PostGIS Database with QGIS : Basic and Spatial Analysis

Dr. Sarawut Ninsawat

RSGIS, AIT

OSGeo Thai Chapter

คำเตือน

- ผู้เข้าอบรมต้องดำเนินการติดตั้ง PostgreSQL/PostGIS
  - https://www.postgresql.org/download/windows/
- โปรแกรม QGIS
  - https://qgis.org/en/site/forusers/download.html
- ข้อมูลที่ใช้
  - https://gitlab.com/i-bitz/gis-marathon-2022/-/tree/main/QGISPostGIS

What is a Spatial Database?

Database that:

- Stores spatial objects
- Manipulates spatial objects just like other objects in the database

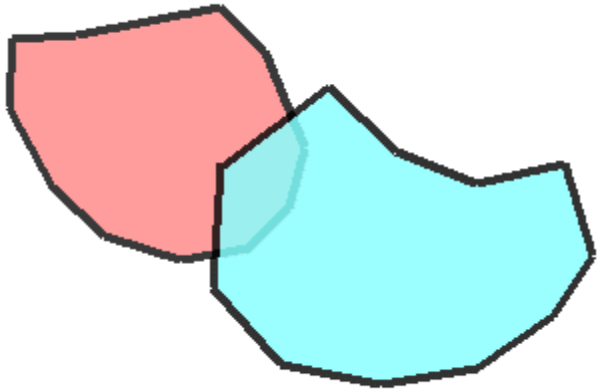
Why Spatial Database?

- Database do **some things** better than files.
  - Support multiple users editing and accessing the same data at the same time.
  - Support large data volumes better than files. (For some kinds of data.)
  - Provide a unified means of accessing and analysing data using **SQL abstraction.**
  - Provide a unified means of access control for data.
  - Provide an integration point for spatial data and enterprise data (usual DB).

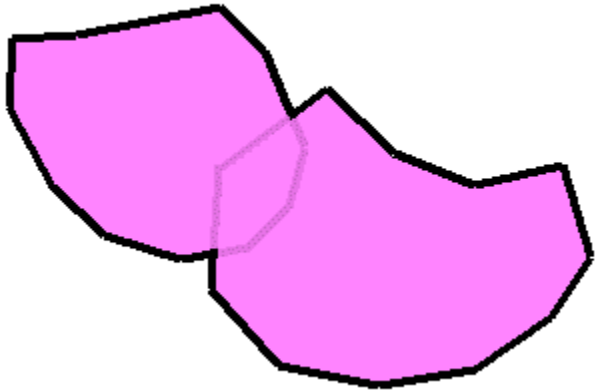
Spatial Relationships

- Not just interested in location, also interested in **"Relationships"** between objects that are very hard to model outside the spatial domain.

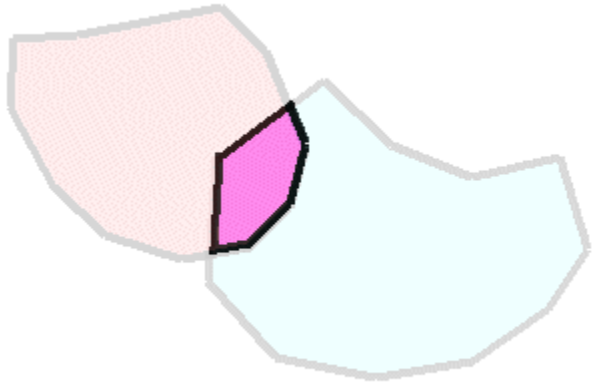
- The most common relationships are
  - Proximity : distance
  - Adjacency : "touching" and "connectivity"
  - Containment : inside/overlapping



Original Polygons

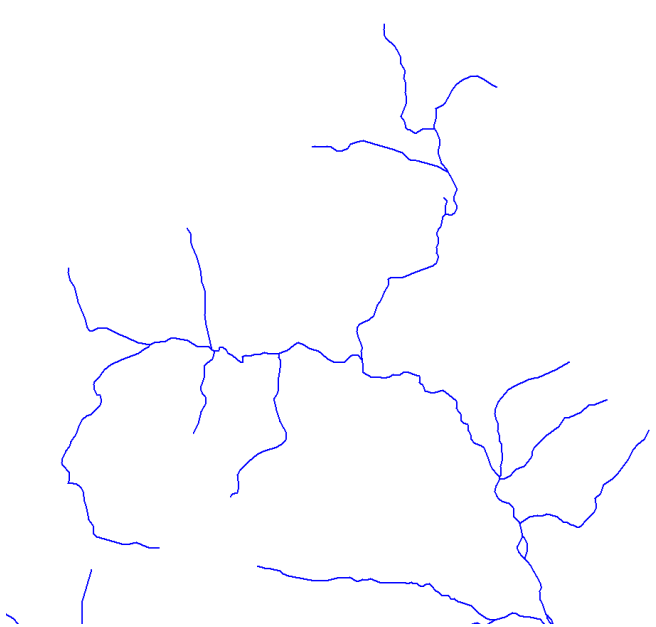


Union

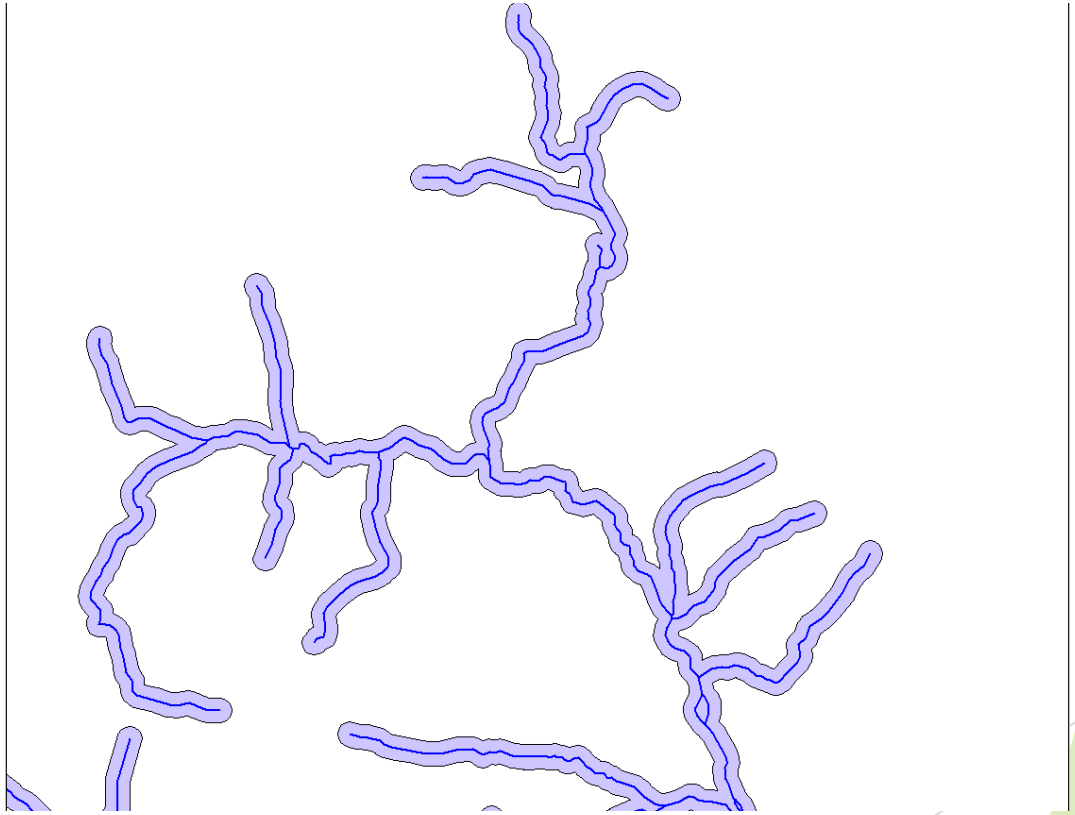


Intersection

Spatial Relationships



Original river network



Buffered rivers

… WHERE distance(<me>,pub_loc) < 1000

SELECT distance(<me>,pub_loc)*$0.01 + beer_cost …

… WHERE touches(pub_loc, street)

… WHERE inside(pub_loc,city_area) and city_name = …

Spatial Relationships query

```
select name, beer_price, distance(location,
GeometryFromText('POINT(1195722 383854)',2167)) from
pubs order by beer_price;
```

```
     name       | beer_price |     distance
----------------+------------+------------------
 Fireside       |       4.25 | 1484.10275160491
 The Forge      |       4.33 | 1533.06561109862
 Rumours        |       4.46 | 2042.00094093097
 Garricks Head  |        4.5 | 669.389105609889
 Slap Happy     |        4.5 | 1882.31910168298
 Old Bailys     |       4.55 | 1147.20900404641
 Black Sheep    |       4.66 | 536.859935972633
 Big Bad Daves  |       4.75 | 907.446543878884
```

Spatial Relationships query

```
select name, beer_price + 0.001 * distance(location,
GeometryFromText('POINT(1195722 383854)',2167)) as
net_price from pubs order by price;
```

name	net_price
Garricks Head	5.16938910560989
Black Sheep	5.19685978338474
Big Bad Daves	5.65744654387888
Old Bailys	5.69720919478127
Fireside	5.73410275160491
The Forge	5.86306553480468
Slap Happy	6.38231910168298
Rumours	6.50200097907794

Disadvantages of Spatial Databases

- Cost to implement can be high
- Some inflexibility
- Incompatibilities with some GIS software
- Slower than local, specialized data structures
- User/managerial inexperience and caution

Spatial Database Offerings

- ESRI ArcSDE (on top of several different DBs)
- Oracle Spatial
- IBM DB2 Spatial Extender
- Informix Spatial DataBlade
- MS SQL Server (with ESRI SDE)
- Geomedia on MS Access
- SpatialLite
- PostGIS / PostgreSQL

PostgreSQL

- PostgreSQL is a powerful, object-relational database management system (ORDBMS).
- It is released under a BSD-style license and is thus free and open source software.
- As with many other open source programs, PostgreSQL is not controlled by any single company, but has a global community of developers and companies to develop it.

PostGIS

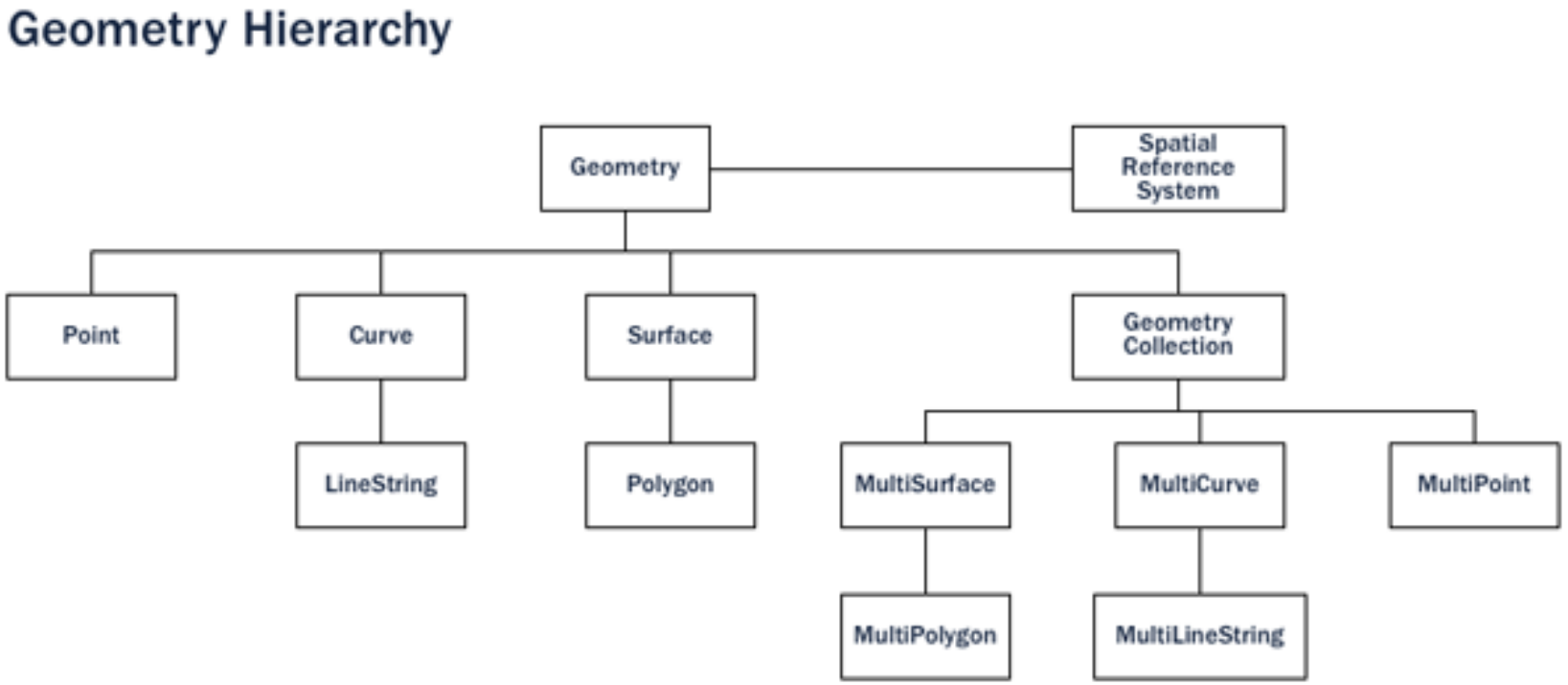
- *PostGIS* is a spatial extension for PostgreSQL
  - Enable PostgreSQL Database Management System into a spatial database by adding adding support for the three features:
    - Spatial types, Indexes and Functions.
- *PostGIS* aims to be an "OpenGIS Simple Features for SQL" compliant spatial database

Spatial Type

- An ordinary database has strings, numbers, and dates. A spatial database adds additional “spatial” types for representing **geographic features.**

- Spatial data types are organized in a type hierarchy. Each sub-type inherits the structure (attributes) and the behavior (methods or functions) of its super-type.

Geometry Hierarchy



Point/Multipoint Geometry

- POINT(0 0)
- MULTIPOINT(0 1,1 0,2 1,1 2)

Linestring/Multilinestring Geometry

- LINESTRING(1 1,2 0,3 1,3 3,2 4)
- MULTILINESTRING((0 2,1 3,2 2),(1 1,2 0,3 1,3 3,2 2))

Simple non-closed linestring

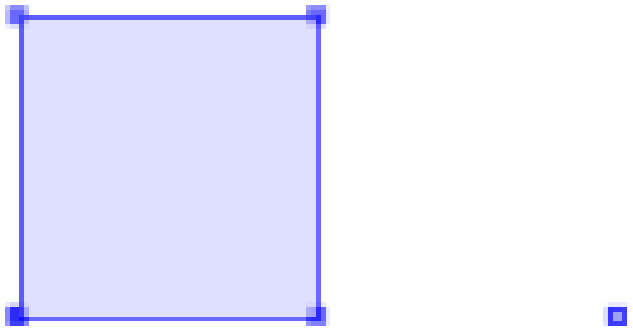
Simple multilinestring defined by 4 endpoints of 2 elements

Polygon/Multipolygon Geometry

- POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
- MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))

Geometrycollection

- GEOMETRYCOLLECTION(POINT(2 0),POLYGON((0 0, 1 0, 1 1, 0 1, 0 0)))



3D Geometries

- PostGIS recognizes and stores 3D geometries, but not yet full support.

