KATHMANDU UNIVERSITY

SCHOOL OF ENGINEERING

DEPARTMENT OF GEOMATICS ENGINEERING

REPORT OF MINI PROJECT ON THE TOPIC

pgRouting

Submitted By

Arbin Chaudhary

Roll no. 07

Group: GE

Level: UNG/3rd year/1st semester

Submitted To

Er. Ajay Thapa

Department of Geomatics Engineering, Kathmandu University

Abstract:

pgRouting is an extension of PostgreSQL and PostGIS that enables geospatial routing functionality, including shortest path search and driving distance calculations. It turns your database into a spatial database for geographic services, making it a valuable tool for developers working with location-based services and logistics. pgRouting provides various functions for routing and network analysis, such as Dijkstra's algorithm and driving distance calculations. To use these functions, you need to have a table in your database that represents a graph, with columns for id, source, target, and cost. pgRouting can be used in various time-series use cases, such as analyzing traffic data over time or optimizing delivery routes based on historical data. It is particularly useful for analyzing movement patterns and optimizing routes in logistics and transportation applications. It employs various algorithms such as All Pairs Short Path, Bidirectional Dijkstra and joining the geometries.

For this mini project pgRouting, we need to install & download the

- Postgres,
- postgis,
- extension pg routing,
- QGIS,
- Java,
- Osm2po.

After installing the required software, we need to open the downloaded

os2po file >> osm2po.config >> Edit Line 179: wtr.finalMask = car, foot, bike >> Edit Line 330: Remove # in the beginning >> Save the file.

After modifying the os2po.config, we have to download the osmdata in .pbf format.

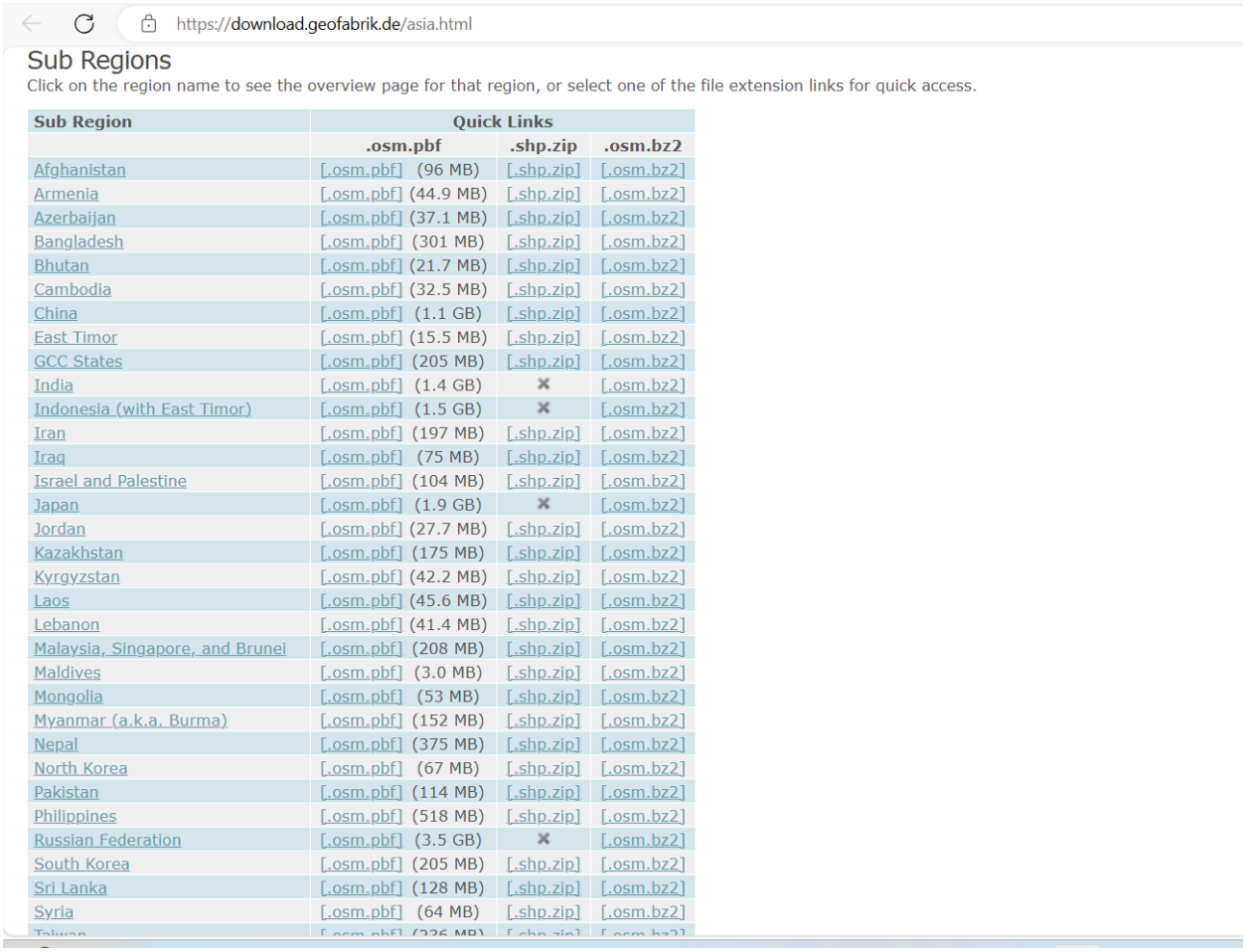


Figure 1:OSM Data

When required osmdata is downloaded, we have proceeded for preprocessing osmdata in osm2po.

