Postgre/PostGIS Tutorial

Spatial Relationships

Created by: Ricardo Oliveira

ricardo.oliveira@ucdenver.edu

University
of Colorado
Denver

December 2014

On the previous tutorial we explored how we can extract spatial data from a specific dataset, now we will learn how we extract spatial relationships from two different datasets. This is a foundational step towards the spatial join process that will be explored on the next tutorial. On this tutorial we will explore three type of relationships: **Intersect, Contains,** and **Within**.

Intersects

The intersect function checks the geometries of the data and return only the ones that intersect each other, let's see the example:

We want to know which streets intersects the CBD neighborhoods. Here is the query.

```
select s.name, n.nbrhd_name
from street_centerline as s, neighborhoods as n
where n.nbrhd_name = 'CBD' AND ST_Intersects(n.geom, s.geom)
```

First things first, we want to know which streets intersects the CBD neighborhood, therefore we want to display both names, did you notice the s. and n.? These are **aliases**, the next line we inform the query that we want to check the relationships of the street_centerline and neighborhoods tables, by using 'as' we give alias to our tables, this save a few keystrokes and makes the query look better. When we want to point our query to a given column on a given table we have to use the following structure: **TABLE.TABLECOLUMN** .

The last row is about which function we will use. In this case we use the ST_Intersects(geometry A, geometry B). The function will compare which features of B intersects with A. and of course we have to inform which neighborhoods we want to check, in this case CBD.

Here is the result:

	name character varying(30)	nbrhd_name character varying(254)
1	LAWRENCE	CBD
2	18TH	CBD
3	ARAPAHOE	CBD
4	15TH	CBD

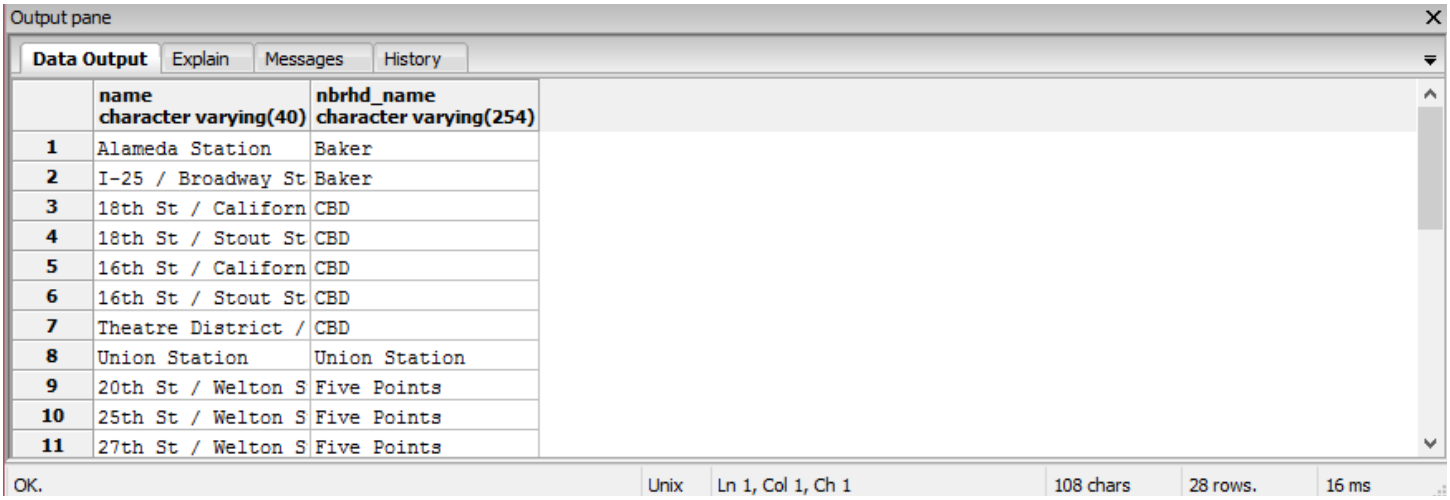
OK. Unix Ln 1, Col 8, Ch 8 175 rows. 32 ms

