# New ways to migrate from Oracle


Laurenz Albe
laurenz.albe@cybertec.at

Cybertec

Prague PostgreSQL Developers' Day 2018

# The problem


Database migration consists of several parts:

- Migration of object definitions
  (`CREATE TABLE`, constraints, indexes …)
- Migration of the table data
- Migration of code stored in the database
  (functions, triggers, packages, …)
- Adapt the application (drivers, SQL dialect)

# DDL migration

CYBERTEC
The PostgreSQL Database Company

- Need to extract metadata to construct DDL
  - `CREATE TABLE` is defined in the SQL standard
  - but not nested tables, partitioning …
- Need to adapt data types
  - Translate `NUMBER` to `integer`, `double precision` or `numeric`?
  - Translate `DATE` to `date` or `timestamp`?
    (Oracle `DATE` has seconds precision)
  - Translate `BLOB` to `bytea` or large objects?

# Data migration


Extracting data from Oracle is difficult:

- sqlplus can be scripted, but is (almost) unusable
- SQL Developer is possible, but not comfortable. Cannot export `BLOB`s
- Oracle foreign data wrapper works well

It seems like Oracle makes this hard by design

# “Stored” code migration

CYBERTEC
The PostgreSQL Database Company

- Functions, procedures, packages, triggers
- Code has to be translated by hand
  - PL/SQL is similar to PL/pgSQL, but only very simple functions need no editing
- Packages can often be translated to functions in a schema
- PL/SQL code that interacts with the OS a lot (manipulate files, send e-mail) is often better translated to PL/Perl or PL/Python
- PL/Java is an option for Java stored procedures

# Adapt the application



- Should be simple: both Oracle and PostgreSQL use SQL, and there is an SQL standard, but
  - nobody supports the complete standard
  - everybody extends the standard
- Need to adapt to changed data types
- Application code needs manual intervention and a lot of testing
- Module “orafce” provides (some) compatibility
- Abstraction layers (ORM) help a lot

# SQL differences: Joins



- Oracle uses a special syntax for outer joins:

```
SELECT b.col1, a.col2
FROM base_table b, attributes a
WHERE b.id=a.b_id(+);
```

- Has to be translated to standard join syntax:

```
SELECT b.col1, a.col2
FROM base_table b
    LEFT JOIN attributes a ON b.id=a.b_id;
```

# SQL differences: empty strings



- Oracle treats empty strings as NULL values
- This leads to different semantics in string concatenation:

```
('astring' || NULL) IS NOT NULL
```

- A workaround in PostgreSQL is to use the function `coalesce`:

```
coalesce(col1, '') || coalesce(col2, '')
```

# SQL differences: Aliases



- Oracle supports the following syntax:

```
SELECT *
    FROM (SELECT …);
```

- PostgreSQL (and the standard) require an alias:

```
SELECT *
    FROM (SELECT …) alias;
```

# SQL differences: sequences



- SQL standard syntax: `NEXT VALUE FOR asequence`
- Oracle syntax: `asequence.NEXTVAL`
- PostgreSQL syntax: `nextval('asequence')`
- Oracle does not allow `NEXTVAL` in a `DEFAULT` clause

  → simplification in PostgreSQL possible
- This difference might become less important with the new identity column syntax supported by both

# Techniques for Oracle migration



- There are some commercial tools
- There is ora2pg (https://ora2pg.darold.net/)
  - Free open source
  - Time tested (has been around for a while)
  - Rich in features (PL/SQL migration, migration cost assessment reports)
  - Works fairly well

# Shortcomings of ora2pg

CYBERTEC
The PostgreSQL Database Company

- Many features lead to many bugs
- Produces a “monolithic” SQL script that usually has to be edited by hand.
- Does not cope well with “moving targets” (schema changes while migration is developed)

# Design goals for a new tool

CYBERTEC
The PostgreSQL Database Company

- Do not attempt to cover everything (PL/SQL code, user defined types, …) Rather, keep it simple
- Comfortable editing of individual object definitions
- Ability to cope with “moving targets” (to some extent)

# Idea: use Oracle FDW

CYBERTEC
The PostgreSQL Database Company

- The Oracle foreign data wrapper is good for data migration (can be used with ora2pg):
  - Translates between data types
  - Translates encoding
  - Direct data transfer without intermediate storage
- If oracle_fdw is good for data, why not also for metadata?
- This way, everything can be done with PL/pgSQL inside PostgreSQL
- Easy to write, no portability issues!

