Loading OSM Data With osm2pgsql

JinXu

2018-11-28

Introduction

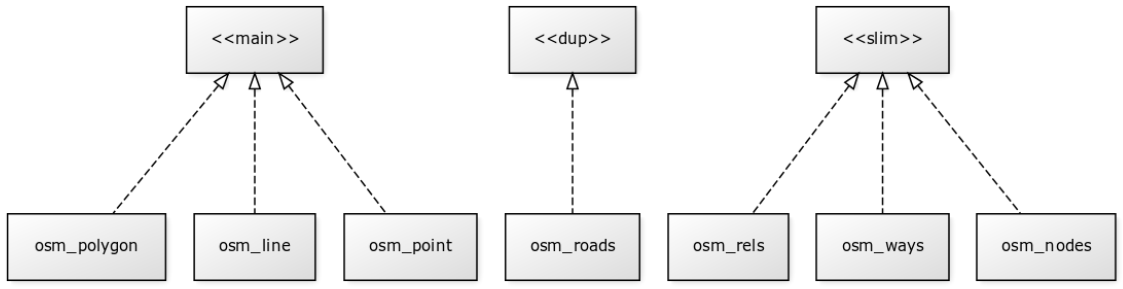
OpenStreetMap (OSM) is a collaborative mapping project to create free editable map of the world. User draw maps with GPS, RS images and other free source data. It is a typical source of VGI (Volunteered Geographic Information) providing up-to-date data as an alternative for authoritative geographic data.
OSM elements, i.e. features on the map, are stored in the database as nodes, ways, and relations:

- Nodes are basically points on the map. It could be independent points such as points of interest (POI), or points on a line, such as the traffic light.

- Ways are lines representing independent lines when it is not closed (such as roads), or polygons when it's closed with tags (such as buildings with tag `building = yes`).

- Relations are logical or spatial relations among OSM elements (whether you can turn right on a road, polygon with a hole, a set of polygons and nodes as one element, etc).

OSM2pgsql is a tool to import OSM data into PostGIS database. It's already installed with the OSGeolive. It can convert OSM data to slim and non-slim mode. The slim mode will keep the data structure as node, ways, and relations, but non-slim mode will transfer data into points, lines, and polygons. In this tutorial, we are only dealing with the non-slim mode because it has geometry column.



The data that we are using here is in pbf format, which is a compressed format similar to xml for geographic data, but it takes less space. Although the OSM database is unstructured, namely it stores attributes with tags other than columns, we can use hstore when loading it with osm2pgsql.

Load Data

Importing OSM data is based on this webpage: SWITCH2OSM, but I skipped some unnecessary steps and made some changes. You can take a look if you are interested.

Data

Original data can be downloaded from Geofabrik where OSM data are updated daily. **But** they are usually large files that take a long time to import. So I clipped the data by a bounding box (N:40.1, S:39.9, E:-75.0, W:-75.2) for only Philadelphia.
FOR THE PURPOSE OF THIS TUTORIAL, PLEASE DOWNLOAD DATA FROM HERE:Philadelphia181108.pb Put the file in your share folder so that you can have access to it in your virtual machine. Run the virtual machine and drag the file to the desktop to make your life easier.
The next steps will all be done on the Linux Terminal. These codes can not be copied, so you have to type these codes on the black window. To avoid misunderstanding and mistyping, here's some explanation for some symbols in the codes:

- $: The beginning of the code. It will come after your username so you **DO NOT** have to type it. I just added it here to make it clear that it's the beginning of the code.

- > : Continuing the code in the previous line. It will appear when you wrap your code to the next line. **DO NOT** type it.

- \ : To continue the code in the next line. You **NEED** to type it at the end of the line to wrap the code. But remember **DO NOT** type blank space after typing it, otherwise the code can not be wrapped.

Remember, Linux Shell is **CASE SENSITIVE.**

Getting Ready

Check Version

Type only the lines after $ and it will show the result of checking versions. You need to make sure you have **at least** these versions:

```
$ proj
Rel. 4.8.0, 6 March 2012
usage: proj [ -beEfiIlormsStTvVwW [args] ] [ +opts[=arg] ] [ files ]
$ psql --version
psql (PostgreSQL) 9.3.4
geos-config --version
3.4.2
```

```
$ osm2pgsql --version
osm2pgsql SVN version 0.85.0 (64bit id space)
$ osmconvert -h | head -n2
osmconvert 0.7T  Parameter Overvie
```

Changing Overcommit (optional)

This step is optional. It makes the import faster.

```
$ sudo tee /etc/sysctl.d/60-overcommit.conf <<EOF
> vm.overcommit_memory=1
> EOF
$ sudo sysctl -p /etc/sysctl.d/60-overcommit.conf
```

Loading OSM Data

Create Database

It creates a database called osmphl, and adds the postgis and hstore extensions. The hstore extension is important here because OSM database are unstructured.

```
$ createdb osmphl
$ psql -d osmphl -c 'CREATE EXTENSION postgis; CREATE EXTENSION hstore;
'
```

- `createdb` create database. Put the name of database after it.

- `-d | --database` database. You can also change the '-d' into '–database' in the code. Put the name of the database you just created here

- `-c | --create` create. You can also change the '-c' into '–create' in the code. The line behind it is SQL query to create extension.

If you open your pgAdmin and refresh the server now, it will appear in the database. Make sure hstore is in the extension.

Import

This code import the pbf file into the database that we just created.

```
$ osm2pgsql -c -d osmphl -C 15000 --hstore --extra-attributes --multi-g
eometry \
> ~/Desktop/Philadelphia181108.pbf
```

- `-c | --create` it create a table in the database.

