Introduction to PostGIS

Tutorial ID: IGET_WEBGIS_002

Introduction to PostGIS

Objective
In this tutorial we will learn, how to register a new server, creating a spatial database in PostGIS, importing data to the spatial database, spatial indexing and some important PostGIS special functions.

Software: OpenGeo Suite 3.0

Level: Beginner

Time required: 3 Hour

Software: OpenGeo Suite 3.0

Level: Beginner

Prerequisites and Geospatial Skills

- Basic computer skills
- IGET_WEBGIS_001 should be completed
- Basic knowledge of SQL is excepted

Readings

1. Introduction to the OpenGeo Suite, Chapter 2: PostGIS, pp. 21 – 38.

   http://presentations.opengeo.org/2012_FOSSGIS/suiteintro.pdf

2. Chapter 13. PostGIS Special Functions Index,
   http://suite.opengeo.org/docs/postgis/postgis/html/PostGIS_Special_Functions_Index.html

Tutorial Data: The tutorial data of this exercise may be downloaded from this link:

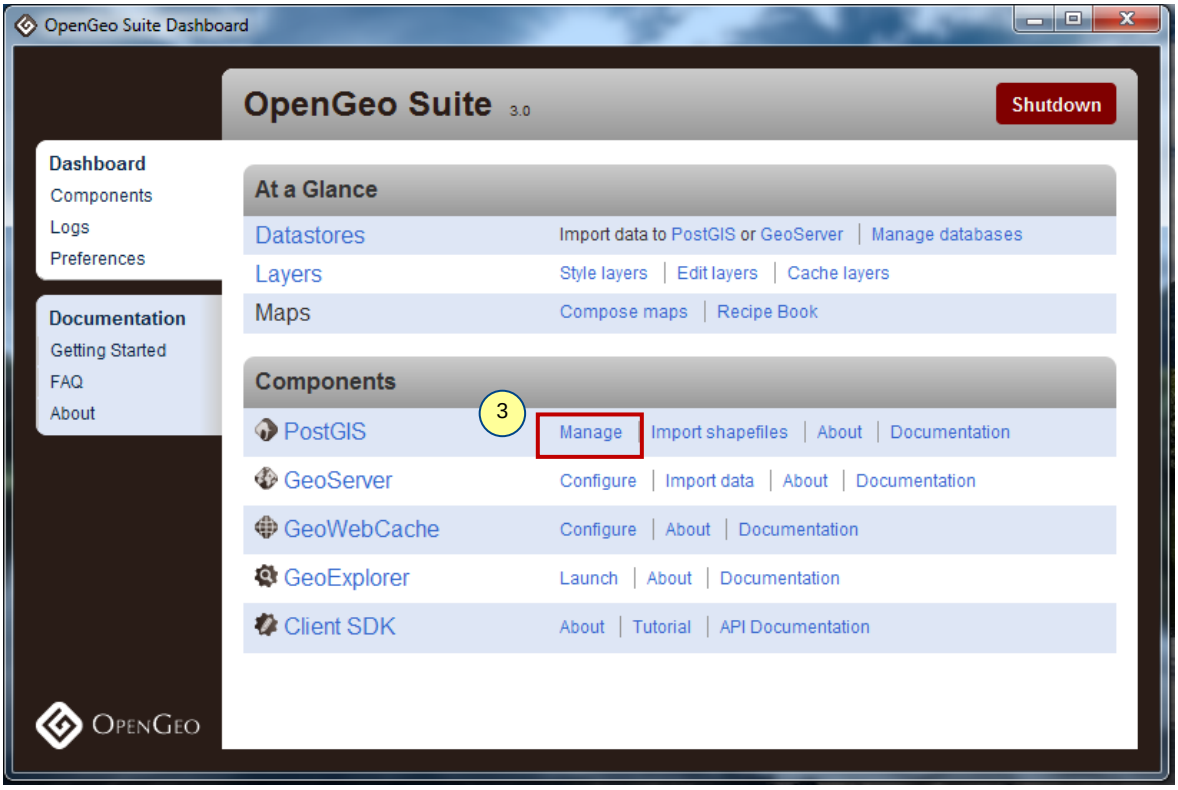
Introduction

PostGIS is an open source spatial database extension that turns PostgreSQL database system into a spatial database. It adds support for geographic objects allowing location queries, analytical functions for raster and vector data, raster map algebra, spatial re-projection, network topology, Geodetic measurements and much more.

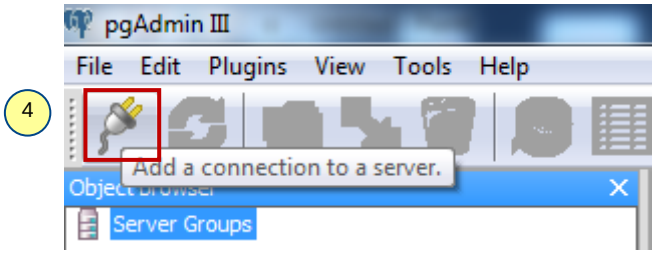
In this tutorial we will learn how to register a server and creation of a spatial database, and importing the shapefiles into it, for this purpose we are using the shapefiles digitized during the tutorial *IGET_GIS_005: Digitization of Toposheet using Quantum GIS*. We will also learn about the Spatial Indexing and PostGIS Special functions in this tutorial.

Registering new server

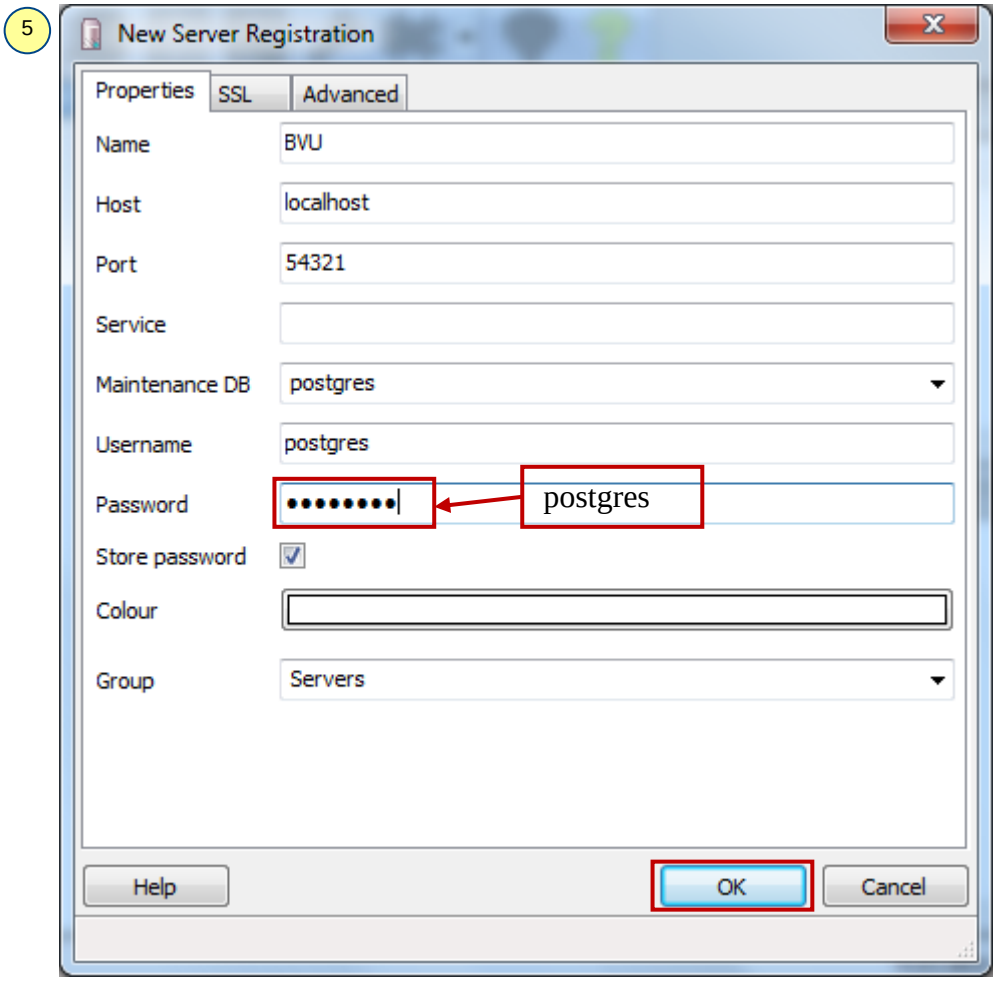
1. Start OpenGeo Suite Dashboard via, *Start → All Programs → OpenGeo Suite 3.0 → OpenGeo Suite Dashboard*.
2. In OpenGeo Suite Dashboard click on Green '***Start***' button to start the server.
3. Start the '***pgAdmin III***' interface by Clicking on '***Manage***' link in '***PostGIS***' section under '***Components***'.



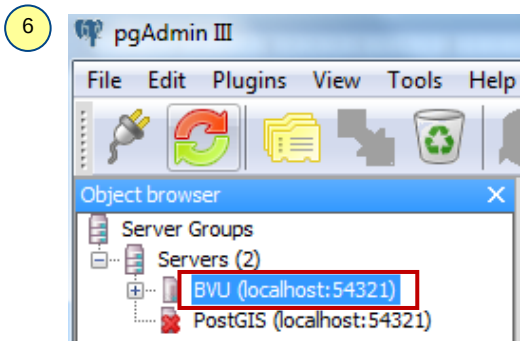
4. Now you will present with '***pgAdmin III***' popup window as shown below. To add new server, click on '*File → Add Server*' or click on  '***Add a connection to a server***' located on the top left corner.



5. ***'New Server Registration'*** window will popup, fill the information as shown in the below figure to register '***BVU***' as new server, click on '***OK***' to finish.



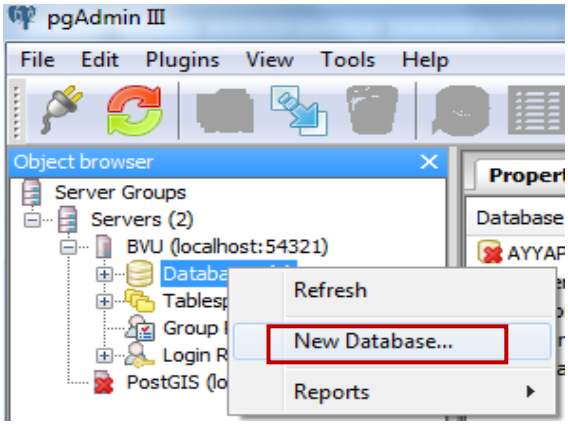
6. Once you clicked on '***OK***', you might be presented with a '***Guru Hint – Saving passwords***' window, click on '***OK***' in it. After few seconds, you will notice '**BVU**' server added to the '*Server Groups*' tree located under '*Object browser window*'.



