MA14
Wir vernetzen Wien.

Use Oracle from PostgreSQL

oracle_fdw in migration scenarios

Laurenz Albe <laurenz.albe@wien.gv.at>

2012-10-04

What is oracle_fdw?

- it allows read access to Oracle tables as if they were PostgreSQL tables
- an SQL/MED Foreign Data Wrapper for Oracle
- a PostgreSQL server extension
- project page: http://oracle-fdw.projects.postgresql.org/

Foreign Data Wrapper concepts

PostgreSQL object	corresponds to
Foreign Data Wrapper	Oracle DB software
Foreign Server	Oracle instance
User Mapping	Oracle credentials
Foreign Table	Oracle table/view

A simple example

```
pgdb=# CREATE EXTENSION oracle_fdw;
pgdb=# CREATE SERVER oradb FOREIGN DATA WRAPPER
   oracle_fdw OPTIONS
   (dbserver '//dbserver.mydomain.com/ORADB');
pgdb=# GRANT USAGE ON FOREIGN SERVER oradb
   TO pguser;
pgdb=# \connect pgdb pguser
pgdb=> CREATE USER MAPPING FOR pguser
   SERVER oradb
   OPTIONS (user 'orauser', password 'orapwd');
pgdb=> CREATE FOREIGN TABLE people (
          id        integer       NOT NULL,
          name      varchar(30),
          birthday  date          NOT NULL
      ) SERVER oradb OPTIONS (table 'PEOPLE');
```

Data migration with oracle_fdw

```
BEGIN;
CREATE TABLE loc_people AS
      (SELECT * FROM people);
ALTER TABLE loc_people
      ADD CONSTRAINT people_pkey
      PRIMARY KEY(id);
DROP FOREIGN TABLE people;
ALTER TABLE loc_people
      RENAME TO people;
COMMIT;
```

Special Features of oracle_fdw

- Automatic encoding management
- Data type conversion
- **`WHERE`** clause push down
- Only fetch required columns
- **`EXPLAIN`** support

New in 9.2:

- Statistics on foreign tables
- No re-check of pushed down **`WHERE`** clauses

Feature: Automatic encoding management

C'est trÃ¨s important!

Automatically sets the Oracle client encoding to the value of the PostgreSQL server encoding.

Override with **`nls_lang`** option on the FDW object (useful for **`SQL_ASCII`**).

Feature: Data type conversion

This could be done with views and casts, but it is more convenient if the FDW supports it.

- Allows conversion of matching data types (e.g. **`NUMBER`** → **`numeric`**/**`integer`**/**`double precision`**/**`boolean`**)
- All except binary data can be converted to textual types
- Does not guarantee that all values can be converted (encoding problems, string length, integer maximum, …)

Feature: **WHERE** pushdown, column elimination

```
EXPLAIN SELECT name FROM people WHERE id=2;

QUERY PLAN
-------------------------------------------------
 Foreign Scan on people  (cost=10000.00..10000.00
     rows=1 width=75)
   Filter: (id = 2)
   Oracle query: SELECT
     /*522d754ad26bc932e0a8984763d2b374*/
     "ID", "NAME" FROM PEOPLE WHERE ("ID" = 2)
(3 rows)
```

Feature: **`EXPLAIN`** support

- **`EXPLAIN`** shows the remote query
- **`EXPLAIN VERBOSE`** shows the remote query plan (requires **`SELECT`** privilege on **`V$SQL`** and **`V$SQL_PLAN`**)

Feature: **EXPLAIN** support

```
EXPLAIN VERBOSE SELECT name FROM people WHERE id=2;

QUERY PLAN
-------------------------------------------------
 Foreign Scan on pguser.people
 (cost=10000.00..10000.00 rows=1 width=75)
   Output: name
   Filter: (people.id = 2)
   Oracle query:
     SELECT /*522d754ad26bc932e0a8984763d2b374*/
     "ID", "NAME" FROM PEOPLE WHERE ("ID" = 2)
   Oracle plan: SELECT STATEMENT
   Oracle plan:   TABLE ACCESS BY INDEX ROWID PEOPLE
   Oracle plan:     INDEX UNIQUE SCAN PEOPLE_PKEY
                                    (condition "ID"=2)
(7 rows)
```

