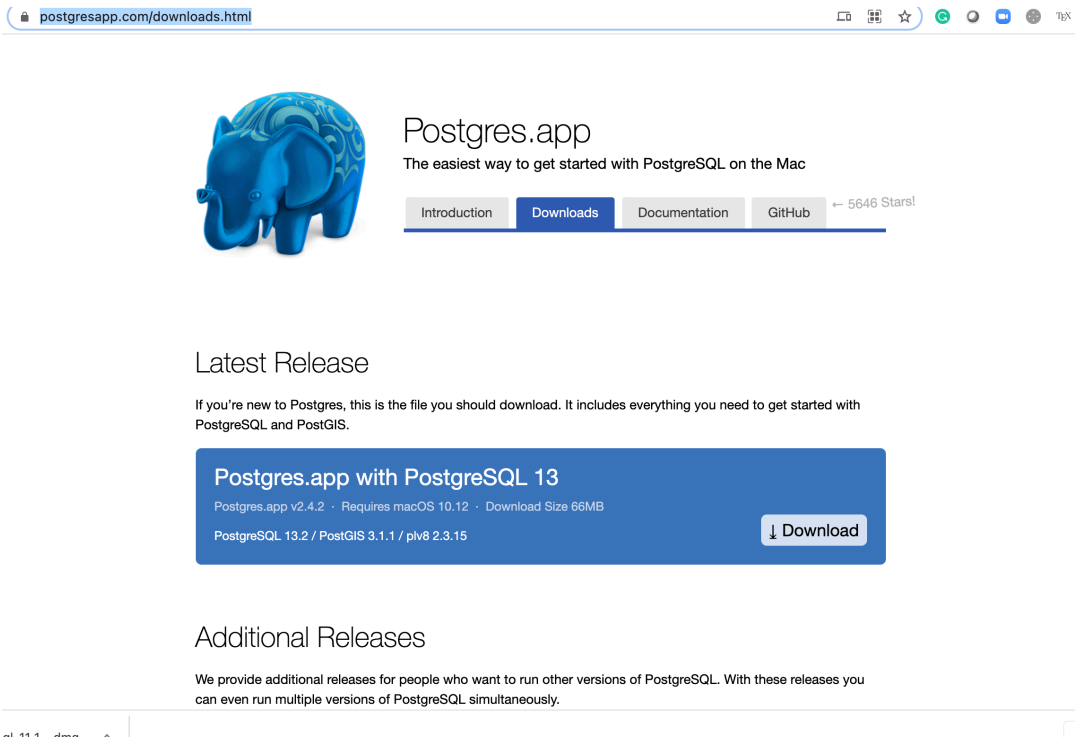
Spatial Database

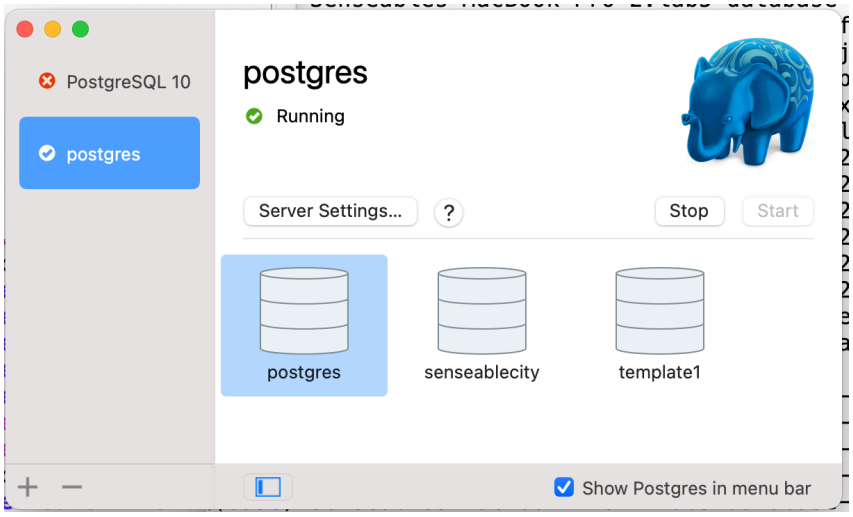
Database systems make spatial operations much simpler, and it is more efficient to handle a large number of records. For example, Excel has a limit of the number of records, however, there is no such limit in the database system. In addition, you probably cannot handle a really large shapefile with millions of records. Using the spatial database system in PostGIS, you can make the spatial operations much more efficient. There are many available database systems and the spatial database system. In this class, we are going to focus on one of the most popular open-source database systems Postgres/PostGIS. We are going to spend two weeks on the database system. The first class is about configuring the Postgres/PostGIS system. The second class is about using Python to query and manipulate the database system. You will learn how to write SQL to do a query in the database.