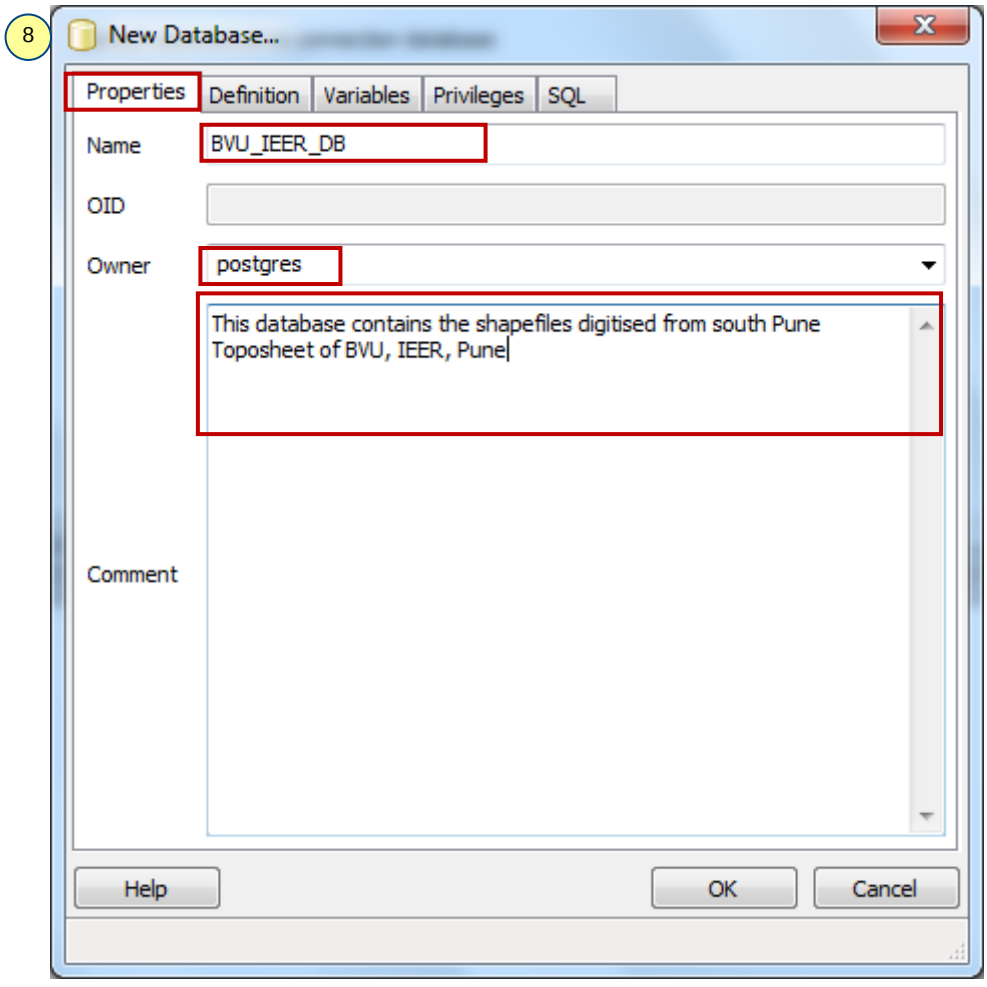
To Create Spatial Database in PostGIS

Spatial database system is a special kind of database system that offers spatial data types in its data model, query language and supports spatial data types in its implementation, providing at least spatial indexing and spatial join methods. Spatial database systems offer the underlying database technology for geographic information systems and other applications (Ralf Hartmut Güting, 1994).

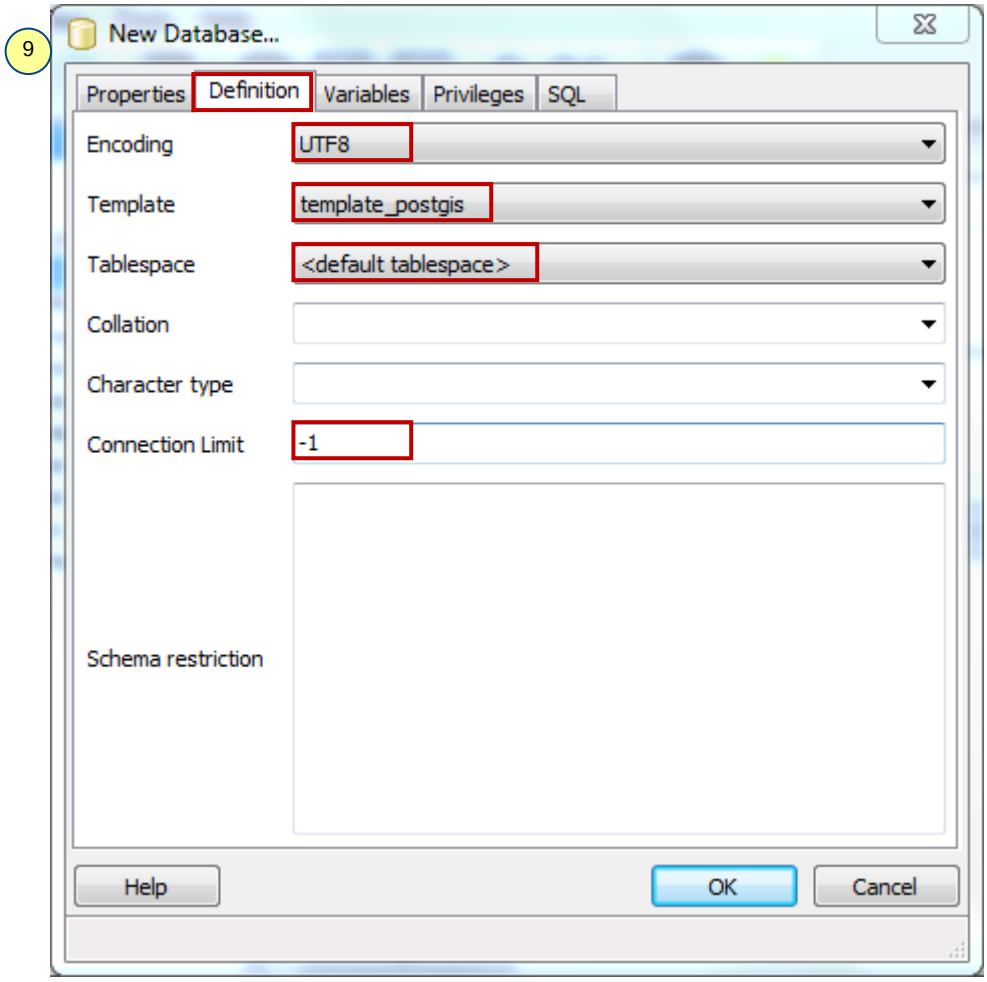
7. Select the ***BVU*** server → Click on the '+' symbol left side to the ***BVU server*** → right click on the '***Databases***' → Click on '***New Database***' to create a new database.



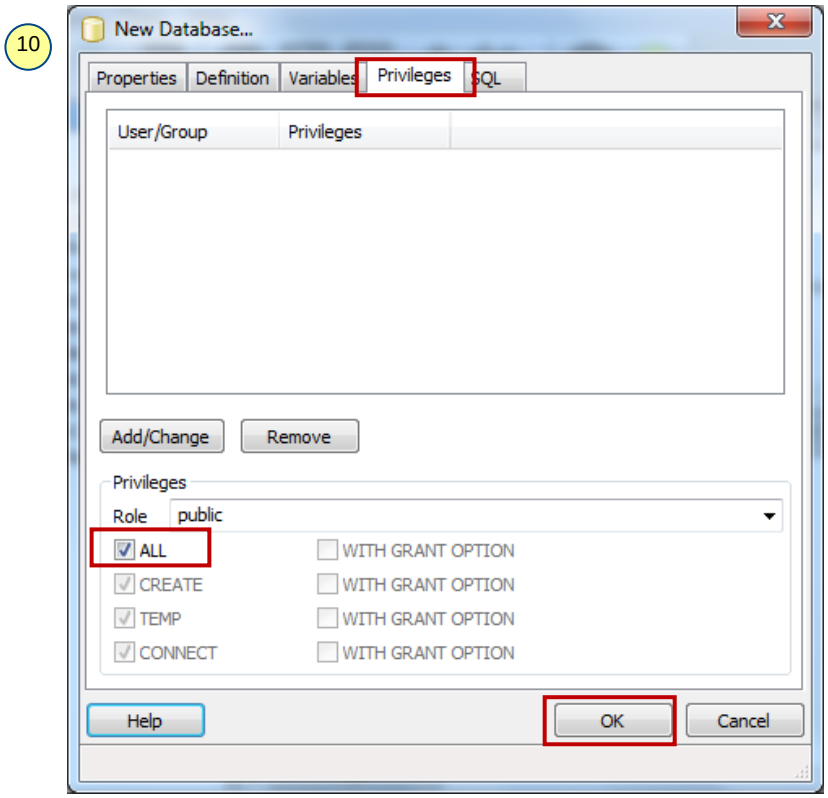
8. Now you will presented with '***New Database***' popup window, Fill the following information under '***Properties***' tab as shown in the below figure to create a database named '***BVU_IEER_DB***'.



9. Now click on '***Definition***' tab and select the following options as shown in below figure from the down drop list.

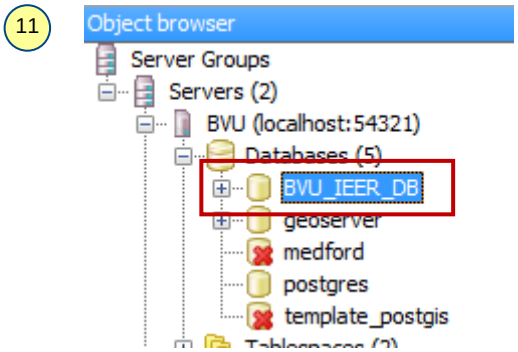


10. Click on the '***Privilege***' tab → '**Check**' the Check box of '***ALL***' → Now Click On '***OK***' in *New Database* Window