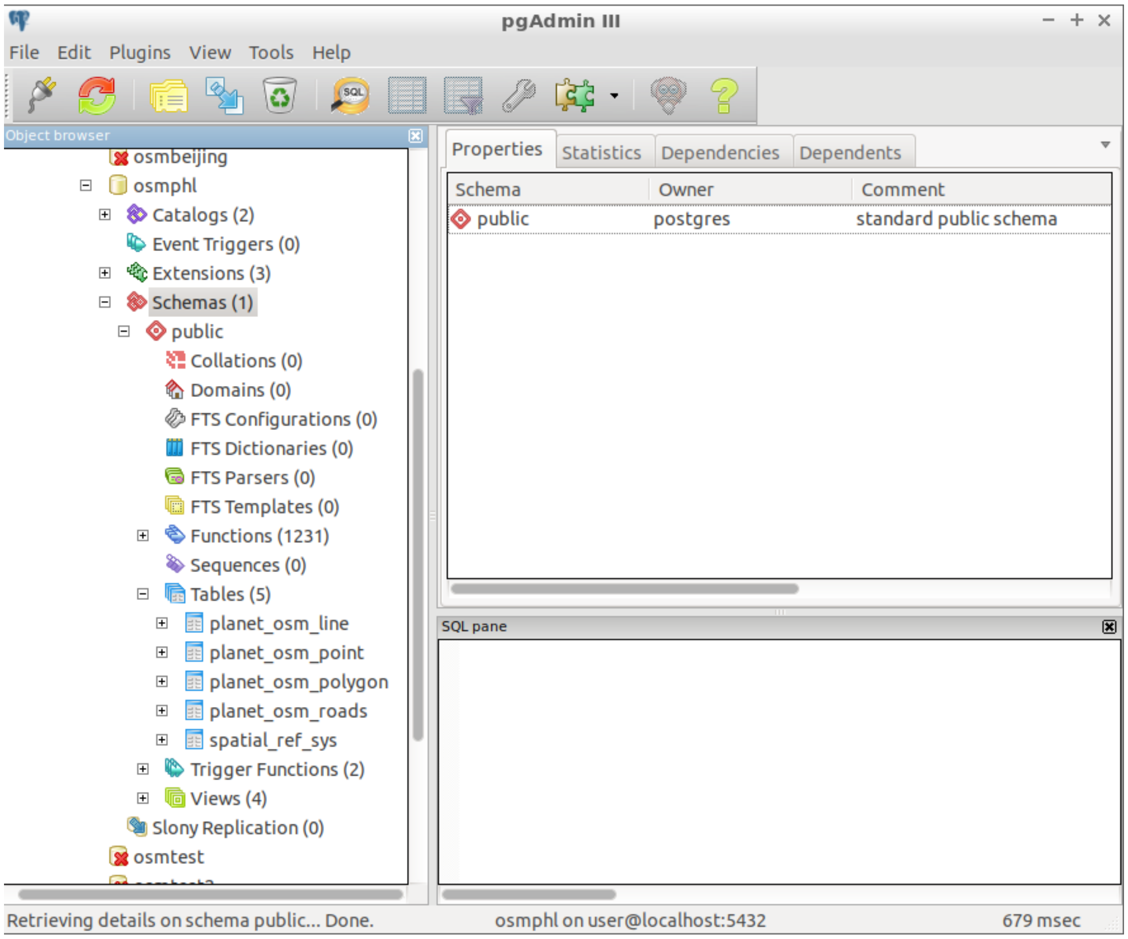
- `-d | --database` to specify which database should the data be imported.
- `-C | --cache` Capital C. It assign some space for importing data. Number after it is the amount of cache assigned. Maximum number is 30,000.

- `-k | --hstore` import tags as hstore. It's important to include this one.

- `--extra-attributes` to keep metadata such as version.

- `--multi-geometry` to keep the multi-geometry in the data. For example, polygon with a hole will not be separated as two polygons.

Now refresh the local server on your pgAdmin and take a look at the public schema in the database. There are 4 tables added into public schema. The points, lines, polygons are OSM features, and roads is duplicating some lines that are tags as roads.



```
osmbeijing
⊟ osmphl
  ⊞ Catalogs (2)
    Event Triggers (0)
  ⊞ Extensions (3)
  ⊟ Schemas (1)
    ⊟ public
        Collations (0)
        Domains (0)
        FTS Configurations (0)
        FTS Dictionaries (0)
        FTS Parsers (0)
        FTS Templates (0)
      ⊞ Functions (1231)
        Sequences (0)
      ⊟ Tables (5)
        ⊞ planet_osm_line
        ⊞ planet_osm_point
        ⊞ planet_osm_polygon
        ⊞ planet_osm_roads
        ⊞ spatial_ref_sys
      ⊞ Trigger Functions (2)
      ⊞ Views (4)
    Slony Replication (0)
osmtest
```

Schema	Owner	Comment
public	postgres	standard public schema

Retrieving details on schema public... Done.   osmphl on user@localhost:5432   679 msec

Simple Manipulation

These steps are done in SQL so we don't need the Linux Terminal anymore and you can copy the code into pgAdmin or QGIS.

Simple Select

Take a look for how it looks like:

```
SET search_path TO public;
SELECT * FROM planet_osm_line;
```

Notice that he tables that we just imported do not have PKs, and you will notice that there are some repeating ids. Although OSM id is the PK in the database, osm2pgSQL will split some features and the OSM id here is not the PK.
The way column here is the geometry and the tags records the key and value. Notice that the user and uid keys are missing values. That's because the data that we are dealing with are from a German website under the EU data protection regulation.