(Mis-)Feature: Estimates in 9.1

```
EXPLAIN ANALYZE SELECT id FROM people
  WHERE name LIKE 'L%'
  AND birthday < now() - '80 years'::interval;

QUERY PLAN
-------------------------------------------------
Foreign Scan on people
     (cost=10000.00..10000.00 rows=4877 width=4)
     (actual time=1.179..102.861 rows=673 loops=1)
   Filter: (((name)::text ~~ 'L%'::text) AND
     (birthday < (now() - '80 years'::interval)))
   Oracle query:
     SELECT /*90af296c03d5552a300f876e9108904d*/
     "ID", "NAME", "BIRTHDAY" FROM PEOPLE
     WHERE ("NAME" LIKE 'L%' ESCAPE '\')
 Total runtime: 103.690 ms
```

Feature: `ANALYZE` in 9.2

- `ANALYZE` collects statistics for remote tables
- Must be called for each foreign table explicitly
- Good estimates even without asking Oracle
- Performs a full table scan on Oracle

Feature: Estimates in 9.2

```
EXPLAIN ANALYZE SELECT id FROM people
  WHERE name LIKE 'L%'
  AND birthday < now() - '80 years'::interval;

QUERY PLAN
-------------------------------------------------
Foreign Scan on people
     (cost=10000.00..10000.00 rows=412 width=4)
     (actual time=1.556..116.143 rows=673 loops=1)
   Filter:
     (birthday < (now() - '80 years'::interval))
   Rows Removed by Filter: 4015
   Oracle query:
     SELECT /*90af296c03d5552a300f876e9108904d*/
     "ID", "NAME", "BIRTHDAY" FROM PEOPLE
     WHERE ("NAME" LIKE 'L%' ESCAPE '\')
 Total runtime: 116.775 ms
```

Problems

- Still beta (awaiting your feedback!)
- **`NCLOB`** and other rare data types not supported
- No Oracle support for some rare server encodings (non-ASCII characters become '**`?`**')
- Bad Oracle cost estimates (disabled by default)
- Incompatible LDAP libraries (build PostgreSQL **`--without-ldap`**)

Usage for migration

- Coexist: integrate with existing Oracle databases
- Migrate data: extract, transform, load (ETL)

Coexist with Oracle

Usually one cannot/does not want to migrate all Oracle databases at once.
Then how can you migrate an Oracle database with database links?

oracle_fdw can save the day!

This can also be a problem for new applications:
“We cannot use PostgreSQL because we have to access this certain Oracle table.”

Migration: extract data from Oracle

Oracle deliberately does not provide tools for that (SQL*Plus does not work well).
You can use third-party tools or write your own.

oracle_fdw does it for you!

Can also be used to extract data from Oracle to text files for other purposes:

```
pgdb=> \copy (SELECT * FROM people)
         TO 'people.csv' (FORMAT 'csv')
```

Migration: transform data

Often data need to be converted during migration:

- different string encoding:
  oracle_fdw does this efficiently
- different data types:
  oracle_fdw does this efficiently
- “data cleansing” or mapping to other values:
  can sometimes be implemented by joins on the PostgreSQL or Oracle side (views).

oracle_fdw can perform simple transformations.

Migration: load into PostgreSQL

Usually done with **`COPY FROM`** SQL statement.

This is the easiest part.

oracle_fdw is slower than **`COPY`**, but can avoid the need for an operating system file as intermediary data store.

Migration: advantages of oracle_fdw

For simple migration scenarios, oracle_fdw is a fast and simple migration tool:

- all written in C
- all can be done in one SQL statement
- Oracle prefetching for fewer client-server round trips
- no intermediary files
- binary values are transferred binary, no conversion necessary
- support for “legacy” data: Oracle 8, deprecated types **`LONG`** and **`LONG RAW`**

Migration: limits of oracle_fdw

oracle_fdw will not help with table/index/function definitions.
ora2pg (http://ora2pg.darold.net/config.html) can generate foreign table definitions for oracle_fdw.

An alternative is a simple "schema converter": PostgreSQL function that uses foreign tables for **`USER_TABLES`** and **`USER_TAB_COLUMNS`** to create foreign tables for everything in an Oracle schema.

What the future could bring

- “join pushdown” of joins between Oracle tables in the same Oracle database
- writeable foreign tables

All this needs added support in core PostgreSQL.

Questions? Suggestions?