# Lab 6
# pgrouting

The installation component of this lab refers to windows. If you have a different system please refer to the links in the Webcourses folder.

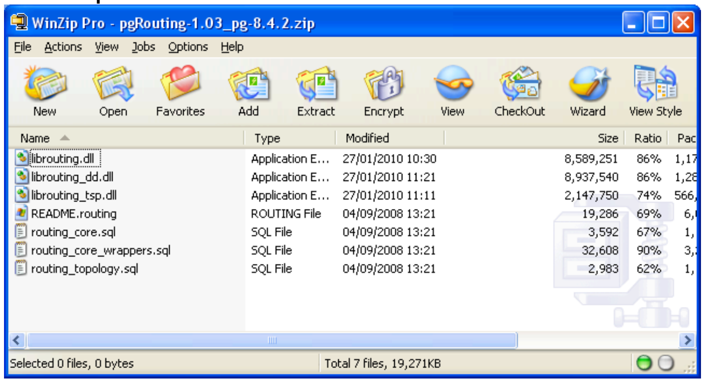
For example, on mac, the easiest way to get pgrouting is through brew install. If you are on El Capitan you might need to change permissions of brew by typing the following into a terminal:

```
sudo chown root /usr/local/bin/brew
```

## 1)Install pgRouting

**<u>Step 1</u>**: Copy and unzip.

- Download pgrouting from one of the links provided, you will receive a zip file on windows
- The Zip contains source SQL files and Windows DLL files

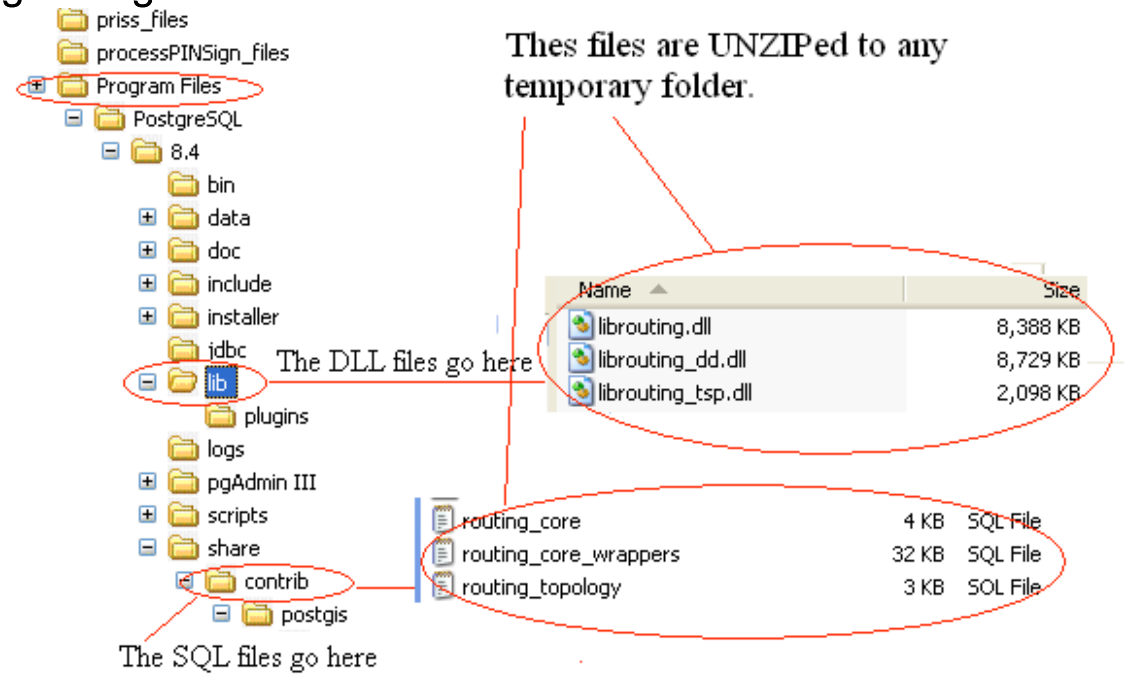

WinZip Pro - pgRouting-1.03_pg-8.4.2.zip
File Actions View Jobs Options Help
New Open Favorites Add Extract Encrypt View CheckOut Wizard View Style

| Name | Type | Modified | Size | Ratio | Pac |
|---|---|---|---|---|---|
| librouting.dll | Application E... | 27/01/2010 10:30 | 8,589,251 | 86% | 1,17 |
| librouting_dd.dll | Application E... | 27/01/2010 11:21 | 8,937,540 | 86% | 1,28 |
| librouting_tsp.dll | Application E... | 27/01/2010 11:11 | 2,147,750 | 74% | 566, |
| README.routing | ROUTING File | 04/09/2008 13:21 | 19,286 | 69% | 6, |
| routing_core.sql | SQL File | 04/09/2008 13:21 | 3,592 | 67% | 1, |
| routing_core_wrappers.sql | SQL File | 04/09/2008 13:21 | 32,608 | 90% | 3, |
| routing_topology.sql | SQL File | 04/09/2008 13:21 | 2,983 | 62% | 1, |

Selected 0 files, 0 bytes    Total 7 files, 19,271KB

- 
- Use Windows Extract All command (or unzip) to copy all the files a temporary folder (say pgTemp).
- The extraction process will make two sub-folders called “lib” and “share”
- You should copy the DLLs and SQL files provided in these folders the correct location under your PostgreSQL installation, which should be:
- Copy the DLL files to C:\Program Files (x86)\PostgreSQL\8.4\lib
- Copy the SQL files to C:\Program Files (x86)\PostgreSQL\8.4\share\contrib
- These folders are shown below:

# Lab 6
# pgrouting



| Name | Size |
|---|---|
| librouting.dll | 8,388 KB |
| librouting_dd.dll | 8,729 KB |
| librouting_tsp.dll | 2,098 KB |