- Lack the volumetric sense of 3D
  - 2D object sitting in 3D space
  - 2.5D

Spatial Indexing

Used the GiST (Generalized Search Tree) index

- Actively being developed
  - Teodor Sigaev and Oleg Bartunov
  - http://www.sai.msu.su/~megera/postgres/gist/
- Fast index creation
- Handles compression
  - use bounding box of the feature
- NULL safe
- Can implement an R-Tree using GiST

R-Tree Indexing

- Generalize all the geometries to their bounding box.
  - small to store
  - operations are simple
- Typical search is to find all the objects that overlap a box
- Result is an approximation
  - too many features are returned
- Used to solve overlap and distance problems

Spatial Functions

- A spatial database provides a complete set of functions for analyzing geometric components, determining spatial relationships, and manipulating geometries.

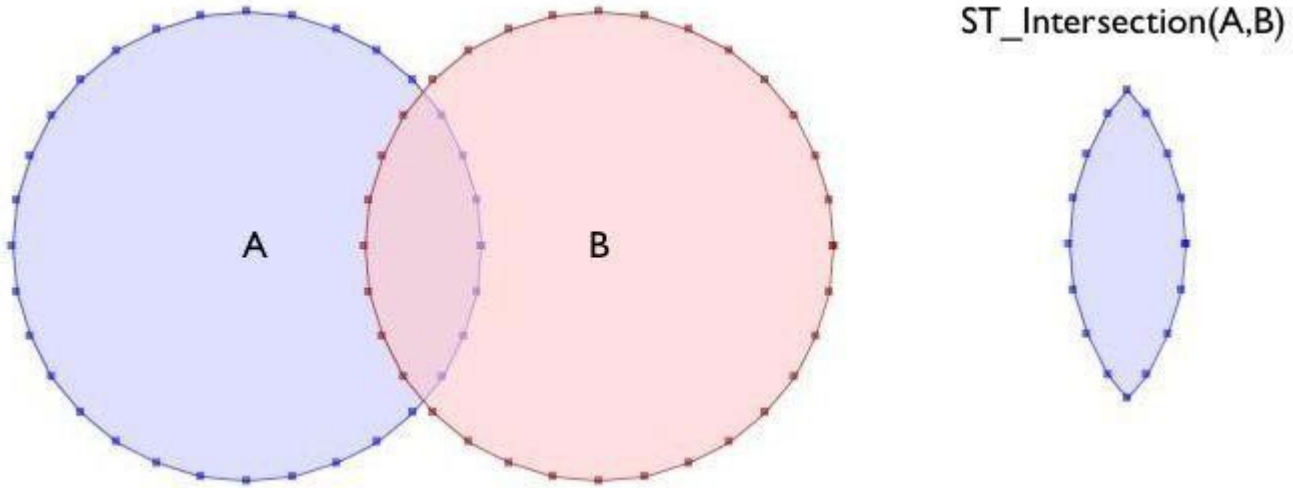
- These spatial functions serve as the building block for any spatial project.

Spatial Functions

- **Conversion**: Functions that *convert* between geometries and external data formats.
- **Management**: Functions that *manage* information about spatial tables and PostGIS administration.
- **Retrieval**: Functions that *retrieve* properties and measurements of a Geometry.
- **Comparison**: Functions that *compare* two geometries with respect to their spatial relation.
- **Generation**: Functions that *generate* new geometries from others.

Spatial Function

- SELECT ST_AsText(ST_Intersection(ST_Buffer('POINT(0 0)', 2), ST_Buffer('POINT(3 0)', 2)));



PostgreSQL Installation

- Free/Open source (http://www.postgresql.org)
- Cross platform
- PL/PgSQL language is required for PostGIS
- Binary package
  - Linux, Mac OS X, Windows etc
  - PostgreSQL
  - PgAdmin III
- Install as Service to allow automatic database start on boot

PostGIS Installation

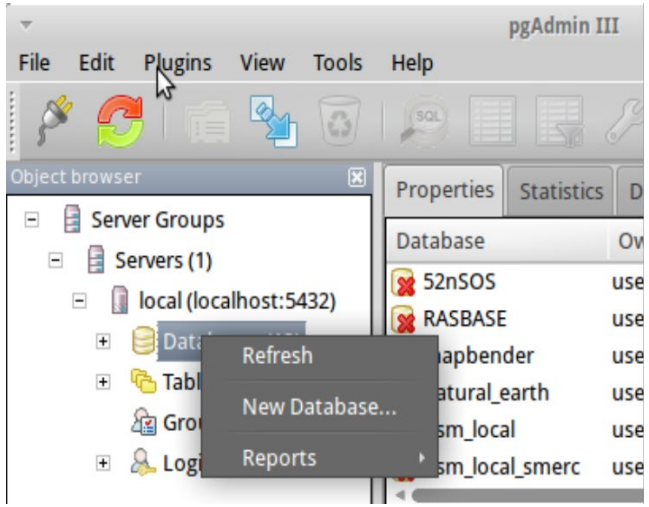
- Free/Open source (http://www.postgis.org/)
- Cross platform
- PL/PgSQL language is required for PostGIS
- Binary package
  - Linux, Mac OS X, Windows etc
- In Window installer, PostGIS bundled with PostgreSQL

Start PostgreSQL instance

- A PostgreSQL instance has one software version and one network port (5432)
- An instance contains many databases
  - Connection string specifies a database
    "-h host –U user –d database –p password"
- A database contains many schemas
  - public
- A schema contains many tables
  - public.geometry_columns
- A table contains many tuples

Spatially Enable PostgreSQL

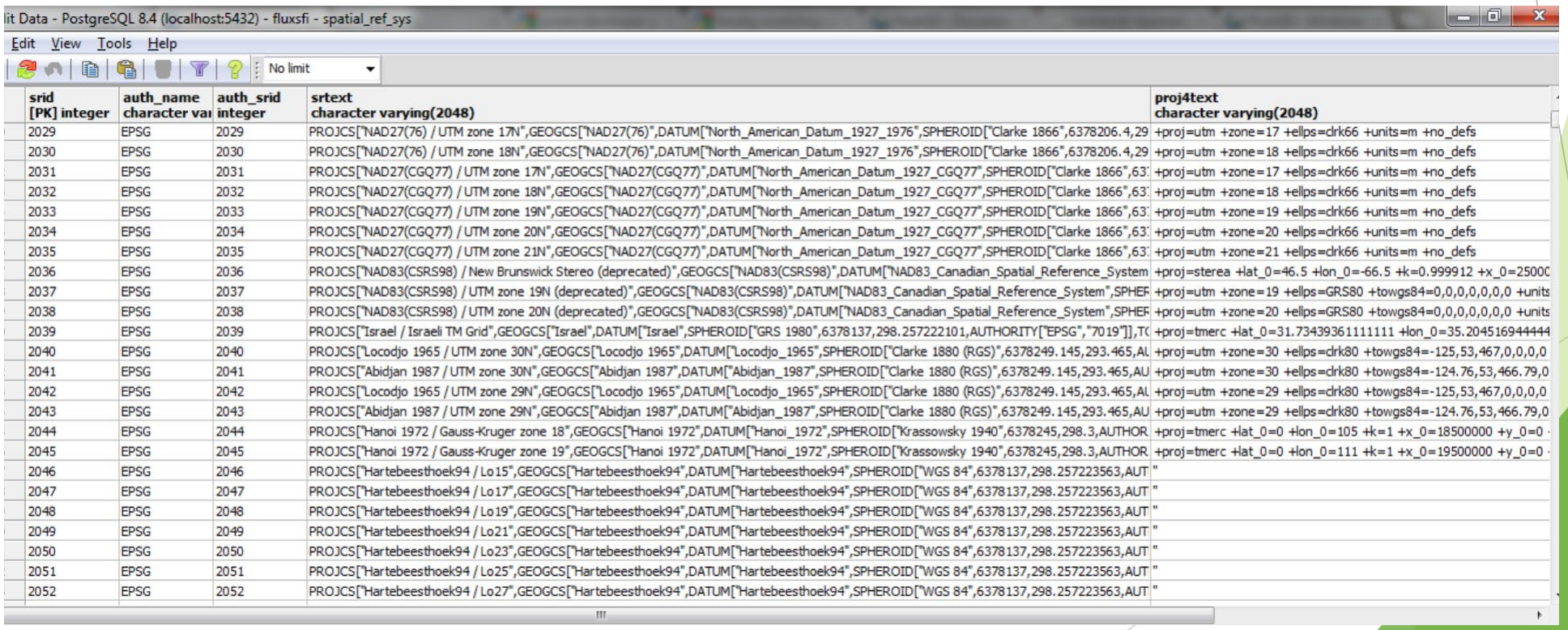
- Create a new database
- GUI in pgAdmin III
  - To enable PostGIS extension
    - CREATE EXTENSION postgis;
    - CREATE EXTENSION postgis_topology;



- Command line
  - createdb nyc_cmd –h localhost –U user –E UTF8 –T template_postgis
    - Remove DB : dropdb nyc_cmd –h localhost –U user

Spatial_ref_sys table

- Defines all the spatial reference systems known to the database and will be described in greater detail later.



