Migrating data from Oracle to PostgreSQL

From PNSNWikiDocs

This page explores tools and procedures for migrating AQMS databases to PostgreSQL. For more general treatment, see Migrating from Oracle to PostgreSQL. To migrate client programs, see Migrating client programs from Oracle to PostgreSQL.

A useful overview of issues can be found at https://severalnines.com /blog/migrating-oracle-postgresql-what-you-should-know

Contents

candidate tools

- convert tools
  - dbconvert - proprietary ~$150 - https://dbconvert.com/oracle /postgresql/
  - #ora2pg - see below.
- db links and foreign wrappers
  - oracle_fdw - foreign data wrapper pg extension for connecting to oracle database: https://pgxn.org/dist/oracle_fdw/ and http://laurenz.github.io /oracle_fdw/

ora2pg

This is a very promising set of perl scripts using perl DBD::Oracle module to read from Oracle database. By default, ora2pg writes to a flat posgreSQL dump-file which can then be imported using psql. You can set up a direct connection using the DBD::pg module on the postgres side.

See: http://ora2pg.darold.net/documentation.html

Configuration is specified in a flat-text file ora2pg.conf. Schemas, Tables, Indexes etc. for copy can be specified here. Column order and data type can be modified with configuration file directives. Tables can be renamed.

The SHOW_REPORT option is useful in determining the scale of the job as well as potential data incompatibilities etc.

installation

Ora2pg is not available from the ordinary yum repositories. It must be downloaded from cpan and requires installing perl modules to connect to both Oracle and PostgreSQL databases.

Oracle connector

This is the hardest part, as it requires Oracle proprietary drivers. These can be installed from the 'free' oracle instantclient development package RPM's available from https://www.oracle.com/technetwork/database/database-technologies/instant-client/downloads/index.html. You will need at least the '-basic' and '-devel' packages, but might as well install -jdbc' and '-sqlplus' while you are there.

Next, you must install the DBD::Oracle module. This is described in Installing DBD::Oracle perl module.

PostgreSQL connector

This is the easiest part:

```
[root@tahoma ~]# yum install perl-DBD-Pg
```

or2pg software

Download source from https://github.com/darold/ora2pg/releases and extract tar contents.

```
kress@babbage:~/software$ cd ora2pg-19.1/
kress@babbage:~/software/ora2pg-19.1$ perl Makefile.PL
```

```
Checking if your kit is complete...
Looks good
Generating a Unix-style Makefile
Writing Makefile for Ora2Pg
Writing MYMETA.yml and MYMETA.json

Done...
---------------------------------------------------------------------------
Please read documentation at http://ora2pg.darold.net/ before asking for help
---------------------------------------------------------------------------
Now type: make && make install
kress@babbage:~/software/ora2pg-19.1$ make
cp lib/Ora2Pg/GEOM.pm blib/lib/Ora2Pg/GEOM.pm
cp lib/Ora2Pg.pm blib/lib/Ora2Pg.pm
cp lib/Ora2Pg/PLSQL.pm blib/lib/Ora2Pg/PLSQL.pm
cp lib/Ora2Pg/MySQL.pm blib/lib/Ora2Pg/MySQL.pm
cp scripts/ora2pg blib/script/ora2pg
"/usr/bin/perl" -MExtUtils::MY -e 'MY->fixin(shift)' -- blib/script/ora2pg
cp scripts/ora2pg_scanner blib/script/ora2pg_scanner
"/usr/bin/perl" -MExtUtils::MY -e 'MY->fixin(shift)' -- blib/script/ora2pg_scanner
Manifying 1 pod document
kress@babbage:~/software/ora2pg-19.1$ sudo make install
[sudo] password for kress:
Manifying 1 pod document
Installing /usr/local/share/perl/5.22.1/Ora2Pg.pm
Installing /usr/local/share/perl/5.22.1/Ora2Pg/PLSQL.pm
Installing /usr/local/share/perl/5.22.1/Ora2Pg/MySQL.pm
Installing /usr/local/share/perl/5.22.1/Ora2Pg/GEOM.pm
Installing /usr/local/man/man3/ora2pg.3
Installing /usr/local/bin/ora2pg
Installing /usr/local/bin/ora2pg_scanner
Installing default configuration file (ora2pg.conf.dist) to /etc/ora2pg
Appending installation info to /usr/local/lib/x86_64-linux-gnu/perl/5.22.1/perllocal.pod
```

As output shows, the default installation is to /usr/local (yes! good for them.).

usage

Read the documentation (http://ora2pg.darold.net/documentation.html#Installing-Ora2Pg) fully before starting in.

The ora2pg perl script is the core of all functionality. Behaviour is controlled by both a plethora of command line options and a specified config file or directory tree of config files and scripts (known as a 'project'). Through these command line options and scripts, the database transfer process can be controlled at any level of detail, including translation of database functions, sequences, indexes and data types for individual table elements. The config file is specified with a '-c' option and defaults to /etc/ora2pg/ora2pg.conf. Tables can be individually included or excluded with wildcard or complete specification.

Output can be in the form of SQL dumped to STDOUT (default) that can be imported into the PostgreSQL database with psql, or the program can input data directly into the target PostgreSQL database.