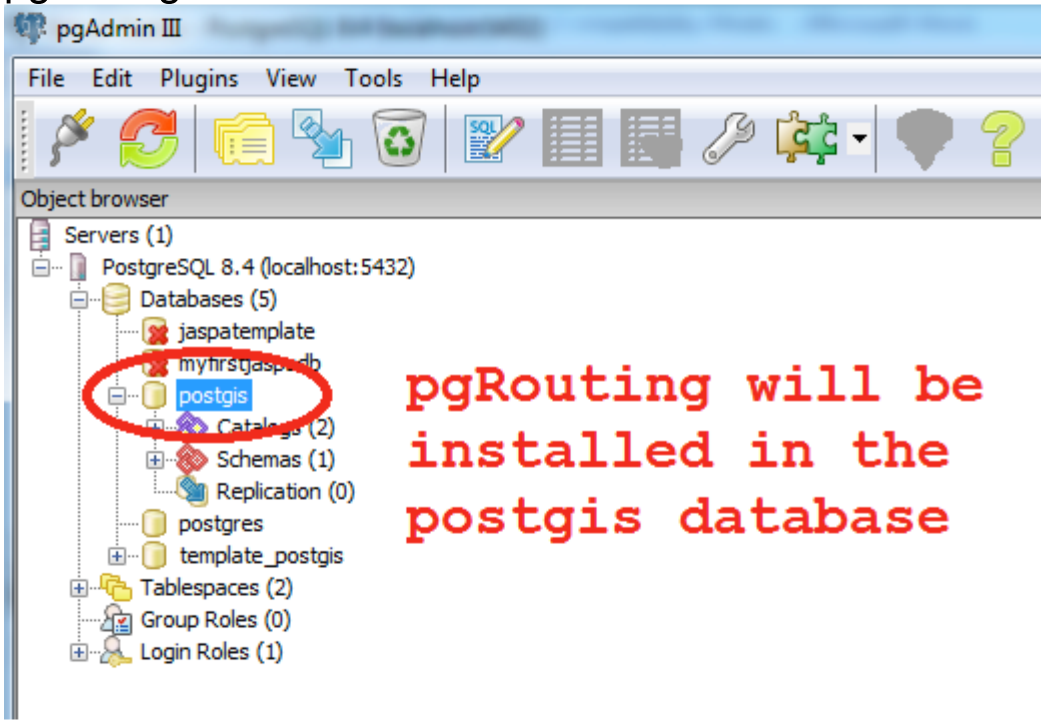
| | | |
|---|---|---|
| routing_core | 4 KB | SQL File |
| routing_core_wrappers | 32 KB | SQL File |
| routing_topology | 3 KB | SQL File |

Start  pgAdmin 4, a graphical tool for administering PostgreSQL databases. This tool is accessible from the Windows Start | All Programs Menu and is necessary for this tutorial.

**<u>Step 2</u>**: Navigate to the postgis database in  pgAdmin 4.

pgAdmin should look something like this:

Lab 6
pgrouting



**Step 3**: Add the core pgRouting functionality to the newly created database.

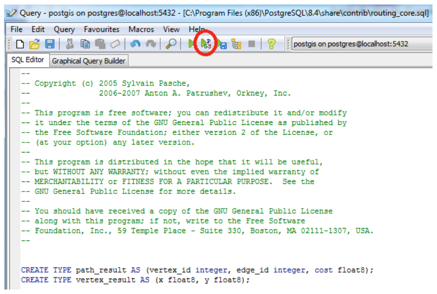
- The postgis  database should be selected in the object browser.

- Look at the top toolbar in pgAdmin III. There is a SQL query tool. Click on this tool to open it, or click “Tools->Query Tool” from the application menu.
- In the SQL query tool window, click “File->Open” and select

 “C:\Program Files\PostgreSQL\8.4\share\contrib\routing_core.sql”

 (the directory structure may differ depending on where you installed your software).

- To execute this file, click the green “play” button or navigate the application menu by clicking “Query->Execute”.


Query - postgis on postgres@localhost:5432 - [C:\Program Files (x86)\PostgreSQL\8.4\share\contrib\routing_core.sql]
File Edit Query Favourites Macros View Help
postgis on postgres@localhost:5432
SQL Editor
Graphical Query Builder

```
--
-- Copyright (c) 2005 Sylvain Pasche,
--               2006-2007 Anton A. Patrushev, Orkney, Inc.
--
-- This program is free software; you can redistribute it and/or modify
-- it under the terms of the GNU General Public License as published by
-- the Free Software Foundation; either version 2 of the License, or
-- (at your option) any later version.
--
-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.
--
-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
--


CREATE TYPE path_result AS (vertex_id integer, edge_id integer, cost float8);
CREATE TYPE vertex_result AS (x float8, y float8);
```

- Repeat the same process for the following files

  "C:\Program Files\PostgreSQL\8.4\share\contrib\routing_core_wrappers.sql".

  "C:\Program Files\PostgreSQL\8.4\share\contrib\routing_topology.sql".

- Now the routing functionality is available in the "postgis" database.

**Step 5**: Import the edges.sql file to the database as a new table at the command prompt.

- cd C:\Program Files (x86)\PostgreSQL\8.4\bin

- Then type the following at the command prompt:

  ```
  psql -d postgis -U postgres  -f  edges.sql
  ```

**Step 7**: Prepare the new table for Dijkstra by adding source, target, and length columns. In this example "length" will be the cost of the edges. This step is required because as it stands the edges table does not contain topology (or connectivity information).

- In the SQL shell, enter the following code and execute:

  ```
  \c postgis

  ALTER TABLE edges ADD COLUMN source integer;

  ALTER TABLE edges ADD COLUMN target integer;

  ALTER TABLE edges ADD COLUMN length double precision;
  ```

**Step 8**: Create the network topology in the "edges" table. This SQL command also populates the "length" field which is to be the edge cost in the network topology.