1. Install and configure the Postgres/PostGIS

The first step is to install the Postgres/PostGIS on your computer. For Mac users, you can download the combo of Postgres/PostGIS from, https://postgresapp.com/downloads.html. Download the Postgres.app which is a combination of Postgres/PostGIS.



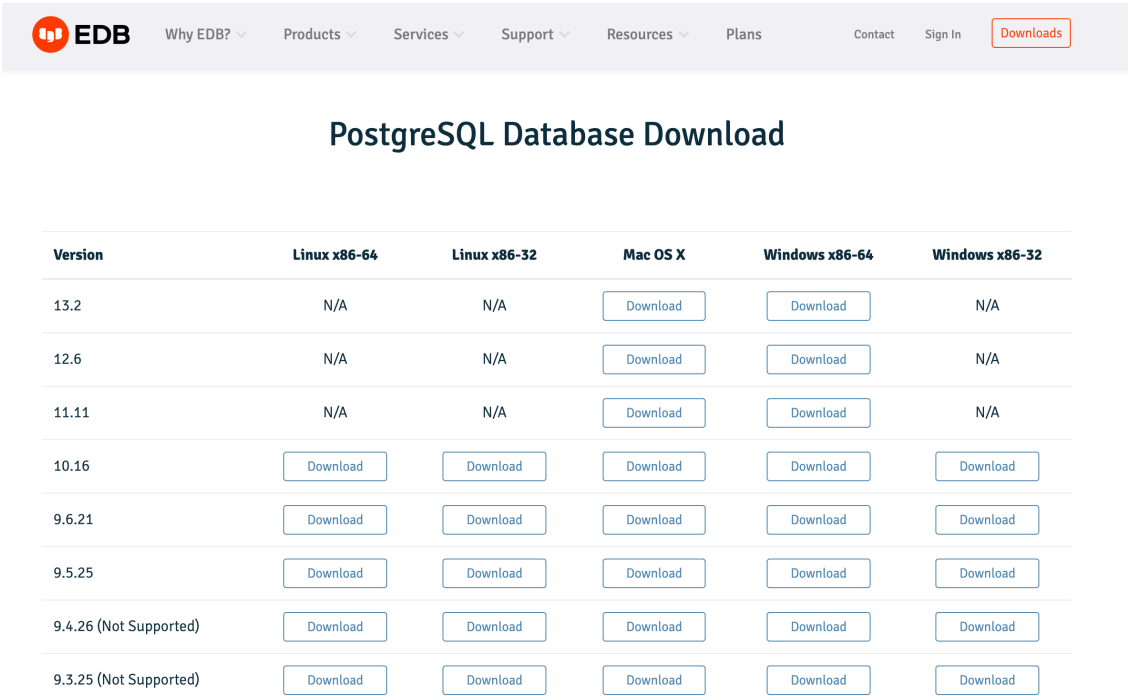
Then on the open your Postgres and click the "+" on the left side, and then you can start your Postgres server,



Then you can just follow the default setting and install the Postgres/PostGIS on your computer. After installing it successfully, you can then start the terminal of psql on mac.

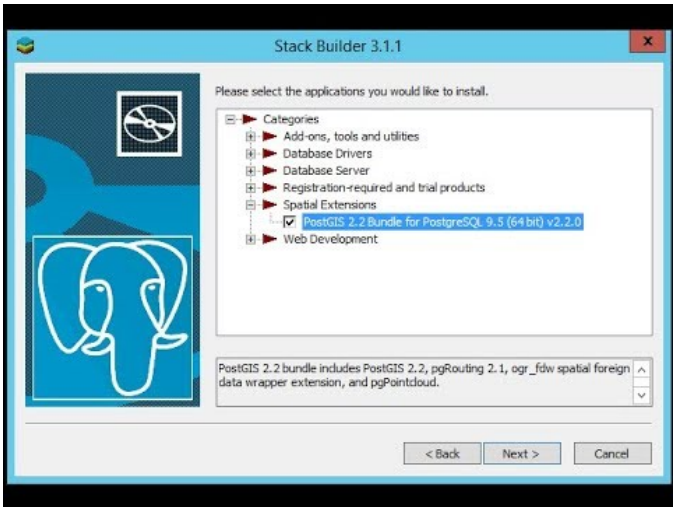
For Windows users download the installer of Postgres/PostGIS from here,

https://www.enterprisedb.com/downloads/postgres-postgresql-downloads



Let's use the version of 10.16. Then you can install the Postgres/PostGIS on your computer.