it Data - PostgreSQL 8.4 (localhost:5432) - fluxsfi - spatial_ref_sys

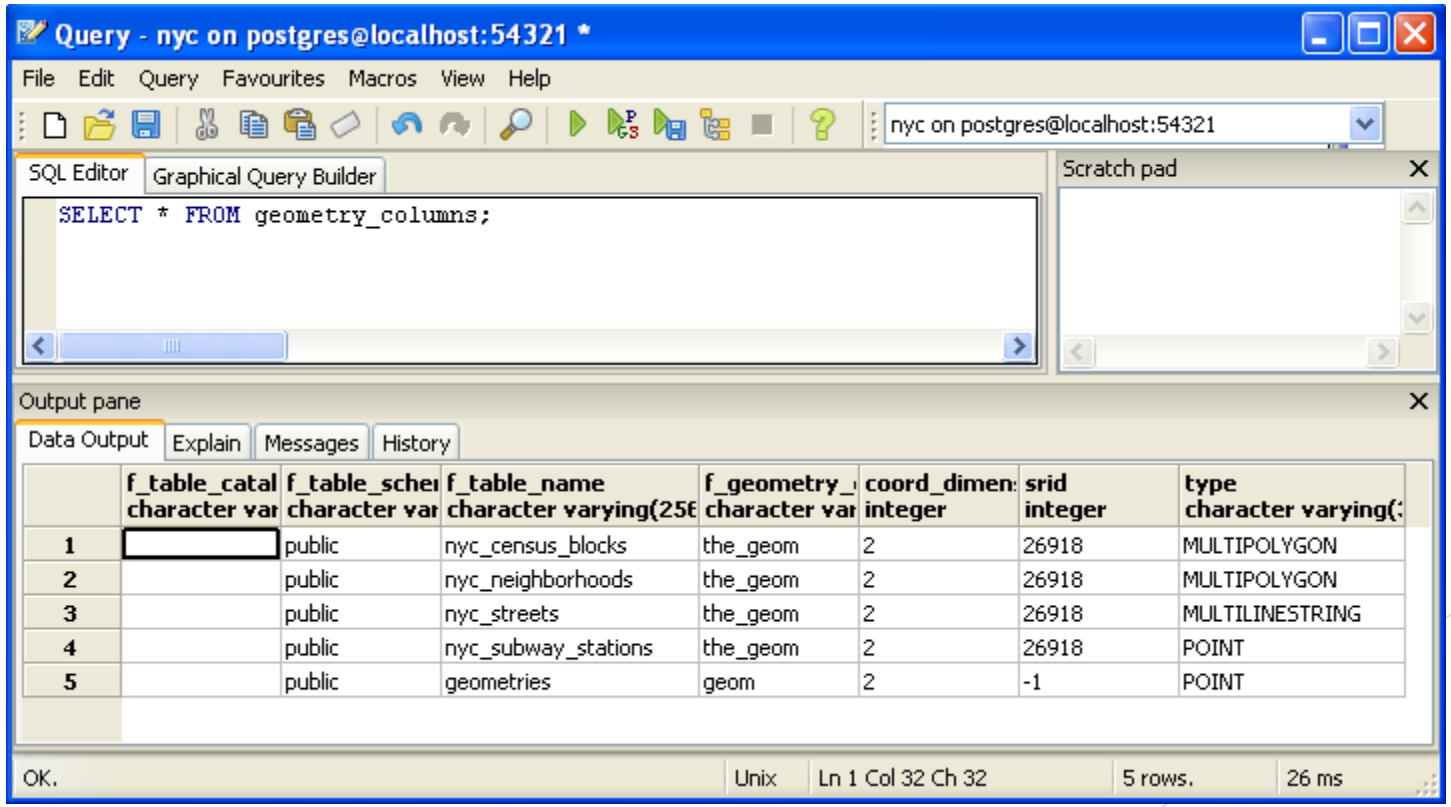
Edit View Tools Help

No limit

	srid [PK] integer	auth_name character var	auth_srid integer	srtext character varying(2048)	proj4text character varying(2048)
	2029	EPSG	2029	PROJCS["NAD27(76) / UTM zone 17N",GEOGCS["NAD27(76)",DATUM["North_American_Datum_1927_1976",SPHEROID["Clarke 1866",6378206.4,29	+proj=utm +zone=17 +ellps=clrk66 +units=m +no_defs
	2030	EPSG	2030	PROJCS["NAD27(76) / UTM zone 18N",GEOGCS["NAD27(76)",DATUM["North_American_Datum_1927_1976",SPHEROID["Clarke 1866",6378206.4,29	+proj=utm +zone=18 +ellps=clrk66 +units=m +no_defs
	2031	EPSG	2031	PROJCS["NAD27(CGQ77) / UTM zone 17N",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1927_CGQ77",SPHEROID["Clarke 1866",63	+proj=utm +zone=17 +ellps=clrk66 +units=m +no_defs
	2032	EPSG	2032	PROJCS["NAD27(CGQ77) / UTM zone 18N",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1927_CGQ77",SPHEROID["Clarke 1866",63	+proj=utm +zone=18 +ellps=clrk66 +units=m +no_defs
	2033	EPSG	2033	PROJCS["NAD27(CGQ77) / UTM zone 19N",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1927_CGQ77",SPHEROID["Clarke 1866",63	+proj=utm +zone=19 +ellps=clrk66 +units=m +no_defs
	2034	EPSG	2034	PROJCS["NAD27(CGQ77) / UTM zone 20N",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1927_CGQ77",SPHEROID["Clarke 1866",63	+proj=utm +zone=20 +ellps=clrk66 +units=m +no_defs
	2035	EPSG	2035	PROJCS["NAD27(CGQ77) / UTM zone 21N",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1927_CGQ77",SPHEROID["Clarke 1866",63	+proj=utm +zone=21 +ellps=clrk66 +units=m +no_defs
	2036	EPSG	2036	PROJCS["NAD83(CSRS98) / New Brunswick Stereo (deprecated)",GEOGCS["NAD83(CSRS98)",DATUM["NAD83_Canadian_Spatial_Reference_System	+proj=sterea +lat_0=46.5 +lon_0=-66.5 +k=0.999912 +x_0=25000
	2037	EPSG	2037	PROJCS["NAD83(CSRS98) / UTM zone 19N (deprecated)",GEOGCS["NAD83(CSRS98)",DATUM["NAD83_Canadian_Spatial_Reference_System",SPHEF	+proj=utm +zone=19 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units
	2038	EPSG	2038	PROJCS["NAD83(CSRS98) / UTM zone 20N (deprecated)",GEOGCS["NAD83(CSRS98)",DATUM["NAD83_Canadian_Spatial_Reference_System",SPHEF	+proj=utm +zone=20 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units
	2039	EPSG	2039	PROJCS["Israel / Israeli TM Grid",GEOGCS["Israel",DATUM["Israel",SPHEROID["GRS 1980",6378137,298.257222101,AUTHORITY["EPSG","7019"]],TC	+proj=tmerc +lat_0=31.73439361111111 +lon_0=35.20451694444
	2040	EPSG	2040	PROJCS["Locodjo 1965 / UTM zone 30N",GEOGCS["Locodjo 1965",DATUM["Locodjo_1965",SPHEROID["Clarke 1880 (RGS)",6378249.145,293.465,AL	+proj=utm +zone=30 +ellps=clrk80 +towgs84=-125,53,467,0,0,0,0
	2041	EPSG	2041	PROJCS["Abidjan 1987 / UTM zone 30N",GEOGCS["Abidjan 1987",DATUM["Abidjan_1987",SPHEROID["Clarke 1880 (RGS)",6378249.145,293.465,AU	+proj=utm +zone=30 +ellps=clrk80 +towgs84=-124.76,53,466.79,0
	2042	EPSG	2042	PROJCS["Locodjo 1965 / UTM zone 29N",GEOGCS["Locodjo 1965",DATUM["Locodjo_1965",SPHEROID["Clarke 1880 (RGS)",6378249.145,293.465,AL	+proj=utm +zone=29 +ellps=clrk80 +towgs84=-125,53,467,0,0,0,0
	2043	EPSG	2043	PROJCS["Abidjan 1987 / UTM zone 29N",GEOGCS["Abidjan 1987",DATUM["Abidjan_1987",SPHEROID["Clarke 1880 (RGS)",6378249.145,293.465,AU	+proj=utm +zone=29 +ellps=clrk80 +towgs84=-124.76,53,466.79,0
	2044	EPSG	2044	PROJCS["Hanoi 1972 / Gauss-Kruger zone 18",GEOGCS["Hanoi 1972",DATUM["Hanoi_1972",SPHEROID["Krassowsky 1940",6378245,298.3,AUTHOR	+proj=tmerc +lat_0=0 +lon_0=105 +k=1 +x_0=18500000 +y_0=0
	2045	EPSG	2045	PROJCS["Hanoi 1972 / Gauss-Kruger zone 19",GEOGCS["Hanoi 1972",DATUM["Hanoi_1972",SPHEROID["Krassowsky 1940",6378245,298.3,AUTHOR	+proj=tmerc +lat_0=0 +lon_0=111 +k=1 +x_0=19500000 +y_0=0
	2046	EPSG	2046	PROJCS["Hartebeesthoek94 / Lo15",GEOGCS["Hartebeesthoek94",DATUM["Hartebeesthoek94",SPHEROID["WGS 84",6378137,298.257223563,AUT	"
	2047	EPSG	2047	PROJCS["Hartebeesthoek94 / Lo17",GEOGCS["Hartebeesthoek94",DATUM["Hartebeesthoek94",SPHEROID["WGS 84",6378137,298.257223563,AUT	"
	2048	EPSG	2048	PROJCS["Hartebeesthoek94 / Lo19",GEOGCS["Hartebeesthoek94",DATUM["Hartebeesthoek94",SPHEROID["WGS 84",6378137,298.257223563,AUT	"
	2049	EPSG	2049	PROJCS["Hartebeesthoek94 / Lo21",GEOGCS["Hartebeesthoek94",DATUM["Hartebeesthoek94",SPHEROID["WGS 84",6378137,298.257223563,AUT	"
	2050	EPSG	2050	PROJCS["Hartebeesthoek94 / Lo23",GEOGCS["Hartebeesthoek94",DATUM["Hartebeesthoek94",SPHEROID["WGS 84",6378137,298.257223563,AUT	"
	2051	EPSG	2051	PROJCS["Hartebeesthoek94 / Lo25",GEOGCS["Hartebeesthoek94",DATUM["Hartebeesthoek94",SPHEROID["WGS 84",6378137,298.257223563,AUT	"
	2052	EPSG	2052	PROJCS["Hartebeesthoek94 / Lo27",GEOGCS["Hartebeesthoek94",DATUM["Hartebeesthoek94",SPHEROID["WGS 84",6378137,298.257223563,AUT	"

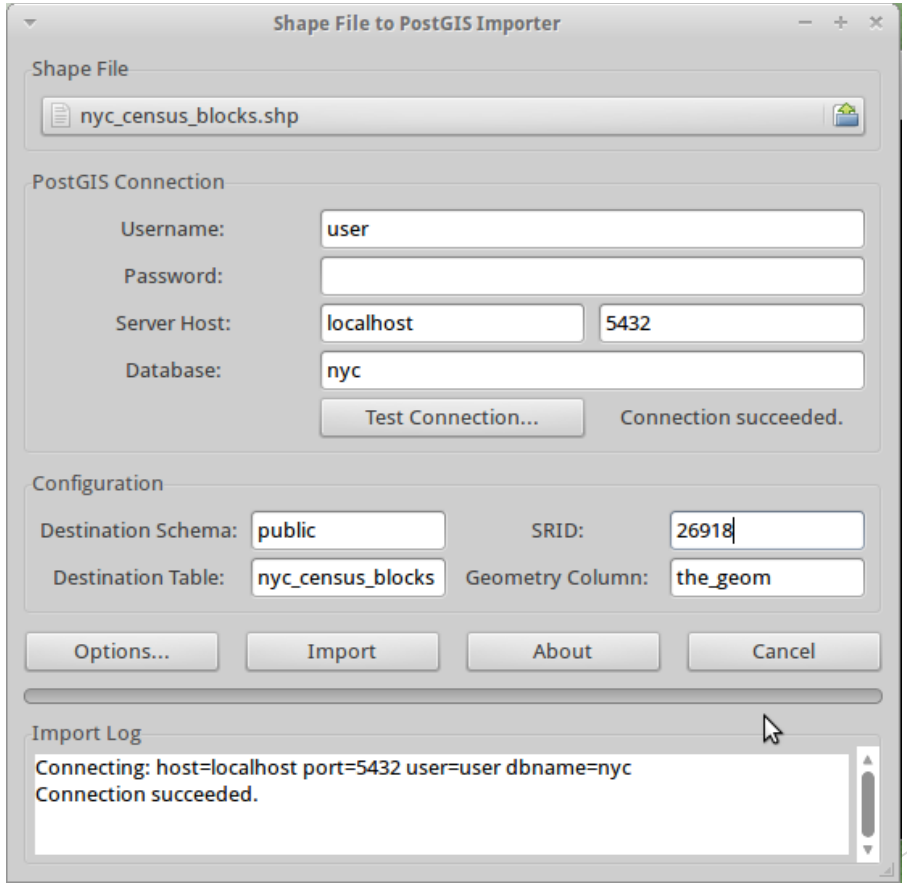
Geometry_columns table

- Provides a listing of all “features” (defined as an object with geometric attributes), and the basic details of those features.



Loading Shape File

- PostGIS provides many options for loading data.
  - The GUI shapefile importer



Loading Shape File

- shp2pgsql [opts] *shapefile tablename*
  - **shp2pgsql -s 26918 -D /home/users/data/nyc/nyc_neighborhoods.shp nyc_neighborhoods > neighborhoods.sql**
- Read in .shp file and Write out .sql file
- Load .sql file into PostgreSQL
  - using psql

    psql –h localhost –U user –d nyc –f neighborhoods.sql
  - using PgAdmin

SQL content (Well-Known Binary, WKB)

neighborhoods.sql

File Edit Search Options Help

```
SET CLIENT_ENCODING TO UTF8;
SET STANDARD_CONFORMING_STRINGS TO ON;
BEGIN;
CREATE TABLE "nyc_neighborhoods" (gid serial PRIMARY KEY,
"boroname" varchar(43),
"name" varchar(64));
SELECT AddGeometryColumn('','nyc_neighborhoods','the_geom','26918','MULTIPOLYGON',2);
COPY "nyc_neighborhoods" ("boroname","name",the_geom) FROM stdin;
Brooklyn        Bensonhurst     0106000020266900000100000001030000202669000001000000110000004
Manhattan       East Village    0106000020266900000100000001030000202669000001000000080000000E94
Manhattan       West Village    0106000020266900000100000001030000202669000001000000340000009E6
The Bronx       Throggs Neck    0106000020266900000100000001030000202669000001000000570000000E6A
The Bronx       Wakefield-Williamsbridge        0106000020266900000100000001030000202669000001
Queens  Auburndale      0106000020266900000100000001030000202669000001000000190000000308396041A5
Manhattan       Battery Park    0106000020266900000100000001030000202669000001000000010000001F00000061F
Manhattan       Carnegie Hill   0106000020266900000100000001030000202669000001000000060000000012B
Staten Island   Mariners Harbor 0106000020266900000100000001030000202669000001000000010000001B0000002A3
Staten Island   Rossville       0106000020266900000100000001030000202669000001000000460000002F67
Manhattan       Harlem  0106000020266900000100000001030000202669000001000000170000000DE75FBF8590
Manhattan       Gramercy        0106000020266900000100000001030000202669000001000000160000004 5D
Queens  Queens Village  0106000020266900000100000001030000202669000001000000140000001BFD764A848
Queens  Middle Village  0106000020266900000100000001030000202669000001000000310000000EF5ACFF7122
Staten Island   Ettingville     0106000020266900000100000001030000202669000001000000220000000900
The Bronx       Morris Park     0106000020266900000100000001030000202669000001000000230000000D49
The Bronx       Baychester      0106000020266900000100000001030000202669000001000000D000000DAFA
Staten Island   Great Kills     0106000020266900000100000001030000202669000001000000250000000CBE0
Staten Island   New Brighton    0106000020266900000100000001030000202669000001000000230000009B21
The Bronx       Fordham 0106000020266900000100000001030000202669000001000000F0000003616F7E20D10
Queens  Nkew Gardens    0106000020266900000100000001030000202669000001000000 2C0000002B9F298D7440
```

SQL Content (Well-Known Text, WKT)

neighborhoods.sql

File Edit Search Options Help

```
SET CLIENT_ENCODING TO UTF8;
SET STANDARD_CONFORMING_STRINGS TO ON;
BEGIN;
CREATE TABLE "nyc_neighborhoods" (gid serial PRIMARY KEY,
"boroname" varchar(43),
"name" varchar(64));
SELECT AddGeometryColumn('','nyc_neighborhoods','the_geom','26918','MULTIPOLYGON',2);
COPY "nyc_neighborhoods" ("boroname","name",the_geom) FROM stdin;
Brooklyn        Bensonhurst     SRID=26918;SRID=26918;MULTIPOLYGON(((582771.425719806 4495167.42
Manhattan       East Village    SRID=26918;SRID=26918;MULTIPOLYGON(((585508.753489015 4509691.26
Manhattan       West Village    SRID=26918;SRID=26918;MULTIPOLYGON(((583263.277659584 4509242.62
The Bronx       Throggs Neck    SRID=26918;SRID=26918;MULTIPOLYGON(((597640.009068814 4520272.71
The Bronx       Wakefield-Williamsbridge        SRID=26918;SRID=26918;MULTIPOLYGON(((595285.2053
Queens  Auburndale      SRID=26918;SRID=26918;MULTIPOLYGON(((600973.008960819 4510338.85744586,6
Manhattan       Battery Park    SRID=26918;SRID=26918;MULTIPOLYGON(((583408.101005476 4508093.11
Manhattan       Carnegie Hill   SRID=26918;SRID=26918;MULTIPOLYGON(((588501.208387079 4515525.87
Staten Island   Mariners Harbor SRID=26918;SRID=26918;MULTIPOLYGON(((570300.108079498 4497031.15
Staten Island   Rossville       SRID=26918;SRID=26918;MULTIPOLYGON(((564664.956782555 4489358.42
Manhattan       Harlem  SRID=26918;SRID=26918;MULTIPOLYGON(((589996.986293491 4517957.74830829,5
Manhattan       Gramercy        SRID=26918;SRID=26918;MULTIPOLYGON(((585709.976851173 4511351.14
Queens  Queens Village  SRID=26918;SRID=26918;MULTIPOLYGON(((607298.145439062 4508590.31406075,6
Queens  Middle Village  SRID=26918;SRID=26918;MULTIPOLYGON(((594825.48400387 4506608.15986962,59
Staten Island   Ettingville     SRID=26918;SRID=26918;MULTIPOLYGON(((570690.335960389 4491182.21
The Bronx       Morris Park     SRID=26918;SRID=26918;MULTIPOLYGON(((594434.019541056 4521319.70
The Bronx       Baychester      SRID=26918;SRID=26918;MULTIPOLYGON(((596434.479156341 4523597.71
Staten Island   Great Kills     SRID=26918;SRID=26918;MULTIPOLYGON(((571214.265898907 4491325.36
Staten Island   New Brighton    SRID=26918;SRID=26918;MULTIPOLYGON(((573781.566285211 4499218.99
The Bronx       Fordham SRID=26918;SRID=26918;MULTIPOLYGON(((592646.943291372 4524122.02609359,5
Queens  Nkew Gardens    SRID=26918;SRID=26918;MULTIPOLYGON(((598074.275708174 4507673.26130292,5
```

Basic sql command – select query

The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

Syntax

The basic syntax of the SELECT statement is as follows.:

```
SELECT column1, column2, columnN FROM table_name;
```

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

```
SELECT * FROM table_name;
```

Basic sql command - WHERE

The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records.