- In the SQL query tool, enter the following code and execute:

  ```
  SELECT assign_vertex_id('edges', 0.001, 'the_geom', 'gid');
  ```

```
     UPDATE edges SET length = ST_length(the_geom);
```

**Step 9**: Create indices for source, target, and geometry columns in the "edges" table.

- In the SQL shell, enter the following code and execute:

```
     CREATE INDEX source_idx ON edges(source);

     CREATE INDEX target_idx ON edges(target);

     CREATE INDEX geom_idx ON edges USING GIST(the_geom
GIST_GEOMETRY_OPS);
```

**Step 10**: Perform the routing operation and store the results in a "dijkstra_result" table.

- In the SQL shell, enter the following code and execute:

```
     DROP TABLE IF EXISTS dijsktra_result;

     CREATE TABLE dijsktra_result(gid int4) with oids;

     SELECT AddGeometryColumn('dijsktra_result', 'the_geom',
'2276', 'MULTILINESTRING', 2);

     INSERT INTO dijsktra_result(the_geom)

     SELECT the_geom FROM dijkstra_sp('edges', 52, 35);
```

- Now the result of the shortest path from node 52 to node 35 can be viewed in QGis by connecting to the "postgis" database and adding the "dijkstra_result" table to QGIS using the query:

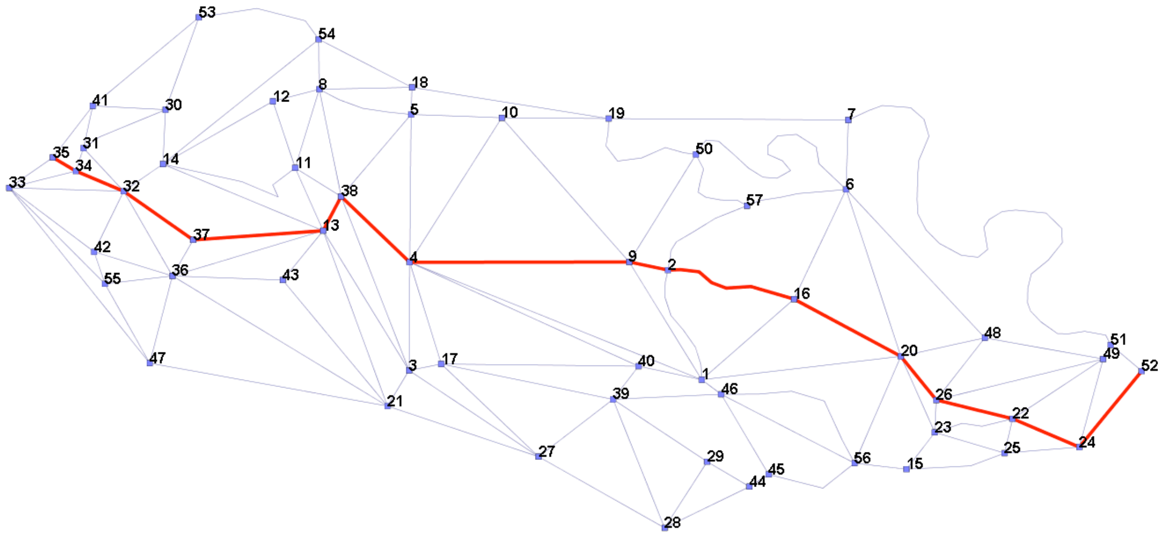
```
  SELECT source,target,length(d.the_geom) FROM dijsktra_result
  AS d, edges;
```

- Below is an example of the results (start node: 52, end node: 35).

**Step 6**: Viewing edges in QGIS.

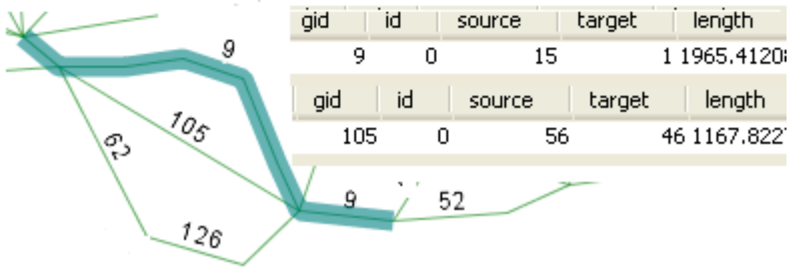
With the edges table as background you should see the following:

# Lab 6
# pgrouting



What is happening here?

```
CREATE VIEW sp1 AS (
SELECT * FROM shortest_path('
SELECT gid as id,
source::integer,
target::integer,
length::double precision as cost
FROM edges',
1, 15, false, false));
```



| gid | id | source | target | length |
|---|---|---|---|---|
| 9 | 0 | 15 | 1 | 1965.4120 |

| gid | id | source | target | length |
|---|---|---|---|---|
| 105 | 0 | 56 | 46 | 1167.822 |

**Examples of route finding using Irish road data.**
Here we look in more detail at the topology. As in the above edges example, we must add the appropriate columns and create topology.
We must have the road data loaded into the postgis database.
At the command prompt change directory:
cd C:\Program Files\PostgreSQL\8.4\bin

Use the following command at the command prompt in:

```
psql -d postgis -U postgres  -f roads.sql
```

# Lab 6
# pgrouting

Now login into PostgreSQL  and connect to postgis.

```
\c postgis

ALTER TABLE roads ADD COLUMN source integer;

ALTER TABLE roads ADD COLUMN target integer;

   -- a column called row already exists in roads.

   -- but we would like to calculate the distances

   -- so we rename the old length and make a new length

ALTER TABLE roads RENAME COLUMN length TO lengthold;

ALTER TABLE roads ADD COLUMN length double precision;
```

In the SQL shell or pgAdmin, enter the following two commands. The first SQL command creates the network topology in the “roads” table. The second SQL command populates the “length” field which will be used as the edge cost in the network topology.

```
SELECT assign_vertex_id('roads', 0.001, 'the_geom', 'gid');

UPDATE roads SET length = ST_length(the_geom);
```

Check the lengths have been calculated
```
select length,ST_length(the_geom) from roads;
```

We can see roads contain MULTILINESTRING data:
```
select name,type,length,ST_LENGTH(the_geom)/2.0,as_text(the_geom)
from roads;
```