Within

The Contains function basically check if a given geometry is inside another bigger geometry. Let's check which lightrail stations are inside each neighborhood.

```
select l.name, n.nbrhd_name
from lightrailstations as l, neighborhoods as n
where ST_Within(l.geom, n.geom)
```

Let's see what this query did. We want to display both the station's names and the neighborhood's names, so we have to inform this on the first line, just like we did with our previous example. The difference now is that we are using the ST_Within functions, this functions compares the geometries of the first with the geometries of the second and returns the comparison. The final result is this:


Output pane

Data Output | Explain | Messages | History

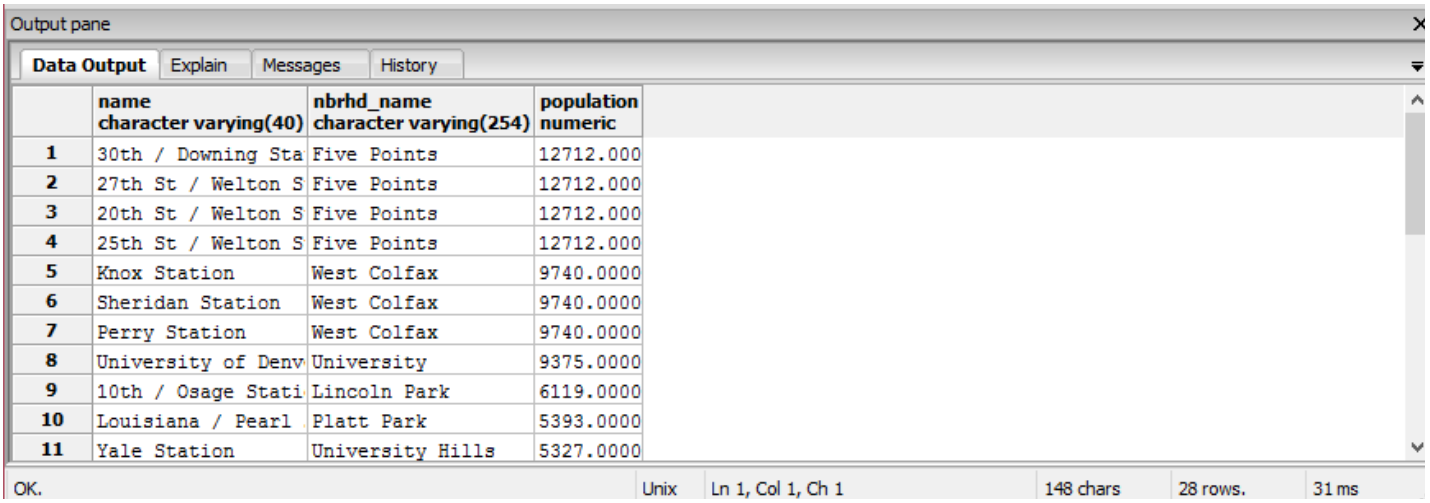
	name character varying(40)	nbrhd_name character varying(254)
1	Alameda Station	Baker
2	I-25 / Broadway St	Baker
3	18th St / Californ	CBD
4	18th St / Stout St	CBD
5	16th St / Californ	CBD
6	16th St / Stout St	CBD
7	Theatre District /	CBD
8	Union Station	Union Station
9	20th St / Welton S	Five Points
10	25th St / Welton S	Five Points
11	27th St / Welton S	Five Points

OK. Unix Ln 1, Col 1, Ch 1 108 chars 28 rows. 16 ms

We can expand the functionality of our query by querying extra columns, let's ask the query to return the neighborhoods population as well.

```
select l.name, n.nbrhd_name, n.population
from lightrailstations as l, neighborhoods as n
where ST_Within(l.geom, n.geom)
order by n.population DESC
```

This new query will return the population of each neighborhood and return it in a descending order.


