



Utrecht University

Applied Data Science Master's degree programme

Spatial Data Analysis and Simulation Modelling course

Instruction manual for Lab 2.3:
spatial databases: pgRouting for emergency response

Document version: 0.9

Document modified: 2020.10.29

Dr. Zhiyong Wang, Dr. Simon Scheider

Department of Human Geography and Spatial Planning

Faculty of Geosciences

z.wang2@uu.nl, s.scheider@uu.nl

Introduction

Spatial databases are an important part of the computational infrastructure used by spatial data scientists. Special technical and query functionality needs to be mastered in order to handle spatial data with such databases. In this lab, you will learn how to manage spatial databases and how to use them as computational infrastructure for spatial network analysis. The lab is focused around a case study in emergency management in Amsterdam (see below). In particular, the learning goals of this lab are:

- Learn to use SQL to query the closest neighbours
- Learn to use Pgrouting to calculate the shortest route
- Learn to use QGIS to visualize query results
- Learn to prepare the network data for routing
- Learn to use Postgis functions to compute geometric attributes and to transform coordinate reference systems

The following software will be used in the lab:

- 1) PostgreSQL/PostGIS
- 2) pgRouting
- 3) QGIS

The following datasets will be used in the lab:

- 1) the road network of Amsterdam (roads_ams_2008.shp)
- 2) the dataset of obstacles (obstacles.shp)
- 3) Fire stations in Amsterdam (firestations_ams_2013.shp)

To save time for the practical, these datasets are already packaged into shapefiles are compressed in a zipped file (Lab 2.3 data.zip). you can download them from the blackboard (Lab 2.3 data).

Some basic knowledge of SQL is a prerequisite for doing this lab. You can study SQL via the following link: <https://www.postgresqltutorial.com/>. The SQL commands below will be used in this lab:

- Select
- Where
- Order by
- Left Join
- Limit
- Update
- Alter table
- Add column

Case study

A strong storm has damaged many houses in the city of Amsterdam, and in many areas help is needed. Some of the streets are already blocked and many more might be blocked in the coming hours. The images below show some of the damages after the storm:



You are appointed as the technical person to manage the routing for emergency responders who will travel from different fire stations to rescue points.



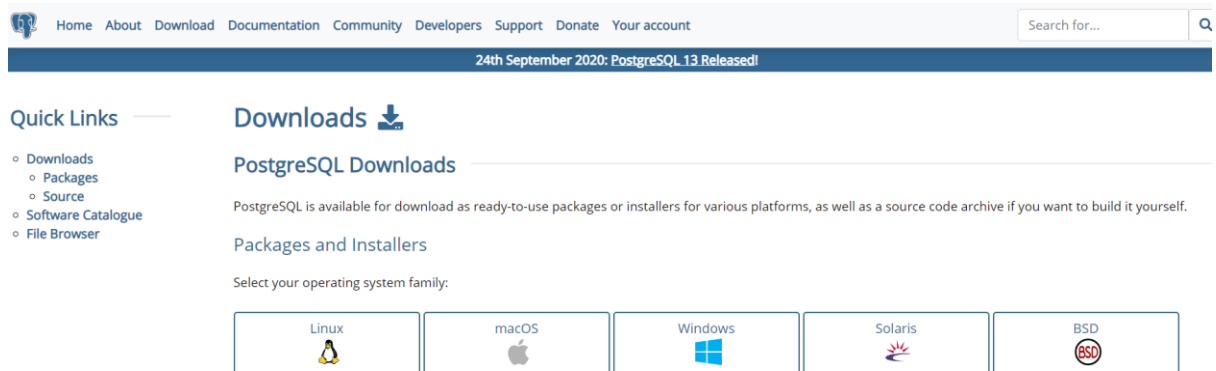
The following tasks need to be performed by you:

- Install the software and prepare the data
- Navigate fire trucks from 2 fire stations to 5 destinations (rescue points). In this tutorial, you will select two fire stations (OBJECTID = 1014 and OBJECTID = 533), but you may pick other fire stations for your case. The coordinates (lon and lat given in WGS 84) of the 5 destination points are listed below:
 - 1). (4.806462559, 52.4438188114)
 - 2). (4.816071043, 52.3480865708)
 - 3). (5.000307123, 52.3539927035)
 - 4). (4.864906827, 52.3127379258)
 - 5). (4.923351096, 52.4117317620)
- Find the closest fire station for each destination
- Plan the shortest routes to the destinations, avoiding obstacles

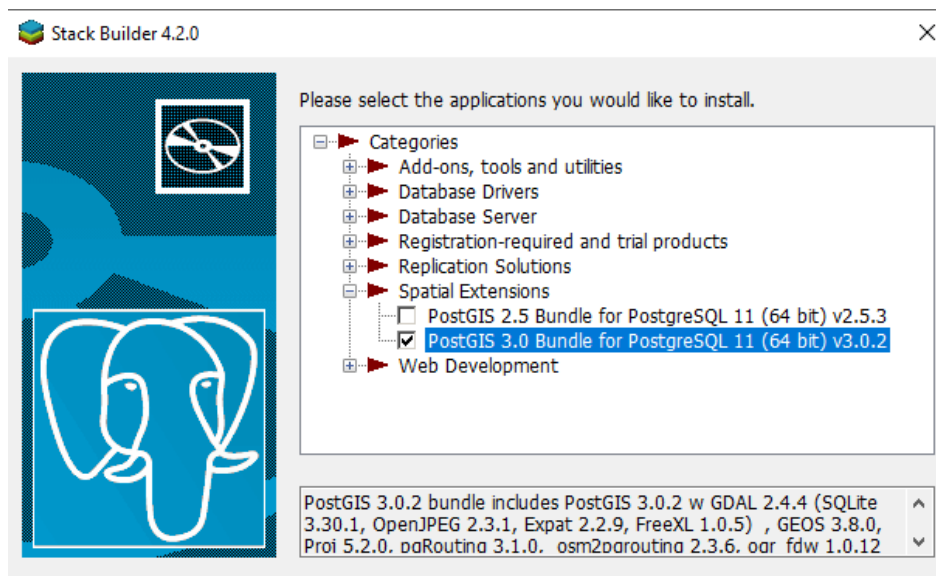
Setup

Prepare a postgresql/postgis database

Download the Postgresql from via the following link and install it.
<https://www.postgresql.org/download/>