Lab 6

pgrouting

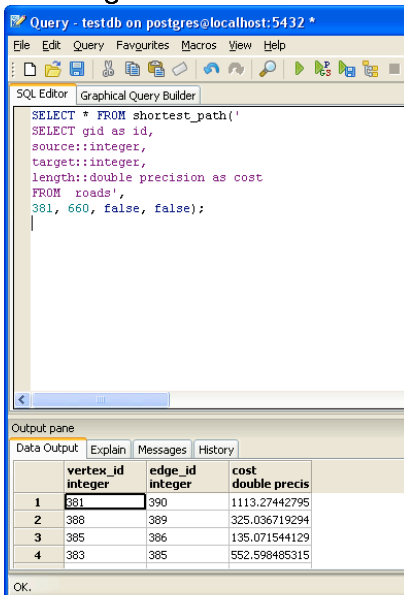
Note the actual vertex ids may differ on your map so you should check visually for the appropriate numbers, before running the following query.
To view the shortest path from Dublin (381) to Waterford (660) in text format, issue the following command in SQL shell:

```
SELECT * FROM shortest_path('
SELECT gid as id,
source::integer,
target::integer,
length::double precision as cost
FROM  roads',
381, 660, false, false);
```

The output consists of vertices, edges, and distances (cost)

```
 vertex_id | edge_id |       cost
-----------+---------+------------------
       381 |     390 | 1113.27442795795
       388 |     389 | 325.036719294703
       385 |     386 | 135.071544129141
       383 |     385 | 552.598485315868
       384 |     387 | 84.8649903547372
```

Or in PgAdmin


Query - testdb on postgres@localhost:5432 *
File Edit Query Favourites Macros View Help
SQL Editor
Graphical Query Builder
Output pane
Data Output
Explain
Messages
History
OK.

```
SELECT * FROM shortest_path('
SELECT gid as id,
source::integer,
target::integer,
length::double precision as cost
FROM  roads',
381, 660, false, false);
```

| | vertex_id integer | edge_id integer | cost double precis |
|---|---|---|---|
| 1 | 381 | 390 | 1113.27442795 |
| 2 | 388 | 389 | 325.036719294 |
| 3 | 385 | 386 | 135.071544129 |
| 4 | 383 | 385 | 552.598485315 |

**Some other Irish locations to query:**

================================

381 = Source in Dublin

71  = source in Cork

# Lab 6
# pgrouting

48  = source in Galway
550 = Source in Limerick
660 = source in Waterford
Note the actual vertex ids may differ on your map so you should check visually in OpenJump. Get the appropriate numbers, before running the queries.

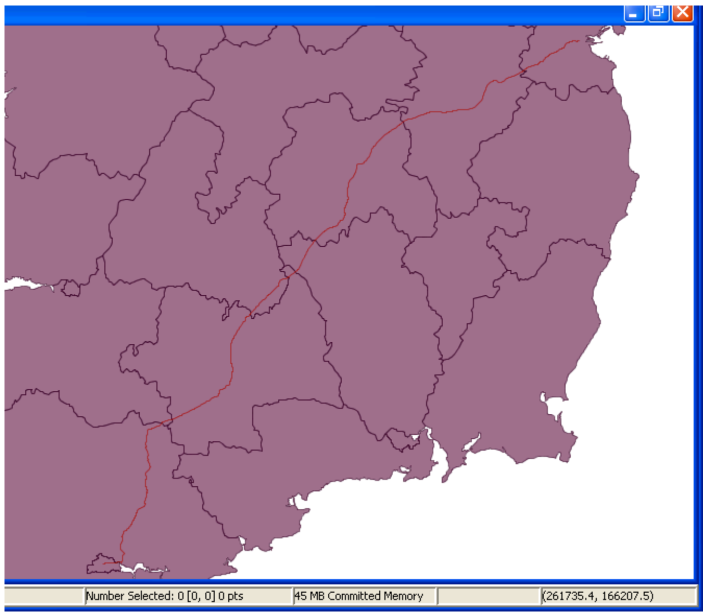
**How to graphically view a route.**
To view the route from Dublin to Cork in QGIS:
First create a view in SQL shell then the view is queries in QGIS

```
CREATE VIEW dublin_cork AS (
SELECT * FROM shortest_path('
SELECT gid as id,
source::integer,
target::integer,
length::double precision as cost
FROM roads',
381, 71, false, false));
```

Then run the following query in QGIS

```
select r.the_geom  from dublin_cork dc, roads r where r.gid =
dc.edge_id;
```

Which should give:



# Lab 6
## pgrouting

Run in SQL shell.

```
CREATE VIEW dublin_wexford AS (
SELECT * FROM shortest_path('
SELECT gid as id,
source::integer,
target::integer,
length::double precision as cost
FROM roads',
381, 660, false, false));
```

Then run the following query in QGIS

```
select r.the_geom from dublin_wexford dw, roads r where r.gid =
dw.edge_id;
```

Run in SQL shell.

```
CREATE VIEW dublin_galway AS (
SELECT * FROM shortest_path('
SELECT gid as id,
source::integer,
target::integer,
length::double precision as cost
FROM roads',
381, 48, false, false));
```

Then run the following query in QGIS

```
Select r.the_geom from dublin_galway dg, roads r where r.gid =
dg.edge_id;
```

Waterford to Galway

```
CREATE VIEW wat_gal AS (
SELECT * FROM shortest_path('
SELECT gid as id,
source::integer,
target::integer,
length::double precision as cost
FROM roads',
660, 48, false, false));

SELECT r.the_geom  FROM wat_gal dc, roads r WHERE r.gid =
dc.edge_id;
```