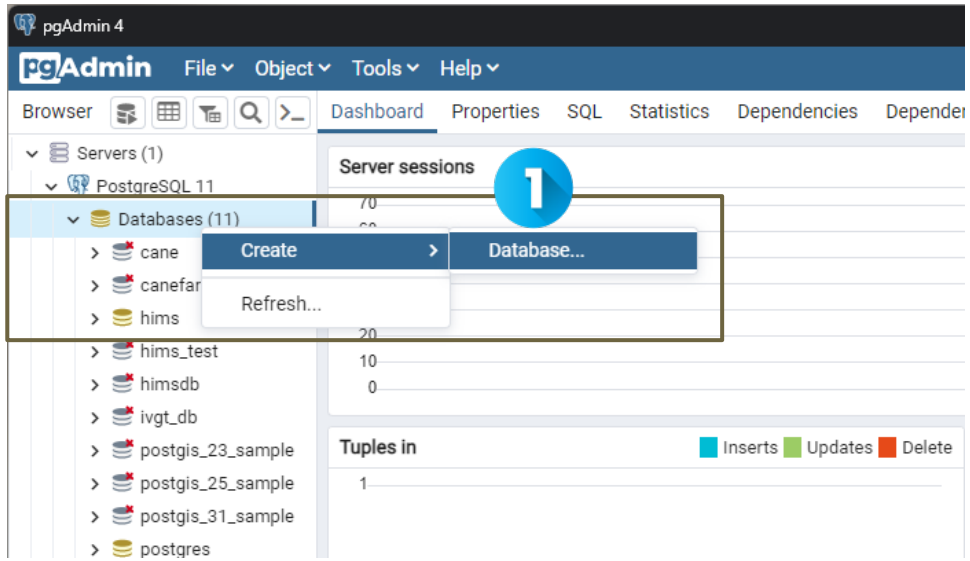
The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc., which we would examine in the subsequent chapters.

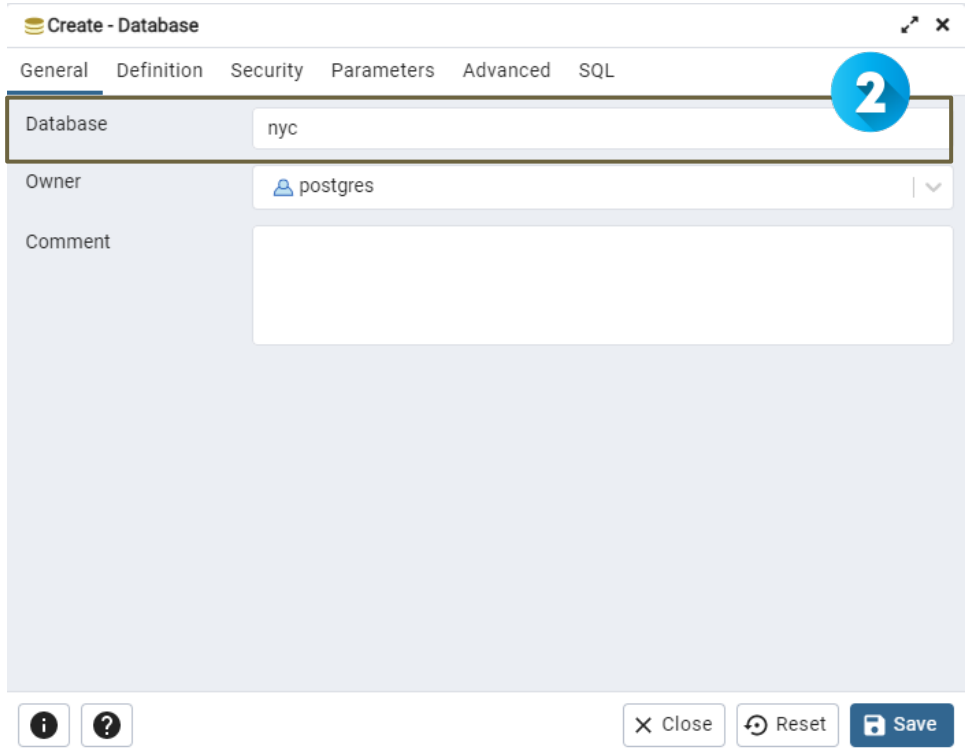
Syntax

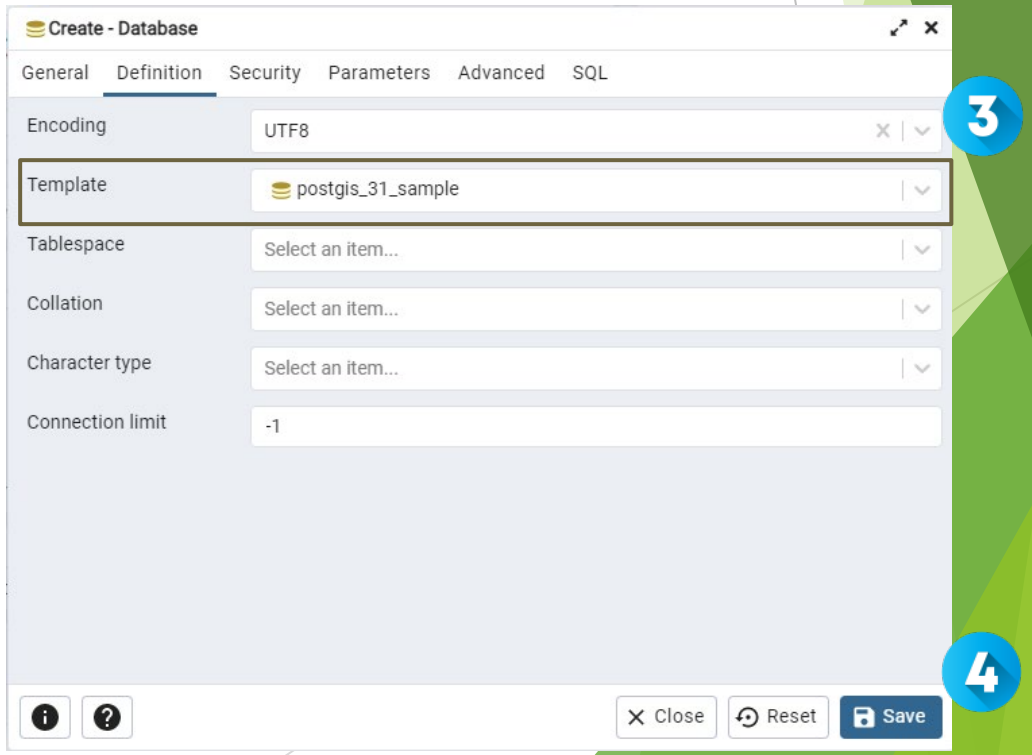
The basic syntax of the SELECT statement with the WHERE clause is as shown below.