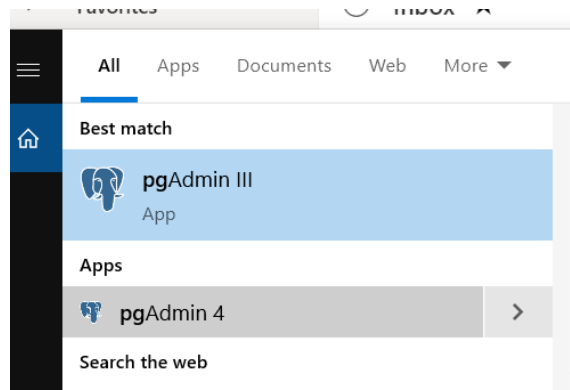


If you select the version for Windows, at the end of the installation, you can install the postgis via Stack Builder (as shown below). Go to **Categories** and expand the **Spatial Extensions**, and select a postgis version you would like to install in your computer. After the selection, just click next to continue and finish the installation.

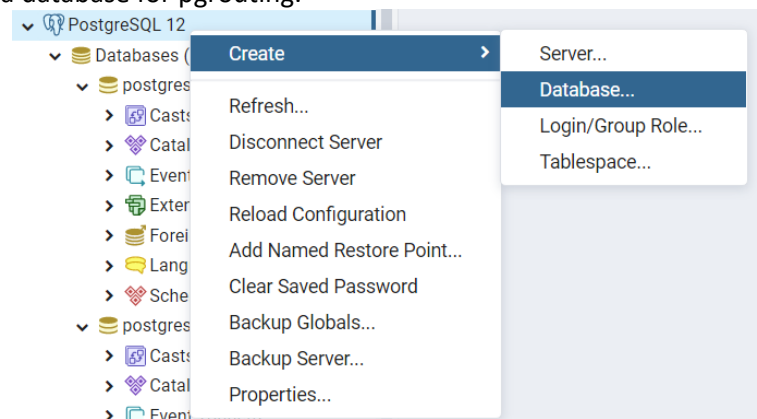


In the case that you already Postgresql installed in your PC, you can download postgis installation package via the following link and install it: <https://postgis.net/install/>. Please always be sure that you already have postgresql installed on your computer before installing the postgis.

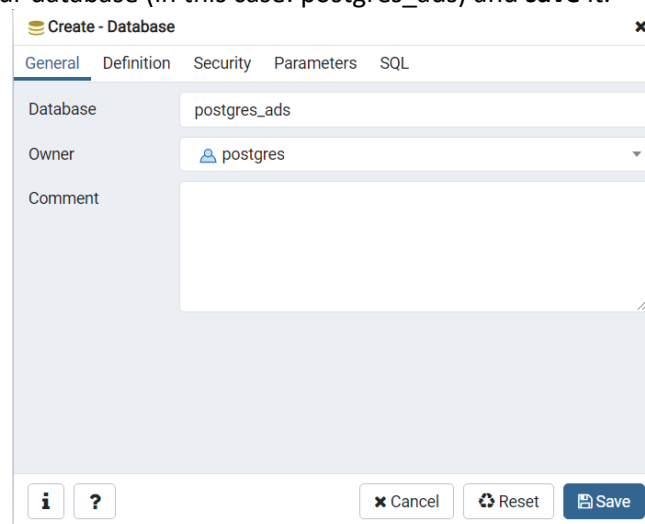
After the installation, you can open **pgAdmin** to run sql queries. **pgAdmin** is a user interface that allows you to access the database. Search **pgAdmin** in the Windows search bar and click it. For pgAdmin 4, a new tab will be opened in your browser (as shown below). For pgAdmin III, the gpAdmin desktop application will be launched.



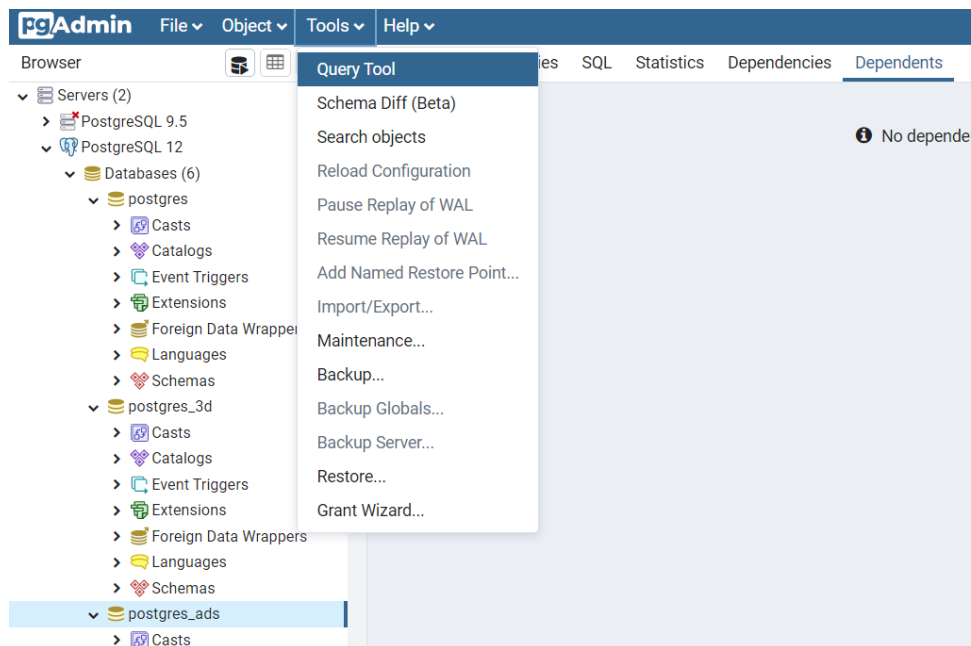
Right click the postgresql server (in this case: PostgreSQL 12), and go to **Create** and select **Database** to create a database for pgrouting.