11. After few seconds you will notice '***BVU_IEER_DB***' database has been added to the '***Databases***' under

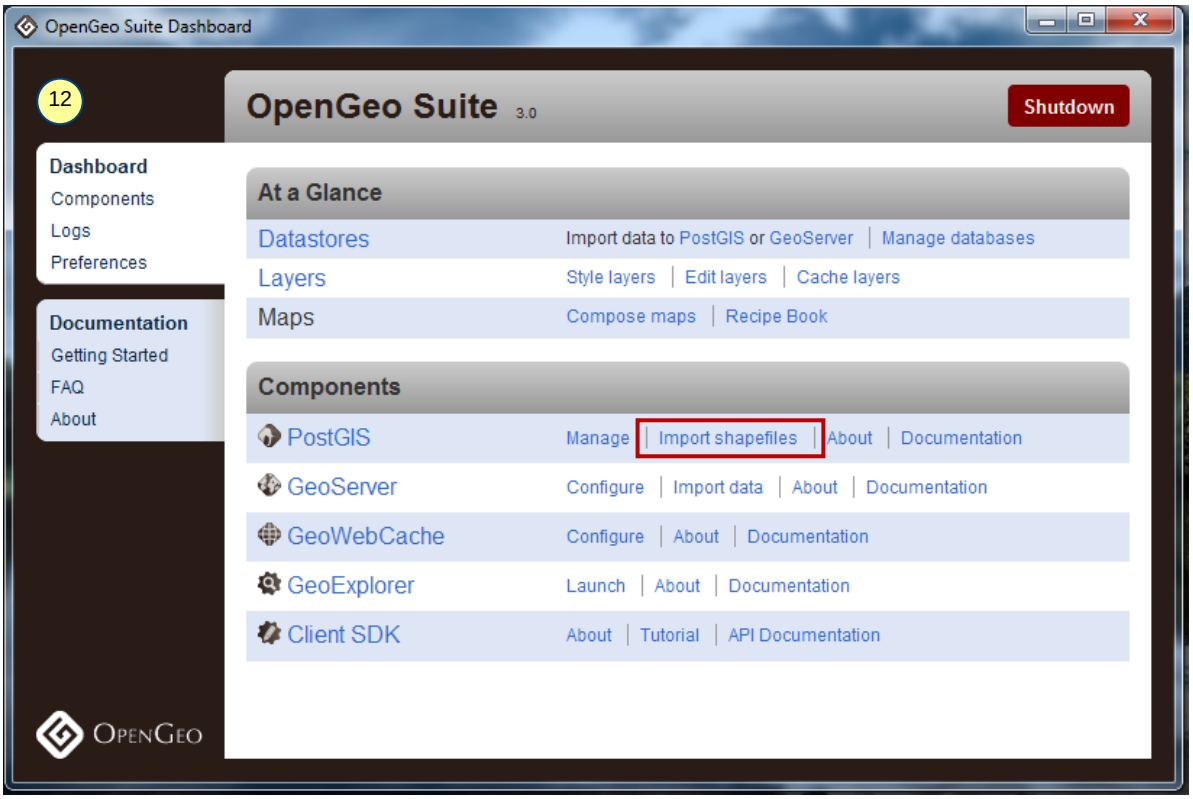
'***BVU***' server.



Importing data to PostGIS

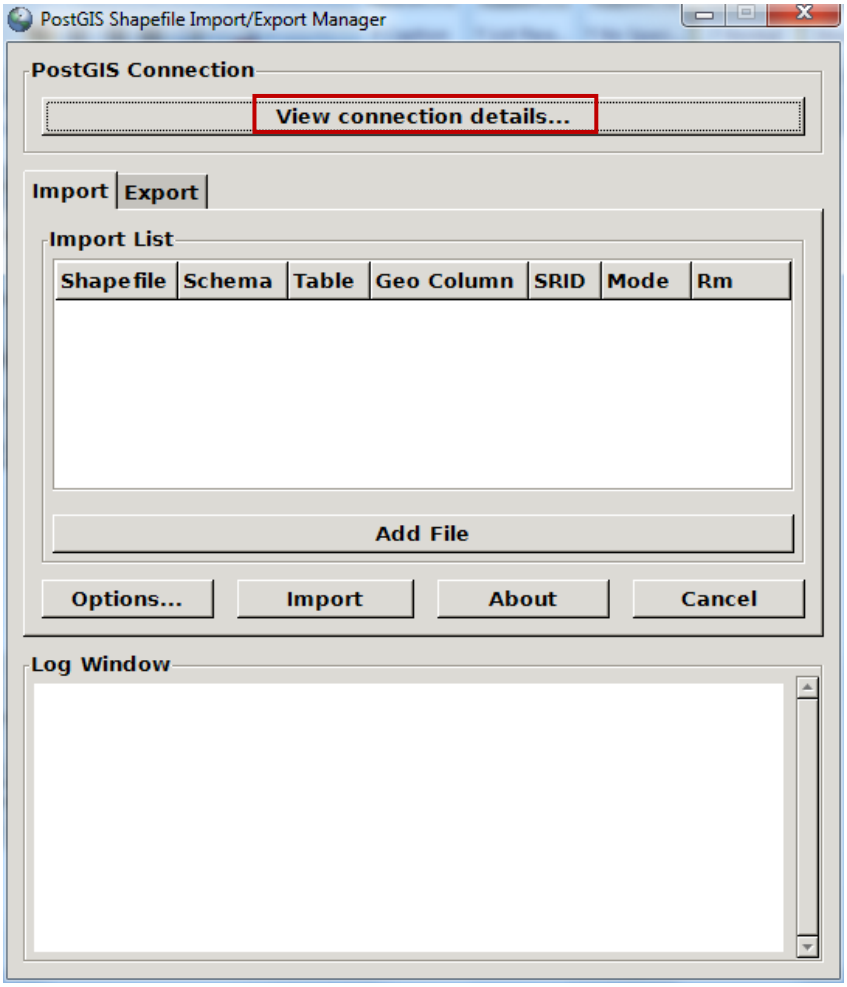
Now we are ready to import the data files into our newly created '***BVU_IEER_DB***' database.

12. Now go to the '*OpenGeo Suite Dashboard*' → Click on '***Import Shapefiles***' under *PostGIS*.



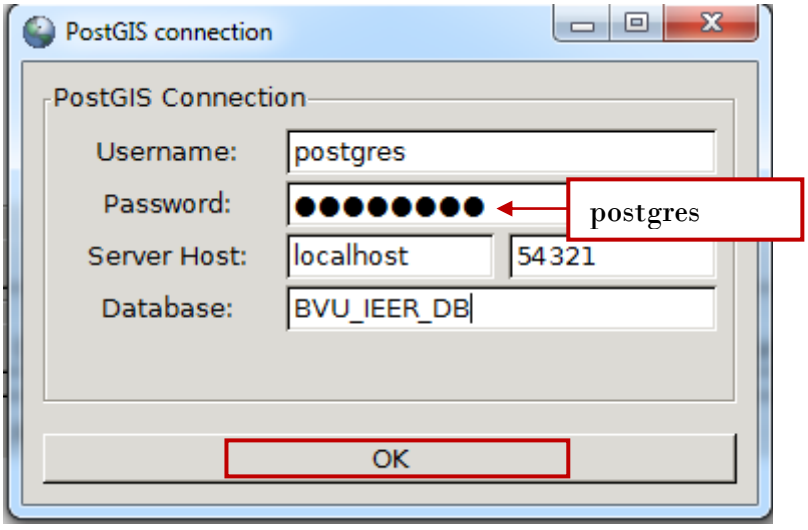
13. Now you will notice '***PostGIS Shapefile Import/Export Manager***' popup window on your screen → Click on '***View connection details…***' under '***PostGIS connection***' section.

13



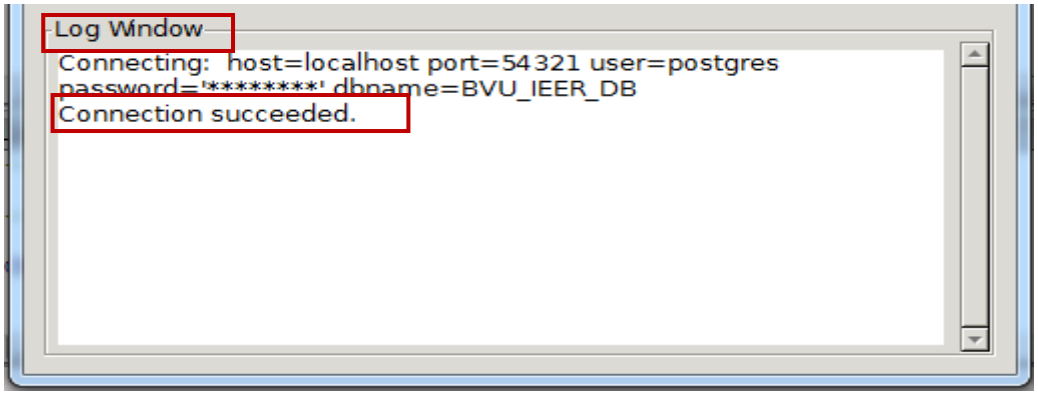
14. Now 'PostGIS Connection' window will popup, enter *username* and *password* as **postgres**, *Server Host* as '**localhost**' & '54321', Database as '***BVU_IEER_DB***' as show in below figure and click on '***OK***'.

14



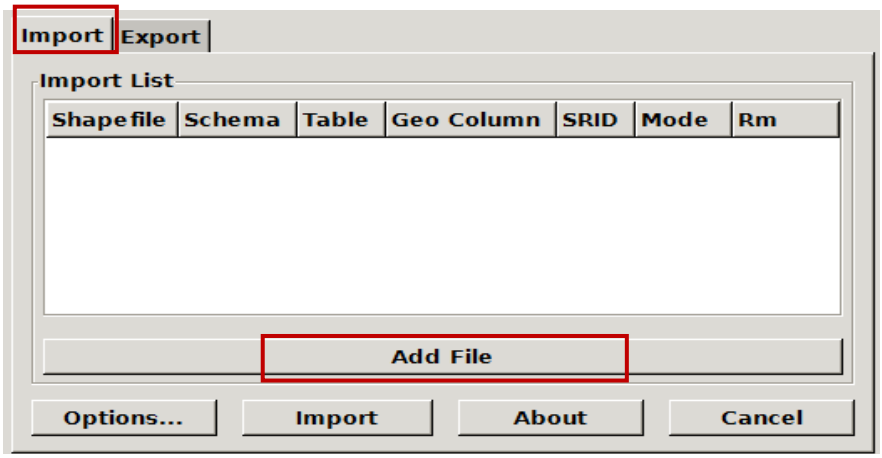
15. If you entered are correct credentials, you will see '***connection succeeded***' message in the '***Log Window***' as shown below. If not, you have to ensure the credentials entered are same as in *Step 5: 'Properties of New server registration'*.