# Lab 6
# pgrouting

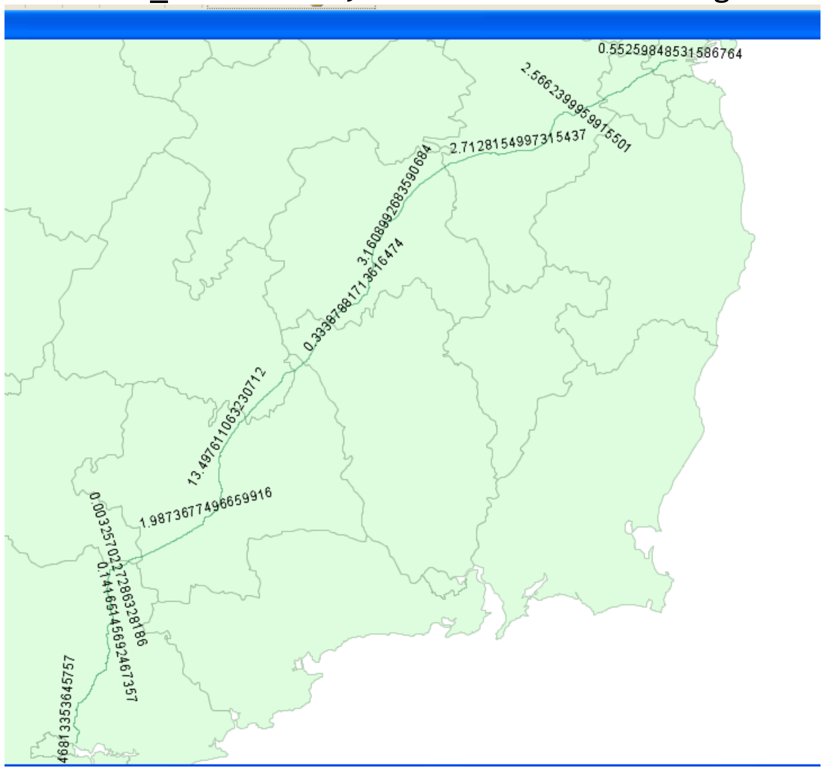
We can make the length column different depending on road type

```
UPDATE roads SET length =
CASE
 WHEN type like '%Motorway%' THEN
 ST_LENGTH(the_geom)/2.0
 WHEN type like '%Dual%' THEN
 ST_LENGTH(the_geom)/1.5
 ELSE
 ST_LENGTH(the_geom)
END;
```

**Measuring the length of a path**

The first example only computes the lengths of individual sections of the path.

```
Select geomunion(r.the_geom,r.the_geom),
ST_length(geomunion(r.the_geom,r.the_geom))/1000 as dist
from dublin_cork as dc, roads as r where r.gid = dc.edge_id;
```



Use the following to compute the <u>length</u> of the shortest path from Dublin to Cork.
Note, the Dublin_Cork view must exist.

```
select MemGeomUnion(r.the_geom),
st_length(MemGeomUnion(r.the_geom))/1000.0 as dist
from dublin_cork as dc, roads as r where r.gid = dc.edge_id;
```

# Lab 6
# pgrouting

The straight lines distance is about 215-220 Kilometres

**Using two versions of Open Street Map data (OSM).**
Here we will use the tables called dubroads2 and dublin_highway1. Look at the geometry and the attributes of each table in QGIS. You must add the source, target and length columns. Then add topology using `assign_vertex_id` . Finally you must compute the lengths. All these steps are described above.

**Compare dubroads2 with dublin_highway1.**

```
SELECT
gid,ST_GeometryType(the_geom),ST_NumPoints(the_geom),ST_NumGeometries(the_geom),as_text(startpoint(the_geom)),
as_text(endpoint(the_geom)) FROM dublin_highway1  where
ST_NumGeometries(the_geom)>1;
(87 rows)
SELECT
gid,ST_GeometryType(the_geom),source,target,ST_NumPoints(the_geom),ST_NumGeometries(the_geom),as_text(startpoint(the_geom)),
as_text(endpoint(the_geom)) FROM dubroads2 where
ST_NumGeometries(the_geom)>1;
(0 rows)
```

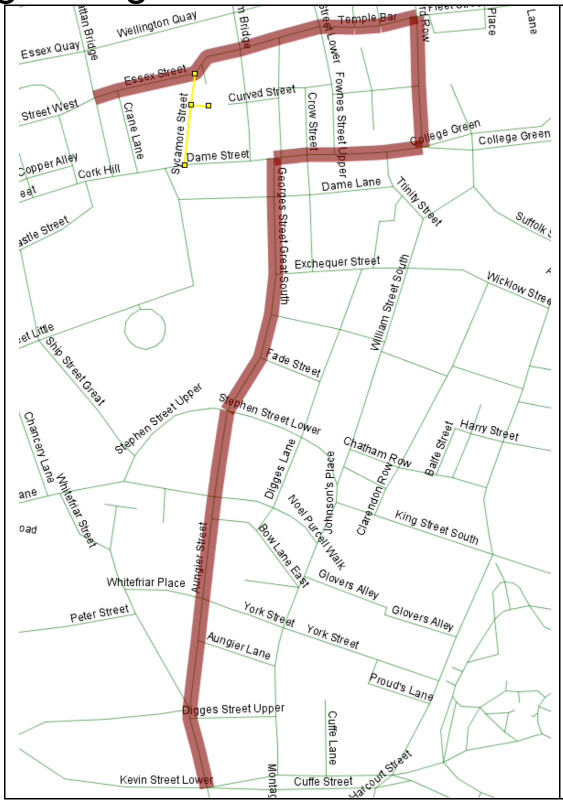
Make a view for route 1.

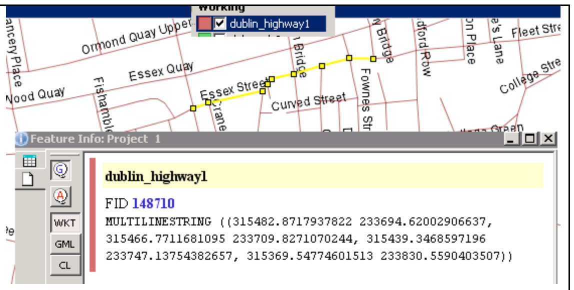
```
CREATE VIEW  r1 AS (
SELECT * FROM shortest_path('
SELECT gid as id,
source::integer,
target::integer,
length::double precision as cost
FROM dublin_highway1',
13287,13320, false, false));
```

```
SELECT r.the_geom
FROM YourStudentNo. r1 as c1, dublin_highway1 r
WHERE r.gid = c1.edge_id;
```

Using dublin_highway1 gives the following route in QGIS. Check out why this route differs from dubroads2.  Here each row contains a named road containing several sections.

