10 WAYS TO LOAD DATA INTO POSTGRES

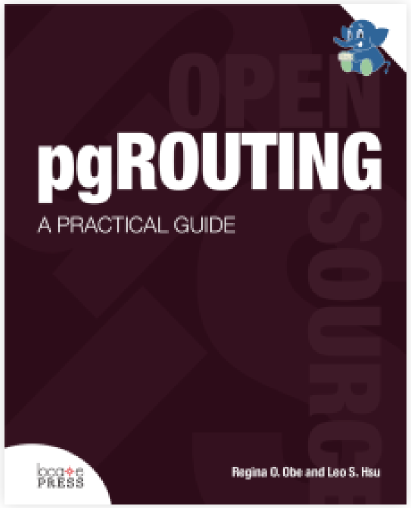
REGINA OBE AND LEO HSU

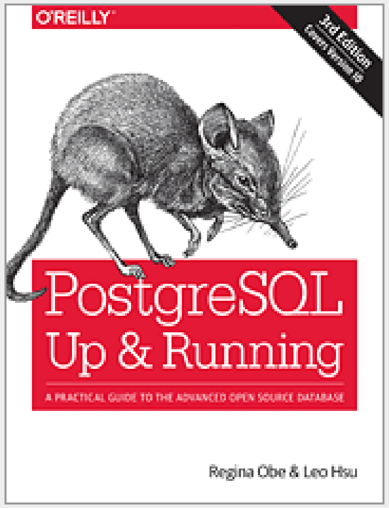
http://www.paragoncorporation.com

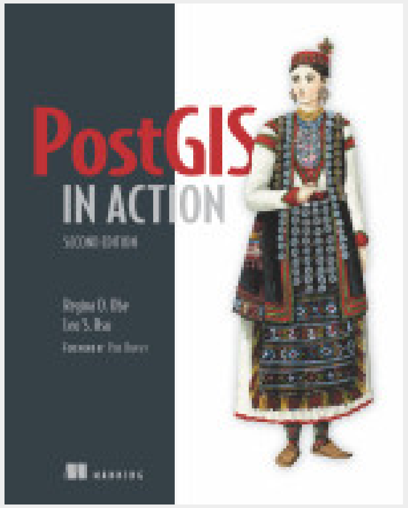
Buy our books! at http://postgis.us/page_buy_book

OUR LATEST BOOK

pgRouting: A Practical Guide http://locatepress.com/pgrouting







CATEGORIES OF LOADING WE'LL COVER

Server-Side

- SQL COPY / COPY FROM PROGRAM
- Large Object storage SQL functions
- Foreign Data Wrappers (FDWs)

Client-Side

- PSQL \copy and \copy FROM PROGRAM
- PSQL Large Object support functions
- Other commandline tools: ogr2ogr, shp2pgsql
- Need not be on same server as Postgres service

LOADING DELIMITED FILES WITH SQL COPY (SERVER SIDE)

- postgres daemon account needs to have access to files
- User has to have super user rights to Postgres service

STEP 1: CREATE STAGING TABLE

Has to match the structure of the file. Using film locations - https://data.sfgov.org/api/views/yitu-d5am/rows.csv?accessType=DOWNLOAD

```
CREATE TABLE film_locations
    (title text ,
     release_year integer ,
     locations text ,
     fun_facts text ,
     production_company text ,
     distributor text ,
     director text ,
     writer text ,
     actor_1 text ,
     actor_2 text ,
     actor_3 text );
```

STEP 2 (FROM FILE): LOAD THE DATA USING SQL COPY

```
COPY film_locations
  FROM '/data_talk/csvs/Film_Locations_in_San_Francisco.csv' HEADER CSV DELIMITER ',';
```

STEP 2 (OUTPUT FROM PROGRAM): LOAD THE DATA USING SQL COPY FROM PROGRAM

Requires PostgreSQL 9.3+

```
COPY film_locations
FROM PROGRAM 'wget -q -O - "$@" "https://data.sfgov.org/api/views/yitu-d5am/rows.csv?accessType=DOWNL
    HEADER CSV DELIMITER ',';
```

LOADING DELIMITED FILES WITH PSQL \COPY (CLIENT SIDE)

- psql client needs to have access to the files
- User initiating does not need super user rights to database, but needs to have permissions to the files
- Could be slow if postgres server is not on same local network as client.