15

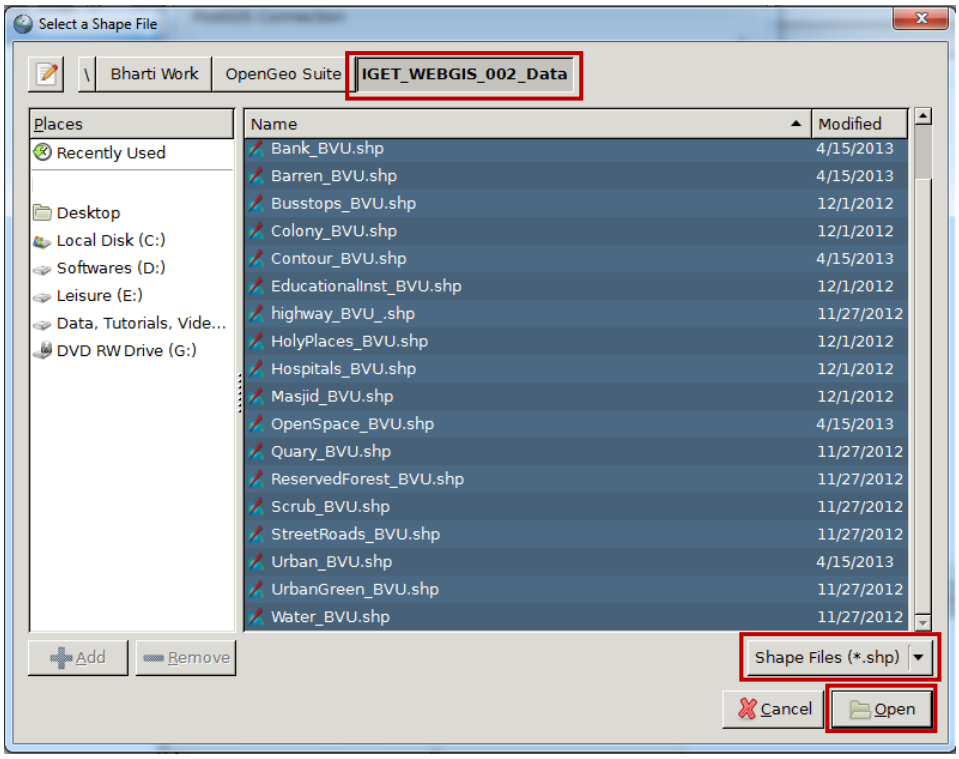


16. Now our database is ready to import the shapefiles, to do this click on '***Add File***' button under '***Import***' tab in '*PostGIS Shapefile Import/Export Manager*' window.

16

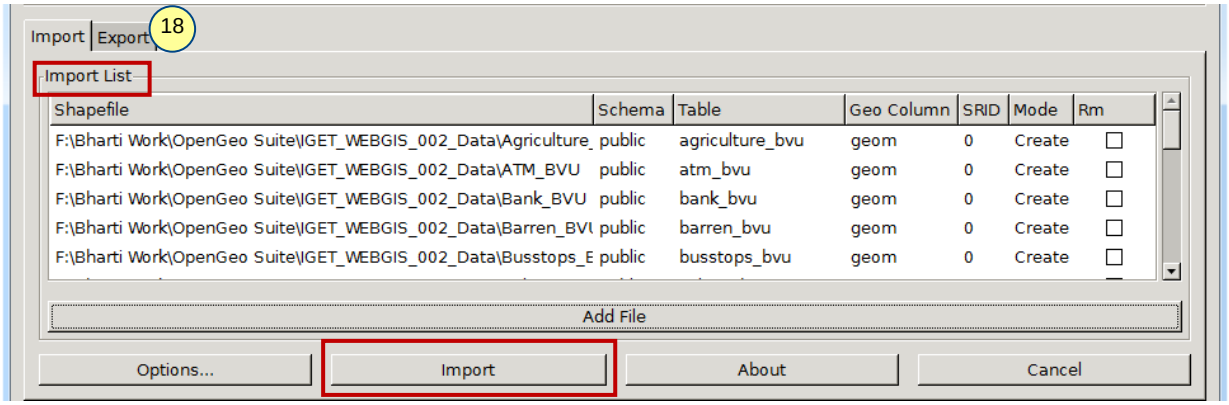


17. Browse the location where you have saved the tutorial data i.e., '***IGET_WEBGIS_002_Data***' folder in browser window and select all the shape files in the folder and click '***Open***'.

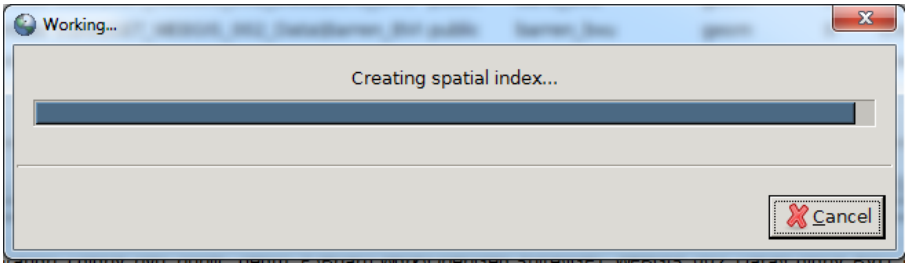


18. Now you can see the list of selected shapefiles under ***Import list***, click on '***Import'*** button to start the

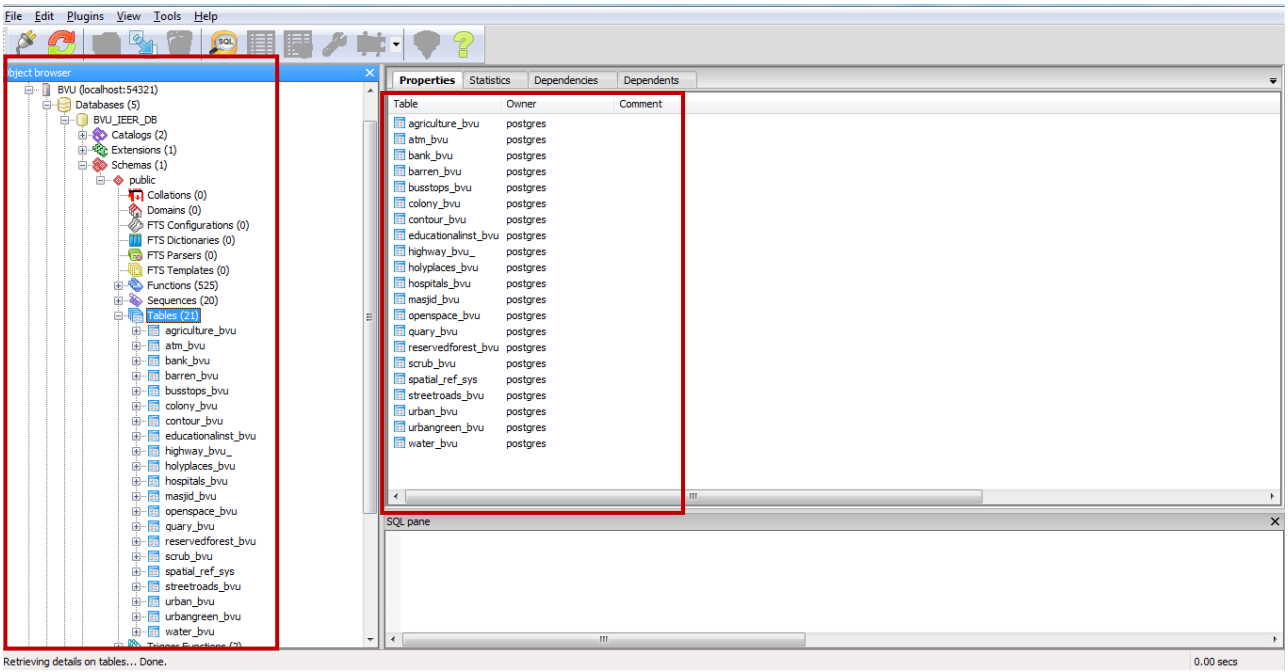
import process.



19. A '***working status***' window flashes on your screen and after some time you can see '***Shapefile import completed'*** message in the '*Log window*'. Close this window after completion of import of all shape files.



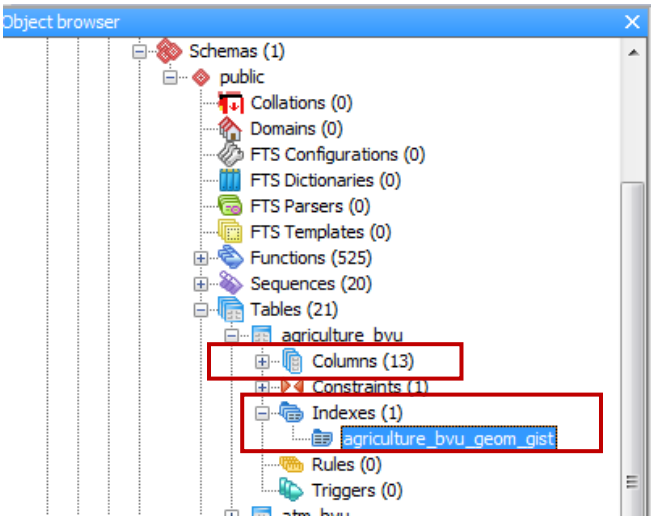
20. Now, go back to the '**pgAdmin III**' Window → Expand the '***Databases***' under '***BVU***' server in '***Object browser'*** → Expand the '***BVU_IEER_DB'*** → expand '***Schemas***' → then expand '***Public***' → expand '***Tables***' to see the list of imported files.



Spatial indexing

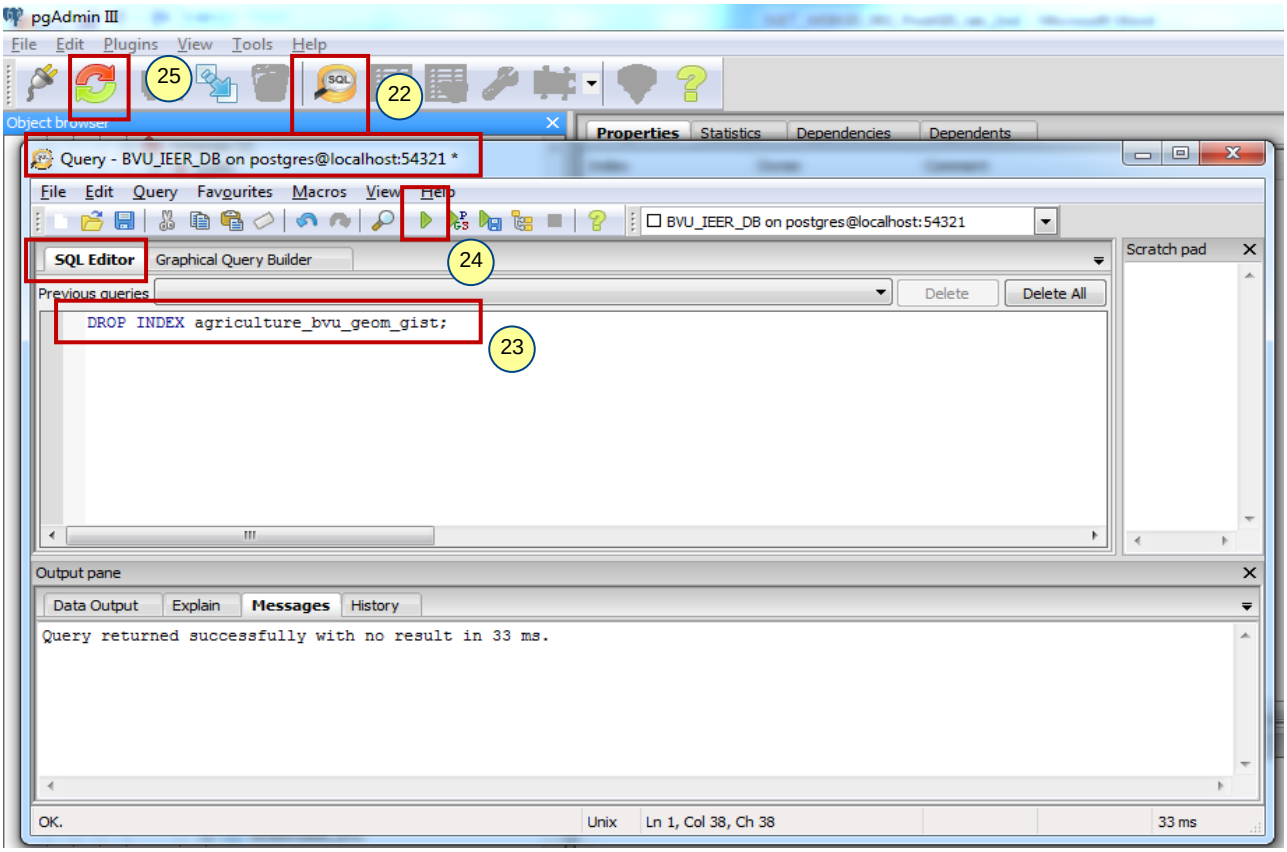
The main purpose of spatial indexing is to support spatial selection, that is, to retrieve from a large set of spatial objects (Ralf Hartmut Güting, 1994) in less time more efficiently.