Lab 6
pgrouting





dublin_highway1

FID 148710

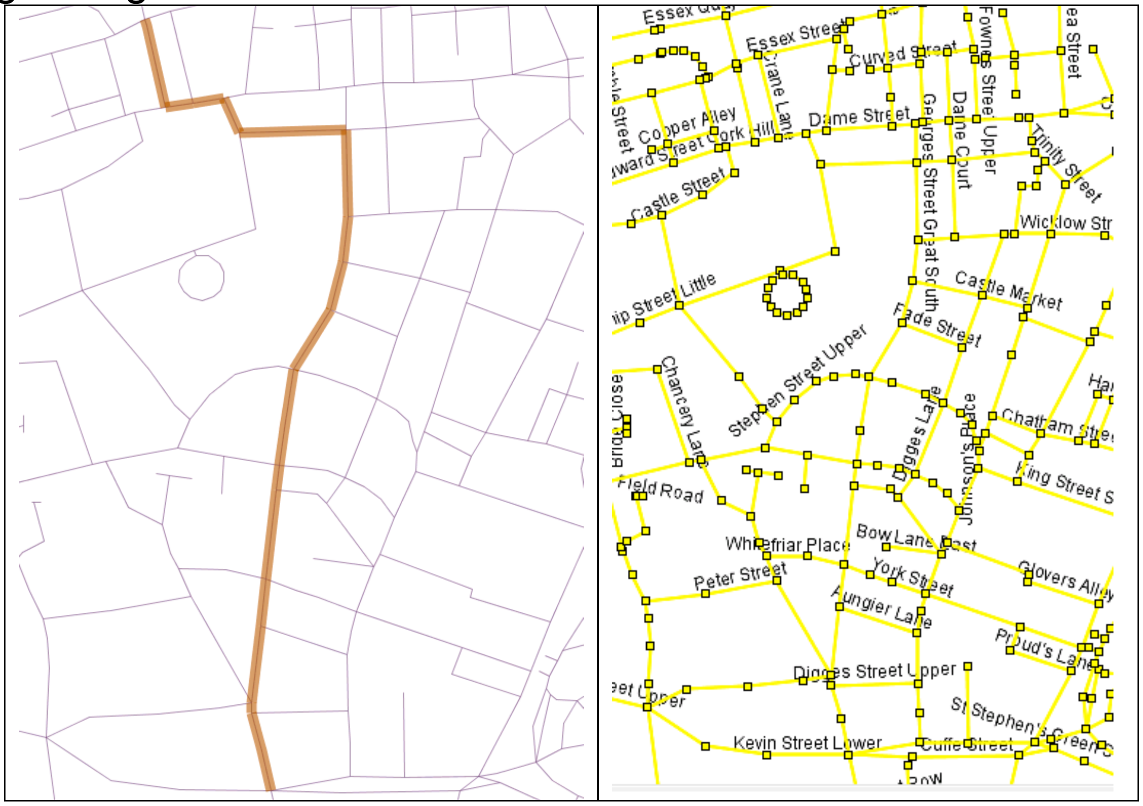
```
MULTILINESTRING ((315482.8717937822 233694.62002906637,
315466.7711681095 233709.8271070244, 315439.3468597196
233747.13754382657, 315369.54774601513 233830.5590403507))
```

```
CREATE VIEW YourStudentNo.r2 AS (
SELECT * FROM shortest_path('
SELECT gid as id,
source::integer,
target::integer,
length::double precision as cost
FROM dubroads2',
13243,13294, false, false));
```

```
SELECT r.the_geom
FROM YourStudentNo.r2 as c1, dubroads2 r
WHERE r.gid = c1.edge_id;
```

Using dubroads2 gives the following route in QGIS. Here each row contains one road segment with a source and target. Hence, every road segment has a source and target.

Lab 6
pgrouting



## Why are the routes different?

**Understanding the topology**
We look at the relationship between the edges, sources, and targets.

```
select target from roads where source = 381;
```

Gives: 382,388, 375
Then if we check each of these we find that their source includes 381
select source  from roads where target = 382;

```
 source
--------
    381
    389
    390
(3 rows)
```

# Lab 6
# pgrouting

```
select source  from roads where target = 388;
 source
--------
   381
(1 row)
```

```
 select source  from roads where target = 375;
 source
--------
   381
(1 row)
```

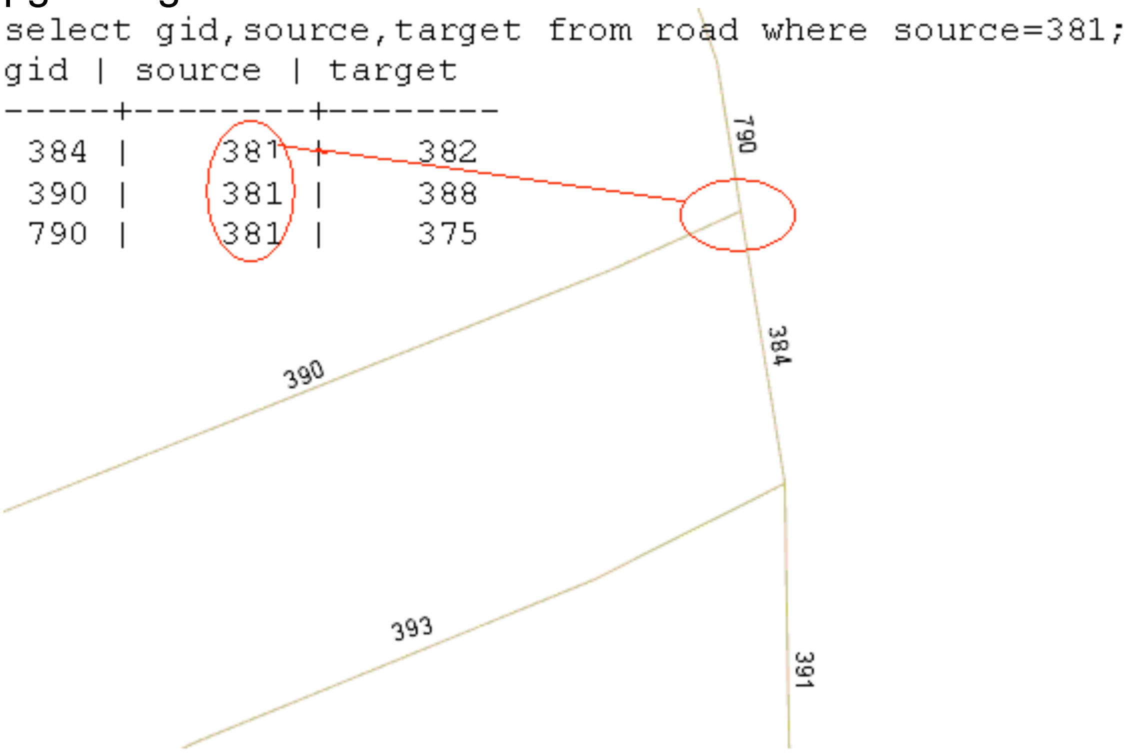
The following is a graphical representation of how edges (gids), sources and targets are represented.

```
select gid,source,target from road where source=381;
gid | source | target
-----+--------+--------
 384 |   381 |   382
 390 |   381 |   388
 790 |   381 |   375
(3 rows)
```