Note: check the PostGIS when you see this window.



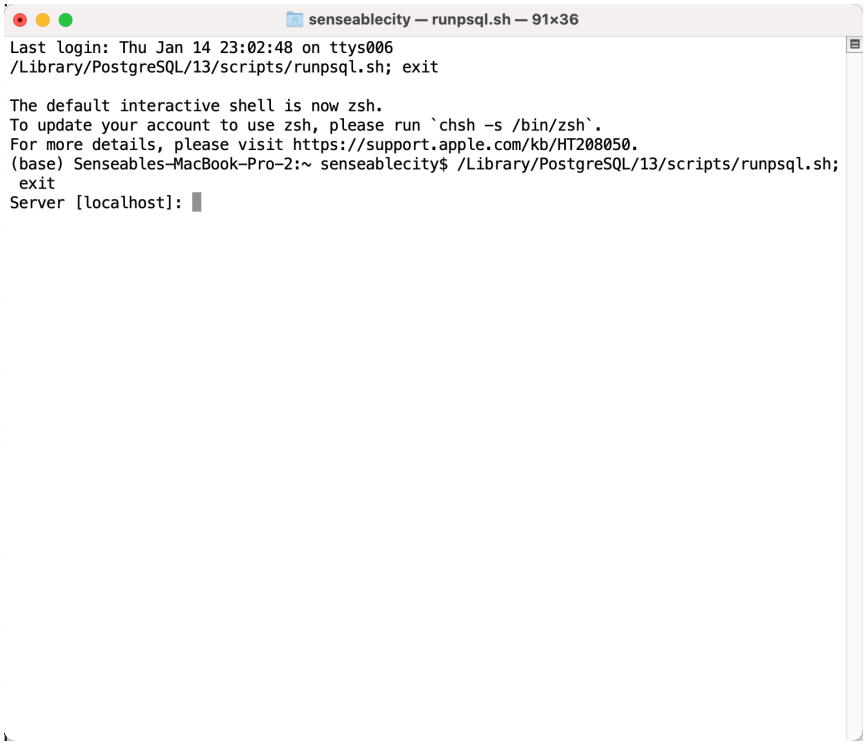
2. Be familiar with the database

For Windows user, Open PSQL shell,

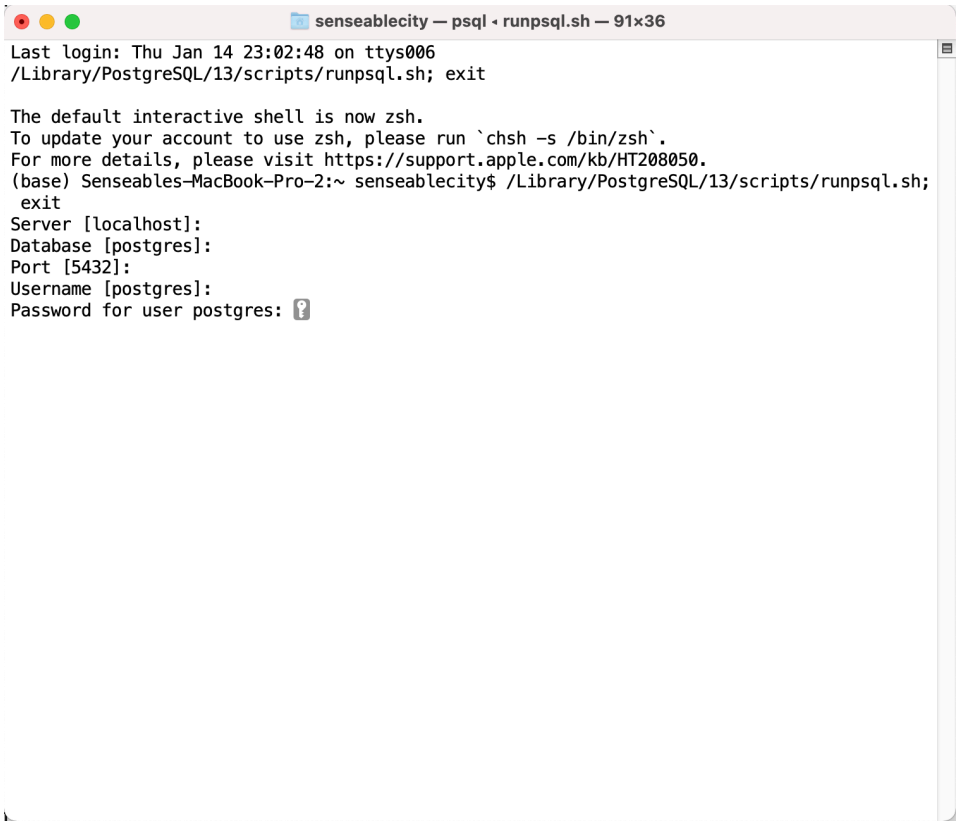
Window ->Postgres -> PSQL Shell

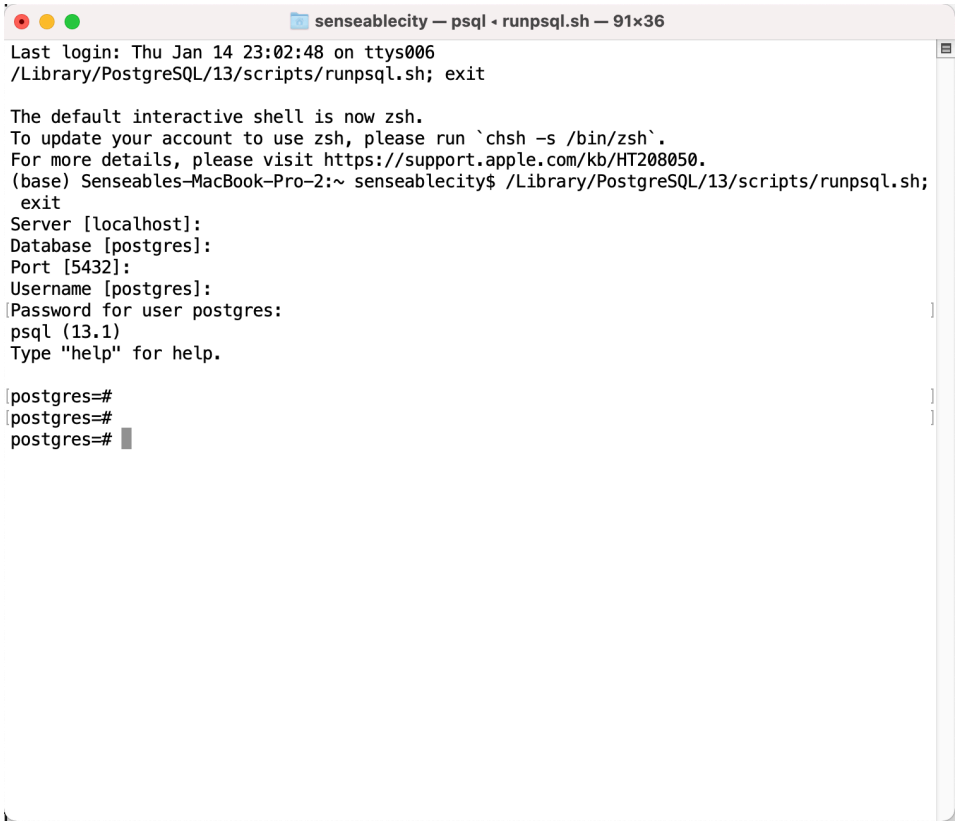
For Mac user, type the command in your terminal,

```
psql -U postgres
```



Type in your password,


senseablecity — psql ◂ runpsql.sh — 91×36
Last login: Thu Jan 14 23:02:48 on ttys006
/Library/PostgreSQL/13/scripts/runpsql.sh; exit
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) Senseables-MacBook-Pro-2:~ senseablecity$ /Library/PostgreSQL/13/scripts/runpsql.sh;
exit
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:


senseablecity — psql ◂ runpsql.sh — 91×36
Last login: Thu Jan 14 23:02:48 on ttys006
/Library/PostgreSQL/13/scripts/runpsql.sh; exit
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) Senseables-MacBook-Pro-2:~ senseablecity$ /Library/PostgreSQL/13/scripts/runpsql.sh;
exit
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (13.1)
Type "help" for help.
postgres=#
postgres=#
postgres=#