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

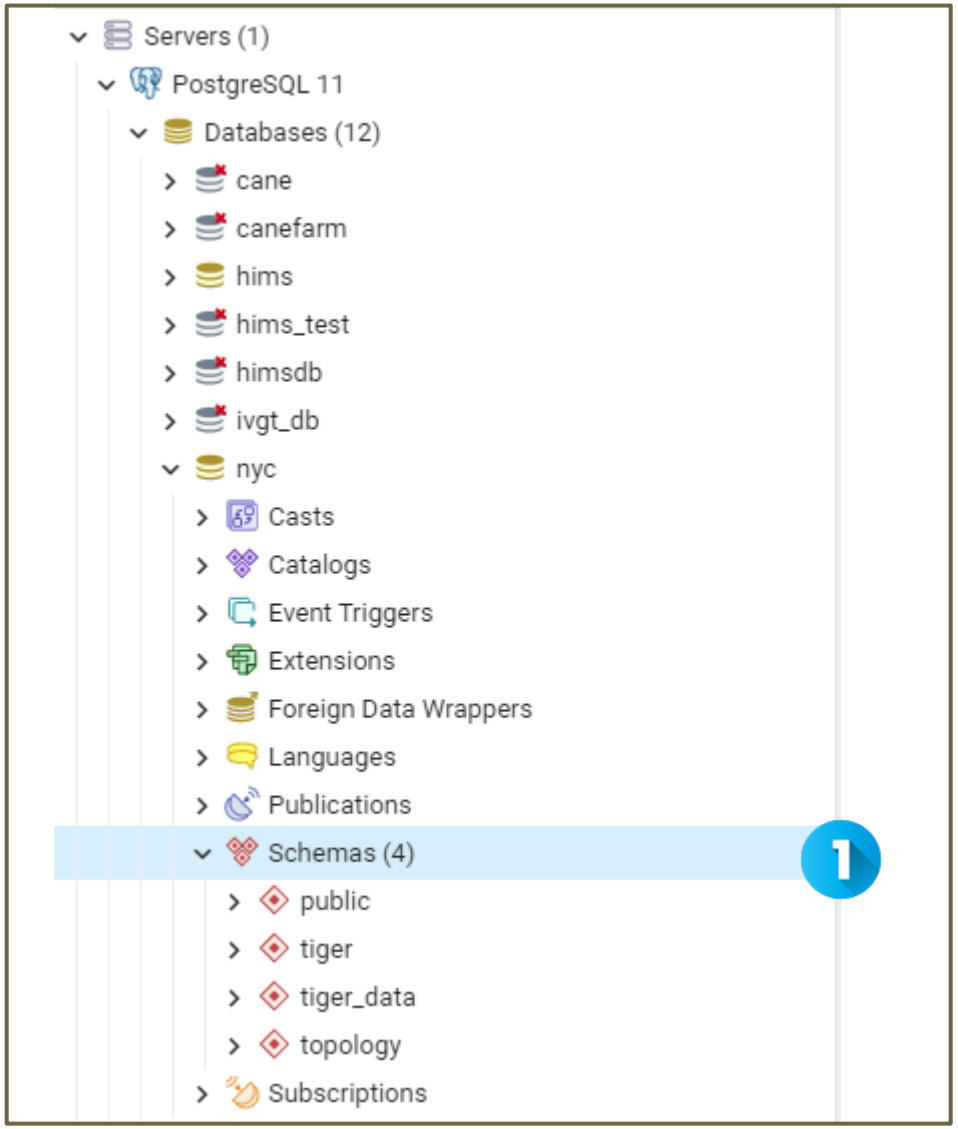
Create Database

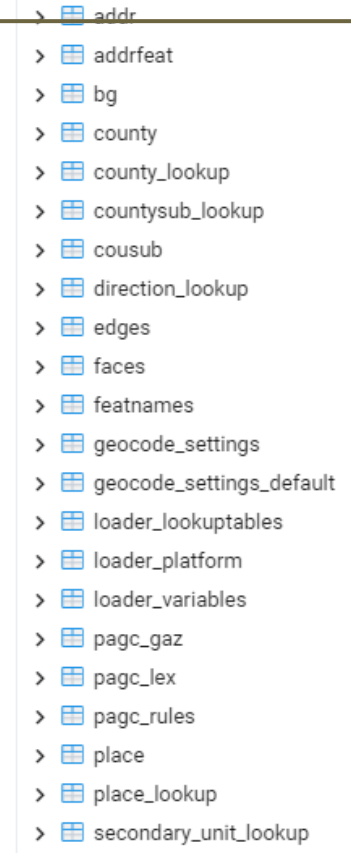




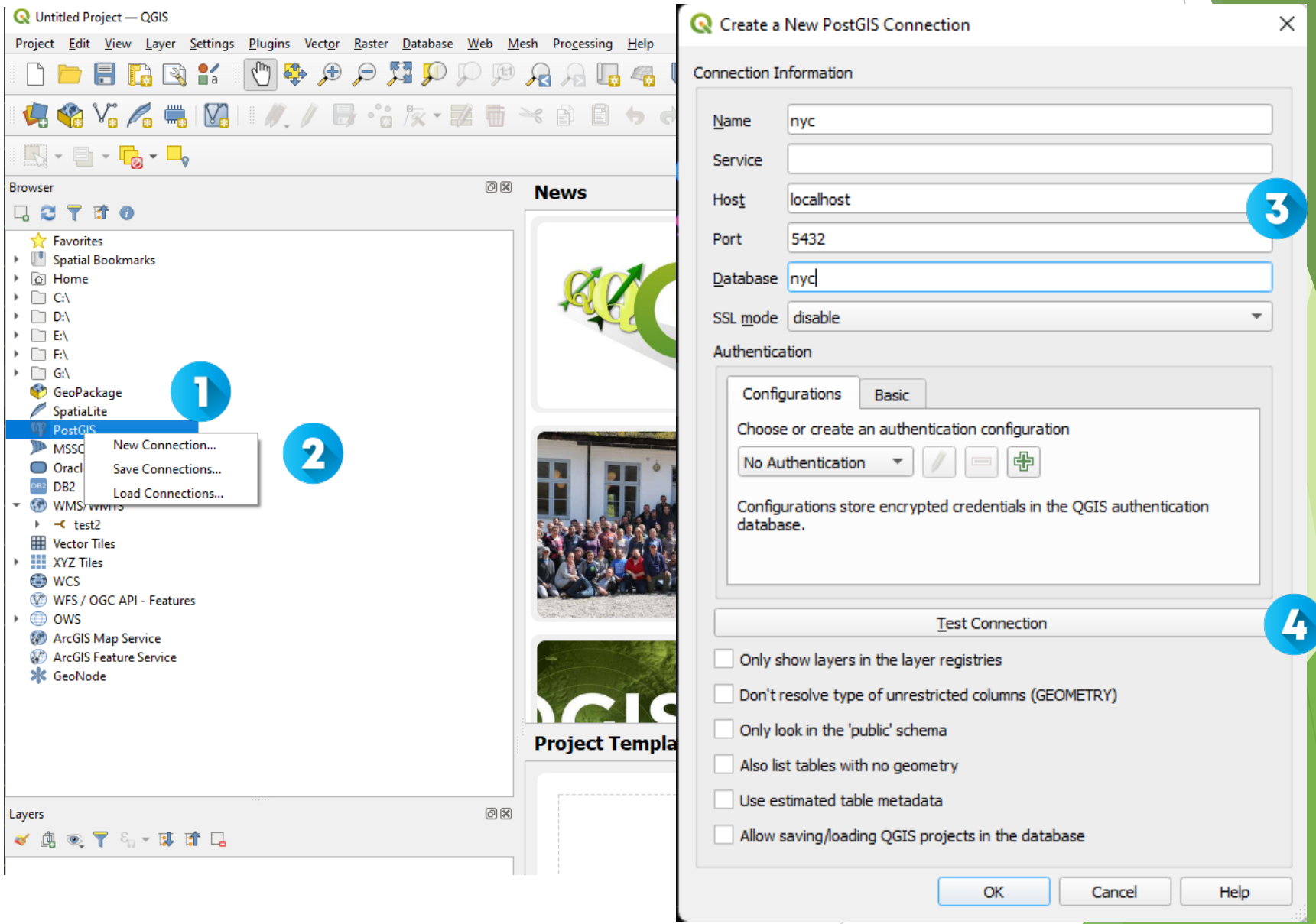


Create Database

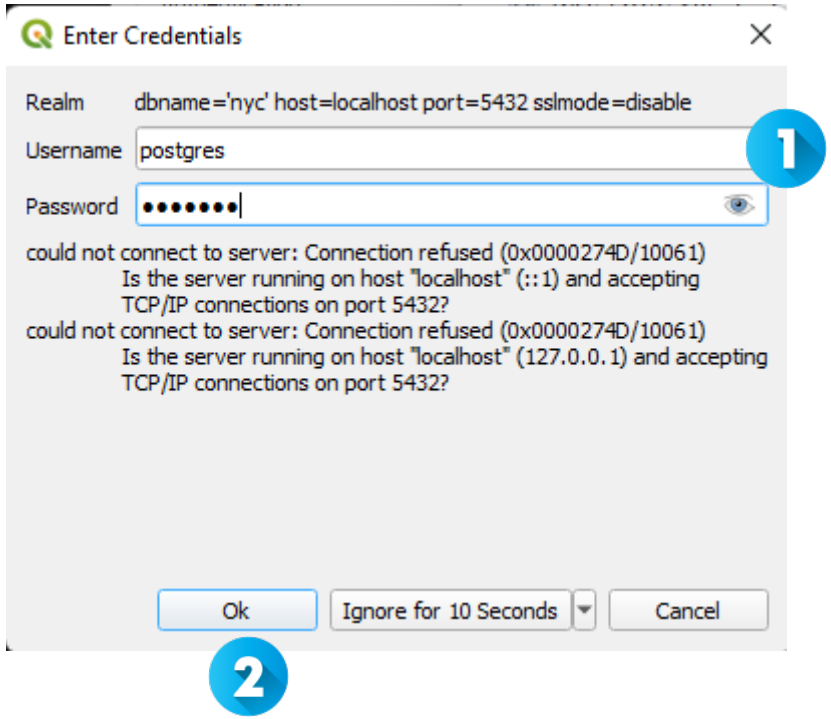


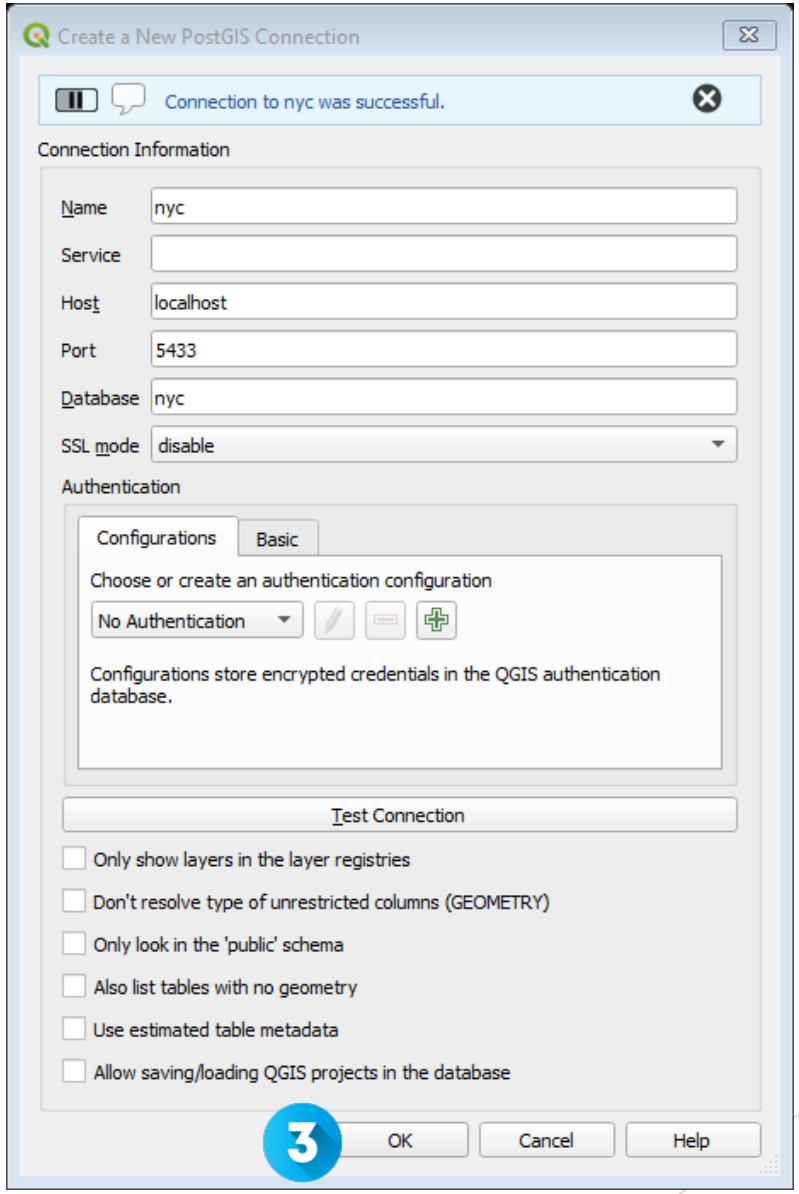


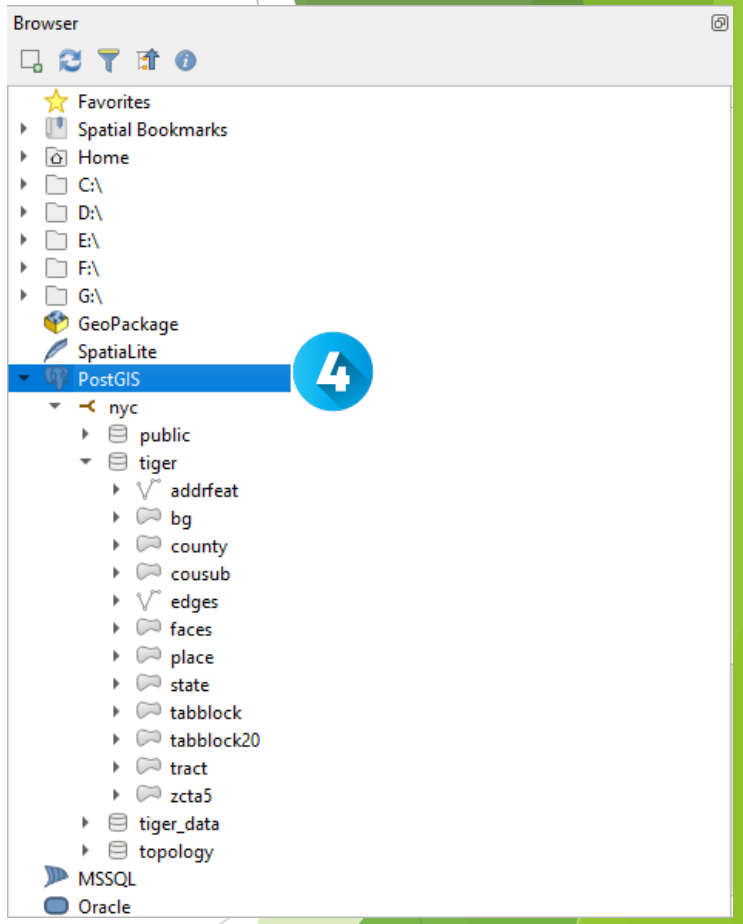
Connect QGIS to PostGIS DB



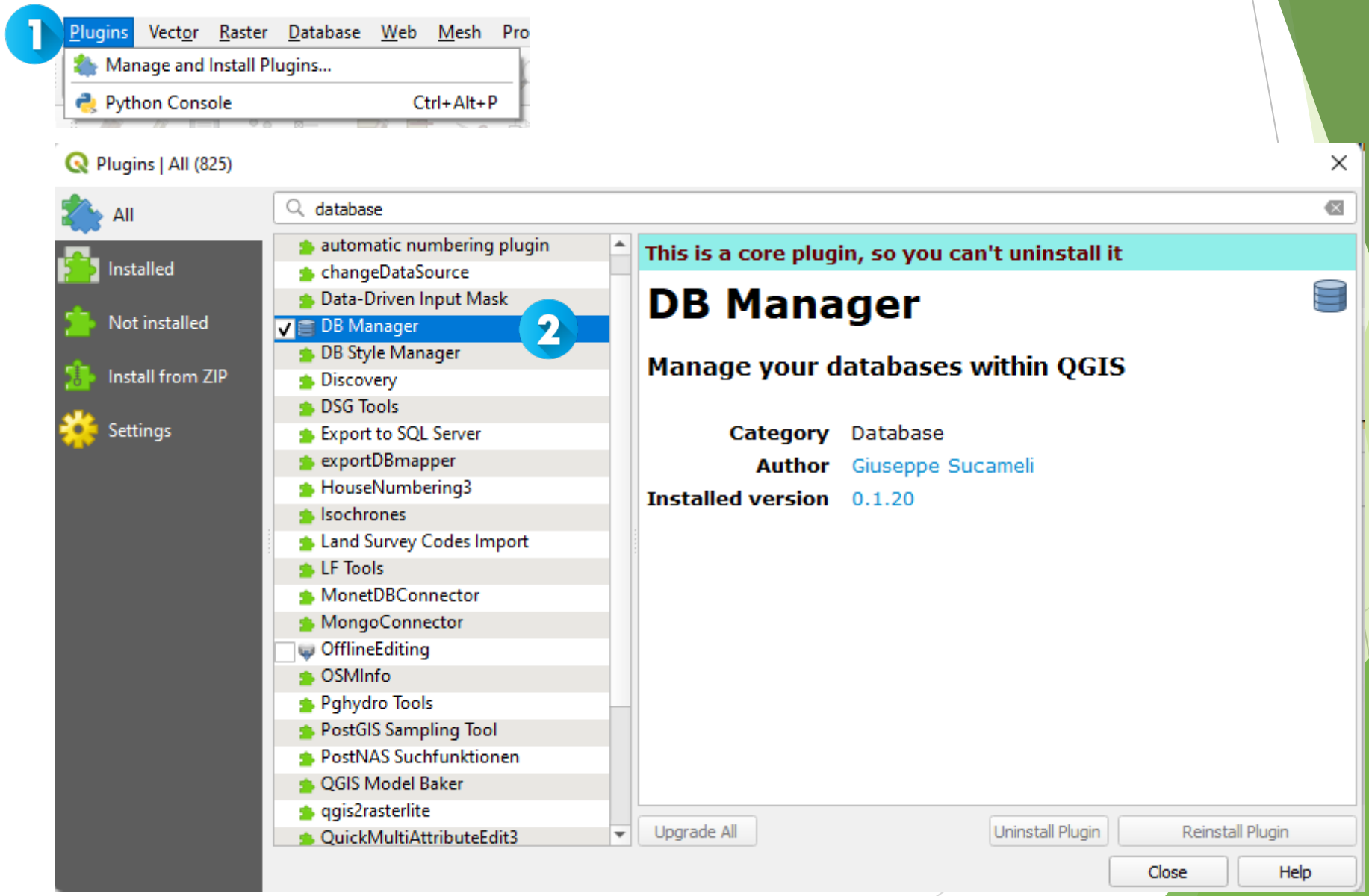
Connect QGIS to PostGIS DB



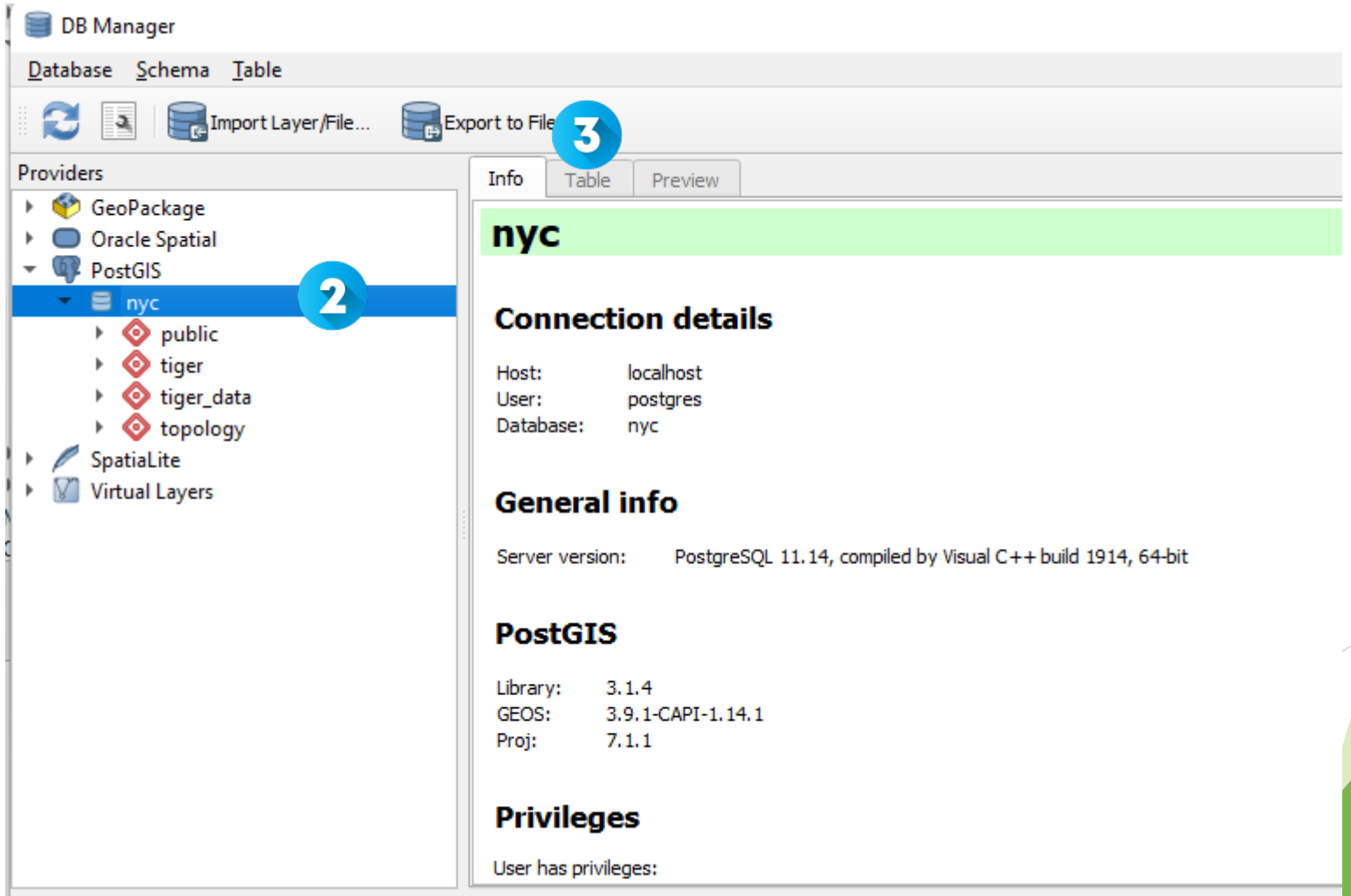




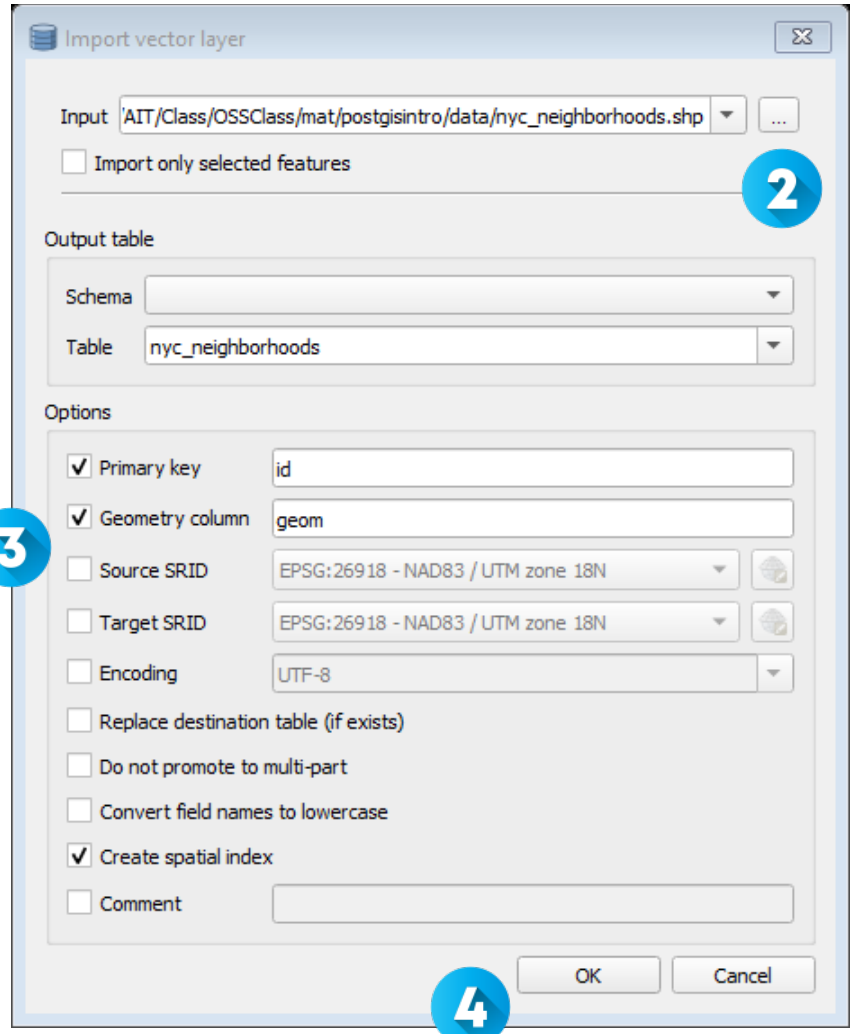
Activate DB Manage plugins

