Migrations to PostgreSQL
(from Oracle)
I am: Venkata B Nagothi

A PostgreSQL consultant for the past 6+ years
Why migrate to PostgreSQL?

- Cost-effective and feature-rich open-source database
- PostgreSQL Community
- Integration capabilities
- Can replace any commercial database
- Cloud adoption
Migration process - overview

- Preliminary Migration Analysis
- Perform Migration
- Functionality Testing
- Performance Testing
Preliminary Migration Analysis

Application Environment
- Evaluate Application Source Code
- Application Architecture

Database Environment
- Evaluate Database Infrastructure
- Schema
- Data
Preliminary Migration Analysis

- Major roadblocks for migration
  - Heavy PL/SQL usage
  - Heavy usage of Large Objects
  - Proprietary application with its own schema
  - Application compatibility issues
  - No application source code

- Evaluate database migration effort
  - Amount of manual migration effort needed

- Evaluate application migration effort
  - This is critical as the Application code change is mostly manual
Migration challenges

- Database Design / Architecture challenges
- High Availability challenges
- Development Challenges
- Data Migration challenges
Database Design / Architectural challenges
Database Design / Architecture

Transaction Log Files

Oracle

Redo Log Files (Multiplexing)

Group 1
/disk1
redo1a
redo1b

Group 2
/disk2
redo2a
redo2b

PostgreSQL

WAL Files (No Multiplexing)

$PGDATA/pg_xlog

00000001000000010000007B
000000010000000100000077
000000010000000100000078
000000010000000100000072
000000010000000100000079

• Database will hang if the pg_xlog Disk space is full
• Each WAL file size is 16 MB
• No multiplexing
• No hard limit on the number of files
• I/O balancing is needed
Database Design / Architecture

Archived Log files

Oracle

Redo Log Files

Group 1

redo 1a

500 MB

PostgreSQL

WAL Files

$PGDATA/pg_xlog

00000001000000010000007B (16 MB)

000000010000000100000077 (16 MB)

Compression needed

WAL Archived File

0000000100000001000000077 (16 MB)
Database Design / Architecture

Data files

Oracle

Data Files

/disk1/datafile1
/disk2/datafile2
/disk3/datafile3
/disk4/datafile4

PostgreSQL

Data Files
(directory)

13157_fsm  13172_fsm  13189
13157_vm   13172_vm   13191
13159     13174      1417
13161     13176      1417_vm
13162     13177      1418

- Storage is Directory Bound
- Data files are auto generated
- DBA has no control over data file management
Database Design / Architecture

Data File structure

Table-1 = 30 GB
Table-2 = 10 GB

Oracle

Tablespace

Table-1, Table-2
/disk1/datafile1
/disk2/datafile2
/disk3/datafile3
/disk4/datafile4
10GB 10GB 10GB 10GB

PostgreSQL

Tablespace-1
Table-1
/disk1/tbs01
30GB

Tablespace-2
Table-2
/disk2/tbs02
10GB

RAID

I/O Balancing is a challenge

(OR)

/data/tbs
### Database Design / Architecture

#### Control File

**Oracle**
- Control File (Multiplexing)
  - /disk1/oradata/control01.ctl
  - /disk2/oradata/control02.ctl

**PostgreSQL**
- pg_control file (No Multiplexing)
  - $PGDATA/global/pg_control

- Loosing pg_control file will result in incomplete recovery
It would be good to have multiple bg-writers for better write performance and scalability in a high-transaction multi-CPU environment.
High Availability challenges
High Availability

Oracle Dataguard

PostgreSQL Streaming Replication

Migration Impact (Streaming Replication)

• More simpler to implement
• Supports all the protection modes in Oracle
• Supports cascading replication with some limitations
High Availability

Role reversal (for disaster recovery) – Limitations in PostgreSQL

Old master can be made standby.