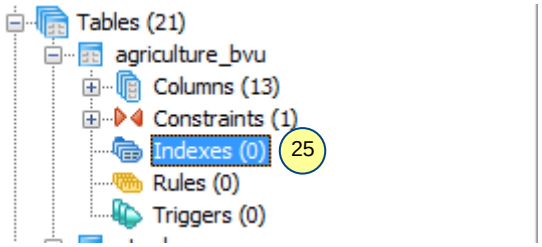
By default '*Shapefile importer*' tool creates the spatial index after importing the data. See the following figure, however, now we will learn manually how to drop and create spatial indexing using SQL command.



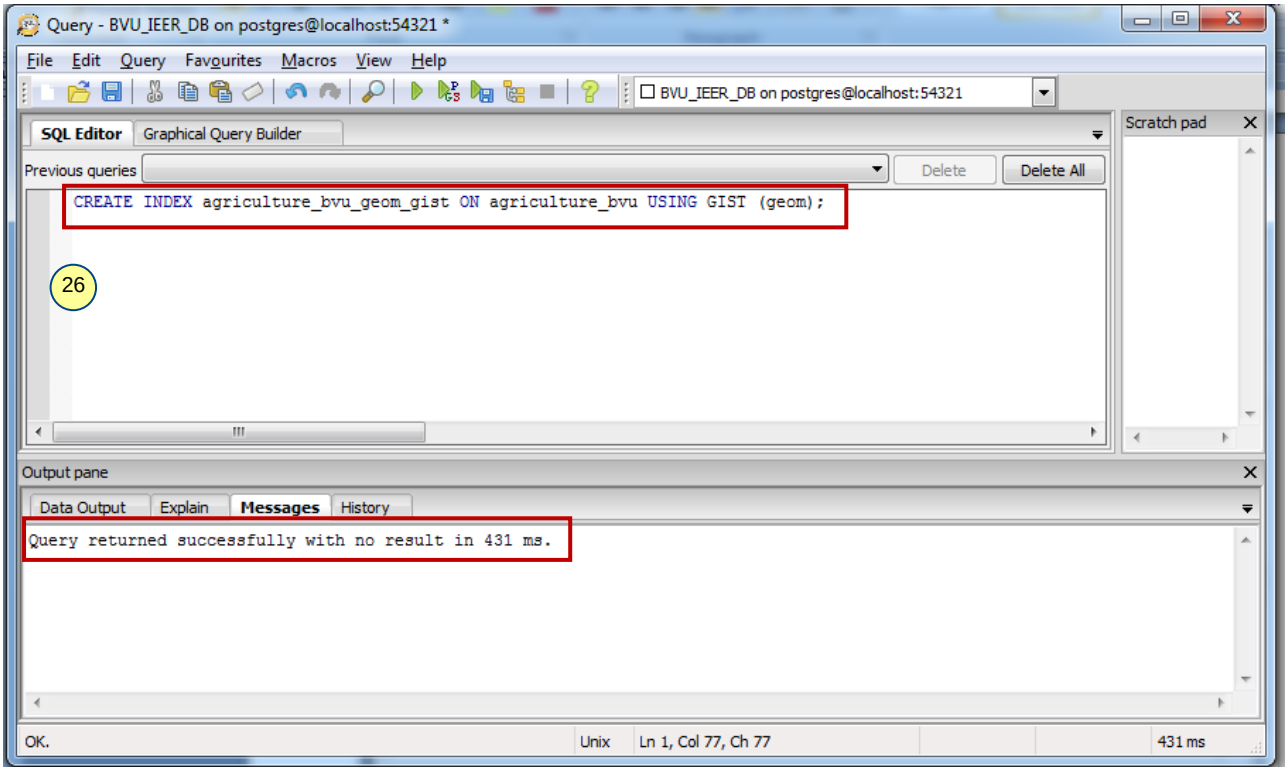
Dropping and Creating spatial index of a table

21. Spatial indexing speeds up the query by applying R-tree algorithm to the tables for more information see http://revenant.ca/www/postgis/workshop/indexing.html
22. Click on the '***SQL queries***' tool to open '***Query***' Window.
23. Now we will drop the spatial indexing of '**agriculture_bvu**' table by using **DROP INDEX** statement, type '***DROP INDEX agriculture_bvu_geom_gist;***' in the text box of *SQL Editor*.
24. Click on to execute the command (to run/ fire the query).
25. Select the table '*agriculture_bvu*' under Tables in '*Object browser*', click on '***Refresh Selected Object***' button from Toolbar in '*pgAdmin III*'. Now you can notice the index of '*agriculture_bvu*' table has been removed.





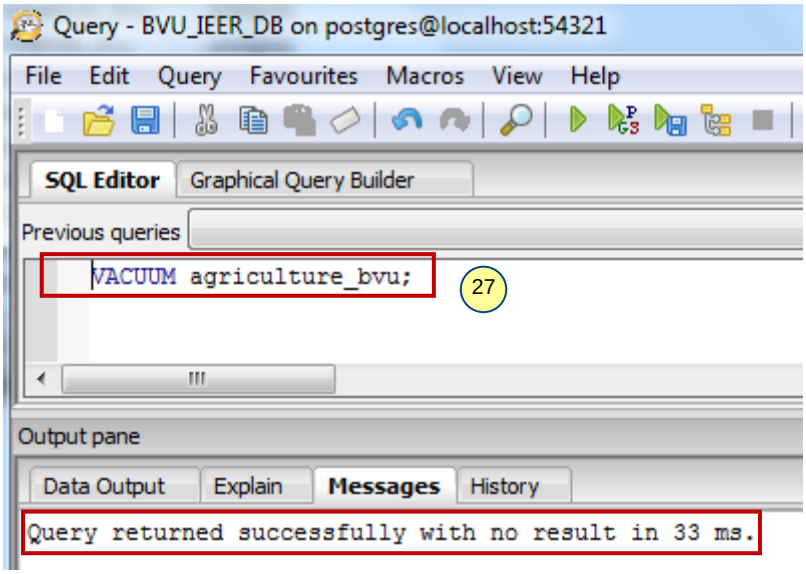
26. Again to create the spatial indexing of '**agriculture_bvu**' table, use '**CREATE INDEX**' statement, type '**CREATE INDEX agriculture_bvu_geom_gist ON agriculture_bvu USING GIST (geom);**' in *SQL Editor* → Click on  to execute the command.



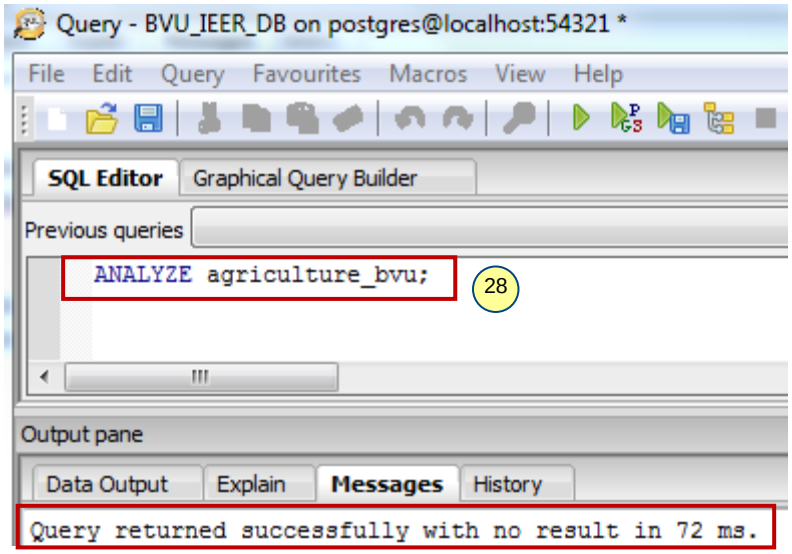
Vacuum and Analyze

Vacuuming the database is a necessary action to perform after creation of Indexes, updates, inserts and deletes. It allows *PostgreSQL* to recover the unused space in the database to work more efficiently. PostgreSQL offers an '*autovacuum*' option, it automatically takes care of recovering the space and update statistics on a sensible time intervals. To recover spaces we have to use '*VACUUM*' command, and we have to use '*ANALYZE*' command to update the statistics of a table after a large number of updates and deletions. It is highly recommended to run '*VACUUM*' and '*ANALYZE*' command manually after creating indexes and data uploads to table/ entire database.

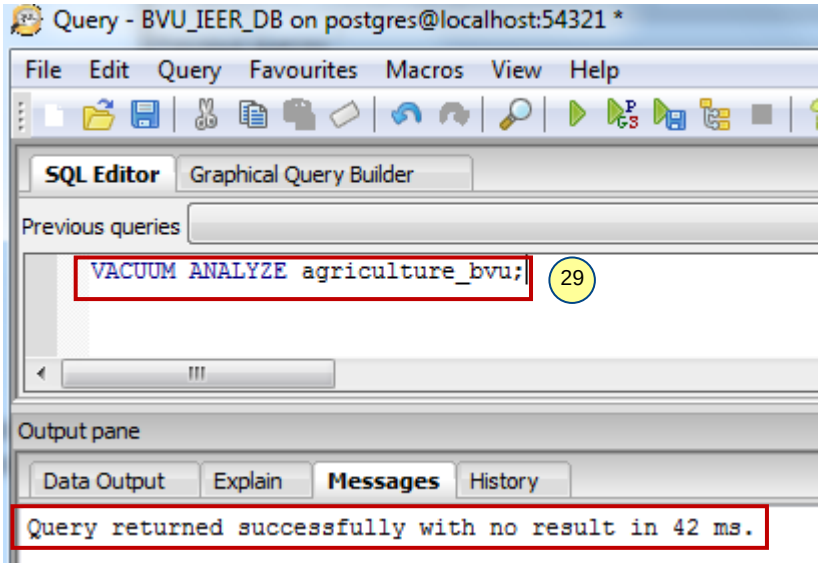
27. Now run '**VACUUM agriculture_bvu;**' command in *SQL Editor* to perform vacuuming on 'agriculture_bvu' table after creation of index.