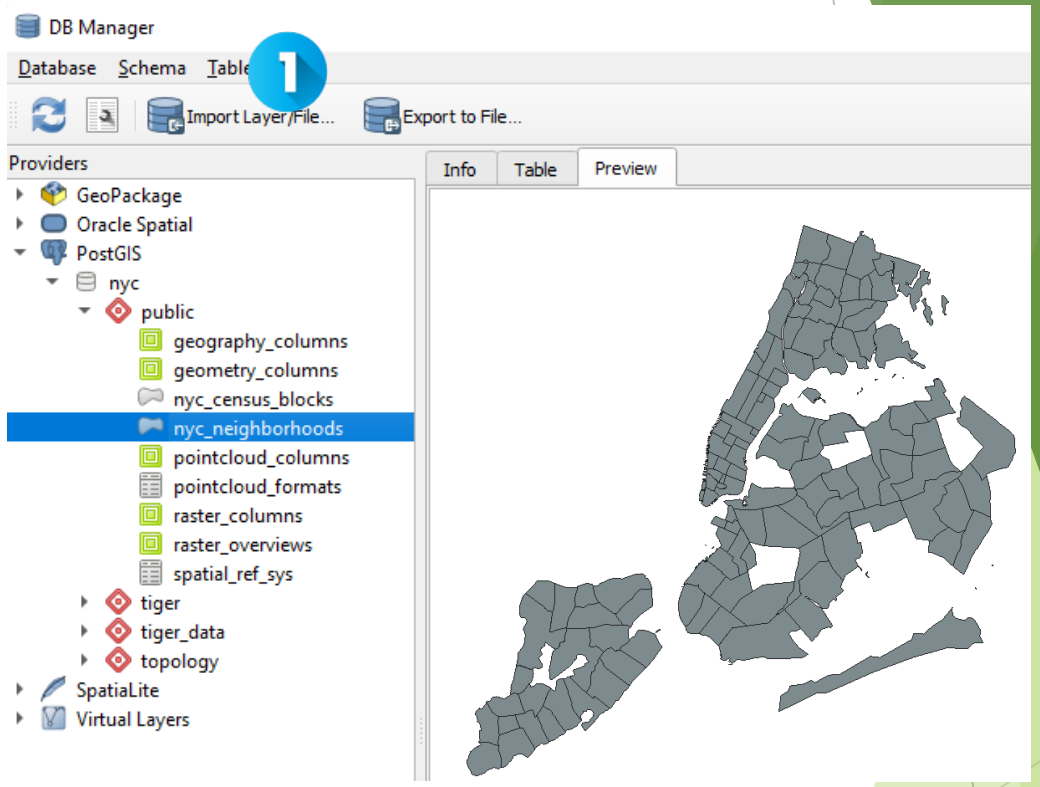


Import Data into DB

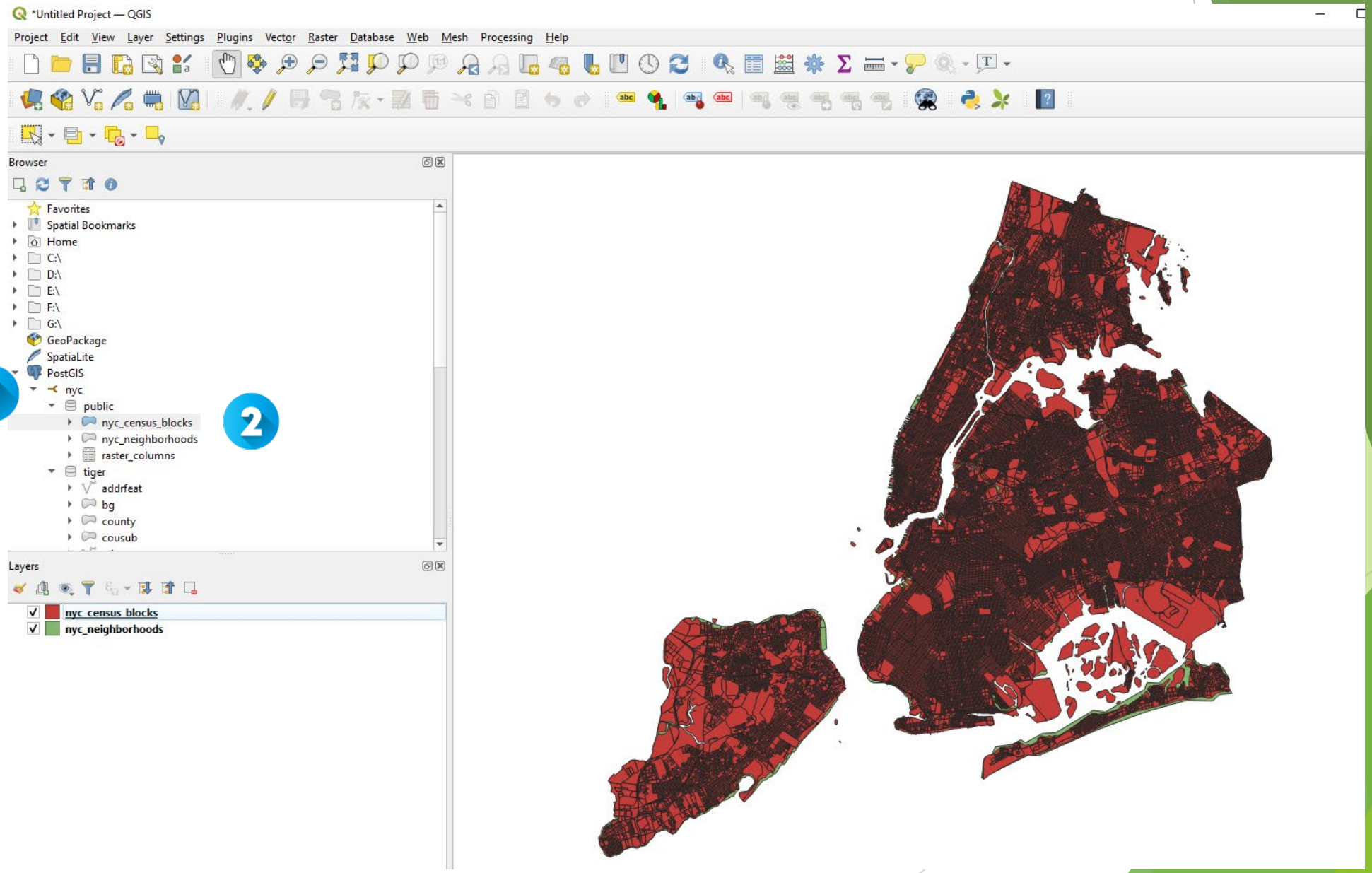


Import Data into DB

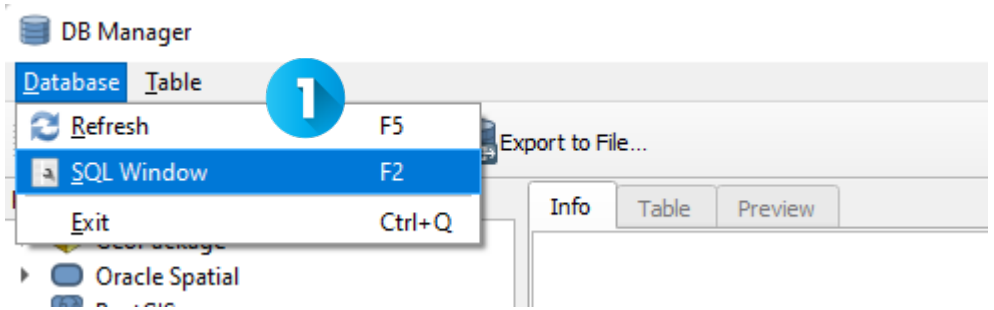


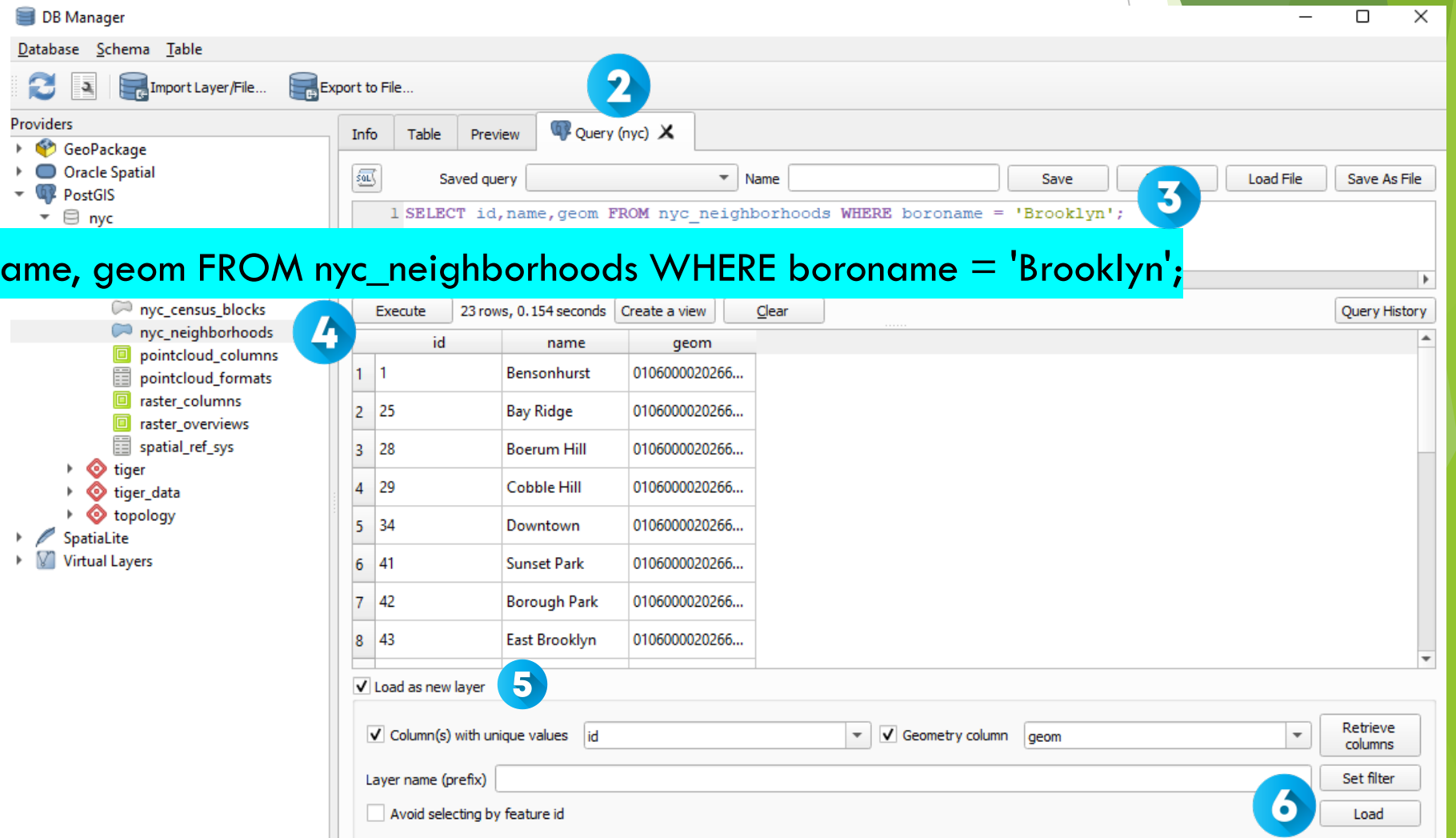


View Data on QGIS

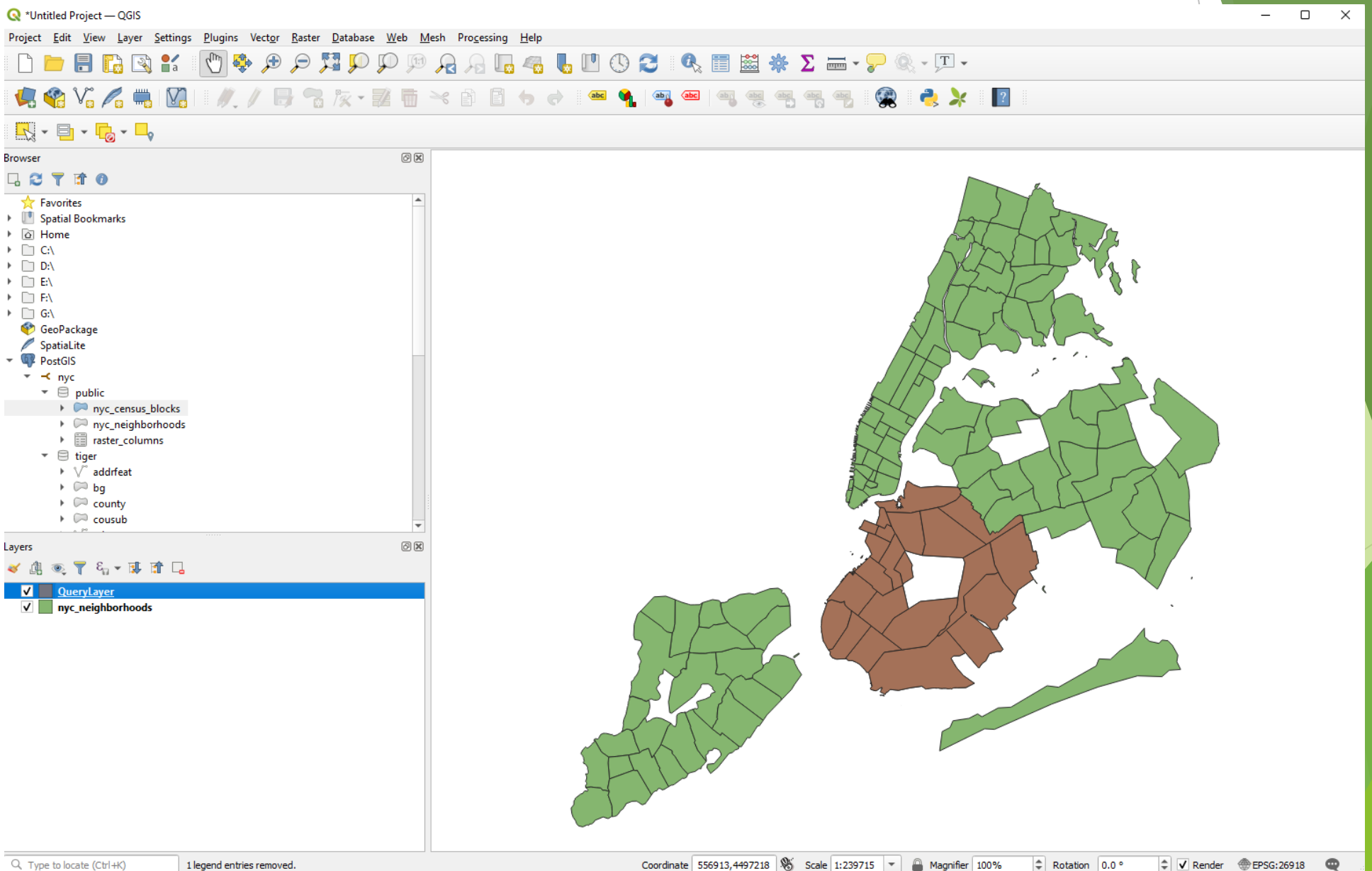


View Data on QGIS

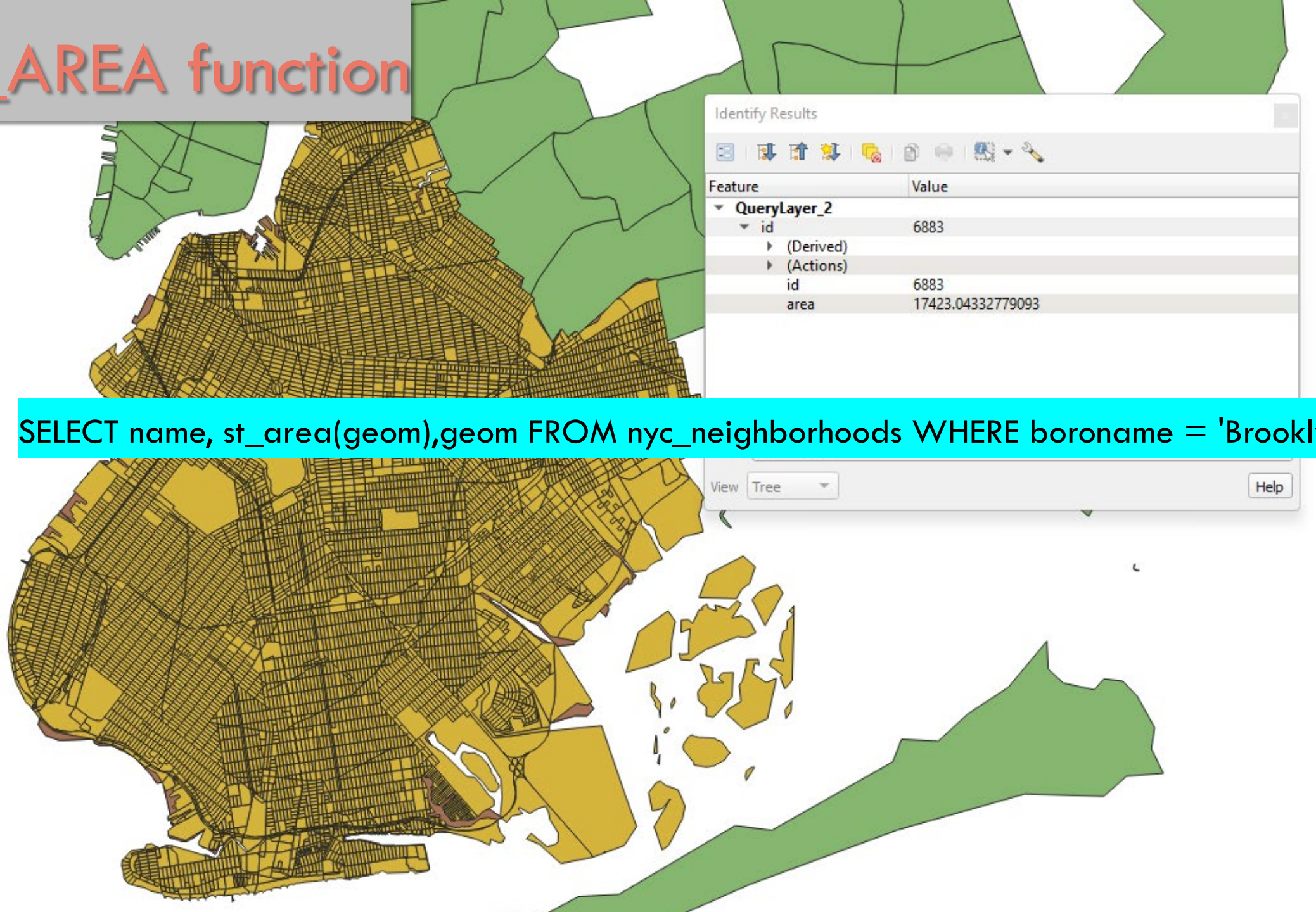




View Data on QGIS



ST_AREA function




```
SELECT name, st_area(geom),geom FROM nyc_neighborhoods WHERE boroname = 'Brooklyn'
```

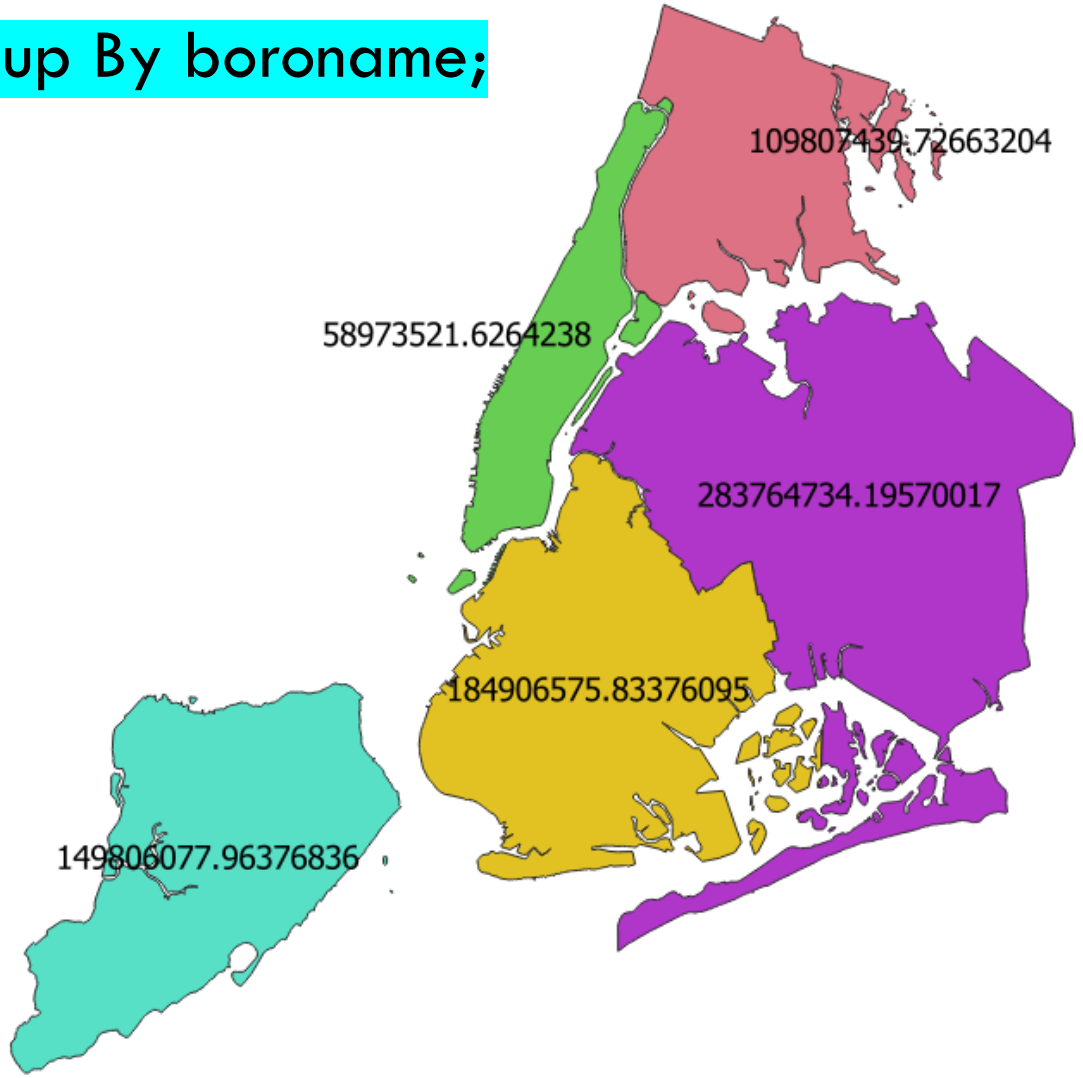
Browser

- Favorites
- Spatial Bookmarks
- Home
- C:\
- D:\
- E:\