STEP 1: CREATE STAGING TABLE

Has to exactly match the structure of the file. Using film locations - https://data.sfgov.org/api/views/yitu-d5am/rows.csv?accessType=DOWNLOAD

```
CREATE TABLE film_locations
    (title text ,
     release_year integer ,
     locations text ,
     fun_facts text ,
     production_company text ,
     distributor text ,
     director text ,
     writer text ,
     actor_1 text ,
     actor_2 text ,
     actor_3 text );
```

STEP 2: LOAD THE DATA WITH \COPY FROM

```
\copy film_locations FROM '/data_talk/csvs/Film_Locations_in_San_Francisco.csv' HEADER CSV DELIMiTER
```

STEP 2 ALTERNATIVE: LOAD THE DATA USING \COPY FROM PROGRAM

Requires psql compiled for PostgreSQL 9.3+

```
\copy film_locations FROM PROGRAM 'wget -q -O - "$@" "https://data.sfgov.org/api/views/yitu-d5am/rows
```

SERVER SIDE: LOADING BINARY FILES

Loading documents and images into a database table from server's file system.

Use COPY FROM PROGRAM (PostgreSQL 9.3+) in conjunction with Large Object support (LO)

STEP 1: CREATE STAGING TABLE

```
CREATE TABLE tmp_docs(file_name text PRIMARY KEY);
```

STEP 2: GET LIST OF FILES

Pull list from folder with COPY FROM PROGRAM

Windows

```
COPY tmp_docs FROM PROGRAM 'dir C:\data /b /S'  WITH (format 'csv');
```

Unix/Linux

```
COPY tmp_docs FROM PROGRAM 'ls /data/* -R'  WITH (format 'csv');
```

STEP 2: ADD FIELDS TO HOLD FILE LINK ID AND BLOB OF THE FILES

```sql
ALTER TABLE tmp_docs ADD COLUMN doc bytea, ADD COLUMN doc_oid oid;
```

STEP 3: LOAD THE BINARY DATA

```sql
-- add the document to large object storage and return the link id
UPDATE tmp_docs SET doc_oid = lo_import(filename);

-- pull document from large object storage
UPDATE tmp_docs SET doc = lo_get(doc_oid);

-- delete the files from large object storage
SELECT lo_unlink(doc_oid)
FROM tmp_docs;
```

CLIENT SIDE: LOADING BINARY FILES USING PSQL

Loading documents and images into a database table from client's file system.

Use PSQL \copy and \lo_* functions and SQL to generate a load script

STEP 1: CREATE STAGING TABLE

Note this is same as what we did for the server side approach

```
CREATE TABLE tmp_docs(file_name text PRIMARY KEY);
```

STEP 2: GET LIST OF FILES

Pull list from folder with PSQL \copy FROM PROGRAM (psql packaged with 9.3+)

Windows

```
\copy tmp_docs FROM PROGRAM 'dir C:\data /b /S'  WITH (format 'csv');
```

Unix/Linux

```
\copy tmp_docs FROM PROGRAM 'ls /data/*'  WITH (format 'csv');
```

STEP 2: ADD FIELDS TO HOLD FILE LINK ID AND BLOB OF THE FILES

```
ALTER TABLE tmp_docs ADD COLUMN doc bytea, ADD COLUMN doc_oid oid;
```

STEP 3: GENERATE A LOAD SCRIPT FILE

\t on returns only tuples (no header), and \x off turns off expanded mode, and \a toggles axis align

```
\o /temp/loadscript.psql
\t on
\x off
\a
SELECT '\lo_import ' || quote_literal(replace(file_name, '\', '/'))
|| '
UPDATE tmp_docs SET doc_oid = :LASTOID
    WHERE  file_name = ' || quote_literal(file_name) || ';'
FROM tmp_docs;
\o
```

STEP 4: RUN THE LOAD SCRIPT FILE GENERATED IN STEP 3

the load script file will look something like this

```
\lo_import '/scans/file1.pdf'
UPDATE tmp_docs SET doc_oid = :LASTOID
    WHERE  file_name = E'/scans/file1.pdf';
\lo_import '/scans/file2.pdf'
UPDATE tmp_docs SET doc_oid = :LASTOID
    WHERE  file_name = E'/scans/file2.pdf';
```

run the load script file generated in step 3

```
\i /temp/loadscript.psql
```

STEP 5: SAME AS SERVER SIDE, USING SERVER SIDE FUNCTIONS GRAB THE BLOB AND DELETE THE FILE FROM LARGE STORAGE

-- pull document from large object storage

```
UPDATE tmp_docs SET doc = lo_get(doc_oid);

-- delete the files from large object storage
SELECT lo_unlink(doc_oid)
FROM tmp_docs;
```