Select Hstore

The tags are stored as hstore. It's store the value of keys rather than creating columns. The default style for importing OSM data into postgres will create several columns for some keys and store the rest into hstore. Therefore, we can not select as how we usually do if we want to select by the value of a tag.
Here's an example for selecting the top 10 roads with the most updates. The tag 'version' represent the time of updates. Once a feature is updated, its version will be added for one more.

```
SET search_path TO public;
SELECT osm_id, name,
       CAST(tags -> 'osm_version' as int) AS osm_version,
       way
FROM planet_osm_roads
WHERE osm_id > 0
ORDER BY CAST(tags -> 'osm_version' as int) DESC
LIMIT 10
;
```

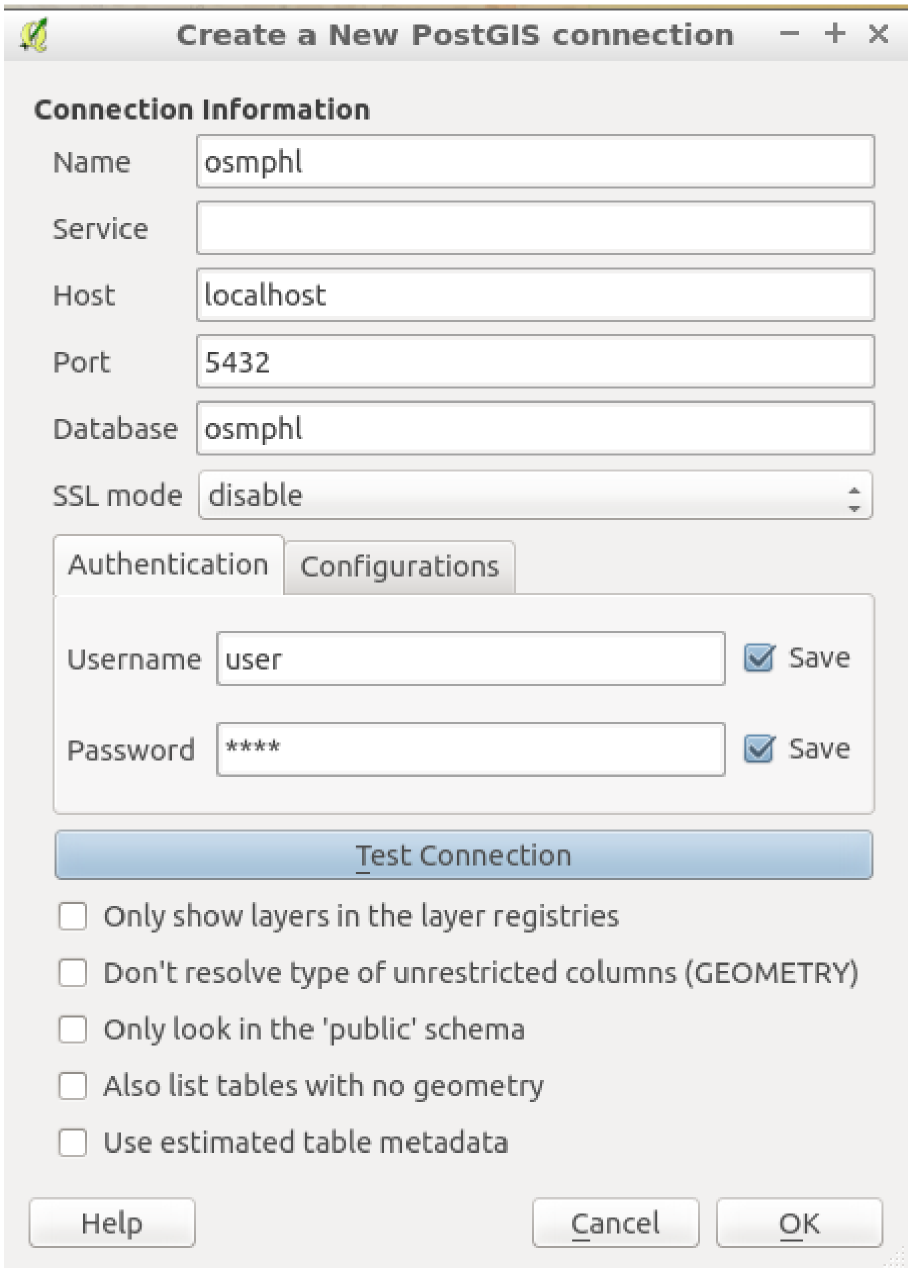
- `tag -> key` select data based on the key of the tag.

- `cast(column, AS datatype)` change datatype in the output.

You can also find help page for hstore on Postgresql help.

Map

Mapping is done in QGIS.
Open QGIS and set connect to the database that we just created. In this case it's called osmphl.



Connection Information

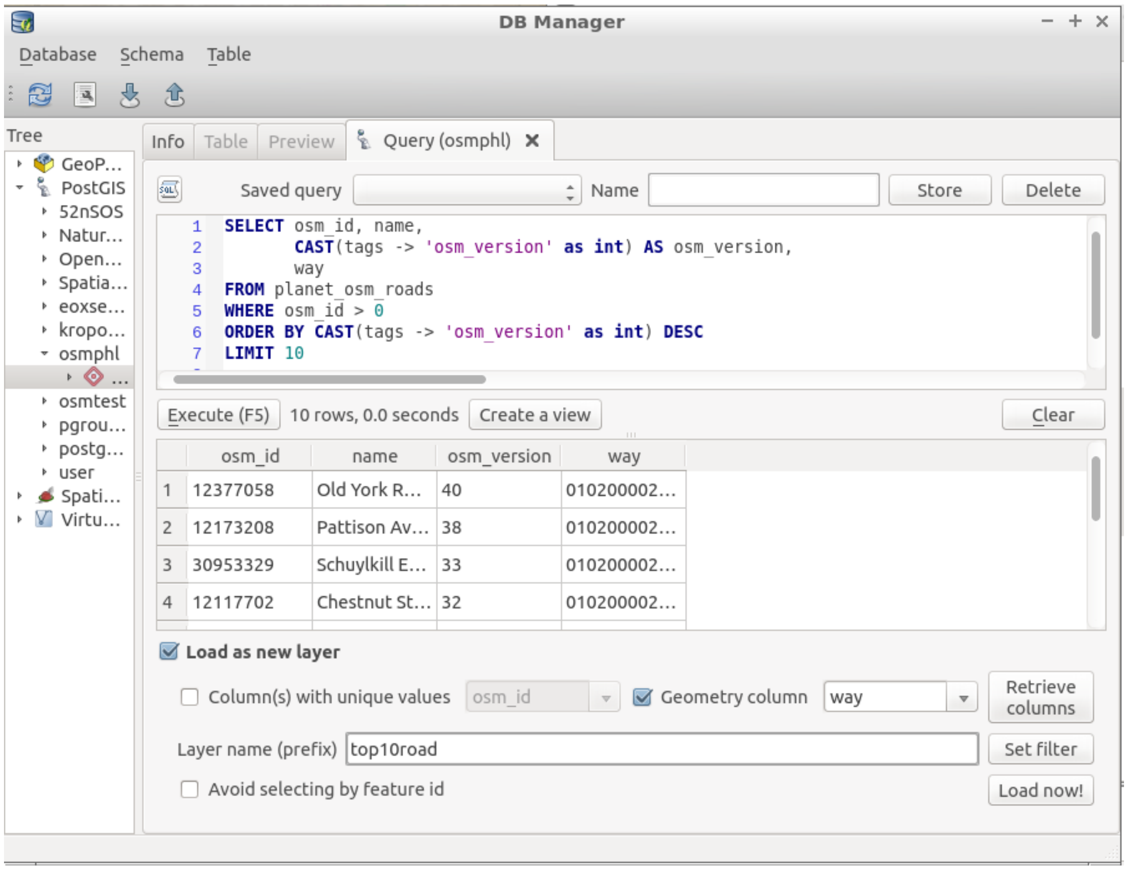
Name	osmphl
Service	
Host	localhost
Port	5432
Database	osmphl
SSL mode	disable