- F:\
- G:\
- GeoPackage
- SpatiaLite
- PostGIS
  - nyc
    - public
      - nyc_census_blocks
      - nyc_neighborhoods
      - raster_columns
    - tiger
      - addrfeat
      - bg
      - county
      - cousub
      - edges
      - faces
      - place

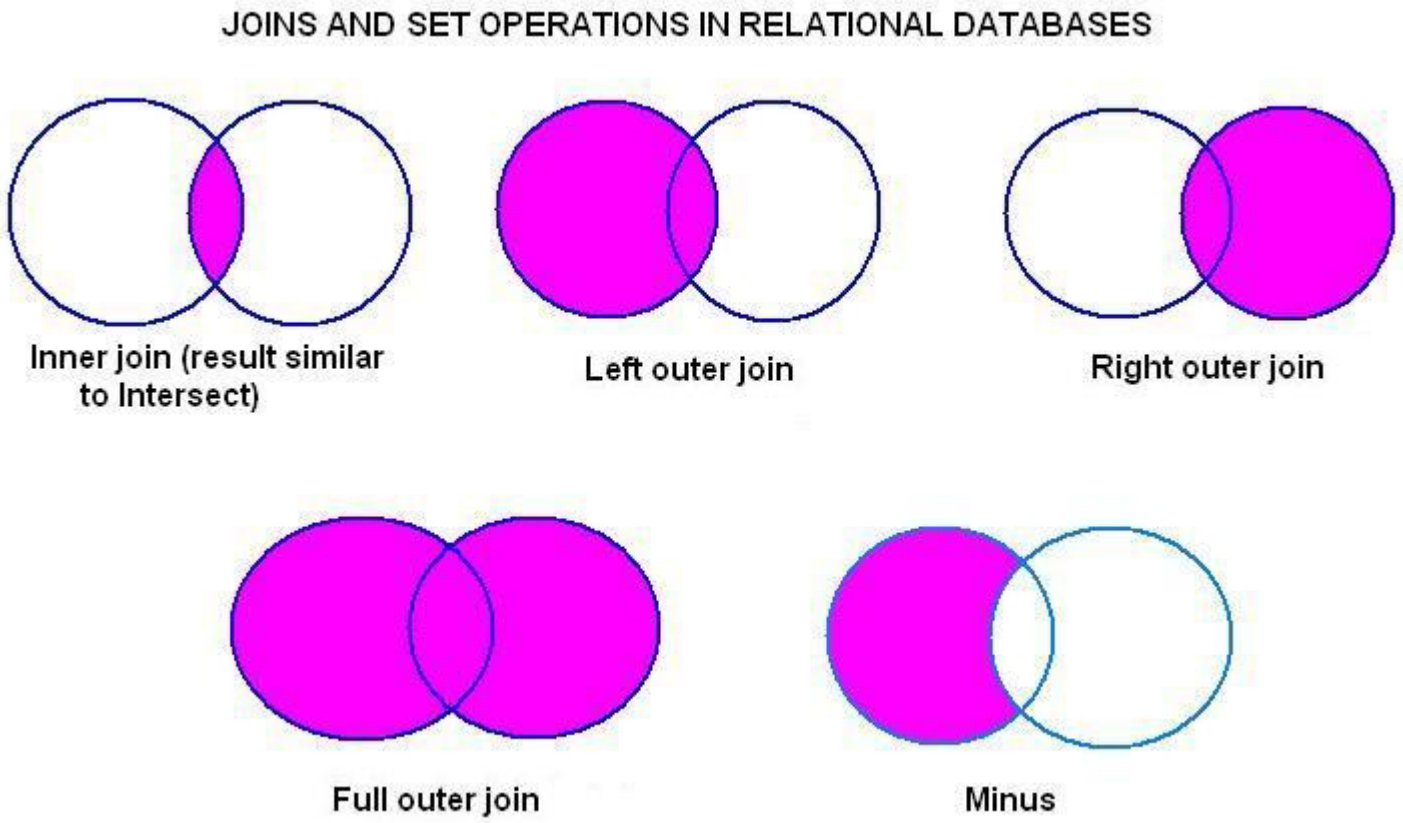
Layers

- QueryLayer_3
  - Brooklyn
  - Manhattan
  - Queens
  - Staten Island
  - The Bronx
  -
- QueryLayer_2
- QueryLayer
- nyc_neighborhoods

```
SELECT SUM(ST_AREA(geom)) as TotalArea, boroname, ST_Union(geom) as
geom FROM nyc_census_blocks Group By boroname;
```