Once you login to the database, let's be familiar with the basic commands in psql,

List all databases,

```
\l
```

Connect to a certain database, \c databasename

```
\c dbname
```

List all tables,

```
\c dbname
\dt
```

The terminal return nothing using the above commands because there is nothing in your database. Let's first create a database for the following datasets.

```
create database phila;
```

Create extension of PostGIS

```
CREATE EXTENSION IF NOT EXISTS PostGIS CASCADE;
```

Then you can check the version of the installed PostGIS,

```
SELECT PostGIS_version();
```

3. Build a database and convert the shapefile into tables,

Although we can create our database and tables using commands, a simpler method to build a database is to import existing tables into the database. This section is about converting shapefiles into a spatial database. There are two shapefiles in the data folder, "census3652.shp" and "philadata3652.shp". Let's input them into our PostGIS database.

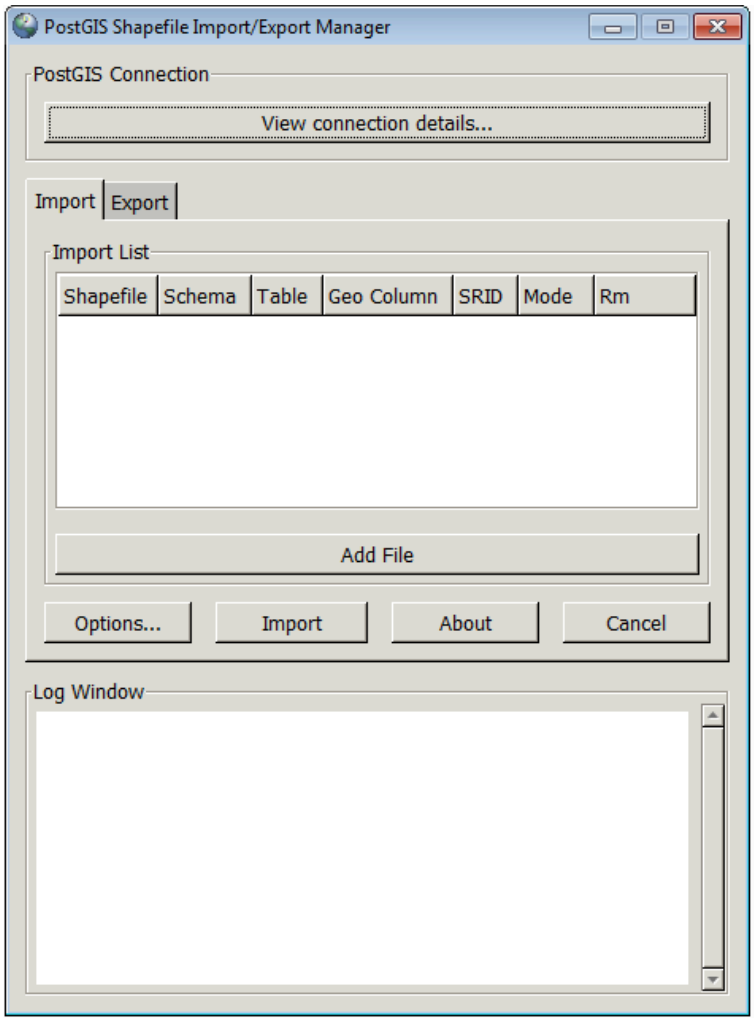
First create a database of phila through the command,

```
create database phila;
```

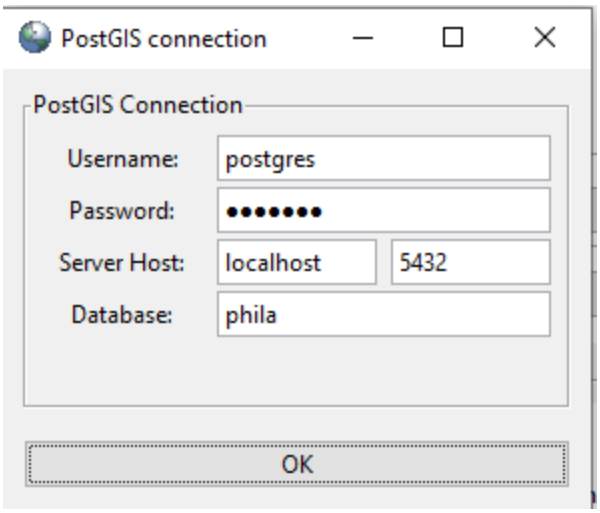
Then you can import the shapefiles into the database of phila.