For Preprocessing of osmdata, we have to enter the command in command prompt as in below figure:

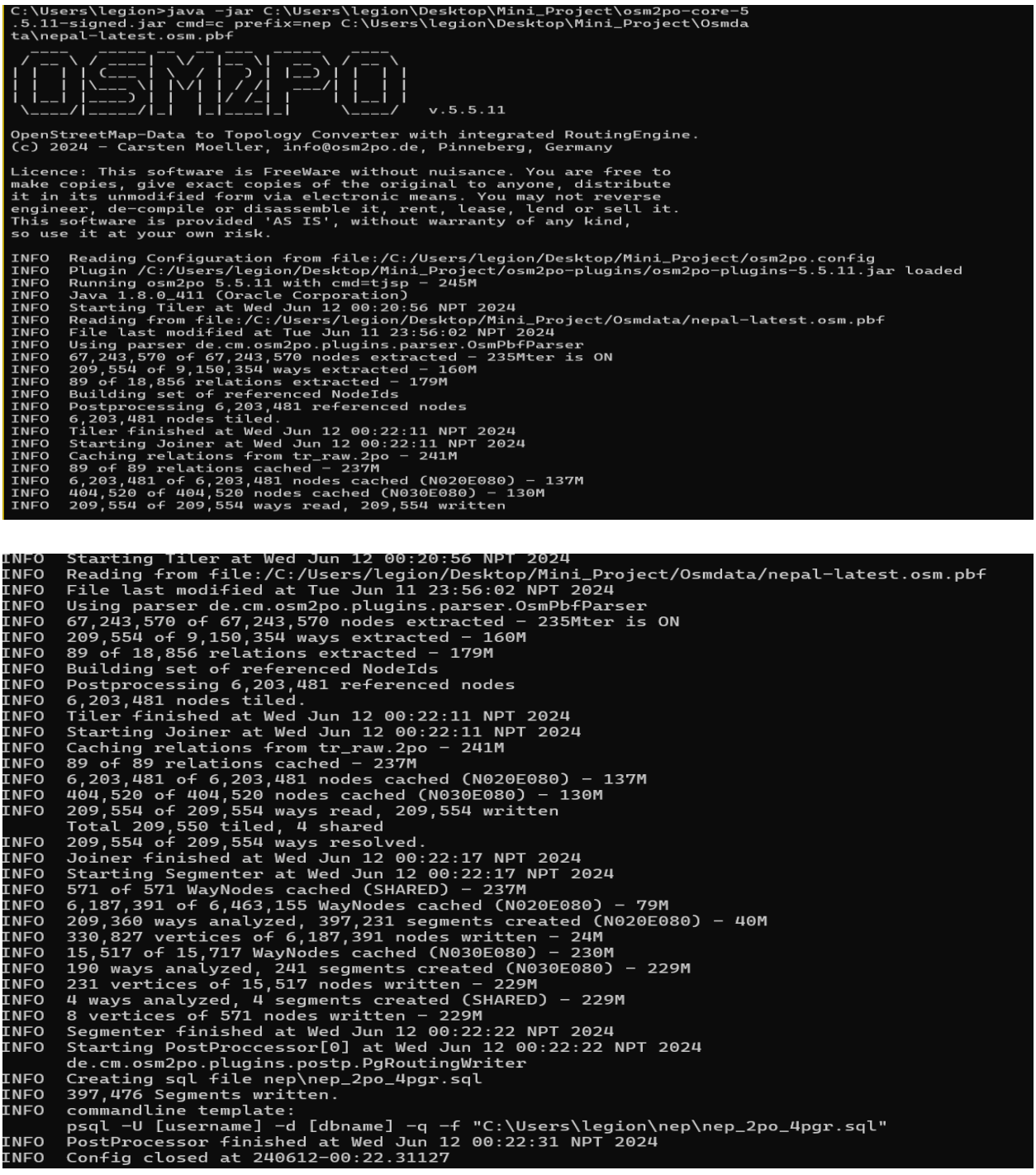


Figure 2: Pre-processing of osmdata in osm2po

After Pre-Processing of data, we obtain the location of converted data.

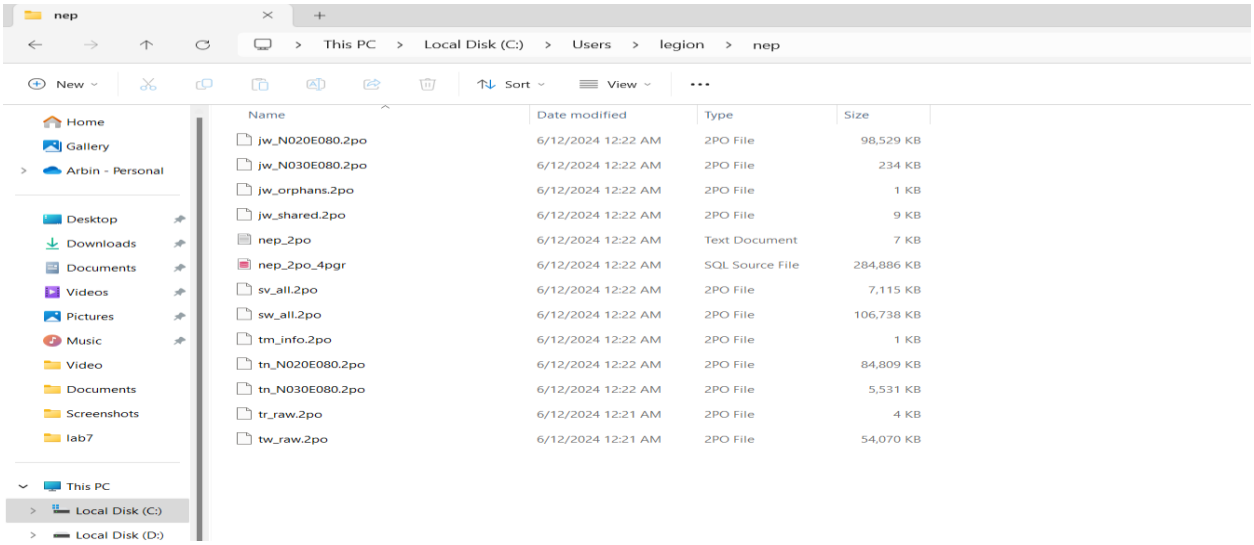


Figure 3:location of converted data

pgAdmin:

Creating the extension postgis and pgRouting:

Query:

Create extension postgis

Explanation:

This command installs the PostGIS extension, which adds support for geographic objects to the PostgreSQL database. PostGIS is an open-source spatial database extender for PostgreSQL, allowing storage and query of information about location and mapping.After running this command, you will be able to use the spatial functions and data types provided by PostGIS in your database.