**Labelled with Edges (gid):**

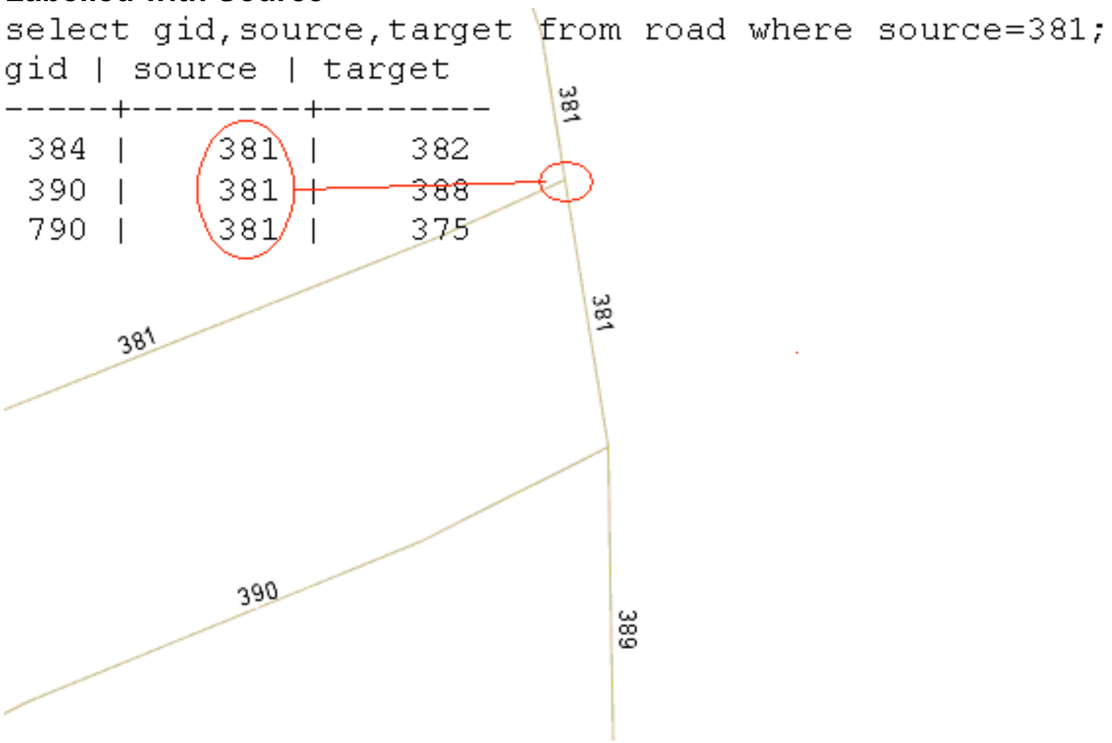
Lab 6

pgrouting



```
select gid,source,target from road where source=381;
gid | source | target
-----+--------+--------
 384 |    381 |    382
 390 |    381 |    388
 790 |    381 |    375
```

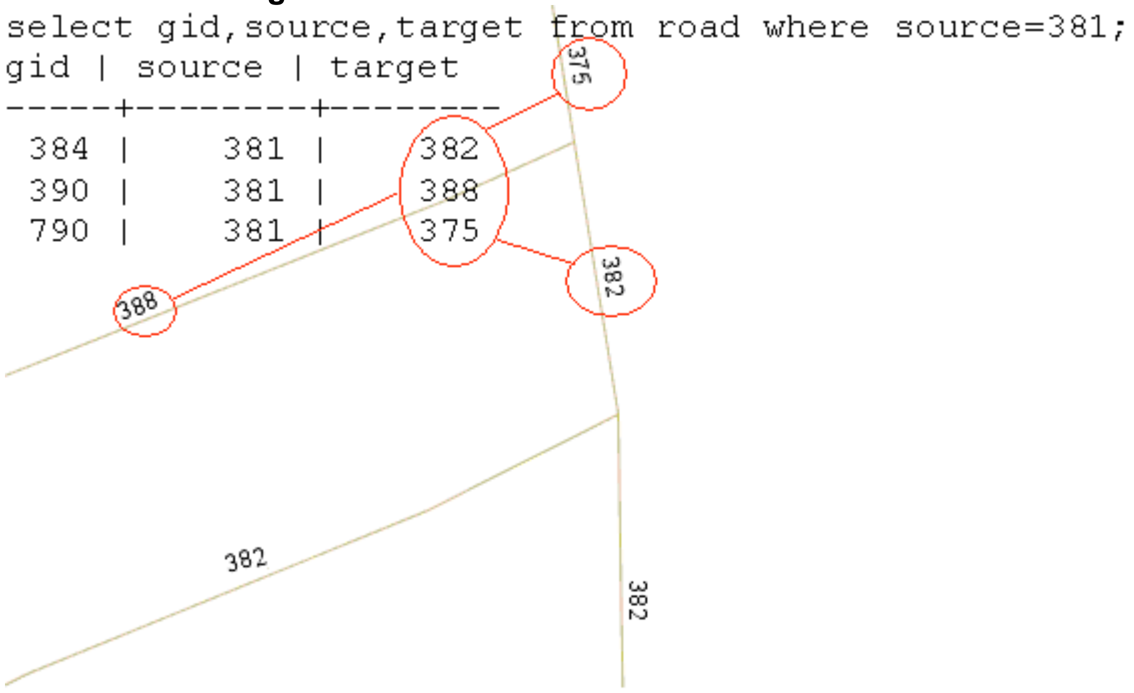
**Labelled with Source**



```
select gid,source,target from road where source=381;
gid | source | target
-----+--------+--------
 384 |    381 |    382
 390 |    381 |    388
 790 |    381 |    375
```