28. Similarly, to update the statistics of 'agriculture_bvu' table, run '**ANALYZE agriculture_bvu;**' command in *SQL Editor* Window.



29. We can also perform both Vacuum and Analyze commands at a time on table/database by running '**VACUUM ANALYZE agriculture_bvu;**' command.

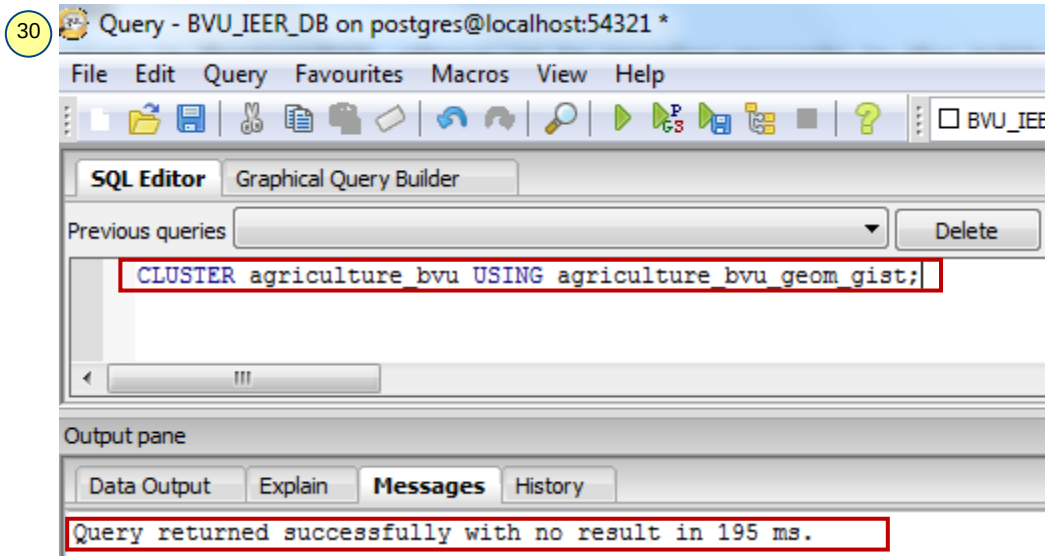


Note: you can also run multiple commands on Query tool.

Clustering data

Clustering feature in *PostgreSQL* allows us to reorder records in the table on disk from unordered state to the order state. This ensures that records with similar attributes have a high likelihood of being found in the same page, reducing the number of pages that must be read into memory for some types of queries.

30. To cluster the data in 'agriculture_bvu' table, run **'CLUSTER agriculture_bvu USING agriculture_bvu_geom_gist;'** command in *SQL Editor*.



PostGIS Special Functions

PostGIS has a 525 special functions, therefore we will discuss few important functions here, which we use frequently. Please refer '*Chapter 13: PostGIS Special Functions Index*', for full list of functions available in PostGIS.

We will cover following special function in this exercise to get an idea of how we can use special functions.

- ST_IsValid
- *ST_IsValidReason*
- ST_ Centroid
- ST_contains
- ST_Length
- ST_Buffer
- ST_Intersects
- ST_Distance
- ST_AsText
- ST_Intersection

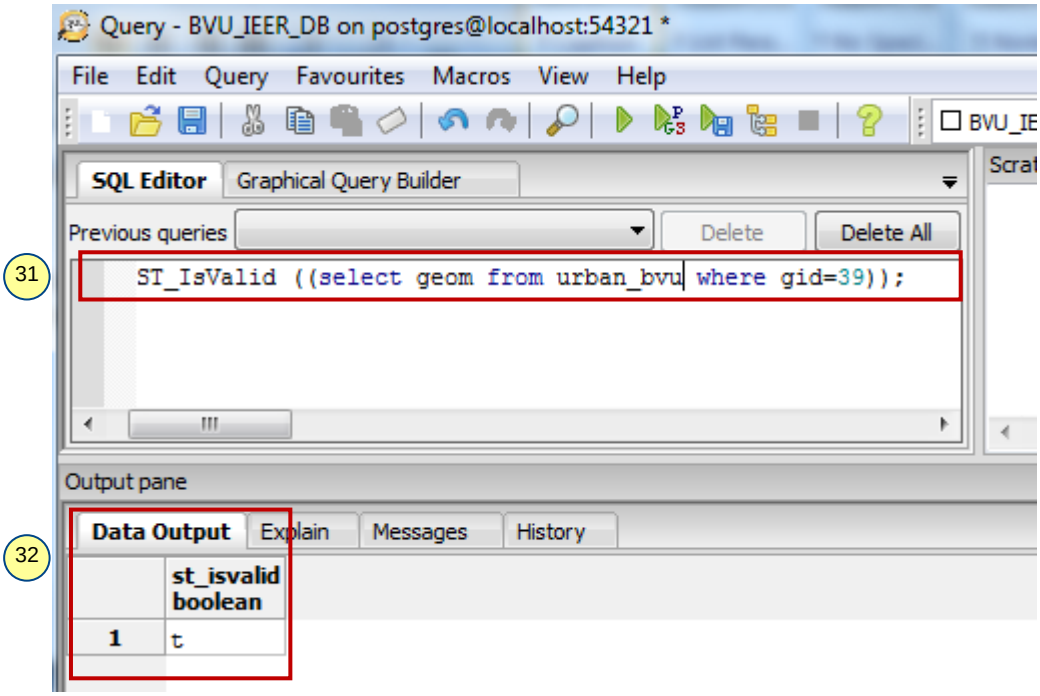
Validation

Before performing any operations on database, it is very important to validate the data in the database to avoid future conflicts. This validation can be by visual inspection or using special function. No one method alone ensures the data is free from errors, this means some errors still exist after visual inspection and vice versa. It is always recommend practicing the both methods will yield good results. In this tutorial we will use the special function called '***ST_IsValid***' to validate the geometry of data present in the 'urban_bvu' table.

31. The following query shows the validity of polygon having '*gid 39*' in 'urban_bvu' table. runthe following command in '*SQL Editor*'

    SELECT ST_IsValid ((select geom from urban_bvu where gid=39));

32. The result will be displayed in '*Output pane*' showing 't', it stands for Boolean *true*, this means the polygon with '*gid 39*' has correct geometry.

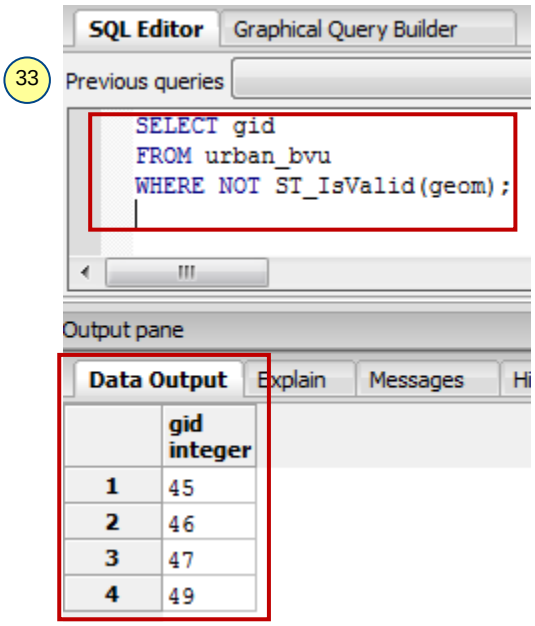


33. It is really a tough job to check the validity for each and every feature individually, if we are dealing with huge data. In order to find all *invalid* polygons in the '*urban_bvu*' table, along with the *gid* of the corresponding polygon run the following query. You can see we used '***NOT***' keyword before '*ST_IsValid*' function to find invalid geometries.

SELECT gid

FROM urban_bvu

WHERE NOT ST_IsValid(geom);



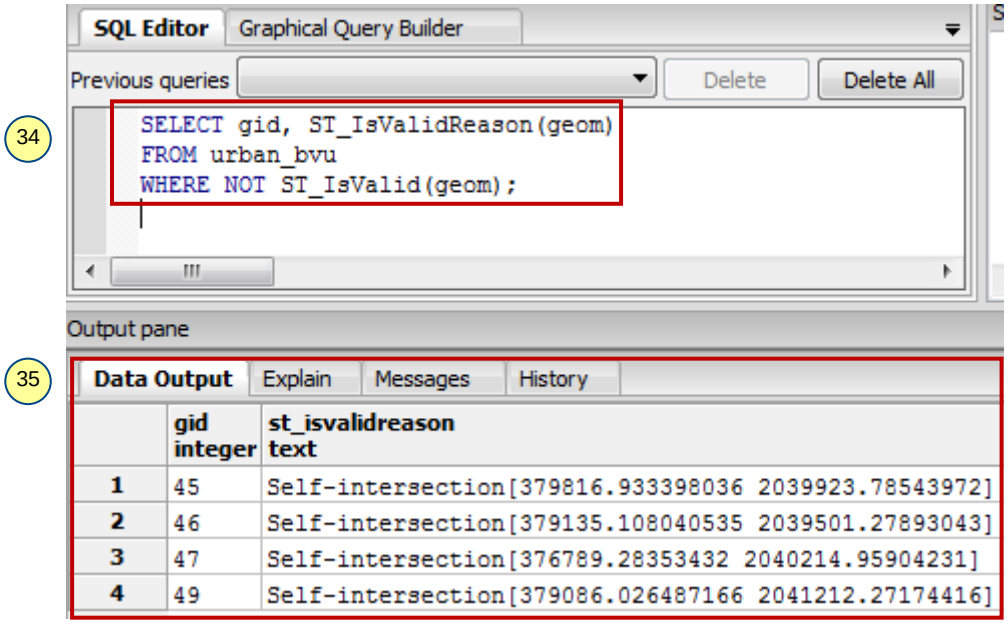
34. We don't know the reason for invalidity of above polygons, in order to find the reason for invalidity use '***ST_IsValidReason***' function. Run the following query in *SQL Editor*.