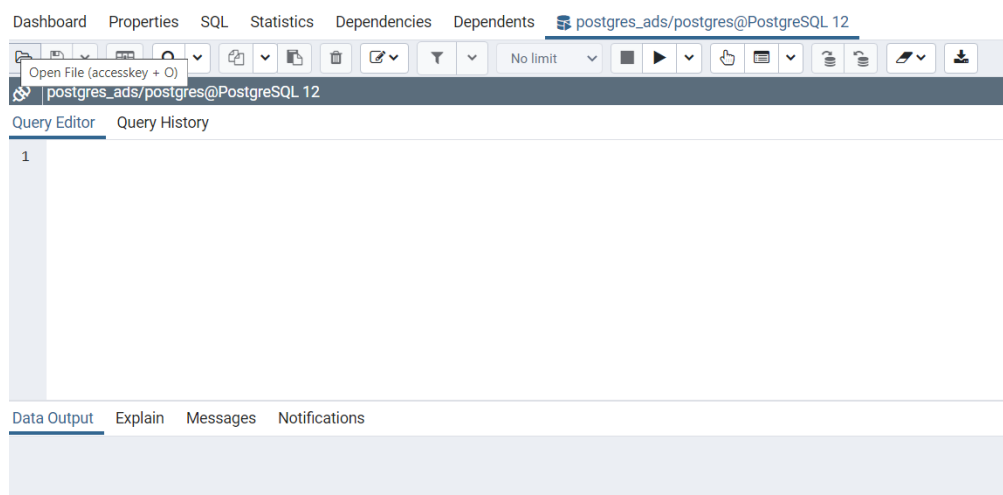
Give a **name** to your database (in this case: postgres_ads) and **save** it.



Left click your newly created database and go to Tools, and select **Query Tool**



You will see a new interface opened as shown below. You can enter your sql commands in the blank space and click  to run the commands.

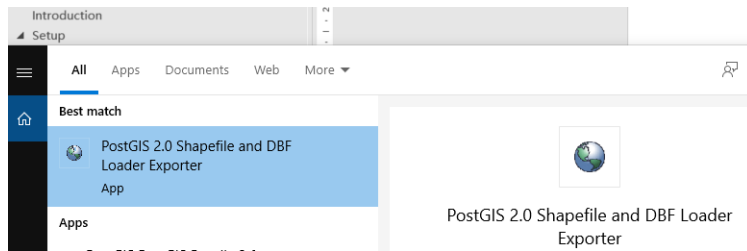


Run the following commands to add *PostGIS* and *pgrouting* extensions to the created database:

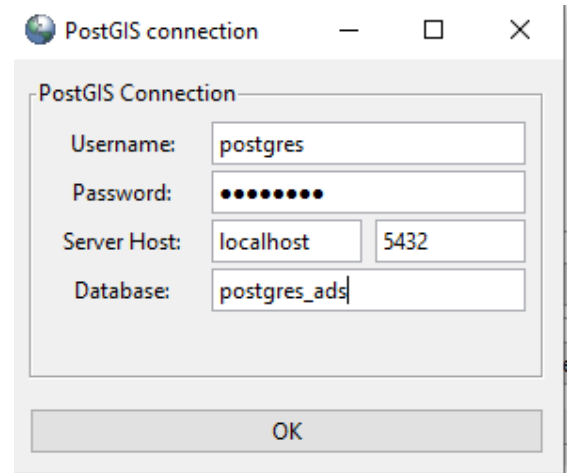
```
CREATE EXTENSION postgis;
CREATE EXTENSION pgRouting;
```

Load the data

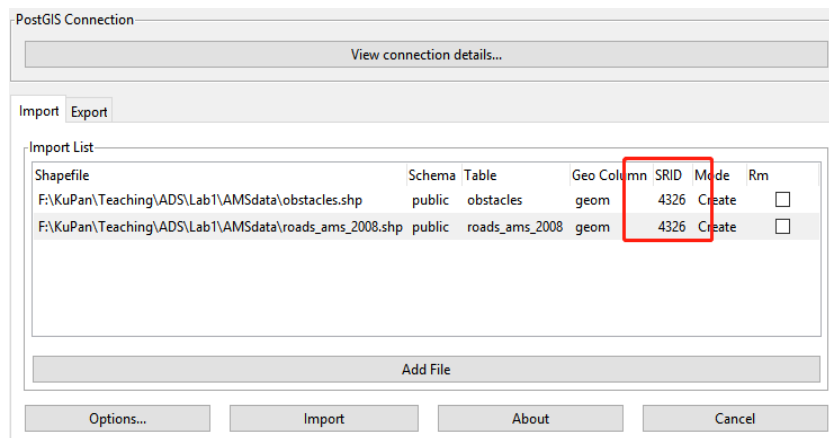
Search for **postgis 2.0 shapefile and dbf loader** and click it.