JOIN operator



Result of JOIN

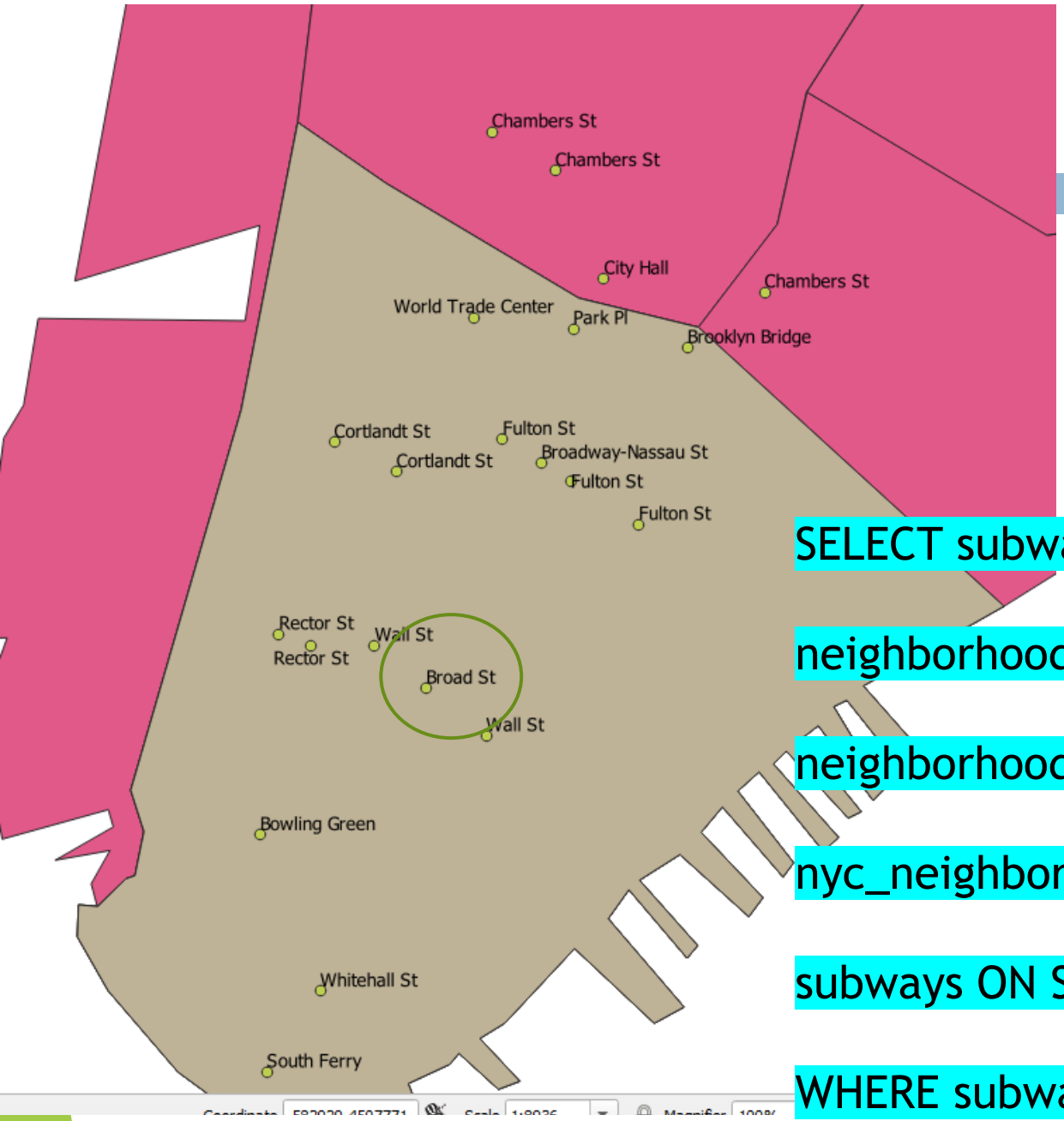
- INNER JOIN: Select only those rows that have values in common in the columns specified in the ON clause.

- LEFT, RIGHT, or FULL OUTER JOIN: Select all rows from the table on the left (or right, or both) regardless of whether the other table has values in common and (usually) enter NULL where data is missing.

Spatial Join

- Allow you to combine information from different tables by using spatial relationships as the join key.
- We can think that “standard GIS analysis” can be expressed as spatial joins.
- Any function that provides a true/false relationship between two tables can be used to drive a spatial join, but the most commonly used ones are:
  - **ST_Intersects, ST_Contains,** and **ST_DWithin**.
- By default, INNER JOIN is used.

ex1



What is neighborhood of subway station Broad Street

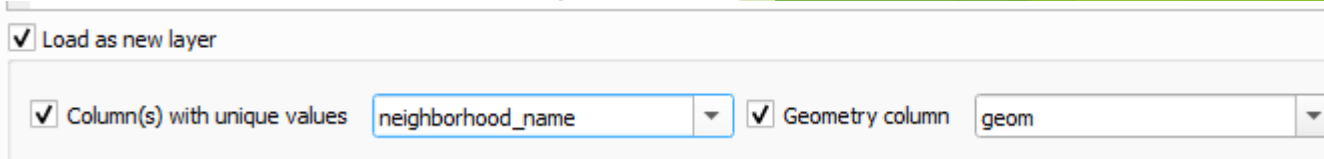
```
SELECT subways.id AS id, subways.name AS subway_name,
neighborhoods.name AS neighborhood_name,
neighborhoods.boroname AS borough, neighborhoods.geom FROM
nyc_neighborhoods AS neighborhoods JOIN nyc_subway_stations AS
subways ON ST_Contains(neighborhoods.geom, subways.geom)
WHERE subways.name = 'Broad St'
```

Functionally speaking

- `ST_Contains(neighborhoods.the_geom, subways.geom)` ???

neighborhoods.geom contain *subways.geom*

```
SELECT subways.id AS id, subways.name AS subway_name, neighborhoods.name AS
neighborhood_name, neighborhoods.boroname AS borough, neighborhoods.geom FROM
nyc_neighborhoods AS neighborhoods JOIN nyc_subway_stations AS subways ON
ST_Contains(neighborhoods.geom, subways.geom) WHERE subways.name = 'Broad St'
```



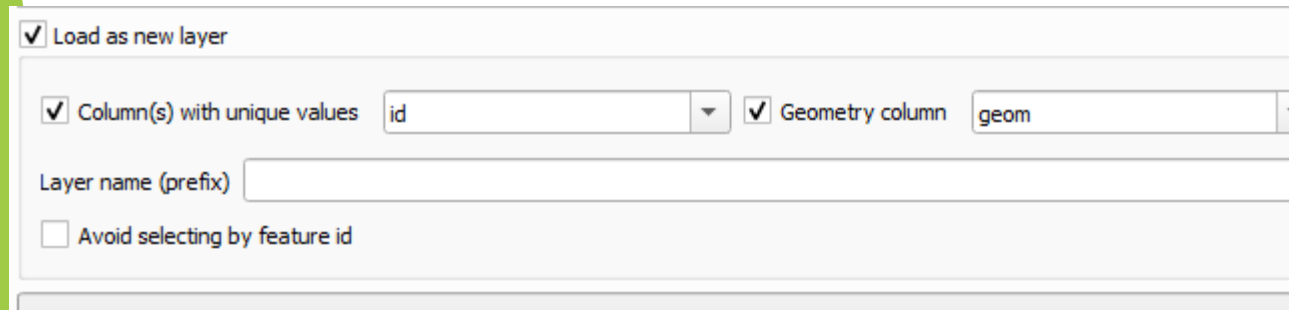
ex2

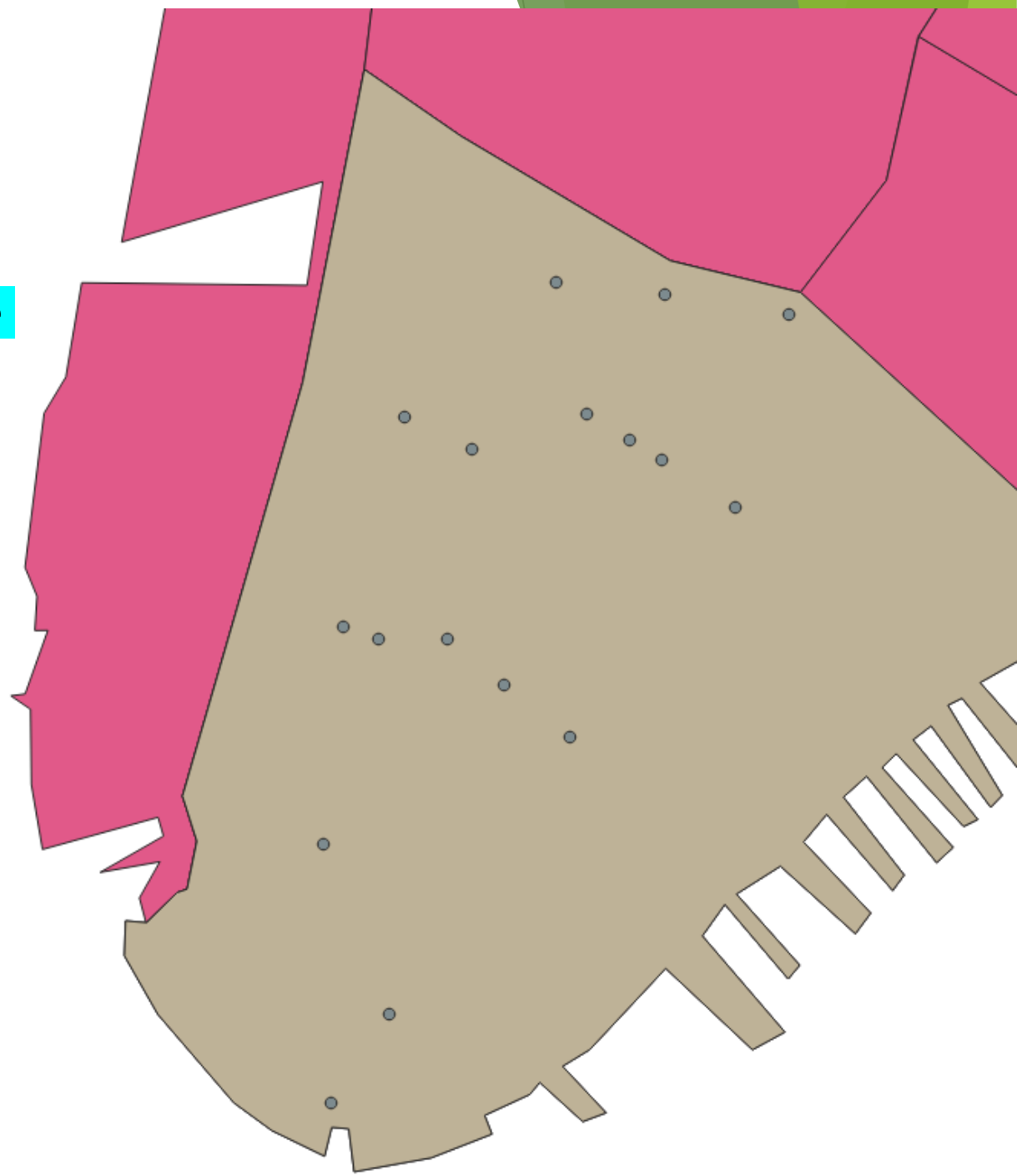
What are subway station in Financial District ?

?

```
SELECT subways.geom, subways.name AS subway_name, subways.id AS id, neighborhoods.name AS
neighborhood_name, neighborhoods.boroname AS borough FROM nyc_neighborhoods AS neighborhoods
JOIN nyc_subway_stations AS subways ON ST_Contains(neighborhoods.geom, subways.geom) WHERE
neighborhoods.name = 'Financial District';
```

```
SELECT subways.geom, subways.name AS
subway_name, subways.id AS id, neighborhoods.name
AS neighborhood_name, neighborhoods.boroname AS
borough FROM nyc_neighborhoods AS neighborhoods
JOIN nyc_subway_stations AS subways ON
ST_Contains(neighborhoods.geom, subways.geom)
WHERE neighborhoods.name = 'Financial District';
```





Join and Summary

- The combination of a JOIN with a GROUP BY provides the kind of analysis that is usually done in a GIS system.

- "What is the population for each borough ?

	gid	boroname	sum	st_union
1	32048	Queens	2229379.0	0106000020266...
2	14730	Brooklyn	2465326.0	0106000020266...
3	36592	Staten Island	443728.0	0106000020266...
4	18374	Manhattan	1537195.0	0106000020266...
5	5255	The Bronx	1332650.0	0106000020266...

Ex2 : Join and summary

```
SELECT max(id) as gid,boroname, sum(popn_total),

ST_Union(geom) as geom FROM nyc_census_blocks

GROUP BY boroname
```

	gid	boroname	sum	st_union
1	32048	Queens	2229379.0	0106000020266...
2	14730	Brooklyn	2465326.0	0106000020266...
3	36592	Staten Island	443728.0	0106000020266...
4	18374	Manhattan	1537195.0	0106000020266...
5	5255	The Bronx	1332650.0	0106000020266...