SELECT gid, ST_IsValidReason(geom)

FROM urban_bvu

WHERE NOT ST_IsValid(geom);

35. The result is shown below; it says self intersection is the reason for invalid of the polygons.



	gid integer	st_isvalidreason text
1	45	Self-intersection[379816.933398036 2039923.78543972]
2	46	Self-intersection[379135.108040535 2039501.27893043]
3	47	Self-intersection[376789.28353432 2040214.95904231]
4	49	Self-intersection[379086.026487166 2041212.27174416]

Rectifying the invalid geometry using ST_Buffer

Buffering is very common proximity analysis tool in Geographic Information Science. You can find a detailed explanation to buffer analysis in the article Buffer Analysis in GIS, written by Nagapramod. PostGIS allows users to perform buffer analysis through '***ST_Buffer***', using this function you can buffer a geometry in 2 ways

- *Positive buffer:* A buffer extending outwards/ away from geometry
- *Negative buffer:* A buffer that extends inside the geometry (this type of buffer don't work on point and line and in case where the negative buffer is greater than the size of polygon)

36. The following query will create a new table named '**BVUHospital_Buffer**', it contains a circular polygon area falling around 750 metres of *Bharati Vidyapeet Hospital* in 'Hospital_BVU' table. This buffer zone contains approximately 1.5 km long Pune- Satara highway, unfortunately if any accident occurs on this road, the victim may be reach this hospital in 2-3 mintues.

CREATE TABLE bvuhospital_buffer AS

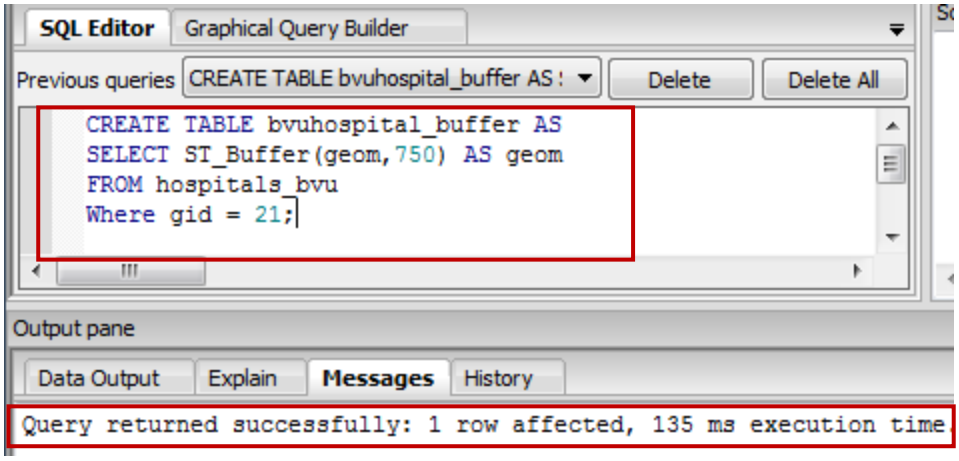
SELECT ST_Buffer(geom,750) AS geom

FROM hospitals_bvu

Where gid = 21;

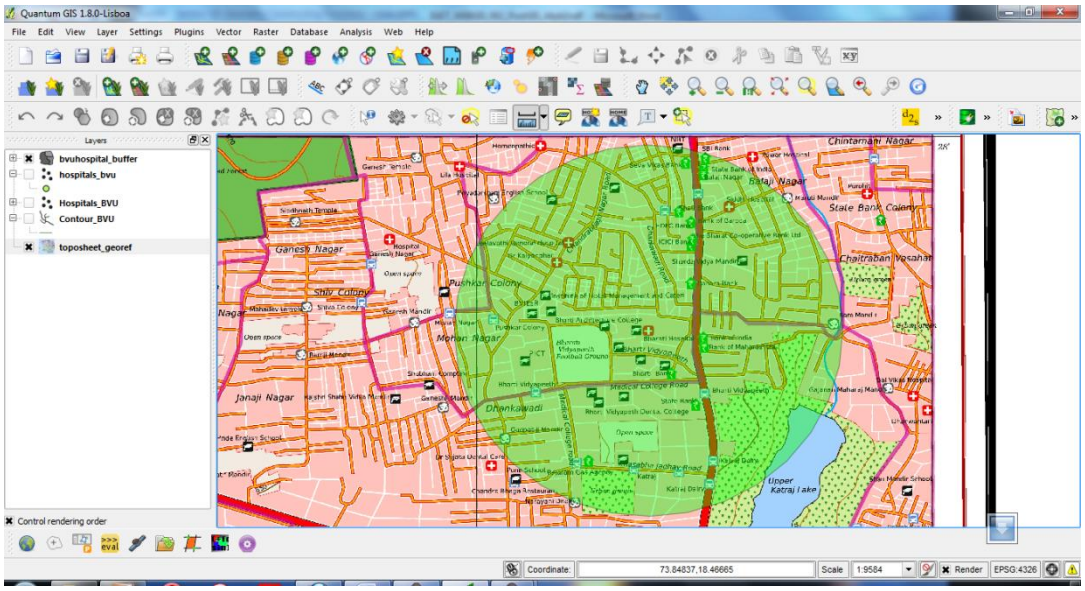
Note: *To see the 'bvuhospital_buffer' table under the list of tables, select Tables in the object browser and click on 'Refresh Selected Object' button from Toolbar in 'pgAdmin III'.*

36



37. The result of above buffer zone can be visualized in QGIS, via Layer → Add PostGIS Layers → establish the connection as per the credentials in *Step 5 & 8* in '*Add PostGis Layer*' window → click on '*Connect*' → select '*bvuhospital_buffer*' table → click on '***Add***' → select 'WGS84 / UTM zone 43N' as CRS → click '*OK*'

37



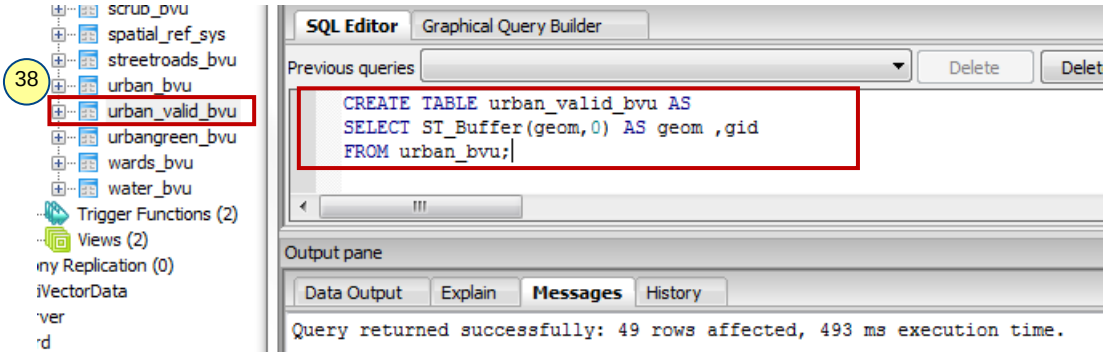
38. We can also use ***ST_Buffer*** function keeping the buffer distance as zero meters to rectify the self intersection error in the geometry. Run the following command to create a valid urban table, i.e., urban_valid_bvu from 'urban_bvu' table.

CREATE TABLE urban_valid_bvu AS

SELECT ST_Buffer(geom,0) AS geom ,gid

FROM urban_bvu;

38



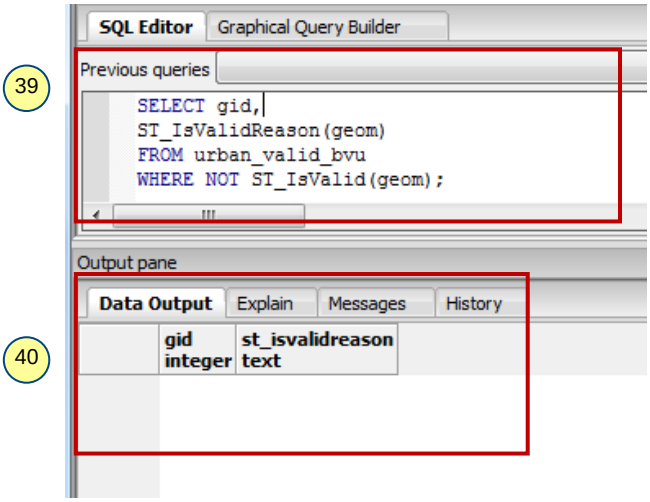
39. By running the following query in the *SQL Editor*, we can see the manipulation of urban polygons with zero buffer distance worked or not?

    SELECT gid,

    ST_IsValidReason(geom)

    FROM urban_valid_bvu

    WHERE NOT ST_IsValid(geom);



40. Yes! From the blank results we can conclude that is there is no invalid geometries exist in the 'urban_valid_bvu' table.

ST_ Centroid: This function will return a point geometry that is placed at the centre of a given geometry.

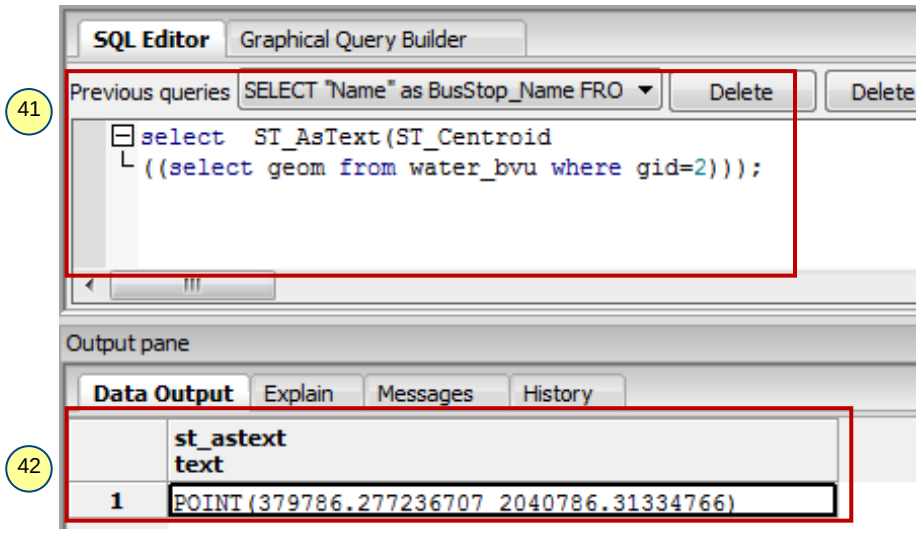
For example, Pune Municipal Corporation (PMC), want to erect a 'Chatrapati Shivaji Monument and a fountain' in the center of 'Upper Katraj lake' to attract people to the park, caould you help them by supply the coordinates of center of lake?

41. Yes, you can use '**ST_Centroid**' function to compute the center of *Upper Katraj Lake*. Run the following query in *SQL Editor*.

    select ST_AsText(ST_Centroid

    ((select geom from water_bvu where gid=2)));

42. In above query we are requesting the center of lake as text by using '***ST_AsText***' function. This will yield result with Easting and Northing values of center of lake in *Data Output* pane.



ST_Contains : It checks for and returns the geometries that are completely inside of another geometry. For example, PMC wants to modernize the bus stops in ward ID number 75. For this purpose they need the name of bus stops exist in ward 75, how you can help PMC?