Click **View connection details** and enter your username/password and your database name



Download the data from the blackboard “Lab 2.3 data.zip” and unzip it to some folder on a local drive, e.g. C:\Temp\dataSpatialDatabase\. Locate the downloaded data files, and add **obstacles.shp** and **roads_ams_2008.shp**. Because all these shp files are in the World Geodetic System (WGS84) as reference coordinate system, we set SRID to 4326 (this is the EPSG number of WGS84). Click **Import** to import the data into the database



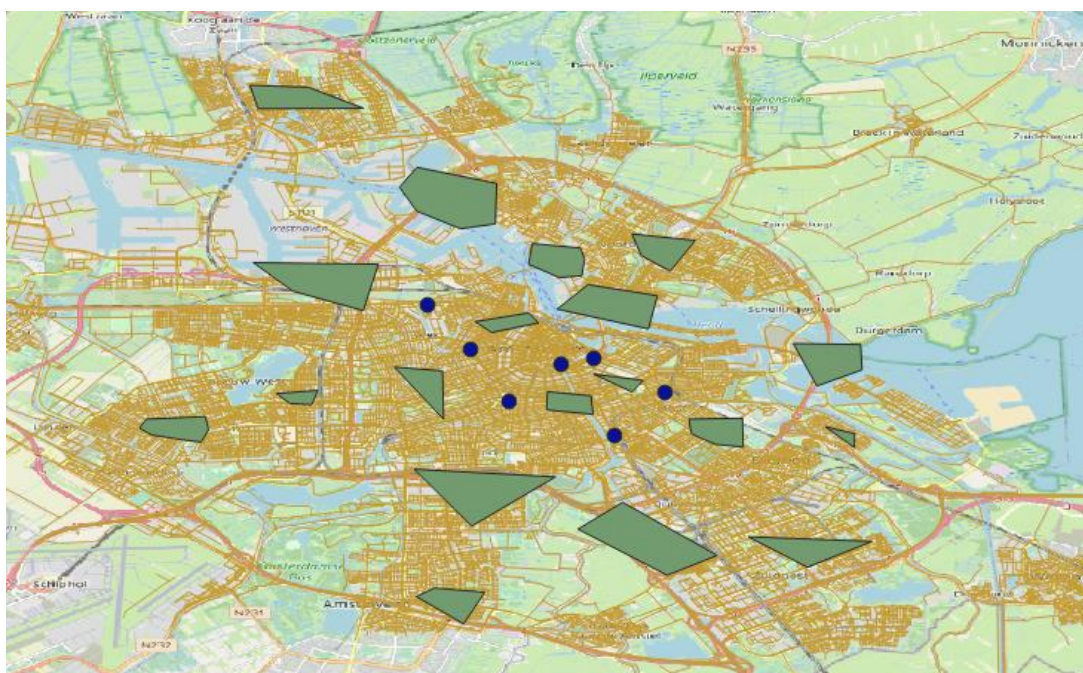
You can run the following sql scripts in pgadmin to see what attributes are already contained the datasets. Note that by using **limit**, you select a subset of the rows returned by the query, so only 10 rows are shown here.

```
Select * from roads_ams_2008 limit 10
Select * from obstacles limit 10
```


postgres_ads/postgres@PostgreSQL 12									
Query Editor Query History									
1 Select * from roads_ams_2008 limit 10									
Data Output Explain Messages Notifications									
	gid [PK] integer	__gid numeric (10)	objectid numeric (10)	feattyp numeric (10)	name character varying (70)	routenum character varying (10)	fow numeric (10)		
1		26448	26429	44426192	4110 [null]	[null]		3	
2		15	13789	44410804	4110 [null]	[null]		10	
3		122	44	44375982	4110 Haarlemmerstraatweg	[null]		3	
4		123	45	44375983	4110 Haarlemmerstraatweg	[null]		3	
5		124	46	44375989	4110 Zwanenburgerdijk	[null]		3	
6		314	237	44391525	4110 Kuiperstraat	[null]		3	
7		356	279	44391863	4110 Westhoff	[null]		3	
8		4863	4799	44400660	4110 Naritaweg	[null]		2	
9		1367	1296	44394773	4110 Grenehout	[null]		3	
10		31249	31241	44431004	4110 Alexanderplein	[null]		3	

1 Select * from obstacles limit 10			
Data Output Explain Messages Notifications			
	gid [PK] integer	id numeric (10)	geom geometry
1		1	0106000020E6100...
2		2	0106000020E6100...
3		3	0106000020E6100...
4		4	0106000020E6100...
5		5	0106000020E6100...
6		6	0106000020E6100...
7		7	0106000020E6100...
8		8	0106000020E6100...
9		9	0106000020E6100...
10		10	0106000020E6100...

To get a better view of the datasets, you can also use QGIS and visualize all data shape files.



Pre-processing the data

Before performing network analysis, we have to create topology for the road network. Topology is a set of rules that defines the spatial relationship between adjacent objects. For the road network, it models how line and points are connected to each other.

Create a topology

First of all we add "source" and "target" column into the network table. We do this in order to store network edge relations between network vertices (road intersections) in the table. Source is the from vertex, and target is the to vertex of the road network segment:

```
ALTER TABLE roads_ams_2008 ADD COLUMN "source" integer;  
ALTER TABLE roads_ams_2008 ADD COLUMN "target" integer;
```

Run the function below to create a topology. This may take a few minutes. A new table **roads_ams_2008_vertices_pgr** will be created and contains all nodes of the road network. Later we will use this table to find the nodes that are closest to the fire stations as well as destinations.

```
SELECT pgr_createTopology('roads_ams_2008', 0.000001, 'geom', 'gid');
```

The function **pgr_createTopology** builds a network topology, based on the geometry information of the network dataset. After the running of this function is finished, you can see that the "source" and "target" column will be filled with numbers, which are IDs of nodes in the table **roads_ams_2008_vertices_pgr**. More details of this function can be found below. https://docs.pgrouting.org/2.2/en/src/topology/doc/pgr_createTopology.html.

