Databases Practical – Week 5

Table of Contents

In this practical session we will be running a number of experiments to test the impact of indexing and de-normalisation on non-spatial and spatial query performance.

However .. we may not see much improvement in query performance as the database server is quite powerful. This will depend on the number of active users when you are running your queries!

Note: For this practical you need to have the entire Open Street Map dataset (points of interest and road networks) imported into your personal database schema. You should also have all the England County boundaries ***('England_Counties' or 'united_kingdom_counties')***. ***If they are not already loaded, all of these datasets should first be loaded into your database using the SPIT tool (QGIS 2.08) or if you are running QGIS 2.18 then follow the data loading instructions posted on the forum.*** **If, on running the queries below, you find that they take too much time, you should revert to the use of the 'subset' datasets and adapt your queries accordingly.**

Note*:* You may also need to fix some issues with some of the queries listed below, due to slight variations in the datasets. For example, in the united_kingdom_highways dataset, you may find that the *name* column is lower case and that "NAME" does not work in any queries. This may differ between the 'subset' datasets and the full datasets as the latter were imported using FME and the former using Quantum GIS.

Part 1 – The Importance of Indexing Non-Spatial Data

Indexing improves database performance by providing a short-cut directly to specific rows of data. It is therefore particularly important for large datasets. We shall be using the Open Street Map United Kingdom Highway data for this exercise, as it contains a total of 1.4 million records.

1. If they are not already loaded into your database, import the full united_kingdom_highway and united_kingdom_poi datasets using the QGIS SPIT tool (2.08) or the instructions from the forum (QGIS 2.18) and SRID 4326.

2. Now make sure that there are no indexes created on the united_kingdom_highway, united_kingdom_poi data and England_Counties data. You can do this by clicking on the table in PGAdmin III and then drilling down to the 'indexes' option. If there are any indexes, right-click on them and 'DROP' them.

3. Once all the indexes are removed, in the *pgAdmin* tool, select *TOOLS > QUERY TOOL* from the menu.

4. First, find out the number of records in your dataset by typing the following into the SQL Editor box:

   ```
   SELECT COUNT(*) FROM public.united_kingdom_highway
   ```

5. Press the *F5* key to run the query (or use the *QUERY* > *EXECUTE* option from the menu). You will see the number of roads

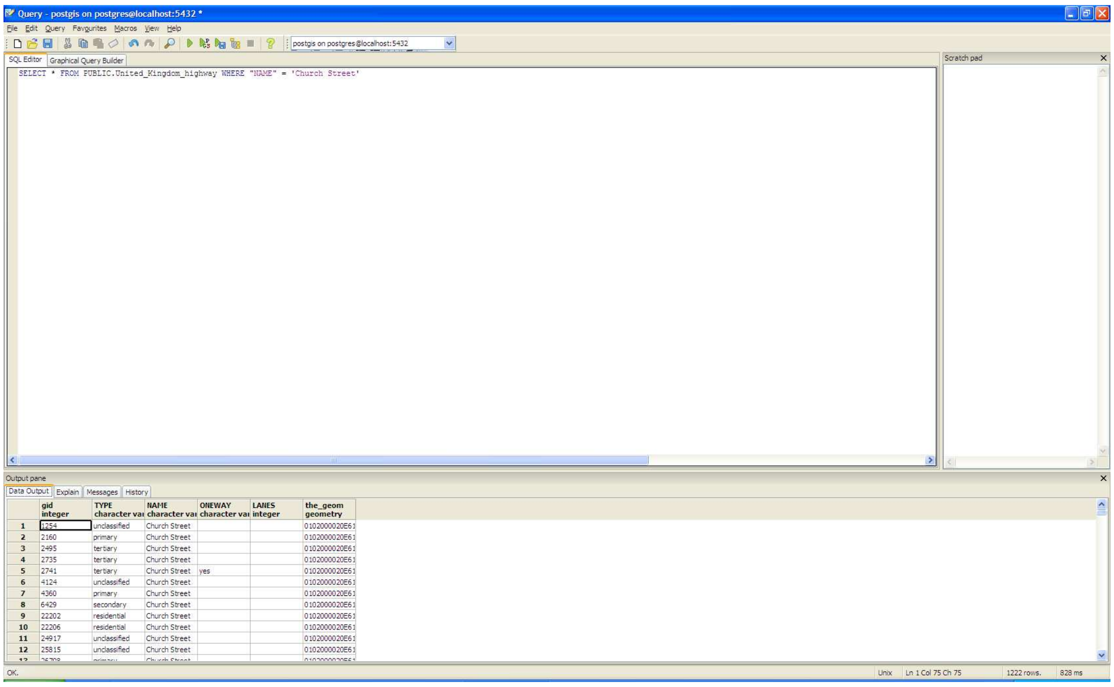
6. Now type the following text into the *SQL Editor* box at the top.