Authentication | Configurations

Username	user	[✓ Save]
Password	****	[✓ Save]

- [ ] Only show layers in the layer registries
- [ ] Don't resolve type of unrestricted columns (GEOMETRY)
- [ ] Only look in the 'public' schema
- [ ] Also list tables with no geometry
- [ ] Use estimated table metadata

Open the SQL scrip and copy the lines that we typed in **Select Hstore** section and set the geometry as way.



Saved query | Name | Store | Delete

```sql
SELECT osm_id, name,
       CAST(tags -> 'osm_version' as int) AS osm_version,
       way
FROM planet_osm_roads
WHERE osm_id > 0
ORDER BY CAST(tags -> 'osm_version' as int) DESC
LIMIT 10
```

Execute (F5) 10 rows, 0.0 seconds Create a view Clear

	osm_id	name	osm_version	way
1	12377058	Old York R…	40	010200002…
2	12173208	Pattison Av…	38	010200002…
3	30953329	Schuylkill E…	33	010200002…
4	12117702	Chestnut St…	32	010200002…

Load as new layer: ✓

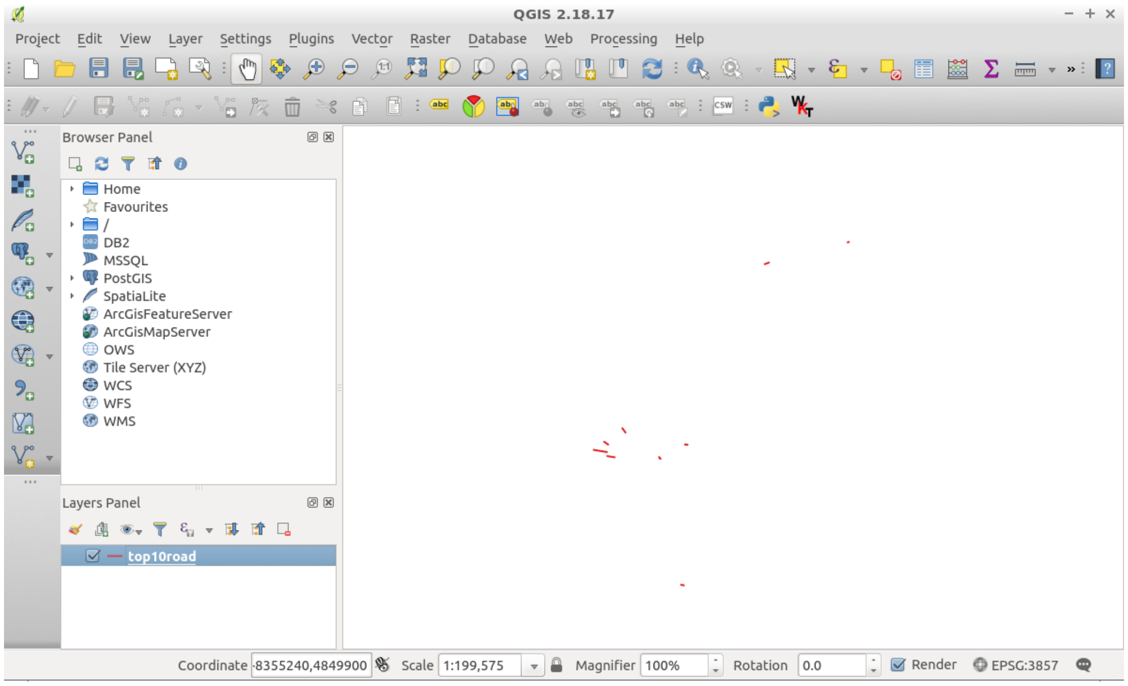
Column(s) with unique values: osm_id

Geometry column: ✓ way

Layer name (prefix): top10road

Avoid selecting by feature id

Load.