# What is a foreign data wrapper?



- Allows to define “foreign tables” which look and feel like they are normal tables.
- SQL statements are “redirected” to an external data source.
- Conforms to the SQL standard.
- Exist (in varying quality) for a lot of external data sources.

# ora_migrator architecture

CYBERTEC
The PostgreSQL Database Company

- Two “helper” schemas: Oracle and PG stage
- Oracle stage contains foreign tables for Oracle metadata (table columns, constraints, etc.)
- PG stage contains tables with a snapshot of the data from the Oracle stage plus “translated” values (lower case names, PostgreSQL data types)
- PG stage can be edited (data types, PL/SQL code)
- PG stage can be “refreshed” from Oracle stage (should work well for “normal” schema changes)

# ora_migrator requirements

CYBERTEC
The PostgreSQL Database Company

- Install oracle_fdw extension
- Create a foreign server and a user mapping with a user that can access Oracle catalog
  (e.g. `SELECT ANY DICTIONARY` privilege)
- Test oracle_fdw configuration with
  ```
  SELECT oracle_diag('servername');
  ```
- Install ora_migrator extension with
  ```
  CREATE EXTENSION
  ```

# Migration steps (1)



- `oracle_migrate_prepare`: creates stages
- Edit tables in the PG stage
- `oracle_migrate_mkforeign`: creates target schemas and foreign tables for the data
- `oracle_migrate_tables`: creates local tables and migrates the data (errors turned to warnings)
- for parallelization, migrate each table by calling `oracle_materialize('schema', 'table')`

# Migration steps (2)



- `oracle_migrate_functions`: creates functions
- `oracle_migrate_triggers`: creates triggers
- `oracle_migrate_views`: creates views
- `oracle_migrate_constraints`: creates constraints and indexes
- `oracle_migrate_finish`: drops staging schemas
- `DROP EXTENSION oracle_fdw CASCADE;`
  to remove all traces of the migration process

# “One click” migration



- For simple databases that need no manual editing, you can migrate with one call:

```
SELECT oracle_migrate(
    server       => 'oraserver',
    only_schemas => '{ORASCHEMA}');
```

- This won’t migrate functions and will only work in very simple cases.

# Data migration problems (1)



- Zero bytes in Oracle strings

```
invalid byte sequence for encoding "UTF8": 0x00
```

- Best solution: fix in Oracle
- Workaround:

```
ALTER FOREIGN TABLE s.tab OPTIONS (
    DROP schema, SET table '(SELECT id,
      replace(str, chr(0), '''') AS str, …
      FROM s.tab)');
```

# Data migration problems (2)



- Illegal bytes in Oracle:

  ```
  invalid byte sequence for encoding "UTF8": 0x80
  ```

- Can happen because Oracle does not check data if client encoding = server encoding
- Possible solution (if “real” encoding is known)
  - Create PostgreSQL DB with “real” encoding
  - Set `nls_lang` option on foreign data wrapper to Oracle server encoding (no translation)

# Making it comfortable

CYBERTEC
The PostgreSQL Database Company

- Editing tables with `UPDATE` statements isn't nice
- Cybertec is currently developing a GUI that handles ora_migrator for you:
  - Calls the appropriate SQL functions for setup and migration
  - Comfortable editor to modify metadata

# How can I get ora_migrator?

CYBERTEC
The PostgreSQL Database Company

- ora_migrator is open source and available on https://github.com/cybertec-postgresql/ora_migrator
- Try it and share your issues and ideas

CYBERTEC
The PostgreSQL Database Company

Thank you for your attention!

Any Questions?