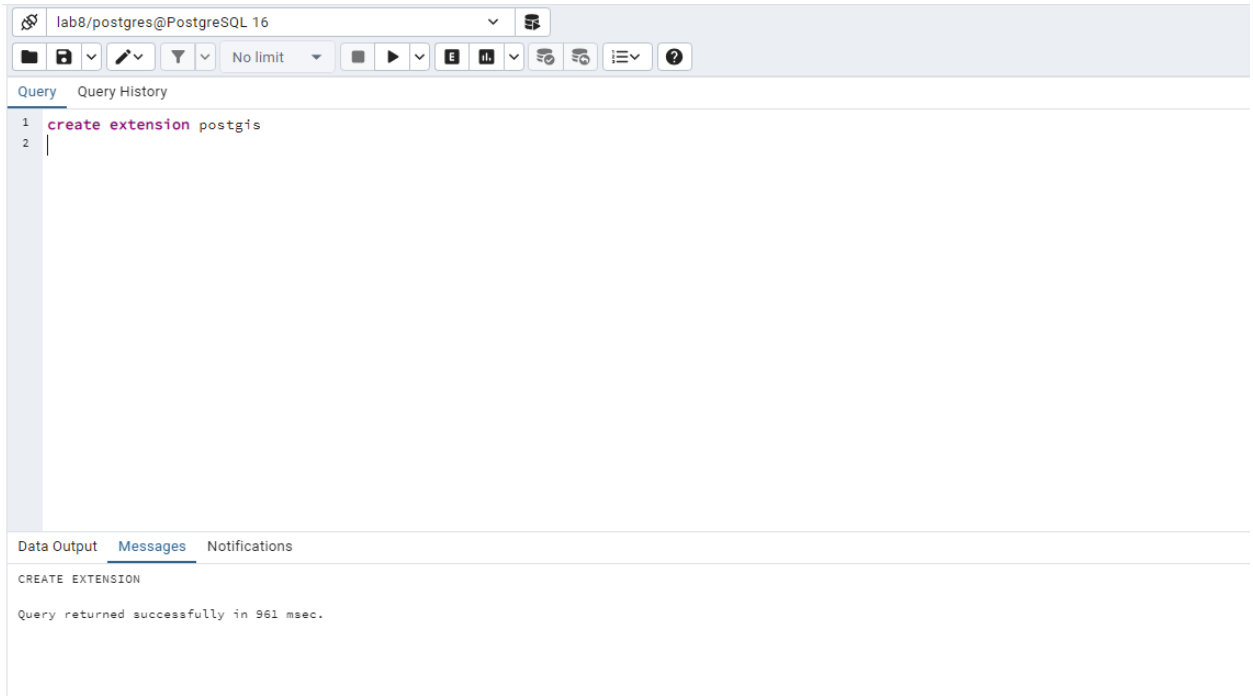


Figure 4:creating the extension postgis

Query:

Create extension pgrouting

Explanation:

This command installs the pgrouting extension, which extends the functionality of PostGIS by providing various functions for solving network problems such as finding the shortest path and the traveling salesperson problem.

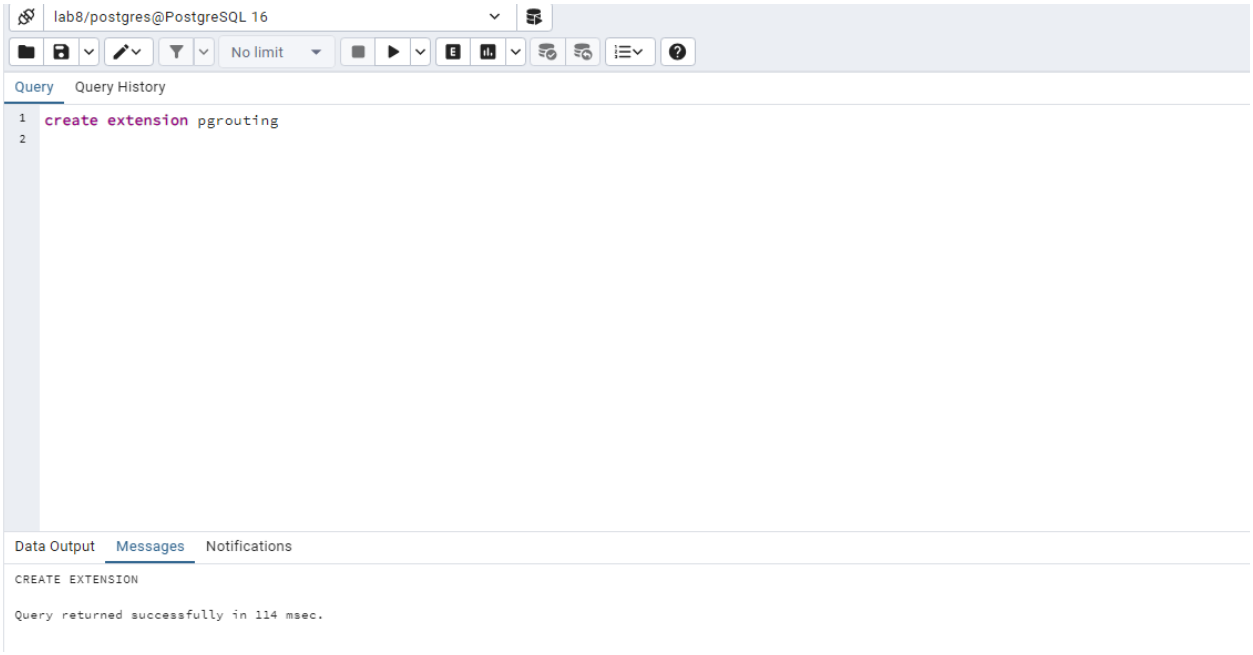


Figure 5: creating of extension

Opening the converted file i.e. "C:\Users\legion\nep\nep_2po_4pgr.sql" using pgAdmin 4 and we execute the query.