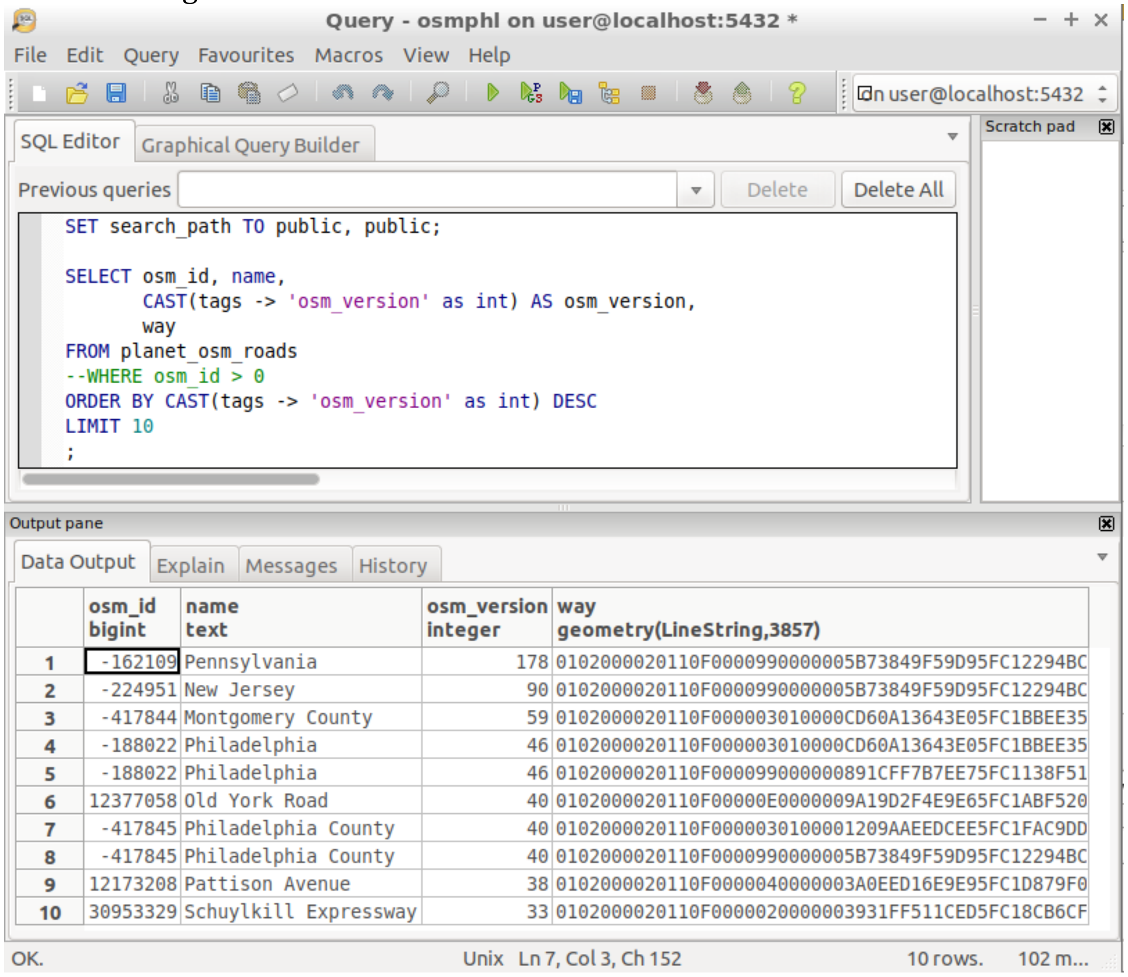
Layers Panel: ✓ top10road

Coordinate -8355240,4849900 Scale 1:199,575 Magnifier 100% Rotation 0.0 ✓ Render EPSG:3857

Here's where these 10 roads with the most updates locate.

Negative OSM ID

You might notice that there is a 'where' clause in the SQL statement above. If we delete the WHERE line, the result will be different. There will be some records in the table with negative OSM ids.

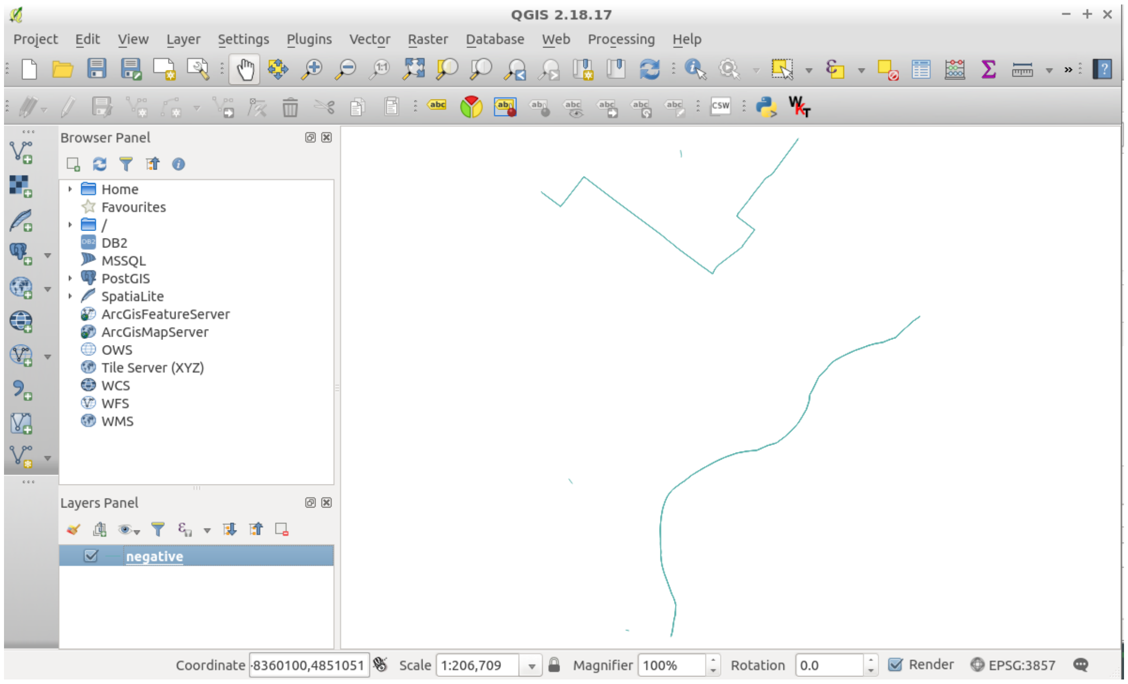


```sql
SET search_path TO public, public;

SELECT osm_id, name,
       CAST(tags -> 'osm_version' as int) AS osm_version,
       way
FROM planet_osm_roads
--WHERE osm_id > 0
ORDER BY CAST(tags -> 'osm_version' as int) DESC
LIMIT 10
;
```