Use the following sql commands to *calculate the length* for each segment in order to add network impedances/costs to the network. However, since length measurements make use of geometric computations in the Euclidean plane, we need to first turn the geometries to planar (projected) coordinates. Because the coordinate reference system in the source data is WGS84, we have to reproject it to the Dutch CRS "RD New / Amersfoort". Units are in meters:

```
Alter table roads_ams_2008 add cost double precision;  
update roads_ams_2008 set cost=st_length(ST_Transform(geom, 28992))
```

To calculate the costs for reverse directions, here we assume that all roads are bi-directional and the forward and backward costs are the same. Use the following sql commands to calculate the reverse costs:

```
ALTER TABLE roads_ams_2008 ADD COLUMN reverse_cost double precision;  
UPDATE roads_ams_2008 SET reverse_cost = cost;
```

After the DB transaction has finished, you should run the sql command below to check that whether the newly added column source, target, cost and reverse_cost are filled with values (as shown below).

```
Select * from roads_ams_2008 limit 10
```

level merit (10)	geom geometry	source integer	target integer	cost double precision	reverse_cost double precision
0	0105000020E6100...	29	30	87.96441310991665	87.96441310991665
0	0105000020E6100...	93	94	66.65141152708902	66.65141152708902
1	0105000020E6100...	232	183	60.88052877278345	60.88052877278345
1	0105000020E6100...	231	232	745.8185301335443	745.8185301335443
1	0105000020E6100...	233	234	182.37419751555427	182.37419751555427
0	0105000020E6100...	412	413	36.710517443398665	36.710517443398665
0	0105000020E6100...	328	442	136.1349513597535	136.1349513597535
0	0105000020E6100...	1059	1287	50.38339730525993	50.38339730525993
0	0105000020E6100...	1870	1935	36.94293207653316	36.94293207653316
0	0105000020E6100...	2132	2133	57.72732002913275	57.72732002913275

Find the access nodes in the network closest to the fire stations:

Load the the dataset of the fires stations (firestations_ams_2013.sh) into your database. Now we are ready to fire a first spatial SQL query. ST_Distance computes the Euclidean distance between different geometries in tables. We are going to use this function to determine the closest access points of the network for each fire station. Using the command below will find the closest network node for the selected fire stations and write the query result (id of the closest node) for each fire station into a new table:

```
SELECT roads_ams_2008_vertices_pgr.id FROM roads_ams_2008_vertices_pgr,
firestations_ams_2013 where firestations_ams_2013.OBJECTID=1014 ORDER BY
ST_Distance(ST_Transform(ST_SetSRID(firestations_ams_2013.geom, 4326),
28992), ST_Transform(roads_ams_2008_vertices_pgr.the_geom, 28992)) ASC
LIMIT 1;
```

The query first used function **ST_Transform** to re-project the geometries to “RD new” (SRID = 28992), and then use the function **ST_Distance** to calculate the distances between roads and fire stations, the distance results are ordered in an ascend order, and only the first row (closest to the fire station OBJECTID=1014) is returned from the query.

For the fire station OBJECTID = 1014, the closest node id is 19354

Data Output		Explain	Messages	Notif
	id [PK] bigint			
1	19354			

Use the same query for the fire station OBJECTID = 533, the closest node id is 22565

Please study the following documentation of the ST_Distance function to perform Nearest-Neighbour Searching <https://postgis.net/workshops/postgis-intro/knn.html>.

Find the nodes in the network closest to the destinations:

Using the command below to find the closest network node for each emergency destination and write

the query result (id of the closest node) for each destination into a new table:

```
SELECT id FROM roads_ams_2008_vertices_pgr ORDER BY
ST_Distance(ST_GeomFromText('POINT(4.806462559
52.4438188114)',4326)::geography, the_geom::geography) ASC LIMIT 1;
```



Now we will store all destinations into a separate table to be used in further queries. Use the commands below to create a new table **destinations** in the database and store the geometric information of all emergency destinations and the nodes that are closest to them in WKT (WGS84) coordinates. **Note that you have to change the coordinates and the closest nodes for each destination.**

```
Drop table if exists destinations;
Create table destinations (
id integer,
nn_id integer, -- nearest node id
geom geometry(point, 4326)
);
```

```
--please change 1300 to the closet node ID for the destination
POINT(4.806462559 52.4438188114), according to your network
INSERT INTO destinations (id, nn_id, geom) values (1, 1300,
ST_GeomFromText('POINT(4.806462559 52.4438188114)',4326));
```

```
--please change XXXX to the closet node ID for the destination
POINT(4.816071043 52.3480865708), according to your network
INSERT INTO destinations (id, nn_id, geom) values (2, XXXX,
ST_GeomFromText('POINT(4.816071043 52.3480865708)',4326));
```

```
--please change XXXX to the closet node ID for the destination
POINT(5.000307123 52.3539927035), according to your network
INSERT INTO destinations (id, nn_id, geom) values (3, XXXX,
ST_GeomFromText('POINT(5.000307123 52.3539927035)',4326));
```

```
--please change XXXX to the closet node ID for the destination
POINT(4.864906827 52.3127379258), according to your network
INSERT INTO destinations (id, nn_id, geom) values (4, XXXX,
ST_GeomFromText('POINT(4.864906827 52.3127379258)',4326));
```

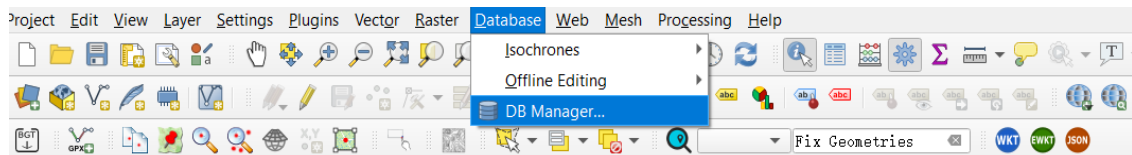
```
--please change XXXX to the closet node ID for the destination
POINT(4.923351096 52.4117317620), according to your network
INSERT INTO destinations (id, nn_id, geom) values (5, XXXX,
ST_GeomFromText('POINT(4.923351096 52.4117317620)',4326));
```