USING FOREIGN DATA WRAPPERS TO LOAD DATA

- **file_fdw**: use to read flat files and flat outputs. New in PostgreSQL 10 can read from commandline programs
- **postgres_fdw**: use to query other postgres servers
- **ogr_fdw** - use to query and load spatial formats and also other relational and flat (e.g. spreadsheets, odbc data sources, dbase files, openstreetmap data
- **file_text_array** - loads each row of data into an array - great where number of columns on each row is not the same like data consisting of orders on one row followed by line items.
- **Honorable mentions**: multicorn, odbc_fdw, mysql_fdw, oracle_fdw, db2_fdw, tds_fdw

FILE_FDW

- https://www.postgresql.org/docs/current/static/file-fdw.html
- Generally available with most PostgreSQL packages, may require installing postgresql-contrib if no by default included
- Requires super user to create a foreign table, but user mappings control access.
- New in PostgreSQL 10: can read from output of programs similar to COPY FROM PROGRAM.

STEP 1: INSTALL EXTENSION AND CREATE FILE_FDW FOREIGN SERVER

```
CREATE EXTENSION file_fdw;
CREATE SERVER svr_file FOREIGN DATA WRAPPER file_fdw;
```

STEP 2 (FILE VERSION): CREATE FOREIGN TABLE TO A FILE SYSTEM FILE

```
CREATE FOREIGN TABLE fdt_film_locations
   (title text ,
    release_year integer ,
    locations text ,
    fun_facts text ,
    production_company text ,
    distributor text ,
    director text ,
    writer text ,
    actor_1 text ,
    actor_2 text ,
    actor_3 text )
    SERVER svr_file
    OPTIONS ( format 'csv', header 'true',
        filename '/data_talk/csvs/Film_Locations_in_San_Francisco.csv',
        delimiter ',',
        null '');
```

STEP 2 (PROGRAM VERSION): CREATE FOREIGN TABLE FROM PROGRAM OUTPUT

Requires PostgreSQL 10+. This will pull the website data on every query of table.

```
CREATE FOREIGN TABLE fdt_film_locations
   (title text ,
    release_year integer ,
    locations text ,
    fun_facts text ,
    production_company text ,
    distributor text ,
    director text ,
    writer text ,
    actor_1 text ,
    actor_2 text ,
    actor_3 text )
    SERVER svr_file
    OPTIONS ( format 'csv', header 'true',
     program 'wget -q -O - "$@" "https://data.sfgov.org/api/views/yitu-d5am/rows.
     delimiter ',',
     null '');
```

POSTGRES_FDW: READ FROM OTHER POSTGRES SERVERS

- Part of standard extension offering so should already have the binaries
- Can read from higher/lower postgres versions, but some features are disabled if both not of same higher version.
- Requires super user to create a foreign table, but user mappings control access.
- New in PostgreSQL 10: Aggregates can be pushed down, which means things like COUNT(*), MAX(*) etc are much faster across databases. More joins can be pushed to remote server thus making cross joins between two databases faster.

STEP 1:INSTALL THE EXTENSION IN YOUR DATABASE

```
CREATE EXTENSION postgres_fdw;
```

STEP 2:CREATE FOREIGN SERVER

```
CREATE SERVER remote_db
 FOREIGN DATA WRAPPER postgres_fdw
 OPTIONS (host 'faraway.host.com', dbname 'db', port '5432');
```

STEP 3:CREATE USER MAPPINGS (CAN BE A GROUP OR USER)

```
CREATE USER MAPPING FOR public SERVER remote_db OPTIONS (user 'pubinfo', passworc
```

STEP 4:LINK IN THE TABLES

```
CREATE SCHEMA remote_public;
-- requires PostgreSQL 9.5
IMPORT FOREIGN SCHEMA public FROM SERVER remote_db INTO remote_public;
```

POSTGRESQL + GDAL (OGR) ~ POSTGIS = OGR_FDW POSTGRESQL MORE THAN SPATIAL FOREIGN DATA WRAPPER

Doesn't require PostGIS to use, but will expose spatial columns as PostGIS geometry if PostGIS is installed.

USE OGR_FDW EXTENSION

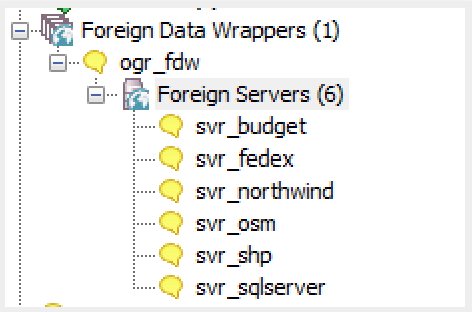
If you have all sorts of data of both a spatial and non-spatial flavor to tame, make sure you have ogr_fdw foreign data wrapper in your tool belt.

