

# Postgre/PostGIS Tutorial

---

## *Spatial Queries*



University  
of Colorado  
Denver

Created by: Ricardo Oliveira  
*ricardo.oliveira@ucdenver.edu*

July 2014

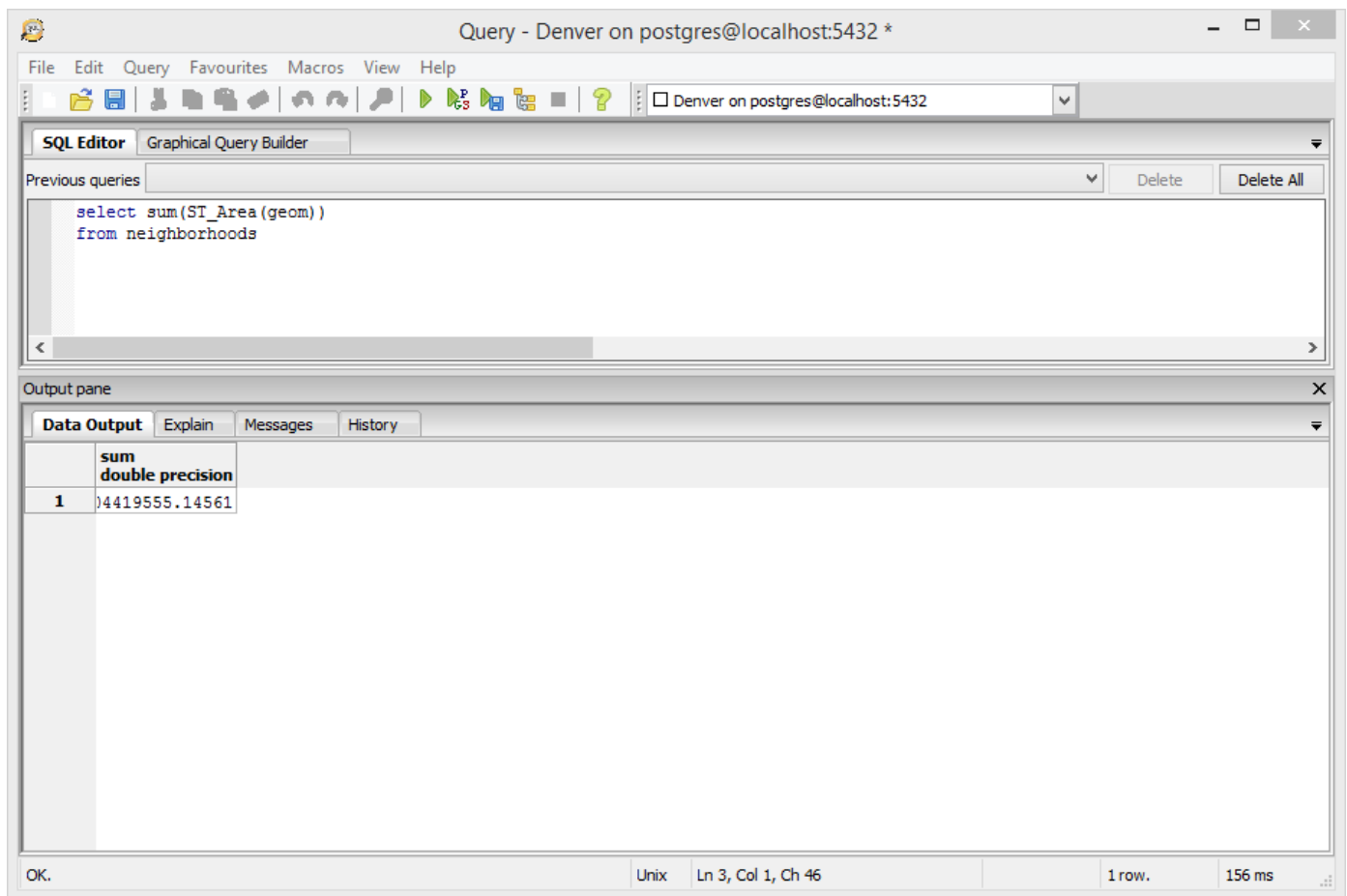
Now that our data was rightly converted into the correct projected coordinate system we are ready to make spatial analyzes with it!

The following tutorials will cover the main aspects of spatial analyzes, those who have prior GIS experience will observe how similar a lot of the concepts will sound, in other words we will learn how to perform known analyzes in new ways.

First we will learn how to call spatial queries in order to analyze the dimension of certain geometries.