Streaming Replication

Standby can be promoted to Standalone after the master is shutdown

Limitation
Role reversal cannot be performed without an Outage to the master.
High Availability

PostgreSQL Streaming Replication – Limitations

- Standby can be built on a different filesystem using `pg_basebackup`
Development challenges
(database)
Database migration

ora2pg

- ora2pg is the most open-source tool used for migrating the Oracle database to PostgreSQL.
- Database migration time and cost can be evaluated by identifying the manual migration effort needed.
- Database migration can be in three phases:
  - Schema migration
  - PL/SQL migration
  - Data migration
Most of the Schema migration can be done automatically using ora2pg.

The Oracle database objects not supported by PostgreSQL must be identified and must be migrated manually.
## Database migration

### Schema Migration

<table>
<thead>
<tr>
<th>Unsupported Object</th>
<th>PostgreSQL Alternative solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Materialized Views</td>
<td>Auto refresh and query re-writing is not supported</td>
</tr>
<tr>
<td>Index-Organized-Tables</td>
<td>This requirement can be partially fulfilled by Clustering a Table.</td>
</tr>
<tr>
<td>Public Synonyms</td>
<td>Public Synonyms are not supported in PostgreSQL. “search_path” can be used as an alternative</td>
</tr>
<tr>
<td>Global Temporary Tables</td>
<td>There is no support for Global Temporary Tables. Unlogged Tables can be used instead.</td>
</tr>
</tbody>
</table>
| Partitioned tables         | Partitioned Tables in PostgreSQL are not quite similar to Oracle style partitioning and cannot be used as an alternative.  
  • Child tables are more like individual tables  
  • Constraints are not inherited to child tables  
  • No support for Global and Local indexes |
Database migration

PL/SQL Migration

- Ora2pg partially migrated PL/SQL objects
- As PostgreSQL does not support objects like Packages, most of the PL/SQL objects must be migrated manually
## PL/SQL Migration

<table>
<thead>
<tr>
<th>Unsupported PL/SQL Object</th>
<th>Alternative PostgreSQL Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packages</td>
<td>Packages are unsupported in PostgreSQL.</td>
</tr>
<tr>
<td>DBMS* Packages</td>
<td>Some of the DBMS packages are supported by orafce external contrib module. The unsupported packages cannot be migrated. A custom function must be built if required.</td>
</tr>
</tbody>
</table>

### PACKAGES

<table>
<thead>
<tr>
<th>PACKAGE DEFINITION</th>
<th>PostgreSQL Alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>must be used as an alternative for Package definitions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GLOBAL VARIABLES</th>
<th>TEMP TABLES must be used as an alternative for GLOBAL VARIABLES</th>
</tr>
</thead>
</table>

| PACKAGE BODY        | Package Body must be converted to FUNCTION(S) |

- Application functionality testing is critical and can poses challenges
- Reaching the expected Performance benchmark can be a challenge too.
- There could be a need to re-write the whole business logic to full fill the application requirements.
## PL/SQL Cursors

<table>
<thead>
<tr>
<th>PostgreSQL Cursors</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROWCOUNT attribute is not supported</td>
<td>MOVE LAST must be used to fetch the count of the rows in a cursor. This could be a significant performance penalty when cursor has millions of rows.</td>
</tr>
</tbody>
</table>

**Cursor position**

1
2
3
4
5  <- Current cursor position
6
7
8
9
1 billion  <- MOVE ALL

Cursor position must be moved to last row of the cursor to fetch the count.
PostgreSQL is rich in data-type

Data migration can be performed using Ora2pg which is very efficient

However, significant challenges can be encountered while migrating **Large objects** and **JSON** data
Database migration

Migration Large Objects

- BLOB
  - BYTEA (1 GB LIMIT)
  - LO
Migrating Large Objects

Migration Impact

- Oracle BLOBs are automatically migrated to PostgreSQL bytea by ora2pg
- BYTEA cannot be streamed. This can lead to unstable Application performance which would have a negative impact on the benchmarking metrics
- Can accommodate more than 1 GB due to PostgreSQL compression algorithm
- Triggers an Application code change
- BYTEA into the memory at a time which will impose performance problems resulting in excessive usage of Memory and Network bandwidth