	osm_id bigint	name text	osm_version integer	way geometry(LineString,3857)
1	-162109	Pennsylvania	178	0102000020110F0000990000005B73849F59D95FC12294BC
2	-224951	New Jersey	90	0102000020110F0000990000005B73849F59D95FC12294BC
3	-417844	Montgomery County	59	0102000020110F000003010000CD60A13643E05FC1BBEE35
4	-188022	Philadelphia	46	0102000020110F000003010000CD60A13643E05FC1BBEE35
5	-188022	Philadelphia	46	0102000020110F000099000000891CFF7B7EE75FC1138F51
6	12377058	Old York Road	40	0102000020110F00000E0000009A19D2F4E9E65FC1ABF520
7	-417845	Philadelphia County	40	0102000020110F0000030100001209AAEEDCEE5FC1FAC9DD
8	-417845	Philadelphia County	40	0102000020110F0000990000005B73849F59D95FC12294BC
9	12173208	Pattison Avenue	38	0102000020110F0000040000003A0EED16E9E95FC1D879F0
10	30953329	Schuylkill Expressway	33	0102000020110F0000020000003931FF511CED5FC18CB6CF

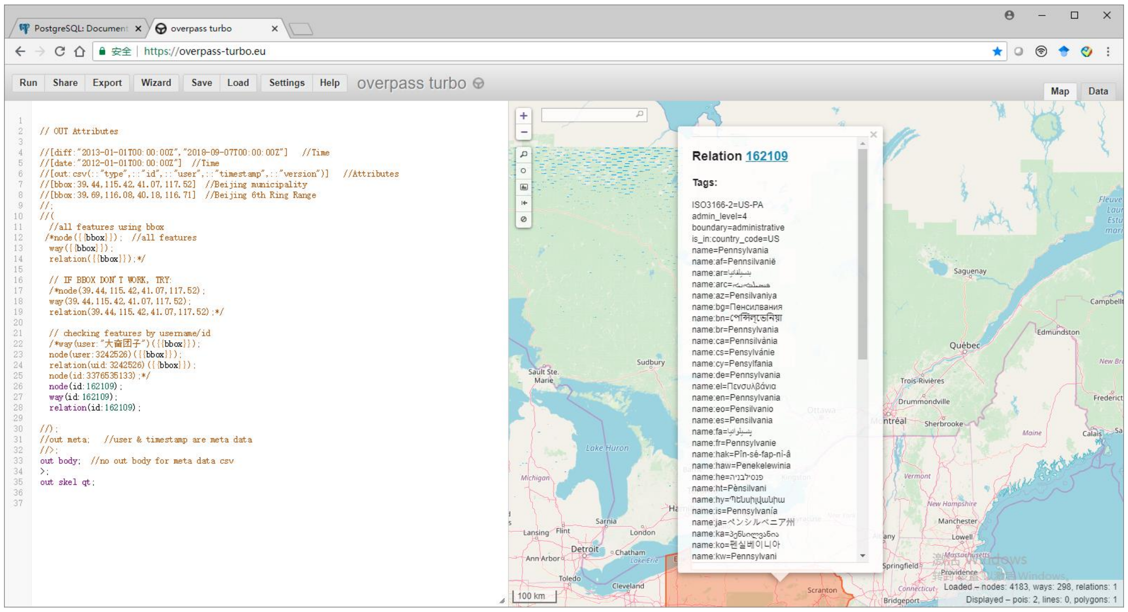
OK. Unix Ln 7, Col 3, Ch 152 10 rows. 102 m...

Do they exist? Let's map it out in the QGIS:



Coordinate -8360100,4851051 Scale 1:206,709 Magnifier 100% Rotation 0.0 Render EPSG:3857

What are they? Let’s take ‘162109’, ‘224951’, and ‘417844’ as an example and take a look on the Overpass Turbo. You can select OSM elements here through Overpass API.



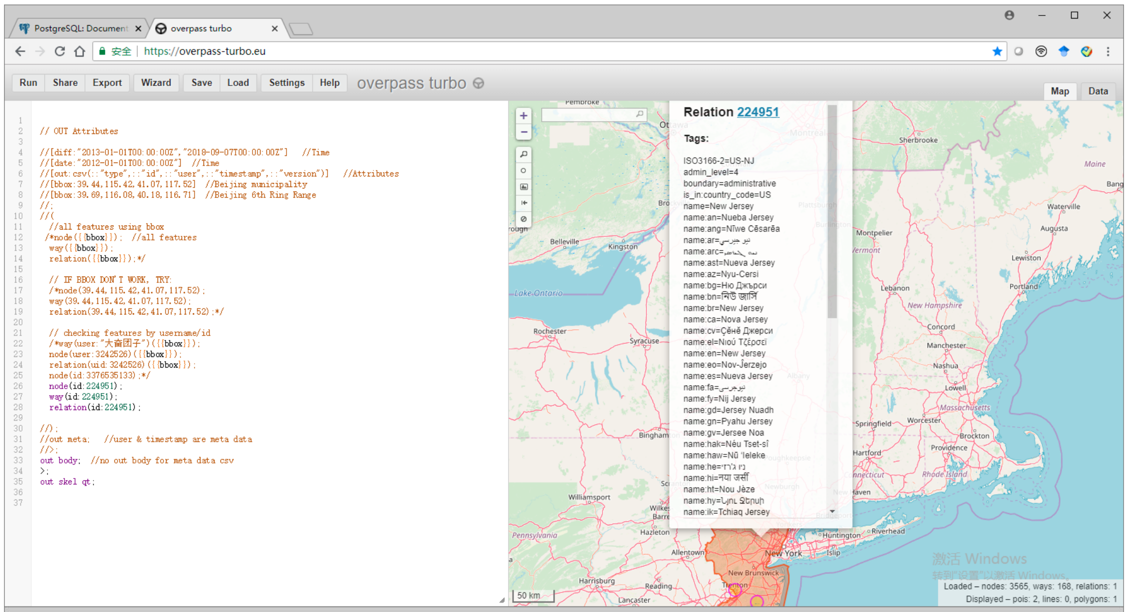
```

// OUT Attributes

//[diff:"2013-01-01T00:00:00Z","2018-09-07T00:00:00Z"]   //Time
//[date:"2012-01-01T00:00:00Z"]  //Time
//[out:csv(::"type",::"id",::"user",::"timestamp",::"version")]   //Attributes
//[bbox:39.44,115.42,41.07,117.52]  //Beijing municipality
//[bbox:39.69,116.08,40.18,116.71]  //Beijing 6th Ring Range
//;
//(
  //all features using bbox
 /*node({{bbox}});  //all features
  way({{bbox}});
  relation({{bbox}});*/

  // IF BBOX DON'T WORK, TRY:
  /*node(39.44,115.42,41.07,117.52);
  way(39.44,115.42,41.07,117.52);
  relation(39.44,115.42,41.07,117.52);*/

  // checking features by username/id
  /*way(user:"大畜团子")({{bbox}});
  node(user:3242526)({{bbox}});
  relation(uid:3242526)({{bbox}});
  node(id:3376535133);*/
  node(id:162109);
  way(id:162109);
  relation(id:162109);

//);
//out meta;   //user & timestamp are meta data
//>;
out body;  //no out body for meta data csv
>;
out skel qt;