Let's start by asking how big is Denver in terms of area.

```
select sum(ST_Area(geom))
from neighborhoods
```



The screenshot shows a PostgreSQL SQL Editor window titled "Query - Denver on postgres@localhost:5432 \*". The window has a menu bar (File, Edit, Query, Favourites, Macros, View, Help) and a toolbar. The SQL Editor pane contains the following query:

```
select sum(ST_Area(geom))
from neighborhoods
```

The Output pane is active, showing the results of the query in a table format:

	sum double precision
1	14419555.14561

The status bar at the bottom indicates "OK.", "Unix", "Ln 3, Col 1, Ch 46", "1 row.", and "156 ms".

You will notice that all spatial functions start with the letters ST, this means Spatial Temporal, this is always a sign that the query is somehow spatial related. Also, most of these operators carry literal names, in this case ST\_Area, an operator that return the geometry's area.

It is also important to remember that all measurements that we make using such operators are in the same unit of the projection system, in this case US Feet, if you forget what projection system the data is in you can always run the "select \* from geometry\_columns" query to check de SRID of our data.

Notice as well that our query was meant to return the whole area of the city and county of Denver, we did this using the SUM() operator. But we can also make a query that will return the area of each neighborhood separate.

```
select nbrhd_name, ST_Area(geom) * 0.000022959 as area
from neighborhoods
order by area DESC
```

In this example we are multiplying the area calculation by 0.000022959 in order to have the result in acres, a much more friendly unit for area. Also, we are renaming the result column as "area" and sorting this column by descending order.

The screenshot shows a PostgreSQL SQL Editor window titled "Query - Denver on postgres@localhost:5432". The SQL Editor contains the following query:

```
select nbrhd_name, ST_Area(geom) * 0.000022959 as area
from neighborhoods
order by area DESC
```

The Output pane shows the results of the query in a table format:

	nbrhd_name character varying(254)	area double precision
1	DIA	26895.03551648
2	Stapleton	5749.735130683
3	Gateway / Green Val	4712.246258428
4	Montbello	3239.944944308
5	Northeast Park Hill	2131.750688188
6	Marston	2028.776966246
7	Hampden South	1863.557637247
8	Hampden	1807.548540500
9	Fort Logan	1698.792283280
10	Elyria Swansea	1660.675613810
11	Lowry Field	1650.702237541
12	Globeville	1318.438508246
13	Virginia Village	1243.360832519
14	Washington Virginia	1170.588644049

The status bar at the bottom indicates "OK.", "Unix", "Ln 4, Col 1, Ch 94", "78 rows.", and "12 ms".

It is interesting to observe how the top two biggest neighborhoods in the Denver are the current international airport and the former airport, this may rise further questions about current and past land uses of our neighborhoods which may explain their sizes. This is the sort of information that you can extract from data simply by questioning it.

Other function that is very handy is ST\_Length to measure line.

```
select fullname, ST_Length(geom)/5280 as length
from street_centerline
order by length DESC
```

The screenshot shows a PostgreSQL SQL Editor window titled "Query - Denver on postgres@localhost:5432 \*". The SQL Editor contains the following query:

```
select fullname, ST_Length(geom)/5280 as length
from street_centerline
order by length DESC
```

The Output pane shows the results of the query in a table format:

	fullname character varying(100)	length double precision
1	PENA BLVD OUTBOUND	2.238498624103
2	N QUENCY ST	1.740774201166
3	PENA BLVD INBOUND	1.684619589746
4	E 114TH AVE	1.479325790618
5	N HUDSON MILE RD	1.096464173009
6	WB I70 TO PEORIA RA	1.072672222619
7	PENA BLVD INBOUND	1.06603250021
8	N HAYESMOUNT RD	1.040470003921
9	E 88TH AVE	1.025835439961
10	N CHAMBERS RD	1.019081053737
11	N BUCKLEY RD	1.008832982546
12	E 128TH AVE	1.007755265688
13	N BUCKLEY RD	1.006739787390
14	N MONAGHAN RD	1.004433573046

The status bar at the bottom indicates "OK.", "Unix", "Ln 3, Col 6, Ch 77", "29174 rows.", and "289 ms".

Again on this example we converted the final result into something that is more suitable such as miles, also we asked the query to sort the results in order to show the longest streets first. But notice how some streets such as Pena Blvd has multiple entries, sometime a query show more than what we expect in this case we have a glimpse of how our data is structured. On the next tutorial we will learn how we can join multiple tables in order to ask more complex questions, questions that a single dataset would not be able to answer.