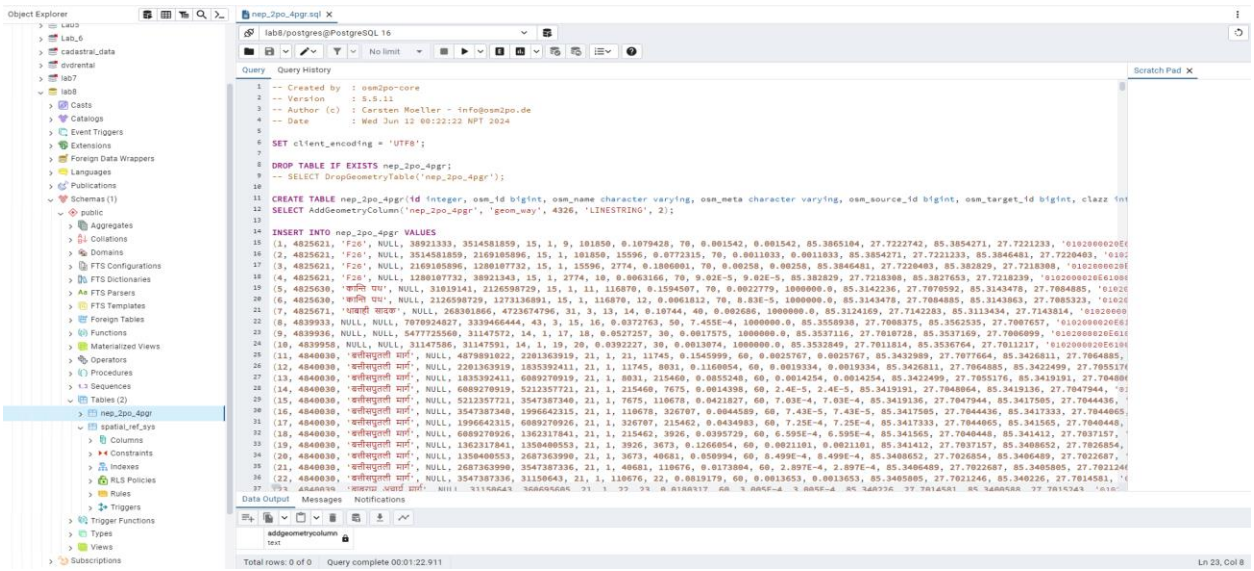
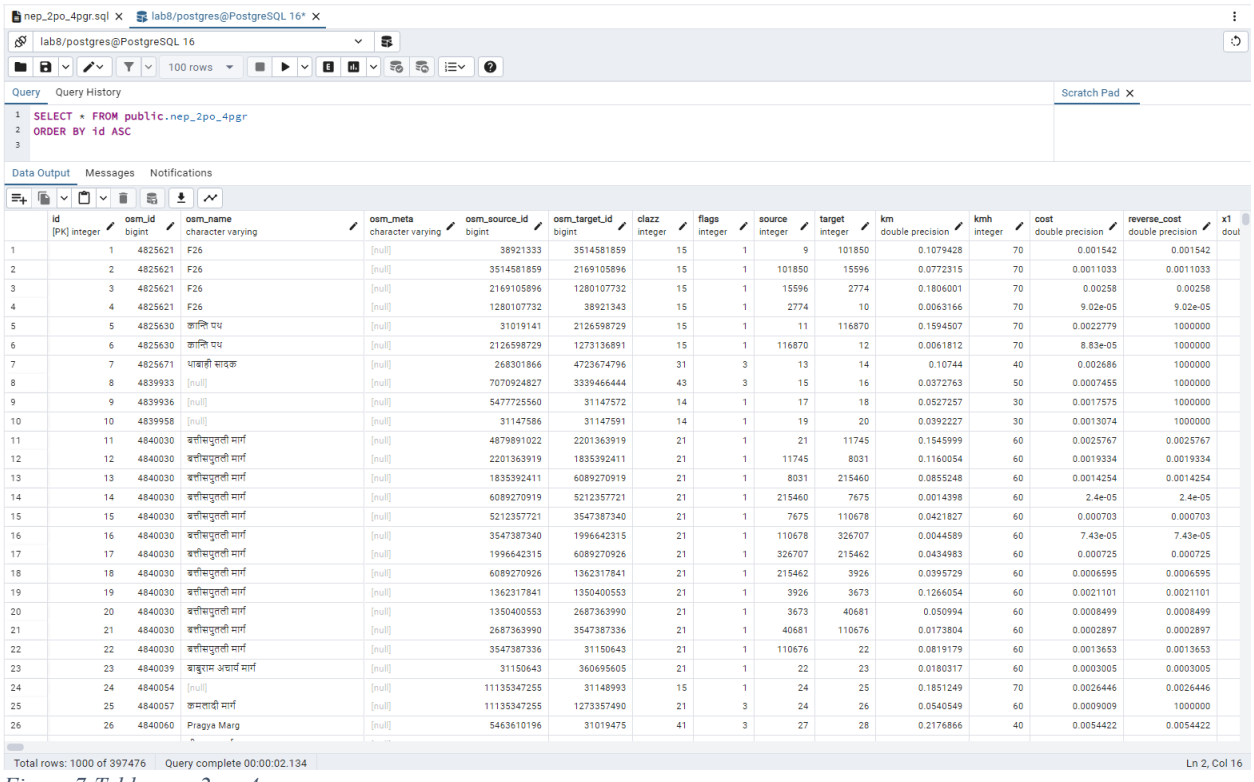