43. You can '***ST_Contains***' function to answer above query. Run the following query to get the results.

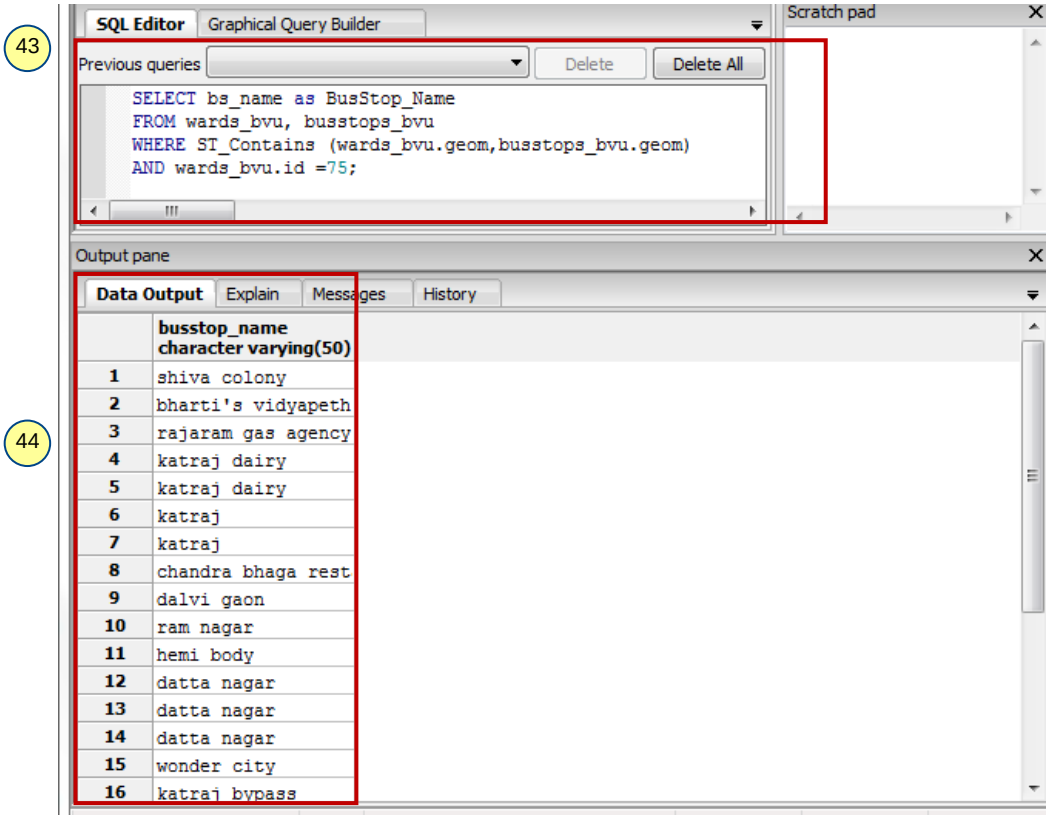
    SELECT bs_name as BusStop_Name

    FROM wards_bvu, busstops_bvu

    WHERE ST_Contains (wards_bvu.geom,busstops_bvu.geom)

    AND wards_bvu.id =75;

44. The above query will result a table of 25 rows in output pane showing the bus stop falling under ward ID:75



	busstop_name character varying(50)
1	shiva colony
2	bharti's vidyapeth
3	rajaram gas agency
4	katraj dairy
5	katraj dairy
6	katraj
7	katraj
8	chandra bhaga rest
9	dalvi gaon
10	ram nagar
11	hemi body
12	datta nagar
13	datta nagar
14	datta nagar
15	wonder city
16	katraj bypass

ST_Intersects: This function yields a Boolean output, it return true, when the geometries overlap, touches or intersects each other and false in case of disjoint.

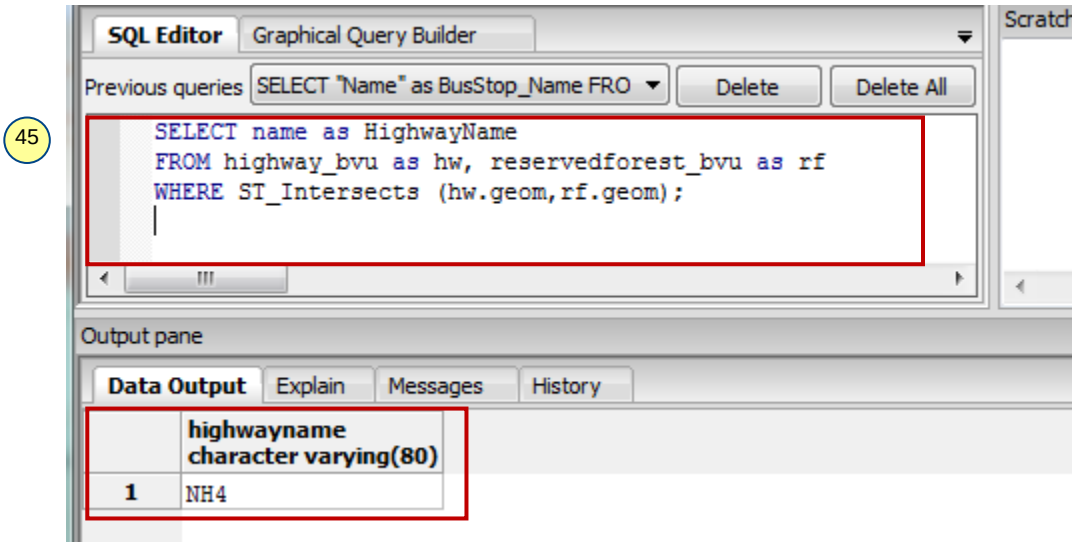
For example: Roads are often fragments the reserved forest areas, this is a barrier for wild life passages. In India road kill of wild life is very common, in order to prevent this situation, we need to take useful measures along the roads passing through the forests. We can use '***ST_Intersects***' function to identify the roads passing through the forests.

45. The following code shows, highway road segment, which are passing through the reserved forest area of south Pune toposheet.

    SELECT name as HighwayName

    FROM highway_bvu as hw, reservedforest_bvu as rf

WHERE ST_Intersects (hw.geom,rf.geom);



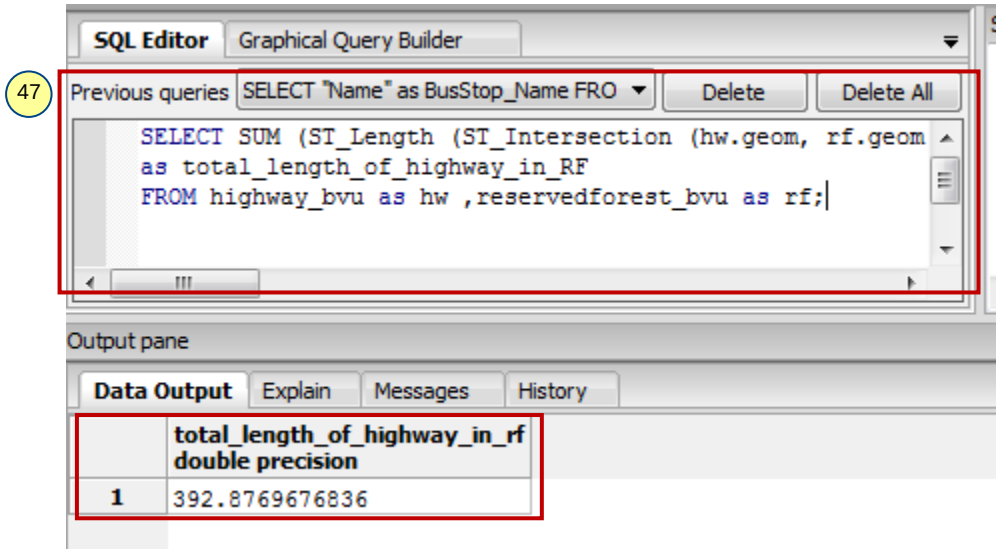
46. The result of query in *step 45*, showed that '*NH4*' highway is passing through the reserved forest area, now we are interested to know the length of the segment of NH4, which is passing through the reserved forest, it helps us to estimate the cost of a project, which is aimed to create a fence along the road and optimal underground wildlife passages. For this purpose we will use '***ST_Length***' function along with '***ST_Intersection***'.

47. Run the following query in *SQL Editor* to get the length of NH4 segment passing through the reserved forest area.

SELECT SUM (ST_Length (ST_Intersection (hw.geom, rf.geom)))

as total_length_of_highway_in_RF

FROM highway_bvu as hw ,reservedforest_bvu as rf;

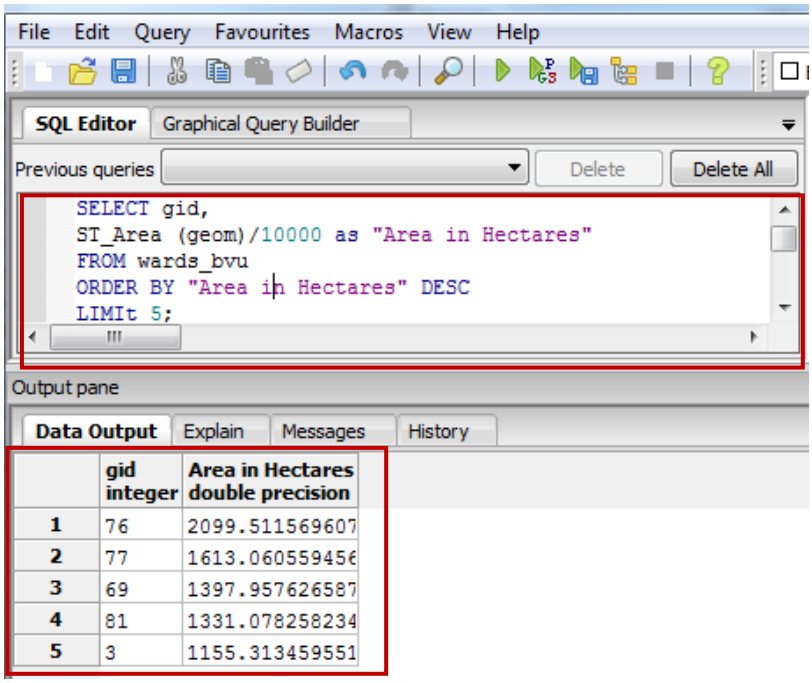


48. Measuring area is a very common task in GIS. '***ST_Area***' function is used to measure area in PostGIS. Following query will yield the areas of five biggest wards of PMC in descending order.

SELECT gid,
ST_Area (geom)/10000 as "Area in Hectares"
FROM wards_bvu
ORDER BY "Area in Hectares" DESC

LIMIt 5;

48



49. Often it is very useful to measure the distance between starting and destination stations. For such tasks we will use '***ST_Distance***'. This spatial function will give cartiesian minimum distance between two geometries. Use the following command to find distance between bus stops having gid 1& 15 in project units, in our case meters.

SELECT ST_Distance
((SELECT geom from busstops_bvu WHERE gid=1),
(SELECT geom from busstops_bvu WHERE gid=15));

49