Path planning

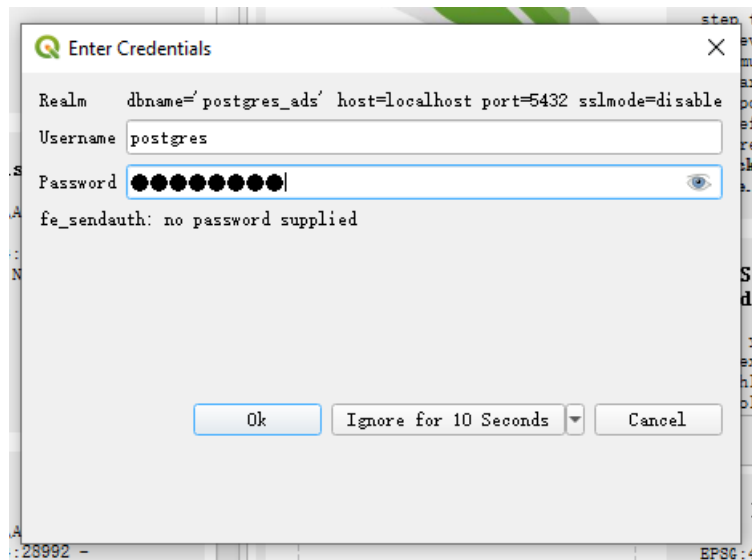
In this section, we first compute the shortest distance from a fire station to a destination point. We use the *pgrouting* function for this purpose and create a SQL query to obtain the route. From this section, we will use QGIS to run SQL queries and to visualize the computed routes.

Using the Dijkstra algorithm to compute shortest route

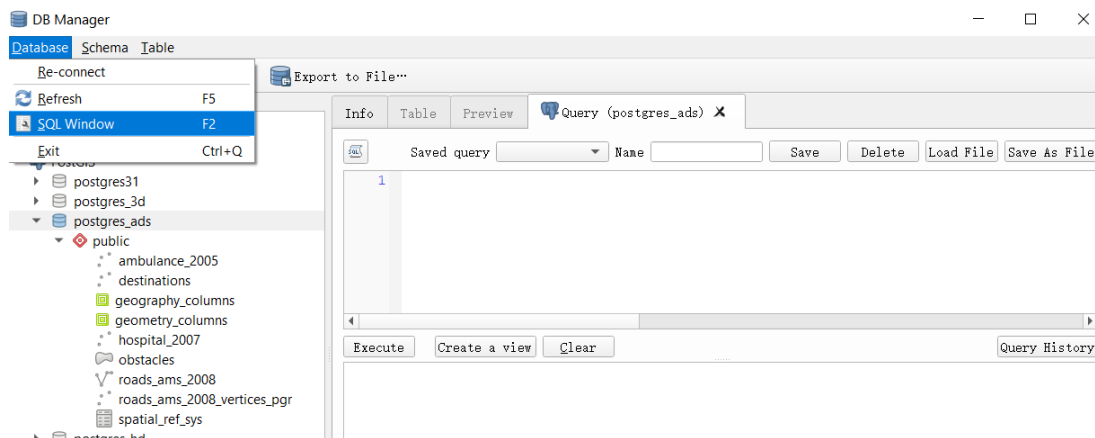
Open QGIS, and go to Database --> DB Manager..



In the DB Manager dialog, double click the database **postgres_ads** to connect it. Type your username and password when you see the prompt as shown below.



After the connection to the database is established, go to the **Database** and Click the **SQL Window** button to launch SQL window.



Here we pick the fire station (OBJECTID=1014, nearest node id=19345) as the origin and select one destination (id=4, nearest node id = 12215). We can use the Dijkstra algorithm to compute the shortest route between the fire station and the network access node of this destination. **Note that you should use the nearest network node id in the pgrouting function for the destination point, not the destination geometry.**

```
SELECT * FROM pgr_dijkstra('
SELECT gid AS id,
    source,
    target,
    cost,
    reverse_cost
FROM roads_ams_2008',
19345, 12215, false);
```

If everything went fine, you can see the results as shown below:

SQL

Saved query

Name

Save

```

1 - SELECT * FROM pgr_dijkstra('
2   SELECT gid AS id,
3     source,
4     target,
5     cost,
6     reverse_cost
7   FROM roads_ams_2008',
8   19345, 12215, false);

```

Execute

122 rows, 1.850 seconds

Create a view

Clear

	seq	path_seq	node	edge	cost	agg_cost
1	1	1	19345	26248	25.7777145659...	0.0
2	2	2	19340	26142	24.8843218922...	25.7777145659...
3	3	3	18901	25573	85.3845471059...	50.6620364582...
4	4	4	18918	25572	6.55506767998...	136.046583564...
5	5	5	18894	25623	10.1127499890...	142.601651244...
6	6	6	18892	26280	191.691818133...	152.714401233...
7	7	7	18964	25631	24.3028709341...	344.406219366...
8	8	8	18968	25633	55.1143933711...	368.709090300...
9	9	9	18970	25648	37.9142822284...	423.823483671...
10	10	10	18983	25647	42.6374577824...	461.737765900...

