3 ioft_emp_perm: 3 ioft_insert_role_perm: 3
TYPE	3	0	2	2 type(s) are concerned by the export, others are not supported. Note that Type inherited and Subtype are converted as table, type inheritance is not supported.	2 nested tables 1 object type
VIEW	4	0	4	Views are fully supported.	
Total	98	1	145	145 cost migration units means approximatively 2 man-day(s). The migration unit was set to 5 minute(s)	

Details of cost assessment per function

- Function emp_mgmt.hire total estimated cost: 11
  - DBLINK => 2 (cost: 4)
  - TEST => 2
  - SIZE => 1
- Function get_tab_ptf total estimated cost: 4
  - TEST => 2
  - SIZE => 1
  - PIPE ROW => 1 (cost: 1)
- Function add_job_history total estimated cost: 3
  - TEST => 2
  - SIZE => 1
- Function emp_mgmt.increase_sal total estimated cost: 3
  - TEST => 2
  - SIZE => 1
- Function emp_mgmt.increase_comm total estimated cost: 3
  - TEST => 2
  - SIZE => 1
- Function get_tab_tf total estimated cost: 3
  - TEST => 2
  - SIZE => 1
- Function emp_mgmt.remove_emp total estimated cost: 3
  - TEST => 2
  - SIZE => 1
- Function emp_mgmt.create_dept total estimated cost: 3
  - TEST => 2
  - SIZE => 1
- Function secure_dml total estimated cost: 3
  - TEST => 2
  - SIZE => 1
- Function emp_mgmt.remove_dept total estimated cost: 3
  - TEST => 2
  - SIZE => 1

Generated by Ora2Pg v14.1

Tools: pgloader vs pg_bulkload

pg_bulkload: http://ossc-db.github.io/pg_bulkload/index.html

- good mostly for MASSIVE INITIAL LOADING
- restrictions due to DIRECT writing into db files
  - *does not work properly in streaming replication environment*
  - *bypasses some internal functionality such as WAL. needs separate recovery procedure before usual PostgreSQL's recovery*

pgloader: http://pgloader.io/index.html

- uses the COPY streaming protocol
- less performing but not so many restrictions
- lots of functionalities, parameters, including transformations
- can load various use cases/formats
- good results even with large files in different scenarios/formats
- well understood and accepted by users/developers
- can load single/multiple files in parallel

Tools: outside the db

- DBeaver CE: Free Universal SQL Client 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/

  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/

  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

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

```
-- 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:

```
-- 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

auto_explain: logging execution plans of *slow* statements automatically
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

```
SELECT blocked_locks.pid AS blocked_pid, blocked_activity.usename AS blocked_user, blocking_locks.pid
  FROM pg_catalog.pg_locks         blocked_locks
  JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid = blocked_locks.pid
  JOIN pg_catalog.pg_locks         blocking_locks ON blocking_locks.locktype = blocked_locks.locktype
        AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
        AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
        AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
        AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
        AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
        AND blocking_locks.transactionid IS NOT DISTINCT FROM blocked_locks.transactionid
        AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
        AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
        AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
        AND blocking_locks.pid != blocked_locks.pid
  JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid = blocking_locks.pid
WHERE NOT blocked_locks.GRANTED;
```

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)