1. **For windows users,**

Download the tool to convert the shapefile to spatial data base table, http://download.osgeo.org/postgis/windows/. Make sure put the exe file to the folder of the your postgres. For example, on my computer, C:\Program Files\PostgreSQL. Then double click it to install the tool,



Then let's connect to the database in the postgis-bundle tool, by click 'View Connection details',

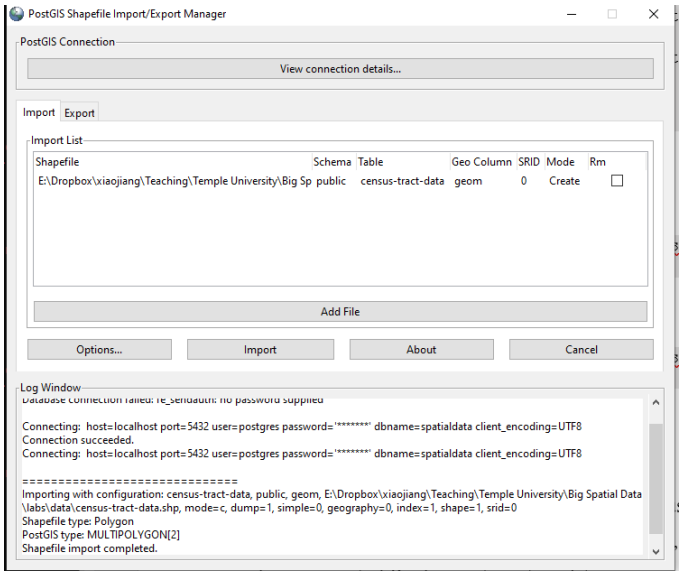


Fill the table using your username, password, localhost, and db name.

Then you need to enable the extension of PostGIS by typing in the PSQL shell,

```
CREATE EXTENSION POSTGIS;
```

After enabling the PostGIS, then you can import the shapefile to PostGIS table. Click the "Add File", and select the shapefile you want to convert to Postgres database table, and set the "SRID" as the epsg code of the shapefile.



Click the add File and select one shapefile, then you can click import. You can then check your database in the PSQL shell,

```
\c phila;
\dt
```

You will find you have the table imported successfully.

Reference: https://postgis.net/workshops/postgis-intro/loading_data.html

For Mac users

You can just stay in the terminal and use the command line to convert the shapefile into Postgres table. First cd to the directory of your shapefile, and then type in,

```
(base) jiang@jiang-K30AD-M31AD-M51AD:~$
(base) jiang@jiang-K30AD-M31AD-M51AD:~$
(base) jiang@jiang-K30AD-M31AD-M51AD:~$ psql postgres
psql (10.15 (Ubuntu 10.15-0ubuntu0.18.04.1))
Type "help" for help.

postgres=#
postgres=#
postgres=#
postgres=# create database phila;
CREATE DATABASE
postgres=#
postgres=# \dt
              List of relations
 Schema |      Name       | Type  |  Owner
--------+-----------------+-------+----------
 public | spatial_ref_sys | table | postgres
(1 row)

postgres=# \l
postgres=#
postgres=#
postgres=#
```

Convert the shapefile of philadata3652.shp into a PostGIS table. First cd to the directory of the philadata.shp file, and type in,

```
shp2pgsql -s 3652 philadata3652> philadata.sql
```

You will find that there is a file "philadata.sql" created. This is a series of Postgres commands. We need to enable the PostGIS before running these commands. Open the "philadata.sql" using text editor, and add the statement on top,

```
CREATE EXTENSION postgis;
```

Then you can run the .sql and import the shapefile into Postgres database table,

```
psql -h localhost -U postgres -d phila -p 5432 -f philadata.sql
```

-h: the host of localhost
-U: the username
-d: the database name
-p: portal, default of 5432
-f: the .sql file

Convert the shapefile of the census tract of Philadelphia into the table

```
shp2pgsql -s 3652 census3652> census3652.sql
```

Then you will find there is a census3652.sql file. The next step is to run the commands and insert the data to your database, make sure you start the PostGIS service first in your psql terminal,

```
CREATE EXTENSION postgis;
```