```

Relation 162109

Tags:

```
ISO3166-2=US-PA
admin_level=4
boundary=administrative
is_in:country_code=US
name=Pennsylvania
name:af=Pennsilvanië
name:ar=بنسلفانيا
name:arc=ܦܢܣܠܒܢܝܐ
name:az=Pensilvaniya
name:bg=Пенсилвания
name:bn=পেন্সিলভেনিয়া
name:br=Pennsylvania
name:ca=Pennsilvània
name:cs=Pensylvánie
name:cy=Pensylfania
name:de=Pennsylvania
name:el=Πενσυλβάνια
name:en=Pennsylvania
name:eo=Pensilvanio
name:es=Pensilvania
name:fa=پنسیلوانیا
name:fr=Pennsylvanie
name:hak=Pîn-sé-fap-nì-â
name:haw=Penekelewinia
name:he=פנסילבניה
name:ht=Pènsilvani
name:hy=Պենսիլվանիա
name:is=Pennsylvanía
name:ja=ペンシルベニア州
name:ka=პენსილვანია
name:ko=펜실베이니아
name:kw=Pennsylvani
```

Loaded – nodes: 4183, ways: 298, relations: 1
Displayed – pois: 2, lines: 0, polygons: 1



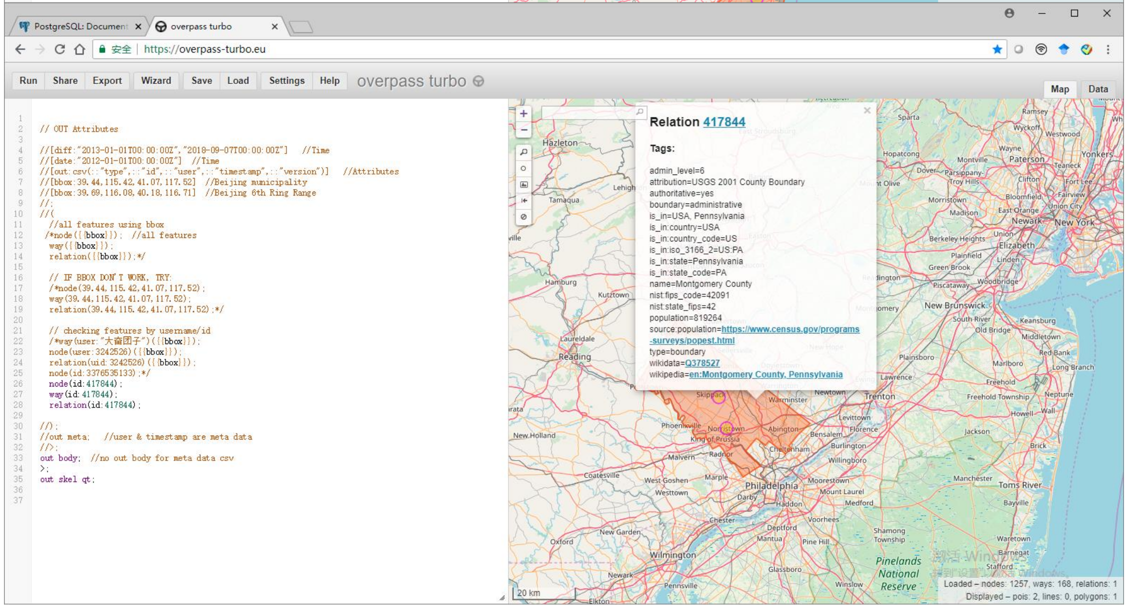
```
// OUT Attributes

//[diff:"2013-01-01T00:00:00Z","2018-09-07T00:00:00Z"]   //Time
//[date:"2012-01-01T00:00:00Z"]  //Time
//[out:csv(::"type",::"id",::"user",::"timestamp",::"version")]   //Attributes
//[bbox:39.44,115.42,41.07,117.52]  //Beijing municipality
//[bbox:39.69,116.08,40.18,116.71]  //Beijing 6th Ring Range
//;
//(
  //all features using bbox
 /*node({{bbox}});  //all features
  way({{bbox}});
  relation({{bbox}});*/

  // IF BBOX DON'T WORK, TRY:
  /*node(39.44,115.42,41.07,117.52);
  way(39.44,115.42,41.07,117.52);
  relation(39.44,115.42,41.07,117.52);*/

  // checking features by username/id
  /*way(user:"大畜团子")({{bbox}});
  node(user:3242526)({{bbox}});
  relation(uid:3242526)({{bbox}});
  node(id:3376535133);*/
  node(id:224951);
  way(id:224951);
  relation(id:224951);

//);
//out meta;   //user & timestamp are meta data
//>;
out body;  //no out body for meta data csv
>;
out skel qt;
```