   ```
   SELECT * FROM PUBLIC.united_kingdom_highway
   WHERE name = 'Church Street'
   ```

Note that PostgreSQL sometimes requires the names of the fields (columns) to be surrounded by double quotation marks (when the names are not lowercase), and always requires any text to be surrounded by single quotation marks.

7. Press the *F5* key to run the query (or use the *QUERY* > *EXECUTE* option from the menu).

8. You will see results as shown below.

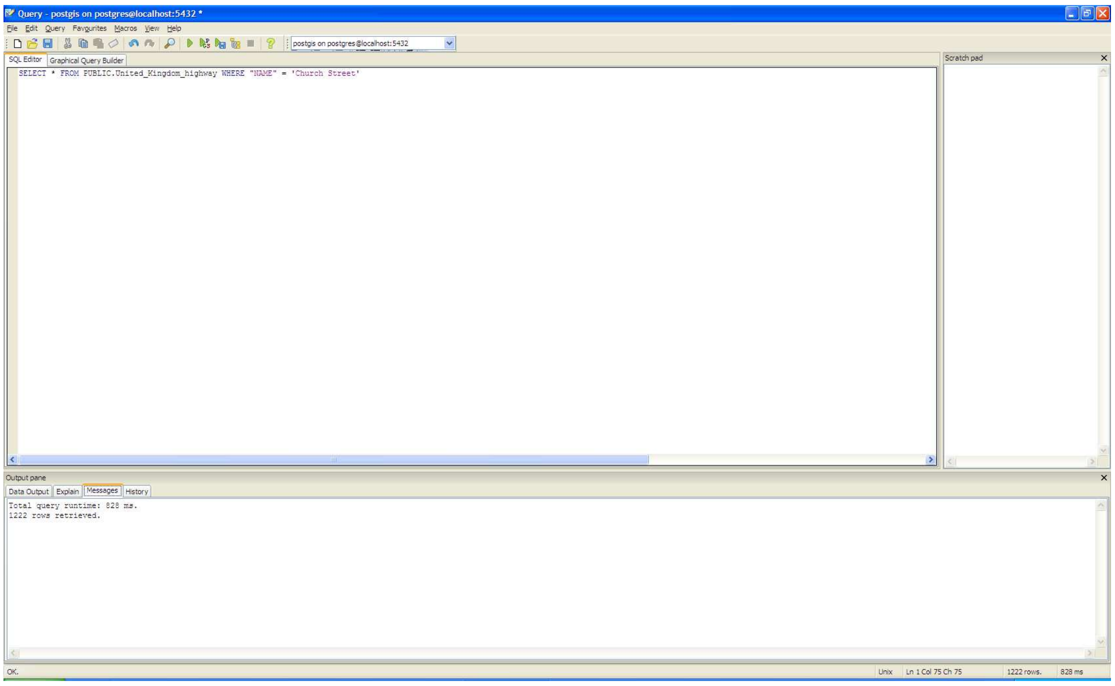

Query - postgis on postgres@localhost:5432 *
File Edit Query Favourites Macros View Help
postgis on postgres@localhost:5432
SQL Editor
Graphical Query Builder
Scratch pad
Output pane
Data Output
Explain
Messages
History

```
SELECT * FROM PUBLIC.United_Kingdom_highway WHERE "NAME" = 'Church Street'
```

	gid integer	TYPE character vai	NAME character vai	ONEWAY character vai	LANES integer	the_geom geometry
1	1254	unclassified	Church Street			0102000020E61
2	2160	primary	Church Street			0102000020E61
3	2495	tertiary	Church Street			0102000020E61
4	2735	tertiary	Church Street			0102000020E61
5	2741	tertiary	Church Street	yes		0102000020E61
6	4124	unclassified	Church Street			0102000020E61
7	4360	primary	Church Street			0102000020E61
8	6429	secondary	Church Street			0102000020E61
9	22202	residential	Church Street			0102000020E61
10	22206	residential	Church Street			0102000020E61
11	24917	unclassified	Church Street			0102000020E61
12	25815	unclassified	Church Street			0102000020E61

```
OK.    Unix    Ln 1 Col 75 Ch 75    1222 rows.    828 ms
```