Then you can run the .sql in your Mac/Linux terminal,

```
psql -h localhost -U postgres -d phila -p 5432 -f census3652.sql
```

You can check if the shapefile has been imported into the database in the terminal,

```
\c phila;
\dt
```

```
(base) Senseables-MacBook-Pro-2:lab5-database-query senseablecity$
(base) Senseables-MacBook-Pro-2:lab5-database-query senseablecity$
(base) Senseables-MacBook-Pro-2:lab5-database-query senseablecity$
(base) Senseables-MacBook-Pro-2:lab5-database-query senseablecity$ psql postgres
psql (12.2, server 12.6)
Type "help" for help.

postgres=#
postgres=#
postgres=# \dt
               List of relations
 Schema |      Name       | Type  |     Owner
--------+-----------------+-------+---------------
 public | spatial_ref_sys | table | senseablecity
(1 row)

postgres=# \c phila
psql (12.2, server 12.6)
You are now connected to database "phila" as user "senseablecity".
phila=#
phila=# \dt
              List of relations
 Schema |      Name       | Type  |  Owner
--------+-----------------+-------+----------
 public | census3652      | table | postgres
 public | philadata3652   | table | postgres
 public | spatial_ref_sys | table | postgres
(3 rows)

phila=#
phila=# select count(*) from census3652;
 count
-------
   384
(1 row)

phila=#
phila=#
phila=#
```

4. A spatial query using PostGIS

After you imported the shapefiles into postgres tables, we can then get started to do our queries using SQL.

4.1 Check the number of records in your point shapefile

```
select count(*) from philadata3652;
```

4.2 Preview the structure of the table

```
\d philadata3652;
```

```
phila=# \d philadata3652;
                                    Table "public.philadata3652"
   Column   |         Type          | Collation | Nullable |                   Default
------------+-----------------------+-----------+----------+-----------------------------------------------
 gid        | integer               |           | not null | nextval('philadata3652_gid_seq'::regclass)
 panoid     | character varying(30) |           |          |
 year       | character varying(4)  |           |          |
 month      | character varying(2)  |           |          |
 lon        | numeric               |           |          |
 lat        | numeric               |           |          |
 pano_yaw   | numeric               |           |          |
 tilt_yaw   | numeric               |           |          |
 tilt_pitch | numeric               |           |          |
 gvi        | numeric               |           |          |
 geom       | geometry(Point,3652)  |           |          |
Indexes:
    "philadata3652_pkey" PRIMARY KEY, btree (gid)
```

Now we are ready for the spatial database tables. Then we can use the database tables to do the spatial join between the point shapefile and the census tract polygon file. Here is an example of counting the number of points in each polygon,

```
SELECT census3652.tractce, count(philadata3652.panoid)
FROM philadata3652
LEFT JOIN census3652 ON ST_Intersects(census3652.geom, philadata3652.geom)
GROUP BY census3652.tractce;
```

We can make our statement shorter,

```
SELECT c.tractce, count(p.panoid)
FROM philadata3652 p
LEFT JOIN census3652 c ON ST_Intersects(c.geom, p.geom)
GROUP BY c.tractce;
```

```
 tractce | count
---------+-------
 000100  |   123
 000200  |   100
 000300  |   127
 000401  |    58
 000402  |    70
 000500  |    73
 000600  |    11
 000700  |    51
 000801  |    21
 000803  |    36
 000804  |    28
 000901  |    21
 000902  |    38
 001001  |    42
 001002  |    87
 001101  |    63
 001102  |    41
 001201  |    94
 001202  |    56
 001300  |   170
 001400  |   115
 001500  |    69
 001600  |    53
 001700  |    67
 001800  |    47
 001900  |    92
 002000  |   105
 002100  |    91
 002200  |    69
 002300  |    61
 002400  |   159
 002500  |   129
 002701  |    71
 002702  |   125
 002801  |    90
:
```

If we want more attribute to be saved, then you can use the statement of,

```
SELECT c2.tractce, c2.countyfp, c2.statefp, c2.geom, t.num
FROM (
        SELECT c.tractce, count(p.panoid) as num
        FROM philadata3652 p
        LEFT JOIN census3652 c ON ST_Intersects(c.geom, p.geom)
        GROUP BY c.tractce) t
JOIN census3652 c2 ON c2.tractce = t.tractce limit 4;
```