Take some time and study the detail of the Dijkstra function:

Reference:

<https://www.interviewbit.com/tutorial/dijkstra-algorithm/>

https://docs.pgrouting.org/2.2/en/src/dijkstra/doc/pgr_dijkstra.html#pgr-dijkstra

Visualization of routes

The previous query only returned edge ids along the route together with the cost. It would be more useful if the actual route is visualized. To do so, we have to modify the above query by joining tables.

```

SELECT s.seq, s.node, s.edge, s.cost,
b.gid, b.geom
FROM
(
SELECT * FROM pgr_dijkstra('
SELECT gid AS id,
      source,
      target,
      cost,
      reverse_cost
FROM roads_ams_2008',
19354, 12215, false)
) s
LEFT JOIN roads_ams_2008 b
ON (b.gid = s.edge)

```

The routing query results from **pgr_dijkstra** contains a column “**edge**” which is the edge ID (**gid**) of the road network. By joining the routing result table and the network table, we select the network edges that are contained in the route, and use their geometries to visualize the route.

Once the calculation is finished, click the checkbox for **Load as new layer** in the panel of DB Manager. Select the ID column of roads (**gid**) as the **Column(s) with unique values** and **geom** as geometry column. Click **Load** to load the results to the canvas of QGIS.



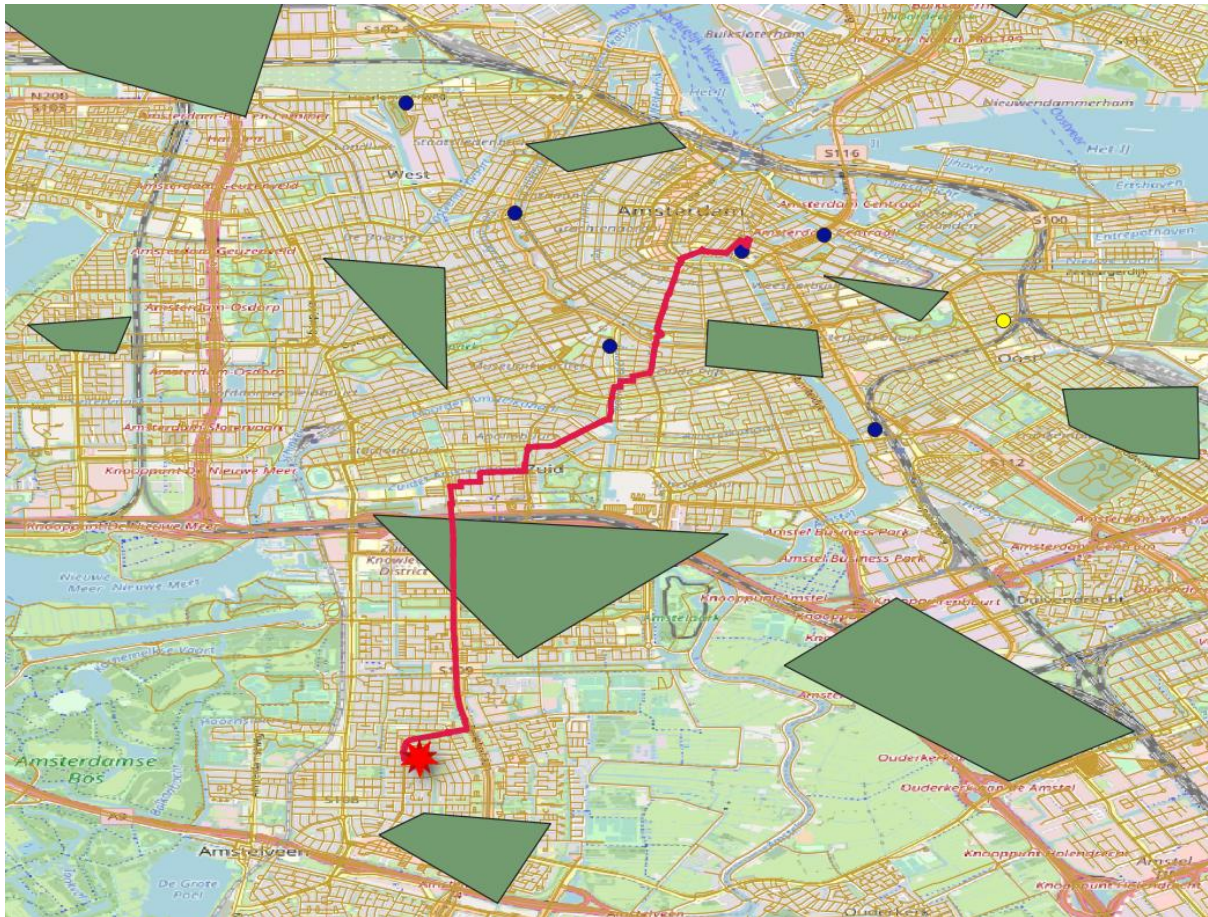
☒ Load as new layer

☐ Column(s) with unique values gid ☒ Geometry column geom Retrieve columns

Layer name (prefix) Set filter

☐ Avoid selecting by feature id Load

A new layer **QueryLayer** will be added to the Layers panel in QGIS. This layer shows the lines connecting the fire station and the destination. As you can see, the shortest route is not feasible as it is blocked by the obstacles.



Take some time to study how to visualize pgRouting results:

Reference: <https://gis.stackexchange.com/questions/122125/how-to-visualize-pgrouting-result>
<https://www.w3resource.com/PostgreSQL/postgresql-join.php>

Path planning in the presence of obstacles

In this practical, we have to navigate emergency responses from the selected two fire stations to the 5 destinations, and provide fire trucks with safe routes that avoid obstacles. These safe routes will be longer than the shortest routes. For each destination, we will therefore need to find a fire station that has the least travel distance when considering obstacles.

Update road cost considering obstacles

Change the **cost** and **reverse_cost** of streets that are affected by obstacles, using the following sql command

```
UPDATE roads_ams_2008 SET cost =999999, reverse_cost=999999 From obstacles
WHERE ST_Intersects(roads_ams_2008.geom, obstacles.geom)
```

The function **ST_Intersects** check if two geometries intersect with each other. In this Lab, we assume a road is not available if it intersects with any obstacle polygons. We assign a big number (999999) to

all the roads that are unavailable as their travel cost, so that these roads would not be considered in the shortest routing.