Figure 6:Execution of the Query stored in the converted file

After Executing the Query, we get the table nep_2po_4pgr,



	id [PK] integer	osm_id bigint	osm_name character varying	osm_meta character varying	osm_source_id bigint	osm_target_id bigint	clazz integer	flags integer	source integer	target integer	km double precision	kmh integer	cost double precision	reverse_cost double precision	x1 dout
1	1	4825621	F26	[null]	38921333	3514581859	15	1	9	101850	0.1079428	70	0.001542	0.001542	
2	2	4825621	F26	[null]	3514581859	2169105896	15	1	101850	15596	0.0772315	70	0.0011033	0.0011033	
3	3	4825621	F26	[null]	2169105896	1280107732	15	1	15596	2774	0.1806001	70	0.00258	0.00258	
4	4	4825621	F26	[null]	1280107732	38921343	15	1	2774	10	0.0063166	70	9.02e-05	9.02e-05	
5	5	4825630	काली पथ	[null]	31019141	2126598729	15	1	11	116870	0.1594507	70	0.0022779	1000000	
6	6	4825630	काली पथ	[null]	2126598729	1273136891	15	1	116870	12	0.0061812	70	8.83e-05	1000000	
7	7	4825671	धाबाही सादक	[null]	268301866	4723674796	31	3	13	14	0.10744	40	0.002686	1000000	
8	8	4839933	[null]	[null]	7070924827	3339466444	43	3	15	16	0.0372763	50	0.0007455	1000000	
9	9	4839936	[null]	[null]	5477725560	31147572	14	1	17	18	0.0527257	30	0.0017575	1000000	
10	10	4839958	[null]	[null]	31147586	31147591	14	1	19	20	0.0392227	30	0.0013074	1000000	
11	11	4840030	बत्तीसपुतली मार्ग	[null]	4879891022	2201363919	21	1	21	11745	0.1545999	60	0.0025767	0.0025767	
12	12	4840030	बत्तीसपुतली मार्ग	[null]	2201363919	1835392411	21	1	11745	8031	0.1160054	60	0.0019334	0.0019334	
13	13	4840030	बत्तीसपुतली मार्ग	[null]	1835392411	6089270919	21	1	8031	215460	0.0855248	60	0.0014254	0.0014254	
14	14	4840030	बत्तीसपुतली मार्ग	[null]	6089270919	5212357721	21	1	215460	7675	0.0014398	60	2.4e-05	2.4e-05	
15	15	4840030	बत्तीसपुतली मार्ग	[null]	5212357721	3547387340	21	1	7675	110678	0.0421827	60	0.000703	0.000703	
16	16	4840030	बत्तीसपुतली मार्ग	[null]	3547387340	1996642315	21	1	110678	326707	0.0044589	60	7.43e-05	7.43e-05	
17	17	4840030	बत्तीसपुतली मार्ग	[null]	1996642315	6089270926	21	1	326707	215462	0.0434983	60	0.000725	0.000725	
18	18	4840030	बत्तीसपुतली मार्ग	[null]	6089270926	1362317841	21	1	215462	3926	0.0395729	60	0.0006595	0.0006595	
19	19	4840030	बत्तीसपुतली मार्ग	[null]	1362317841	1350400553	21	1	3926	3673	0.1266054	60	0.0021101	0.0021101	
20	20	4840030	बत्तीसपुतली मार्ग	[null]	1350400553	2687363990	21	1	3673	40681	0.050994	60	0.0008499	0.0008499	
21	21	4840030	बत्तीसपुतली मार्ग	[null]	2687363990	3547387336	21	1	40681	110676	0.0173804	60	0.0002897	0.0002897	
22	22	4840030	बत्तीसपुतली मार्ग	[null]	3547387336	31150643	21	1	110676	22	0.0819179	60	0.0013653	0.0013653	
23	23	4840039	डाबुराम अचार्य मार्ग	[null]	31150643	360695605	21	1	22	23	0.0180317	60	0.0003005	0.0003005	
24	24	4840054	[null]	[null]	11135347255	31148993	15	1	24	25	0.1851249	70	0.0026446	0.0026446	
25	25	4840057	कमलादी मार्ग	[null]	11135347255	1273357490	21	3	24	26	0.0540549	60	0.0009009	1000000	
26	26	4840060	Pragya Marg	[null]	5463610196	31019475	41	3	27	28	0.2176866	40	0.0054422	0.0054422	

Figure 7:Table nep_2po_4pgr

Geometry of the Data:

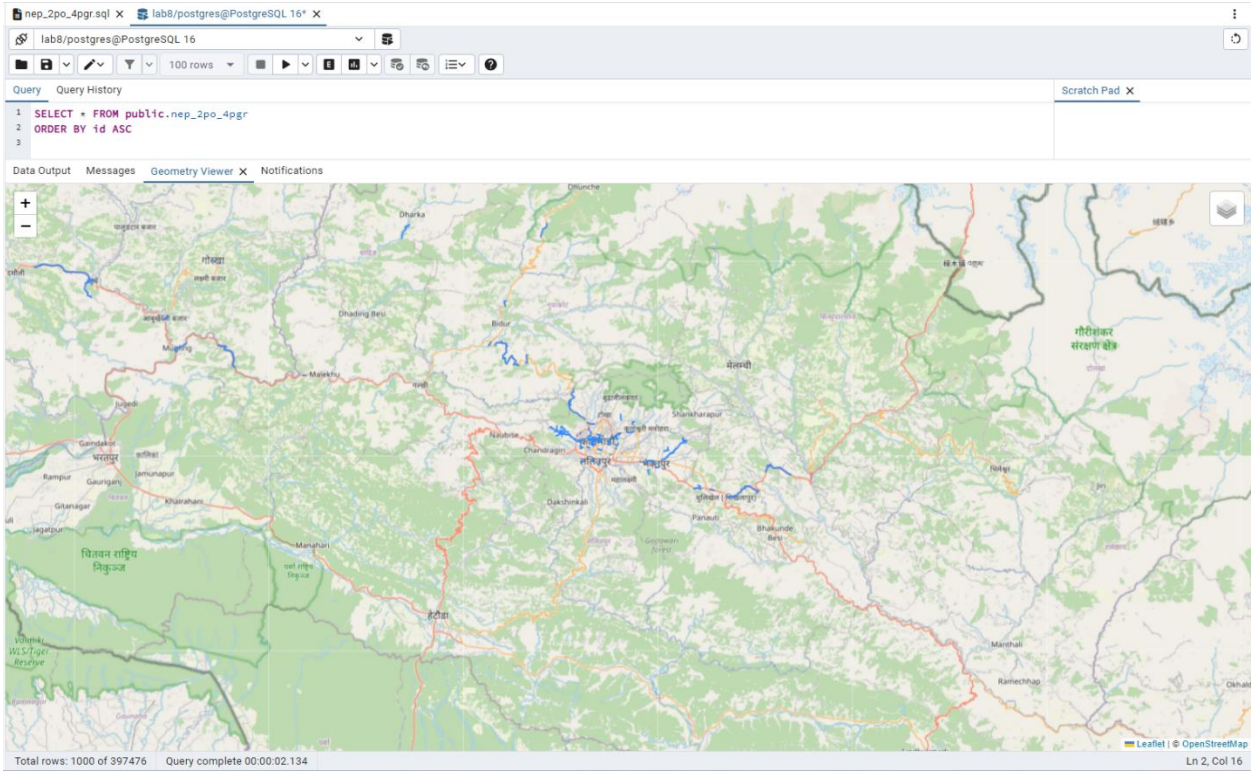


Figure 8:Geometry of the nep_2po_4pgr

Now, we are using QGIS for further process:

Making connection to QGIS with database in pgAdmin:

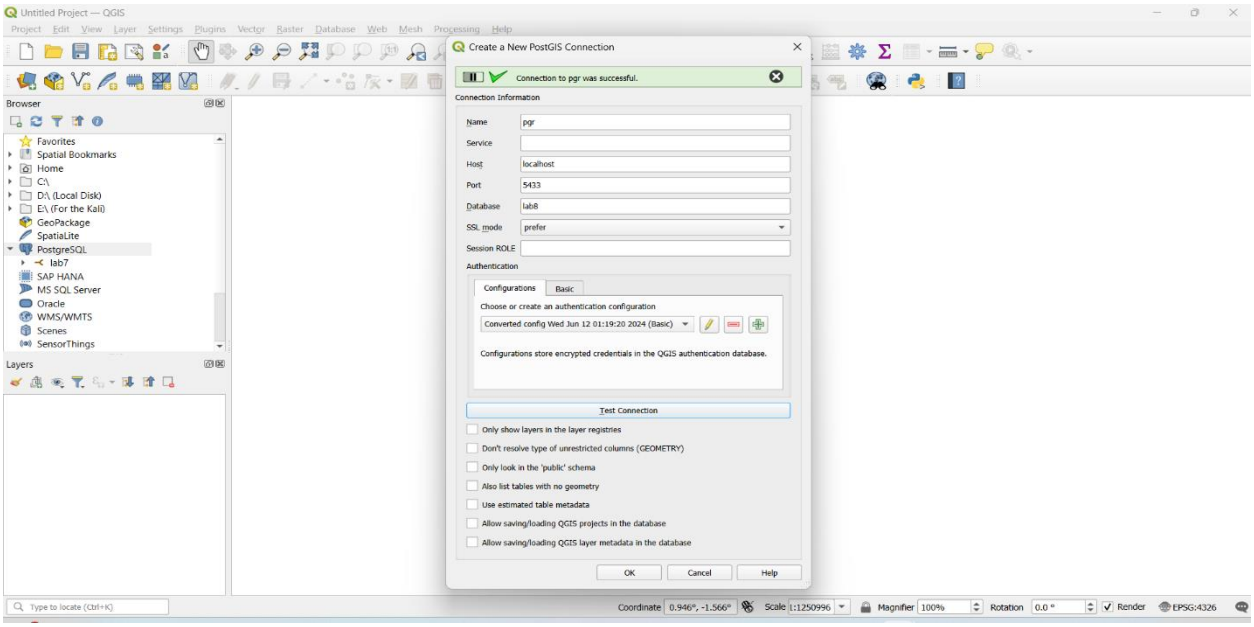
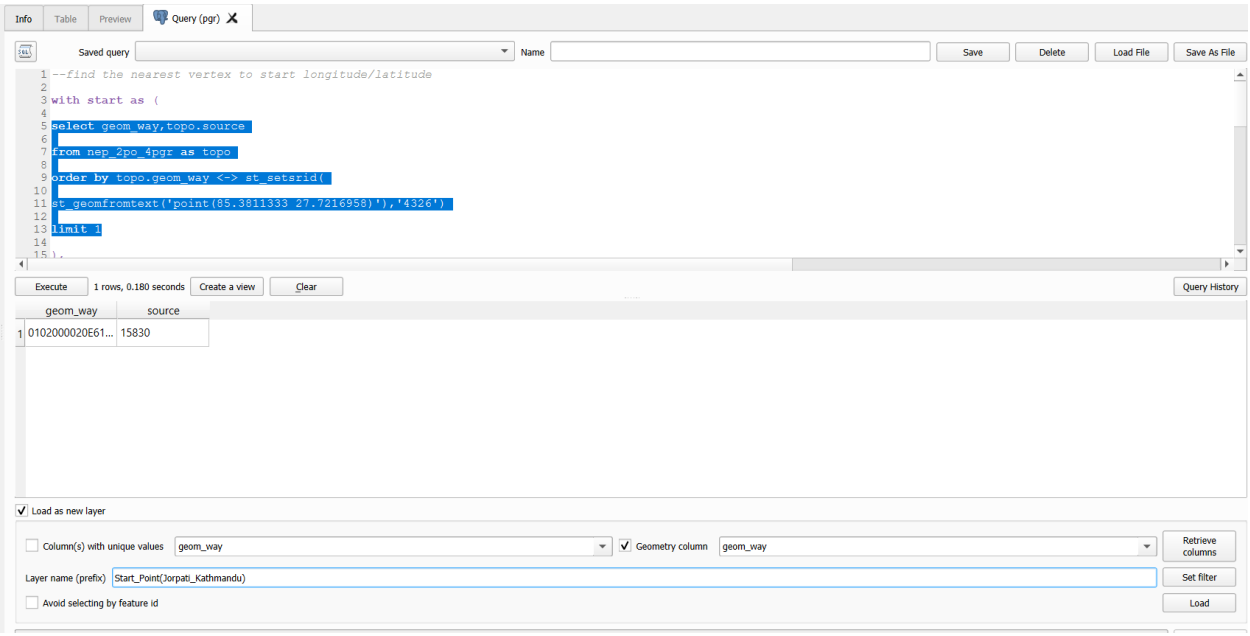


Figure 9:Connection with database

Setting the Origin/Start point/node:

Here, I take the Jorpati, Kathmandu location as Origin having coordinates:

- Latitude 27.72144°
- Longitude 85.37343°



Query:

--find the nearest vertex to start longitude/latitude

with, start as (

select geom_way,topo.source

from nep_2po_4pgr as topo

order by topo.geom_way <-> st_setsrid(

st_geomfromtext('point(85.3811333 27.7216958)'),'4326')

limit 1

)

Explanation:

This SQL query finds the `geom_way` and `topo.source` that is closest to a specific point (85.3811333, 27.7216958) in the `nep_2po_4pgr` table and saved as layer named start point.