Since we want to save the result as a new shapefile, we need to create a new table of the queried result,

```
CREATE TABLE CensusPntNum AS
        SELECT c2.tractce, c2.countyfp, c2.statefp, c2.geom, t.num
        FROM (
                SELECT c.tractce, count(p.panoid) as num
                FROM philadata3652 p
                LEFT JOIN census3652 c ON ST_Intersects(c.geom, p.geom)
                GROUP BY c.tractce) t
        JOIN census3652 c2 ON c2.tractce = t.tractce;
```

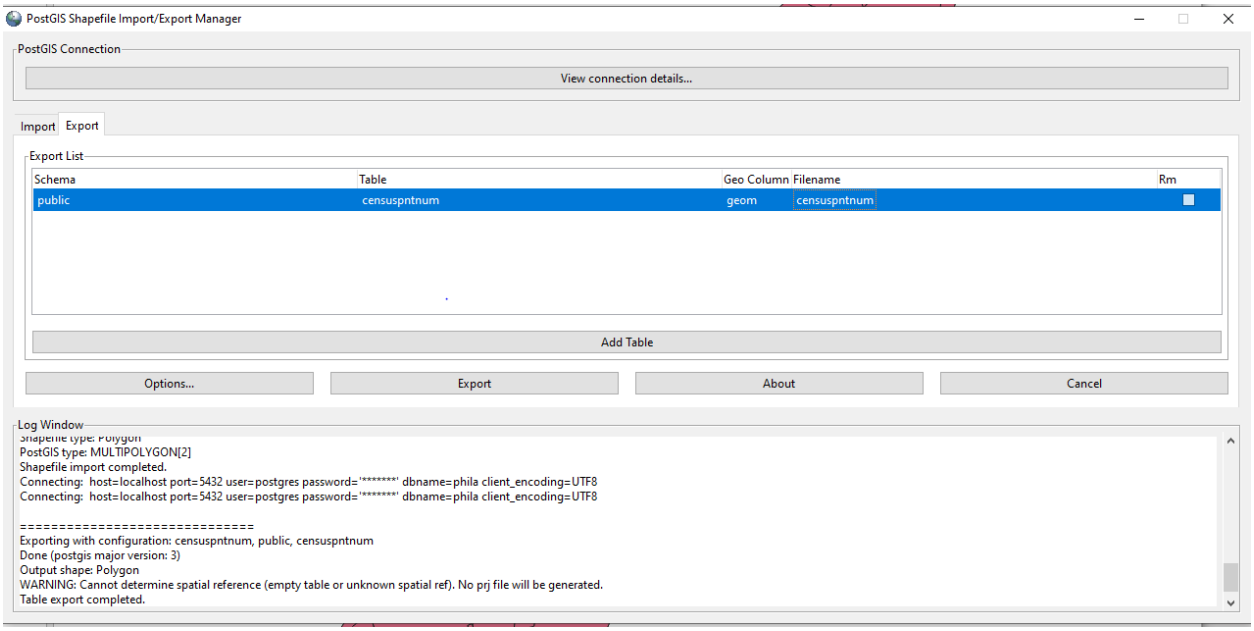
Then you will find you have a new table “censuspntnum” created,

```
phila=# \dt
              List of relations
 Schema |      Name       | Type  |  Owner
--------+-----------------+-------+----------
 public | census3652      | table | postgres
 public | censuspntnum    | table | jiang
 public | philadata3652   | table | jiang
 public | spatial_ref_sys | table | jiang
(4 rows)
```

You can check the structure of the newly created table by,

```
phila=# \d censuspntnum;
                    Table "public.censuspntnum"
  Column   |            Type             | Collation | Nullable | Default
-----------+-----------------------------+-----------+----------+---------
 tractce   | character varying(80)       |           |          |
 countyfp  | character varying(80)       |           |          |
 statefp   | character varying(80)       |           |          |
 geom      | geometry(MultiPolygon,3652) |           |          |
 num       | bigint                      |           |          |
```

We can use the same commands to export the table to a shapefile,



Let's save the table as a shapefile, using the following command in terminal (not your psql command),

```
pgsql2shp -u postgres -h localhost -P 5424796 -f pntshp phila "SELECT
* FROM censuspntnum;"
```

-u: the username of the database
-h: the host, here is the localhost
-P: is your password
-f: the file you going to save
phila: the database name