There are three parts to an ordinary database port:

1. export the Oracle Database Design Language (DDL) to SQL.
2. import DDL to postgres
3. copy data

Here the DDL refers to definition of tables, constraints, etc. Essentially the overall structure of the database. In our case, we define all the tables independently and all we want to do is copy data so only #3 is relevant. This greatly simplifies our task. We can set up a direct transfer of data from the Oracle to PostgreSQL database by specifying postgres connection parameters in the conf file, but safer to first dump to an SQL file that we can pre-inspect before importing.

Useful command line options include:

- -c | --conf conf_file : to specify non-standard configuration file
- -o | --out base_sql_outfile : defaults to output.sql
- --project_base DIR : base dir for ora2pg project trees. Default is current directory.

In setting up the export configuration file, it is useful to run tests to make sure it is doing what you want. Items specified on the command line take precedence, so if for example you have a 'TYPE COPY' entry in the conf file, specifying '-t SHOW_COLUMNS' on the command line will override this and output all the columns that would be exported and the input and output types.

After importing lots of new data, it is useful to run ANALYZE and VACUUM on all modified tables to set up proper indexing foreign keys etc.

AQMS data transfer

Though it is possible to read from the Oracle db directly into the postgres database, we will do this in two steps.

data export

When exporting to file, ora2pg will create a master output.sql file (default) in current directory (default) which includes other generated files with names like 'TABLENAME_output.sql'.

First test export from archdb on shuksan was not particularly resource intensive (low network, cpu, mem). Peak net was < 10Mb/sec and most of a cpu.

88 tables were exported:

```
Total number of rows: 102884567
```

```
Top 10 of tables sorted by number of rows:
        [1] TABLE ASSOCWAE has 19854881 rows
        [2] TABLE WAVEFORM has 19625225 rows
        [3] TABLE ASSOCAMO has 18699168 rows
        [4] TABLE AMP has 11599192 rows
        [5] TABLE AMPSET has 10184095 rows
        [6] TABLE FILENAME has 6312910 rows
        [7] TABLE TRIG_CHANNEL has 4279048 rows
        [8] TABLE ASSOCARO has 2836939 rows
        [9] TABLE ARRIVAL has 2564892 rows
        [10] TABLE ASSOCAMM has 1369480 rows
```

Average export rate was close to 10000 recs/sec, which suggests 10288 seconds or 2:51:28.

The first try failed on an active table due to an Oracle consistency failure from exceeding rollback capacity after 1.5 hours of transfer. This is probably because both the active table should not have been exported and because the transaction isolation level is set very high by default (see http://elliot.land/post/sql-transaction-isolation-levels-explained). Oracle only supports strict 'serializable' or 'read commited' (a.k.a. 'committed') https://blogs.oracle.com/oraclemagazine/on-transaction-isolation-levels. 'TRANSACTION committed' is probably sufficient for our purposes, though our second attempt with volatile tables removed and 'TRANSACTION serializable' succeeded.

Final test actual export of 9.3G data took 2 hour 49 minutes.

data load

For this part, from the directory containing sql dump:

```
kress@babbage:~/BabProjects/ora2pg/test_project/test1$ psql -h 192.168.122.11 -U trinetdb archdb < output
```

First try failed because it could not truncate table with foreign references. Suggested truncating at one time or using CASCADE option. This is not an option in ora2db, but a global change to sql dump files can be made with something like:

```
kress@babbage:~/BabProjects/ora2pg/test_project/test1$ sed -i '/^TRUNCATE/s/;/ CASCADE;/' *.sql
```

This tells the loader to truncate any tables with foreign key references in this table, which is what we intend to do anyway. If instead you want to just eliminate the truncate, just specify 'TRUNCATE_TABLE 0' in the conf file.

Needed to modify order of includes quite a bit in output.sql to satisfy foreign key and parent/child relationships. Modified version is in modded_output.sql, and should not need to be modified for the final export. Should just swap this in place of the output.sql generated by ora2pg.

Lots more issues related to inconsistent constraints between our oracle database and the more canonical pg schema. Most of these were handled by modifying data on oracle or making constraints on our pg database more consistent with our usage. One exception is that Event.version defaulted to NULL on Oracle, but on pg the constraint was 'not null default 0'. The easiest way to fix this particular issue was to modify the EVENT sql file with a command like:

```
kress@babbage:~/BabProjects/ora2pg/test_project/test1$ awk -F'\t' '$10=="\\N" {$10=0} 1' OFS='\t' EVENT_o
```

This command is combined with the above sed command in the process_sql.sh script.

For some reason the eventtype constraint eventtype02 constraint never got 'le', 're' or 'ts' types, yet these were plentiful in the etype field. This constraint was clearly not active in oracle, but was in postgres (with the obsolete constraints). These should not be in the database and should be verified as all converted before final export.

Final data load into postgres database took a remarkable 23 minutes. This suggests total downtime between start of export and end of import could be as short as 3.5 hours.

Retrieved from "https://internal.pnsn.org/LOCAL/WikiDocs/index.php?title=Migrating_data_from_Oracle_to_PostgreSQL&oldid=49280"
Categories: Database | PostgreSQL | Oracle | AQMS

- This page was last modified on 16 October 2018, at 15:51.
- This page has been accessed 274 times.