Setting the Destination point/node:

For the destination, we take the Kathmandu University having coordinates

- Latitude 27.6187°
- Longitude 85.53735°

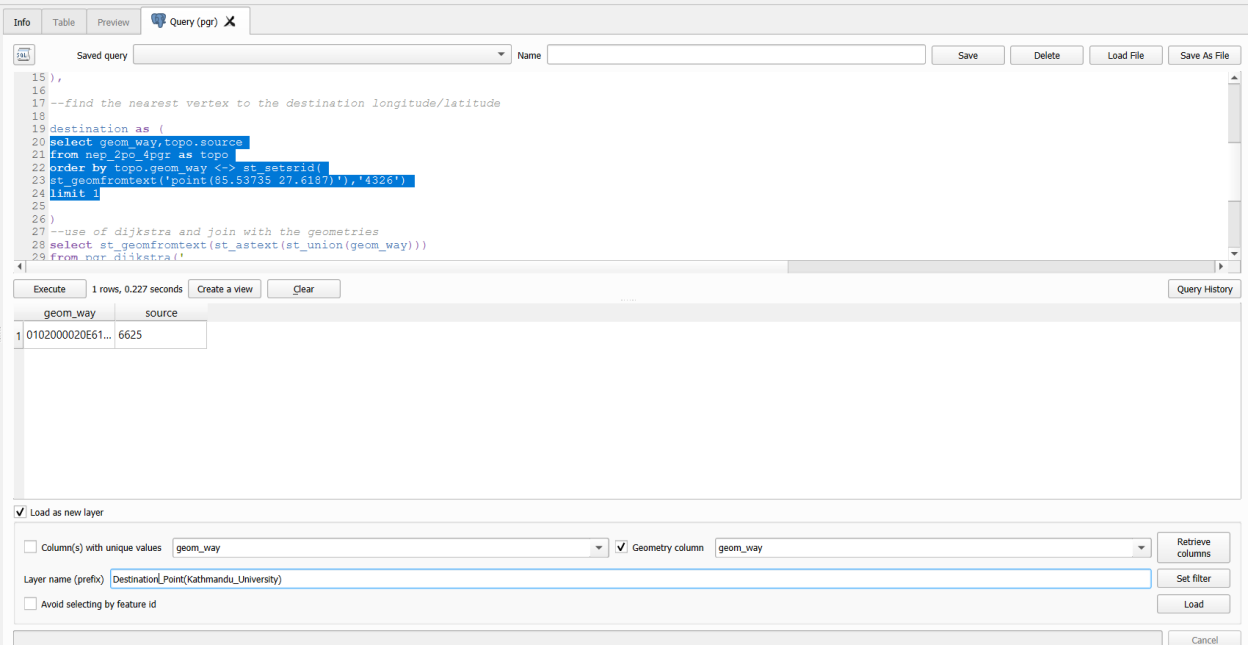


Figure 10:Creating extension pgrouting

Query:

--find the nearest vertex to the destination longitude/latitude

destination as (

select geom_way,topo.source

from nep_2po_4pgr as topo

order by topo.geom_way <-> st_setsrid(

st_geomfromtext('point(85.53735 27.6187)'),'4326')

limit 1

),

Explanation:

This SQL query finds the `geom_way` and `topo.source` that is closest to a specific point (85.53735, 27.6187) in the `nep_2po_4pgr` table and saved a layer named destination point.

After executing above query in QGIS, we save that result as a layer naming start and destination. Here, the **start point** is shown with **green dot** and **destination point** with **black dot** in the map and is shown below.

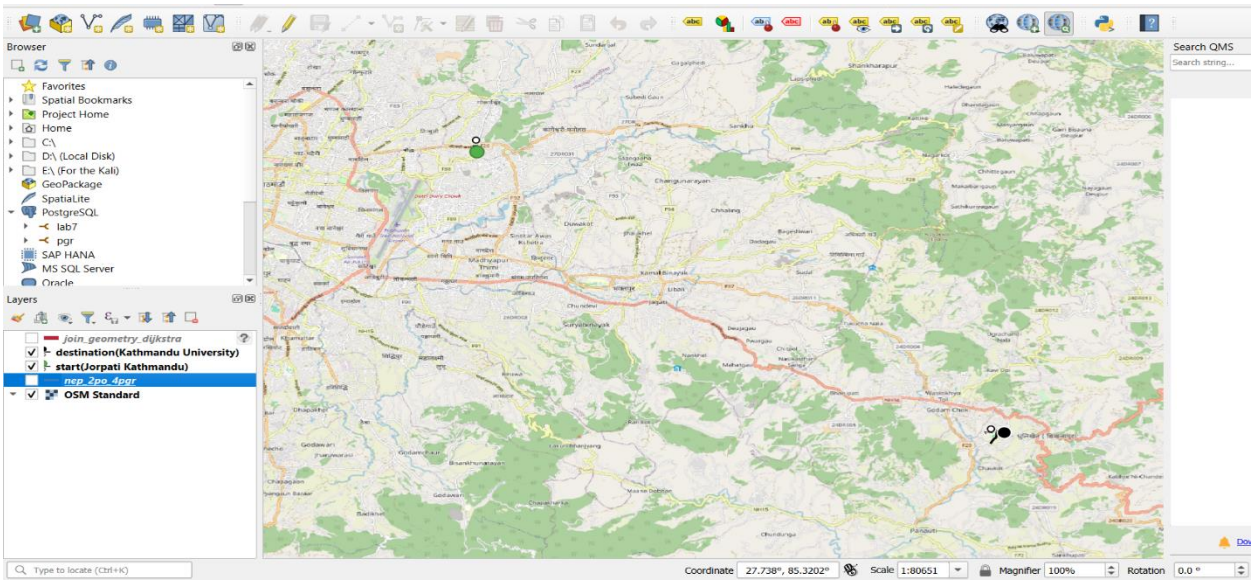


Figure 11:Starting & Destination Point

Finally, we apply the Dijkstra concept for shortest route and joins for joining the geometry. Here we run the combined queries i.e. Start node, Destination node, Dijkstra and join with geometry to get the shortest route and joining the geometry.