Relation 224951

Tags:

ISO3166-2=US-NJ
admin_level=4
boundary=administrative
is_in:country_code=US
name=New Jersey
name:an=Nueba Jersey
name:ang=Nīwe Cēsarēa
name:ar=نيو جيرسي
name:arc=ܢܝܘ ܓܪܣܝ
name:ast=Nueva Jersey
name:az=Nyu-Cersi
name:bg=Ню Джърси
name:bn=নিউ জার্সি
name:br=New Jersey
name:ca=Nova Jersey
name:cv=Çĕнĕ Джерси
name:el=Νιού Τζέρσεϊ
name:en=New Jersey
name:eo=Nov-Ĵerzejo
name:es=Nueva Jersey
name:fa=نیوجرسی
name:fy=Nij Jersey
name:gd=Jersey Nuadh
name:gn=Pyahu Jersey
name:gv=Jersee Noa
name:hak=Néu Tset-sî
name:haw=Nū ʻIeleke
name:he=ניו ג'רזי
name:hi=नया जर्सी
name:ht=Nou Jèze
name:hy=Նյու Ջերսի
name:ik=Tchiaq Jersey

Loaded – nodes: 3565, ways: 168, relations: 1
Displayed – pois: 2, lines: 0, polygons: 1



```
// OUT Attributes

//[diff:"2013-01-01T00:00:00Z","2018-09-07T00:00:00Z"]   //Time
//[date:"2012-01-01T00:00:00Z"]  //Time
//[out:csv(::"type",::"id",::"user",::"timestamp",::"version")]   //Attributes
//[bbox:39.44,115.42,41.07,117.52]  //Beijing municipality
//[bbox:39.69,116.08,40.18,116.71]  //Beijing 6th Ring Range
//;
//(
  //all features using bbox
 /*node({{bbox}});  //all features
  way({{bbox}});
  relation({{bbox}});*/

  // IF BBOX DON'T WORK, TRY:
  /*node(39.44,115.42,41.07,117.52);
  way(39.44,115.42,41.07,117.52);
  relation(39.44,115.42,41.07,117.52);*/

  // checking features by username/id
  /*way(user:"大畜团子")({{bbox}});
  node(user:3242526)({{bbox}});
  relation(uid:3242526)({{bbox}});
  node(id:3376535133);*/
  node(id:417844);
  way(id:417844);
  relation(id:417844);

//);
//out meta;   //user & timestamp are meta data
//>;
out body;  //no out body for meta data csv
>;
out skel qt;
```

Relation 417844

Tags:

admin_level=6
attribution=USGS 2001 County Boundary
authoritative=yes
boundary=administrative
is_in=USA, Pennsylvania
is_in:country=USA
is_in:country_code=US
is_in:iso_3166_2=US:PA
is_in:state=Pennsylvania
is_in:state_code=PA
name=Montgomery County
nist:fips_code=42091
nist:state_fips=42
population=819264
source:population=https://www.census.gov/programs-surveys/popest.html
type=boundary
wikidata=Q378527
wikipedia=en:Montgomery County, Pennsylvania

Loaded – nodes: 1257, ways: 168, relations: 1
Displayed – pois: 2, lines: 0, polygons: 1

They are relations.
The tool osm2pgsql will split the boundary of relations and make it into negative to indicate that it's actually not a road but the boundary of a relation.

If You Are Interested

This part of the workbook is not going to be in the tutorial. But you can also take a look if you are interested.

Slim Mode

As I mentioned in the introduction, OSM data can be converted in two modes, i.e. slim and non-slim. We are dealing with the slim mode in this tutorial but what

about slim mode?
Load OSM data for the slim mode with a additional '-s | –slim' in the code after creating database:

```
$ osm2pgsql -c -d osmtest -C 15000 --hstore --extra-attributes --multi-
geometry \
> ~/Desktop/philadelphia.osm.pbf
```

Three tables are created in the database. Records in these tables don't have repeated id but they also missing geometry. Moreover, they store the tags into text strings.

Clipping Data With osmconvert

The data we are using in this tutorial is clipped from OSM PA data by the bounding box with osmconvert. It's a multipurpose tool for OSM data. It's not related to osm2pgsql so I did not included in the tutorial. But you can try it if you are interested. More command can be found in a blog here.
You might need to install a set of tools in the Linux Terminal first before using it:

```
$ sudo apt-get install osmctools
```

To clip the area with bounding box, you can do this:

```
$ osmconvert /media/sf_SHAREFOLDERS_Linux_OSGEO/history-latest.osm.pbf
\
> -b=116.08,39.69,116.71,40.18 -o=beijinghistory.pbf
```

- This code is an example for clipping data within Beijing 6th Ring highway from OSM history dump stored in my sharefolder.

- -b means bounding box, put the coordinate in this order: W,S,E,N

- -o is the output file.

Export as csv Data With osmconvert

Osmconvert can also be used in exporting only intended data in a csv format, but it might not contain geometry.

```
$ osmconvert /media/sf_SHAREFOLDERS_Linux_OSGEO/history-latest.osm.pbf
\
> --out-csv --csv="@id @user @uid @timestamp @version" --csv-headline -
-csv-separator=, \
> -o=beijinghistory.csv
```

- This code is an example for exporting csv data within Beijing 6th Ring highway from OSM history dump stored in my sharefolder.

- `--csv-headline` is followed by the header in the csv

- `--csv-separator=` defines what is the separator for csv columns. In this case we used comma as the separator.

The import of csv file can be done with psql. Type the first line before continuing the second line.

- This code is an example for importing csv data within Beijing 6th Ring highway to a database preciously created called osmhistory_planet.

- `delimiter` is used to specify the separator for columns. In this case we used comma as the separator.