<table>
<thead>
<tr>
<th>BYTEA Column</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text File</td>
<td>10K</td>
</tr>
<tr>
<td>Text File</td>
<td>10K</td>
</tr>
<tr>
<td>Image</td>
<td>20M</td>
</tr>
<tr>
<td>Image</td>
<td>300M</td>
</tr>
<tr>
<td>Text file</td>
<td>100K</td>
</tr>
</tbody>
</table>
Database migration

Migration Large Objects

**Migration Impact**

- LO objects can be streamed in multiple chunks resulting in effective performance
- Custom ETL scripts must be built
- Migration effort and time can be a challenge
- Application code change is expected
- Application functionality and Performance testing
pg_largeobjects is used to store the large objects in PostgreSQL

There is a serious limitation around this.

All the Large objects of all the tables in the database are stored in a single table called pg_largeobjects. This can turn out to be a serious design flaw and can be a significant performance bottleneck.
Migrating JSON Data

- Generally, Oracle uses BLOB or CLOB to store JSON data.
- Technically, JSON is treated as a Large object.
- As PostgreSQL’s support towards JSON is powerful and efficient, JSON data can be migrated to PostgreSQL’s JSON/JSONB data types.
- Unfortunately, this is a manual and time consuming process. Significant application code change is also required.
Migrating JSON Data

Migration Impact

- BLOB migration to JSONB cannot be done directly and no ETL can make this possible
- Can result in an heavy Application design and code change
- Migration effort and time can be a challenge
- Heavy development can be on the cards
Migrating JSON Data

Oracle CLOB with JSON data ➔ AUTOMATIC (Ora2pg) ➔ PostgreSQL TEXT ➔ Manual ➔ PostgreSQL JSONB

Migration Impact

- Can result in an heavy Application design and code change
- Migration effort and time can be a challenge
Development challenges
(Migrating Oracle SQLs for Application)
SQLs Migration for application

- Application migration is the most critical and challenge episode of whole migration project

- Migration SQLs in the Application source code is very critical and can take up majority of the migration time.

- Application source code must be analysed to identify the SQLs that needs to be changed which is not an straight forward way unfortunately.

- A single SQL syntax change can trigger a change at hundreds of places in the application source code files
## Hierarchical queries

<table>
<thead>
<tr>
<th>Oracle</th>
<th>PostgreSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECT BY START WITH SYS_CONNECT_BY_PATH</td>
<td>• WITH RECURSIVE is a straight alternative for migrating Hierarchical queries</td>
</tr>
<tr>
<td>CONNECT_BY_ROOT</td>
<td>• Depending on the complexity of the query logic, connectby() function part of tablefunc contrib module can be of use</td>
</tr>
<tr>
<td></td>
<td>• pl-pgsql functions is another alternative if the Hierarchical queries cannot be converted using WITH RECURSIVE alone</td>
</tr>
</tbody>
</table>
### Migration Impact

**AUTOCOMMIT OFF**

- Application behaviour is different in **autocommit off** mode
- Multiple transactions will be automatically part of a transaction block, which means COMMIT ALL or NONE
- Heavy application code change may be required
- Functionality testing
- Performance testing
- Changing legacy code might impose further more challenges
Implicit type casting can be highly beneficial and can help reduce the need to change the application code to a greater extent.

- Increases the possibility usage of Indexes
- Avoids the need to create function-based Indexes

**Table**

<table>
<thead>
<tr>
<th>varchar</th>
<th>varchar</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Sydney</td>
</tr>
<tr>
<td>3</td>
<td>London</td>
</tr>
<tr>
<td>4</td>
<td>Singapore</td>
</tr>
</tbody>
</table>

**PostgreSQL**

```sql
select * from table where col1::int > 2;
create cast(varchar as integer) with inout as implicit;
select * from table where col1 > 2;
```

**Oracle**

```sql
select * from table where col1 > 2;
3       London  
4       Singapore 
```

SQLs Migration for application
Questions ?