9. Now click on the *Messages* tab. You will see a message stating how long the query took to execute and also giving the number of rows returned.



```
SELECT * FROM PUBLIC.United_Kingdom_highway WHERE "NAME" = 'Church Street'
```

```
Total query runtime: 828 ms.
1222 rows retrieved.
```

10. Repeat the above process for a ‘GROUP BY’ query to see all the different street names in the data set:

    ```
    SELECT COUNT(*),  name from public.united_kingdom_highway
    GROUP BY name
    ```

11. Again, find out how long the query took to execute.

12. We will now create an index on the *NAME* field in the dataset, to improve search performance. From the *pgAdmin* tool, click TOOLS > QUERY TOOL to open another query window.

13. Now type in the following text to create the index

    ```
            create index united_kingdom_highway_name_idx
            on united_kingdom_highway(name);
    ```

14. Run the last two queries again. Once the results have been returned, view the *Messages* output to see if search performance has improved.

15. Run the following queries and time them.

    ```
    SELECT COUNT(*)
    FROM public.united_kingdom_highway
    WHERE type = 'path' OR type='track';
    ```

```
SELECT COUNT(*)
FROM public.united_kingdom_highway
WHERE type = 'path' OR type='track' or type = 'tertiary';

SELECT COUNT(*)
FROM public.united_kingdom_highway
WHERE type = 'secondary' AND name is not null AND oneway='true';

SELECT COUNT(*)
FROM public.united_kingdom_highway
WHERE type = 'secondary' AND name is not null AND (oneway='true' OR oneway =
'yes');
```

16. Now create an index on the ‘type’ column and re-run the queries. Have the results improved?

17. Run a query to find the number of points in each category in the united_kingdom_poi dataset. Make a note of the time taken to run the query.

18. Create an appropriate index for the above query and run the query again. Has the index made a difference on timing?

19. Repeat 17 and 18 to find the number of different names of points in the united_kingdom_poi dataset. Does the index make a difference?

20. Why do you think an index hasn’t made such a big different to the POI as it did for the highways (if the index did make a different at all)?

Part 2 – Using Spatial Indexes

For the second part of this practical exercise, we will test the impact of spatial indexes on spatial query performance. We will be using the united_kingdom_highways data, the united_kingdom_poi data and the England_Counties data[1]. (NB: If the queries take exceptionally long to execute, you may wish to use the _subset datasets you created in Weeks 3 and 4 of the course.)

1. Check that there are no spatial indexes on your data. You can do this in the PGAdmin III tool by clicking on the datasets and then drilling down to the 'indexes' option. If there are any spatial indexes, right-click on them and 'DROP' them. (Spatial indexes will mention "gist" in their operator class).

2. Run the following SQL statements and make notes of the time taken for each one (be patient, some of these make take some time). NB: To get an average result, run each query once, then re-run the entire list, and re-run the entire list again – at the end, you should have three time measurements for each query[2].

```
--- First make sure that the Counties table has the correct SRID
UPDATE public."England_Counties" SET geom=ST_SetSRID(geom,27700);

select UpdateGeometrySRID('public', 'England_Counties' , 'geom', 27700) ;


UPDATE public.united_kingdom_highway SET geom=ST_SetSRID(geom,4326);

select UpdateGeometrySRID('public', 'united_kingdom_highway' , 'geom', 4326) ;


UPDATE public.united_kingdom_poi SET geom=ST_SetSRID(geom,4326);

select UpdateGeometrySRID('public', 'united_kingdom_poi' , 'geom', 4326) ;


--- Find the Counties containing a High Street
SELECT distinct A.name
FROM public."England_Counties" A, public.united_kingdom_highway B
WHERE B.name = 'High Street'
and ST_INTERSECTS(A.geom, ST_TRANSFORM(B. geom,27700)) = 't'
```