- For windows users using EDB distribution, it's part of PostGIS bundle (versions 2.2 and up) on application stackbuilder.
- For windows/linux/mac desktop users, it's part of the BigSQL PostGIS package.
- For CentOS/Red Hat/Scientific etc, it's available via yum.postgresql.org
- Available via debian and apt.postgresql.org
- For others, if you have PostGIS with GDAL support, just need postgresql dev package to compile. Download the source https://github.com/pramsey/pgsql-ogr-fdw

WHAT CAN OGR_FDW READ?

You have the combined power of Geospatial Data Abstraction Layer (GDAL), PostgreSQL, and any PostgreSQL extension you want (including PostGIS) working seamlessly together. So many kinds of data you can query and take advantage of PostgreSQL functions and any extension functions and types such as PostGIS, hstore, built-in json/jsonb to tame your data.

- Spreadsheets
- ODBC datasources
- Other relational
- OSM files (OSM, PBF)
- Dbase files
- ESRI Shapefiles
- Spatial web services
- Many more


Foreign Data Wrappers (1)
ogr_fdw
Foreign Servers (6)
svr_budget
svr_fedex
svr_northwind
svr_osm
svr_shp
svr_sqlserver

INSTALL BINARIES

Make sure to use version for your PostgreSQL, examples below are for 10

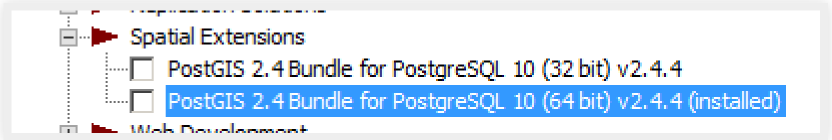
- Yum (CentOS, RedHat going via yum.postgresql.org) -

```
yum install ogr_fdw10
```

- Debian/Ubuntu (via apt.postgresql.org) -

```
apt install postgresql-10-ogr-fdw
```

- Windows via application Stackbuilder - included as part of PostGIS bundle 2.2 and above bundles


Spatial Extensions
PostGIS 2.4 Bundle for PostgreSQL 10 (32 bit) v2.4.4
PostGIS 2.4 Bundle for PostgreSQL 10 (64 bit) v2.4.4 (installed)

.

- BigSQL (linux/windows/Mac) - included as part of the PostGIS install:

```
pgc install postgis24-pg10
```

ENABLE IT IN YOUR DATABASE

```
CREATE EXTENSION ogr_fdw;
```

LOAD IN FOLDER OF CSV FILES

```
CREATE SCHEMA IF NOT EXISTS staging;
CREATE SERVER svr_csv FOREIGN DATA WRAPPER ogr_fdw
OPTIONS (datasource '/fdw_data/csvs', format 'CSV');
-- requires PostgreSQL 9.5+
IMPORT FOREIGN SCHEMA ogr_all FROM SERVER svr_csv INTO staging;
```

OTHER RELATIONAL DATABASES

Format for SQL Server ODBC 'ODBC:your_user/your_password@yourDSN,table1,table2'. ODBC can be slow with a lot of tables (more than 150) so filter list if you have over 200 tables

```
CREATE SERVER svr_sqlserver FOREIGN DATA WRAPPER ogr_fdw
OPTIONS (datasource 'ODBC:pguser/whatever@MSSQLTest,dbo.IssueLog,dbo.IssueNotes',
format 'ODBC'
);
CREATE SCHEMA IF NOT EXISTS ss;
IMPORT FOREIGN SCHEMA "dbo."
FROM SERVER svr_sqlserver INTO ss;
```

```
\dE ss.*
```

```
                  List of relations
Schema |       Name       |      Type       |  Owner
--------+-----------------+----------------+----------
ss     | dbo_issuelog   | foreign table | postgres
ss     | dbo_issuenotes | foreign table | postgres
(2 rows)
```

SPREADSHEETS

Each workbook is considered a server and each sheet a table

```
CREATE SERVER svr_currency_rates
   FOREIGN DATA WRAPPER ogr_fdw
  OPTIONS (datasource '/fdw_data/ExchangeRates.xlsx',format 'XLSX',
    config_options 'OGR_XLSX_HEADERS=FORCE');
CREATE SCHEMA staging;
-- link only 2 spreadsheets preserve headers (requires PostgreSQL 9.5 to use IMPO
IMPORT FOREIGN SCHEMA ogr_all  LIMIT TO (EUR, USD)
        FROM SERVER svr_currency_rates INTO staging
        OPTIONS (launder_column_names 'false');
```

FILE_TEXT_ARRAY: LOADING DELIMITED FILES AND JAGGED FILES WITH FILE_TEXT_ARRAY

- Usually not available from distros but fairly easy compile
- We have a windows 32/64-bit builds https://tinyurl.com/y8bojebk
- Source code here: https://github.com/adunstan/file_text_array_fdw (note different branch for each version of PostgreSQL)
- New in PostgreSQL 10: can read from output of programs similar to COPY FROM PROGRAM.