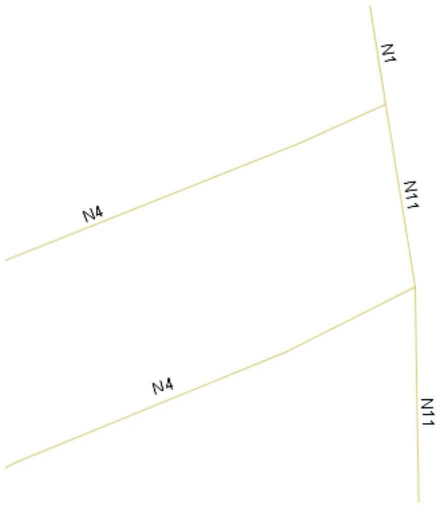
Lab 6
pgrouting

**Labelled with Target**

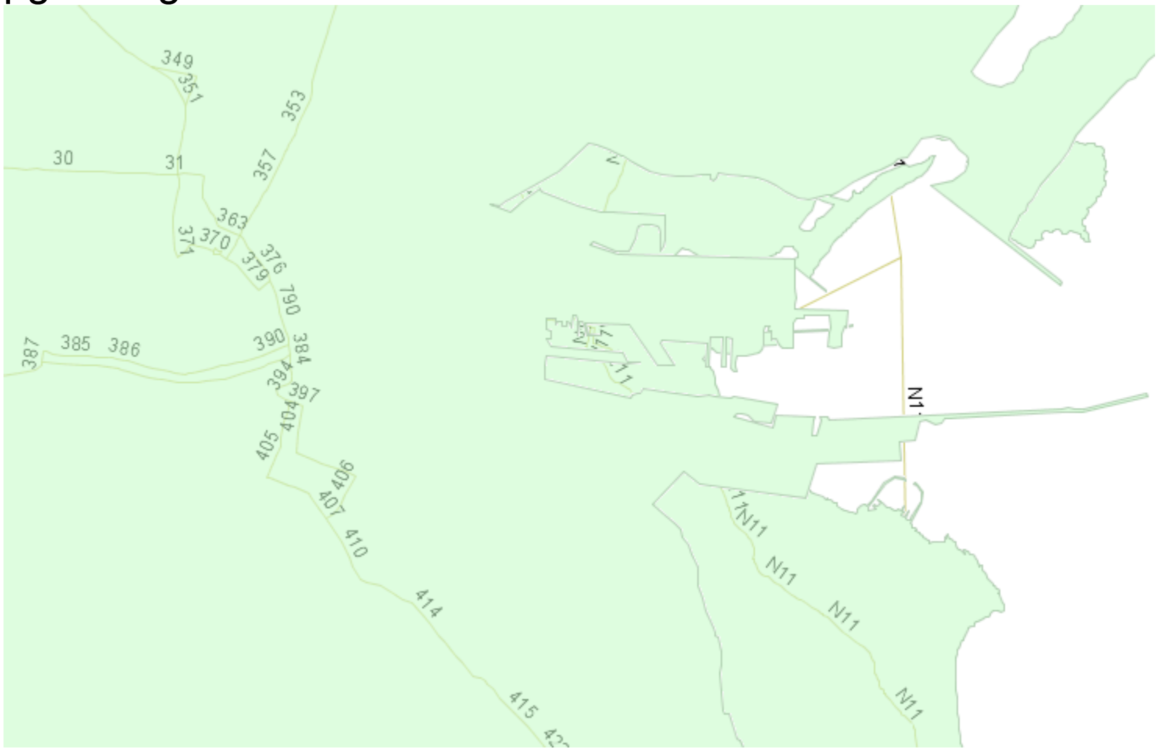


```
select gid,source,target from road where source=381;
gid | source | target
-----+--------+--------
 384 |    381 |    382
 390 |    381 |    388
 790 |    381 |    375
```

Road classification National Primary (N1,N11, N4)



# Lab 6
# pgrouting



## pgRouting for Postgres

This section may be of use to Mac users or students who already have Postges installed. Routing is broadly the same for Postgres, but with slightly different syntax.

```
-- If you have Postgis already installed then execute the following at the SQL-Shell or pgAdmin
CREATE EXTENSION pgrouting;
SELECT pgr_version();

-- Add networking information
ALTER TABLE edges ADD COLUMN source integer;
ALTER TABLE edges ADD COLUMN target integer;
ALTER TABLE edges ADD COLUMN length double precision;
UPDATE edges SET length = st_length(the_geom);
SELECT source,target,length FROM edges;

-- Create topology
SELECT pgr_createTopology('edges', 0.00001, 'the_geom', 'gid');

-- Create route
SELECT pgr_dijkstra('SELECT gid AS id,
                        source::integer,
                        target::integer,
                        length::double precision AS cost
                       FROM public.edges', 52, 35);

SELECT seq, id1 AS node, id2 AS edge, cost FROM pgr_dijkstra('
                SELECT gid AS id,
                        source::integer,
                        target::integer,
                        length::double precision AS cost
                       FROM edges',
                52, 35, false, false);
```

# Lab 6
# pgrouting

```
CREATE VIEW route AS (
SELECT * FROM pgr_dijkstra('
                SELECT gid AS id,
                         source::integer,
                         target::integer,
                         length::double precision AS cost
                        FROM edges',
                52, 35, false, false));


select st_astext(e.the_geom)  from route r, edges e where e.gid = r.id2;
select e.the_geom  from route r, edges e where e.gid = r.id2;
```

-- see
-- http://workshop.pgrouting.org/chapters/topology.html
-- http://workshop.pgrouting.org/chapters/shortest_path.html
-- http://docs.pgrouting.org/dev/doc/index.html