[1] Change England_Counties to united_kingdom_counties if this is the name of your table
[2] If you get the error: Input geometry has unknown (0) SRID check the practical from Week 4 to set your Counties geometry to BNG (27700) and the highway geometry to WGS 84 (4236).

```
--- Find the nearest road to each Point of Interest[3]
select distinct on (b.name) b.name as closest_road_name, st_distance(b.geom, s.geom) as
distance,  s.name as poi_name, s.category as poi_category
from public.united_kingdom_highway b, public.united_kingdom_poi s
order by b.name, st_distance(b.geom, s.geom);


--- Find the total length of roads in the dataset
Select sum(st_length(geom)) from public.united_kingdom_highway;
--- Find the total length of roads within one county
SELECT sum(st_length(b.geom))
FROM public."England_Counties" a, public.united_kingdom_highway b
WHERE a.name = 'Cambridgeshire'
and ST_INTERSECTS(a.geom, ST_TRANSFORM(b. geom,27700)) = 't';
```

3. Now create the spatial indexes on the above tables, adapting the following example SQL:

```
CREATE INDEX spatial_table_highway_gidx
ON united_kingdom_highway
USING GIST(geom);
```

   You may wish to also create non-spatial indexes if there are specific fields used in the query that are not already indexed.

[3] Without indexes, this query is measuring the distance between 100,000 points of interest and nearly 1 million roads, so this will take some time. If the time is excessive (over 10 minutes), then modify the query as follows:

```
select distinct on (b.name) b.name as closest_road_name, st_distance(b.geom, s.geom) as
distance,  s.name as poi_name, s.category as poi_category
from public.united_kingdom_highway b, public.united_kingdom_poi s
where s.category = 'Automotive'
order by b.name, st_distance(b.geom, s.geom)
```

or use:

```
select distinct on (b.name) b.name as closest_road_name, st_distance(b.geom, s.geom) as
distance,  s.name as poi_name, s.category as poi_category
from public.united_kingdom_highway b, public.united_kingdom_poi s
where s.category = 'Automotive' and s.name = 'Fuel:M1 Service Area'
order by b.name, st_distance(b.geom, s.geom)
```

4. Re-run the queries above. Has the spatial index made a difference?

Part 3 – Querying Normalized versus Un-Normalized Data

For these queries, we will artificially create a non-normalised addition to the united_kingdom_highways table to contain the name of the County in which the highway lies. We will then compare the timing for three versions of a query to list all the highways in Cheshire.

NB: Some of these queries may take time to run – you can use the subset datasets to speed things up if necessary. You will need to change some of the queries as the column names are different and Cheshire and Cumbria are not included in the subset of the Counties.

1. Add a new column to the united_kingdom_highway table, called 'county_name'.

2. Add a second column to the united_kingdom_highway table, called 'county_id'.

3. Run the following queries to update the column to the correct value (note that two steps are required as there is a many:many relationship between highways and counties).

```
---	First create a join table for the many:many relationship
create table highway_counties as
select b.oid as county_id, a.oid as highway_id
from  uk_counties b, united_kingdom_highway a
where st_contains(b.geom,st_transform(a.geom, 27700));

--- Then take the first entry for each Highway and put it in the Highways table
--- Limit to 1 result, as some highways are in more than one county
update united_kingdom_highway a set county_id =
(select b.county_id
from  highway_counties b inner join united_kingdom_highway a
on a.oid = b.highway_id limit 1);
```

4. Adapt the above queries to set the county_id field to the OID of the containing County.

5. On paper, write three queries to list the street names in Cheshire:

   a. Option 1 – Using the data from one table
   b. Option 2 – Using a non-spatial join on the OID from England_Counties with the county_id field in the united_kingdom_highway. You will need to make use of the new interim table you created above for this.
   c. Option 3 – Using a spatial join between the two tables

6. Run the queries. Are the results as you expected?

7. Repeat the above exercise to find all the points of interest in Cumbria, using three different SQL queries

8. If you haven't already done so, repeat the above exercises using the subset datasets rather than the full datasets, selecting appropriate Counties to replace Cheshire and Cumbria (which are not in the subset dataset). Does using non-normalised data make such a difference on the smaller datasets?

9. The operations in Part 2 involved transforming the data into British National Grid. The final query in this exercise will test whether pre-transforming the data will improve query performance. This is a slightly different example of de-normalisation, as we will now have two copies of the geometry in the same table.

```
alter table united_kingdom_highway add geom_osgb (geometry);

update united_kingdom_highway set geom_osgb =
st_transform(geom,27700);

CREATE INDEX spatial_table_highway_gidx
ON united_kingdom_highway
USING GIST(geom_osgb);
```

Once the updates are run, re-run the above queries with and without the ST_Transform operations to see if pre-transforming the data makes a difference.