Reference: https://postgis.net/docs/ST_Intersects.html

Path planning with obstacles

After the road network is updated with new cost by obstacles, we can simply use the pgrouting query to obtain the route avoiding the obstacles.

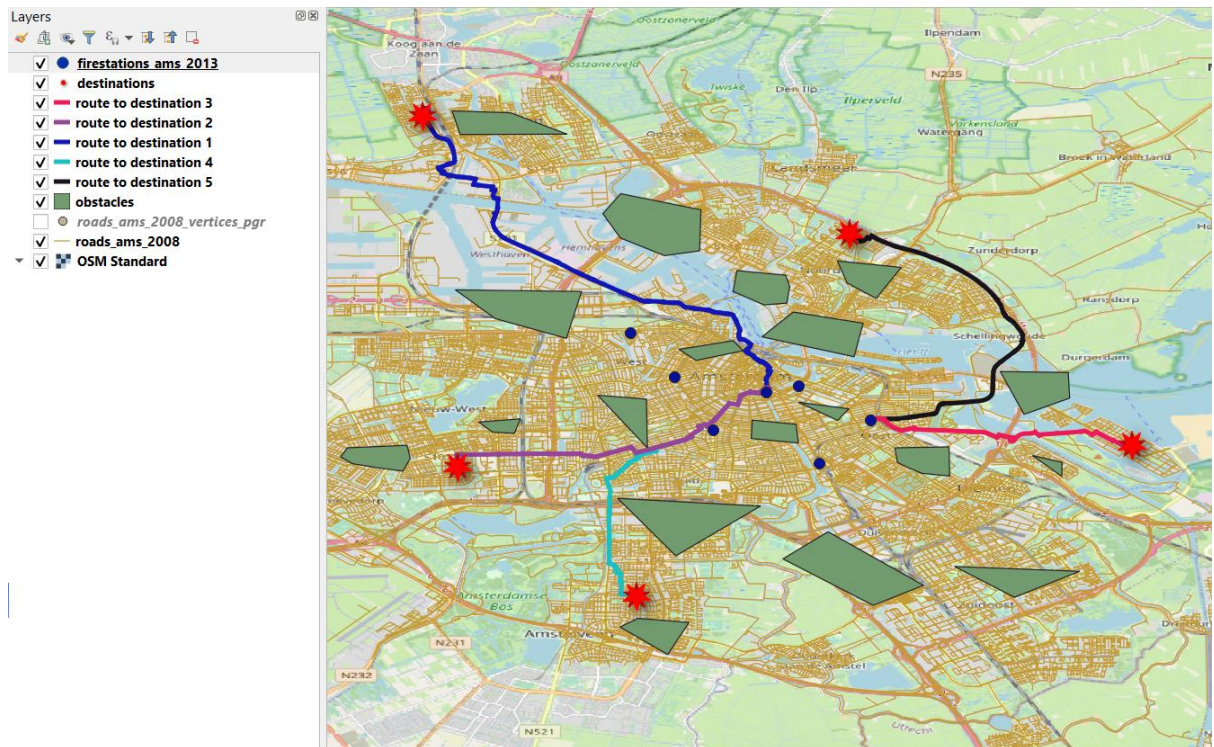
```
SELECT s.seq, s.node, s.edge, s.cost,
b.gid, b.geom
FROM
(
SELECT * FROM pgr_dijkstra('
SELECT gid AS id,
      source,
      target,
      cost,
      reverse_cost
FROM roads_ams_2008',
19354, 12215, false)
) s
LEFT JOIN roads_ams_2008 b
ON (b.gid = s.edge)
```

As we did earlier, the query above joins the routing result table and the network table. By matching edge IDs (**edge** from the routing result table and **gid** from the network table), we select the network edges that are contained in the route, and use their geometries to visualize the route.



Find the closest fire station for each destination, considering obstacles

Finally, for each of 5 given destinations, we will use the road network updated with new cost to find the closest fire station, and calculate the *shortest safe* route from the fire station to the destinations, avoiding the obstacles. In this case, the routes provided for fire trucks should look as follows.



You should figure out how to do this on your own for other fire stations and other destinations, based on what you have learned so far and based on modifying the SQL queries.

References:

- 1) pgRouting Project <https://pgrouting.org/>
- 2) A Beginner's Guide to pgRouting <https://anitagraser.com/2011/02/07/a-beginners-guide-to-pgrouting/>
- 3) PGrouting PostGIS Postgresql – No need for Google routing <https://www.igismap.com/pgrouting-postgis/>
- 4) pgRouting Manual (2.2) <https://docs.pgrouting.org/2.2/en/doc/src/tutorial/tutorial.html>

Assignment: routing for an ambulance in the presence of obstacles

In this assignment, you do a similar kind of analysis and navigate *an ambulance* from its station in Amsterdam to the same 5 destinations (as used in this Lab). For your convenience, you may use the dataset *ambulance stations* (ambulanceposten) we provided in the zipped data file or download the dataset *ambulance stations* (ambulanceposten) of the Netherlands via the following link (using ArcGIS to export the data) <https://shorturl.at/qvzf4>. To help select an ambulance station in Amsterdam, you may visualize the dataset by importing the shp data into QGIS and use OBJECTID to locate an ambulance station in Amsterdam. You can reuse the AMS street network dataset and adapt your codes if necessary. We assume that the status of the street network will not change significantly in the next 30 mins. So you can reuse the dataset of the obstacles as well. In your submitted document, please include the following: 1). A screenshot of the used datasets (Amsterdam area), and please indicate which ambulance station in the city of Amsterdam is selected (ID, coordinate, address, etc) ; 2). Screenshots of the safe routes from the selected ambulance station to the five given destinations (avoiding the obstacles). 3). A table with the details of the safe routes (e.g., travel distance).