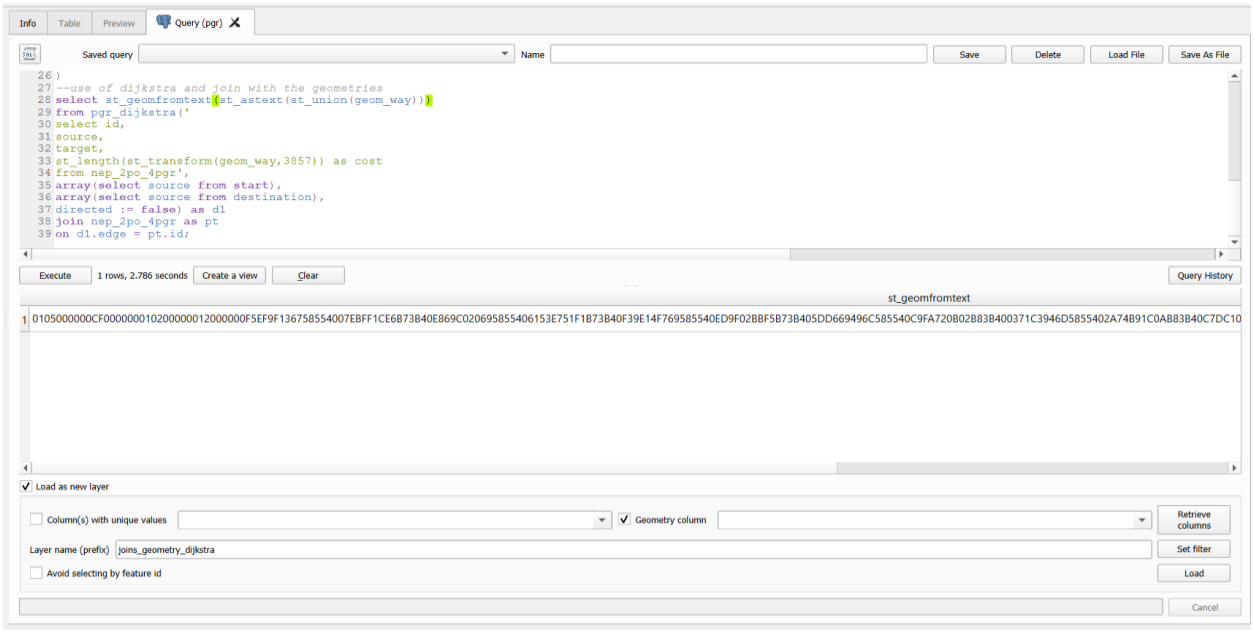


Figure 12:Finding the shortest route using Dijkstra and joins with geometry

Query executed for the shortest route using dijkstra:

```
--find the nearest vertex to start longitude/latitude
with, start as (
select geom_way,topo.source
from nep_2po_4pgr as topo
order by topo.geom_way <-> st_setsrid(
st_geomfromtext('point(85.3811333 27.7216958)'),'4326')
limit 1
),
--find the nearest vertex to the destination longitude/latitude
destination as (
select geom_way,topo.source
from nep_2po_4pgr as topo
order by topo.geom_way <-> st_setsrid(
st_geomfromtext('point(85.53735 27.6187)'),'4326')
limit 1
),
--use of dijkstra and join with the geometries
select st_geomfromtext(st_astext(st_union(geom_way)))
from pgr_dijkstra('
select id,
source,
target,
st_length(st_transform(geom_way,3857)) as cost
from nep_2po_4pgr',
array(select source from start),
array(select source from destination),
directed := false) as d1
join nep_2po_4pgr as pt
on d1.edge = pt.id;
```

Explanation:

This SQL query finds the nearest vertices to specific points in the `nep_2po_4pgr` table. It uses the `st_setsrid` function to convert the points to the WGS 84 spatial reference system and then calculates the distance between each vertex and the points using the `<->` operator. The results are ordered by distance and limited to the first row, which corresponds to the closest vertex. The query is executed twice, once for the start point and once for the destination point, and the results are stored in the `start` and `destination` tables, respectively.

Here, we saved the layer each of the start node, destination node and Dijkstra & join with geometry query in QGIS. To find the shortest route, we need to combine the layers, the combined layer as a map is shown below:

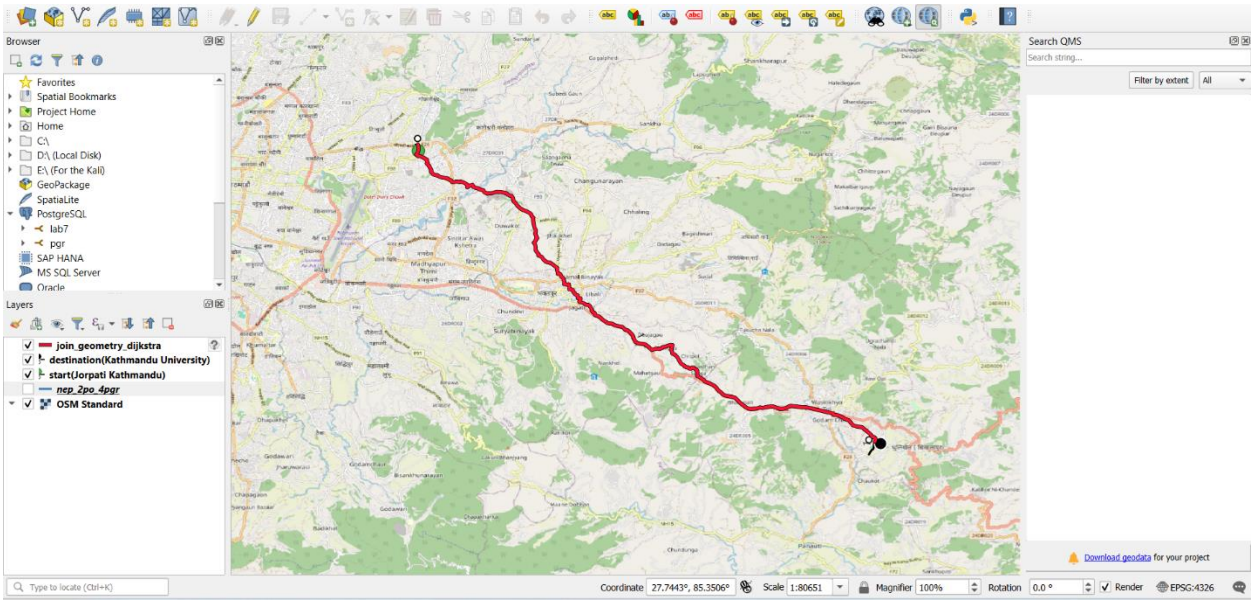


Figure 13:Shorest Route from start node to destination node

In the map above,

- **black** dot shows **destination point/node** i.e. Kathmandu University, Dhulikhel-Kavre.
- **green** dot shows **start point/node** i.e. Jorpati, Kathmandu.

As a whole, this concludes this one simple demonstration of **determining the shortest route** using **pgRouting.**