STEP 1: CREATE EXTENSION AND SERVER

```
CREATE EXTENSION file_textarray_fdw;
CREATE SERVER file_ta_server FOREIGN DATA WRAPPER file_textarray_fdw;
CREATE USER MAPPING FOR public SERVER file_ta_server;
```

STEP 2 (FILE VERSION): CREATE FOREIGN TABLE

```
CREATE FOREIGN TABLE fdt_film_locations_ta( x text[] ) SERVER file_ta_server
OPTIONS (filename '/data_talk/csvs/Film_Locations_in_San_Francisco.csv', encoding
```

STEP 2 (PROGRAM VERSION): CREATE FOREIGN TABLE

Requires PostgreSQL 10+

```
CREATE FOREIGN TABLE fdt_film_locations_ta( x text[] ) SERVER file_ta_server
OPTIONS (program 'wget -q -O - "$@" "https://data.sfgov.org/api/views/yitu-d5am/r
```

COMMANDLINE TOOLS

Commonly available Open source command-line when you have PostgreSQL / PostGIS installed.

- shp2pgsql- use to load Dbase and ESRI shapefiles, generally part of based postgis or postgis-gui package
- ogr2ogr - Binaries and packages available for most server and Desktop OS (https://trac.osgeo.org/gdal/wiki/DownloadingGdalBinaries). Use to load any kind of data, specially designed for spatial vector data.

SHP2PGSQL

Part of PostGIS project, often packaged separately as postgis-gui. On EDB Windows Stackbuilder, part of PostGIS bundle. On BigSQL part of postgis package.

Pipe to psql for load

```
export PGPORT=5432
export PGDATABASE=pgopen2018
export PGUSER=postgres
export PGPASSWORD=xxxx
exportPGHOST=localhost
shp2pgsql -s 4269 -D -d data\BART_Lines\Bart_13 bart_lines | psql
```

Output as an sql script

```
shp2pgsql -s 4269 -D -d > bart_lines.sql
```

OGR2OGR: SWISS ARMY KNIFE FOR DATA LOADING

```
ogr2ogr --formats
```

```
Supported Formats:
  JP2ECW -raster,vector- (rov): ERDAS JPEG2000 (SDK 5.3)
  OCI -vector- (rw+): Oracle Spatial
  SOSI -vector- (ro): Norwegian SOSI Standard
  PCIDSK -raster,vector- (rw+v): PCIDSK Database File
  netCDF -raster,vector- (rw+s): Network Common Data Format
  JP2OpenJPEG -raster,vector- (rwv): JPEG-2000 driver based on OpenJPEG library
  PDF -raster,vector- (rw+vs): Geospatial PDF
  DB2ODBC -raster,vector- (rw+): IBM DB2 Spatial Database
  ESRI Shapefile -vector- (rw+v): ESRI Shapefile
  MapInfo File -vector- (rw+v): MapInfo File
  UK .NTF -vector- (ro): UK .NTF
  OGR_SDTS -vector- (ro): SDTS
  S57 -vector- (rw+v): IHO S-57 (ENC)
  DGN -vector- (rw+): Microstation DGN
  OGR_VRT -vector- (rov): VRT - Virtual Datasource
  REC -vector- (ro): EPIInfo .REC
  Memory -vector- (rw+): Memory
  BNA -vector- (rw+v): Atlas BNA
```

OGR2OGR LOAD DATA INTO POSTGRESQL

Can use psql variables or be specified on commandline

Load an OpenStreetMap protobuf file

```
ogr2ogr -f "PostgreSQL" \
        "PG:host=localhost user=postgres password=xxx dbname=pgopen2018" sf.osm.pbf
```

Load a folder of CSV files (folder is called csvs)

```
ogr2ogr -f "PostgreSQL" \
        "PG:host=localhost user=postgres password=xxx dbname=pgopen2018" /data_csv
```

FIN

BUY OUR BOOKS

HTTP://WWW.POSTGIS.US