Output pane

Data Output | Explain | Messages | History

	name character varying(40)	nbrhd_name character varying(254)	population numeric
1	30th / Downing Sta	Five Points	12712.000
2	27th St / Welton S	Five Points	12712.000
3	20th St / Welton S	Five Points	12712.000
4	25th St / Welton S	Five Points	12712.000
5	Knox Station	West Colfax	9740.0000
6	Sheridan Station	West Colfax	9740.0000
7	Perry Station	West Colfax	9740.0000
8	University of Denv	University	9375.0000
9	10th / Osage Stati	Lincoln Park	6119.0000
10	Louisiana / Pearl	Platt Park	5393.0000
11	Yale Station	University Hills	5327.0000

OK. | Unix | Ln 1, Col 1, Ch 1 | 148 chars | 28 rows. | 31 ms

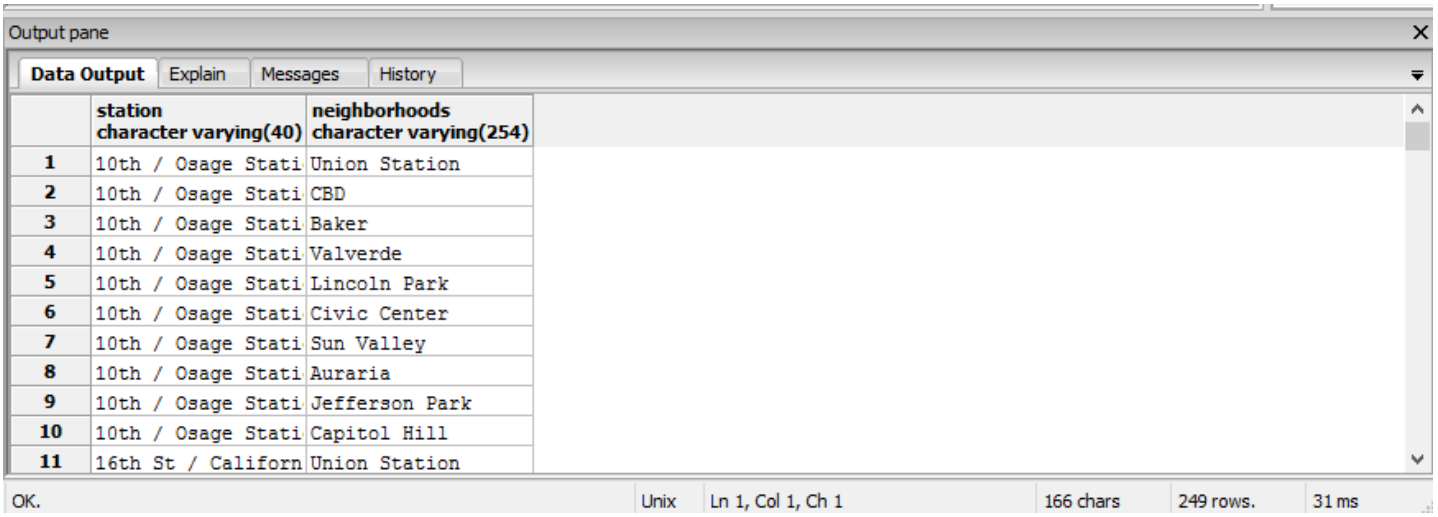
Notice that now we are getting closer to gather all the tools that we need to analyze the relationship between the RTD's light rail stations and the rest of the city of Denver. There is one extra function that can be very useful in our analysis.

Distance Within

The distance within function is called by using ST_DWithin, this function will compare geometry A and geometry B based on a distance input by the user. The generic form of the function is, ST_DWithin(geometry A, geometry B, radius). Remember that the radius must be informed using the same unit of measurement of the data being used. Let's check which neighborhoods are within one mile from all the lightrail stations in Denver.

```
select st.name as station, n.nbrhd_name as neighborhoods
from neighborhoods as n, lightrailstations as st
where ST_DWithin(n.geom, st.geom, 5280)
order by station ASC
```

On this query we created aliases for our tables and also for how the query should return the results, we want our columns to make sense by calling them simply stations and neighborhoods. Notice that the radios inside the function is in feet, that is because our data is in state plane projection. Lastly we want the results to be displayed in ascending order just to improve the readability. The final result is this:



Output pane

Data Output | Explain | Messages | History

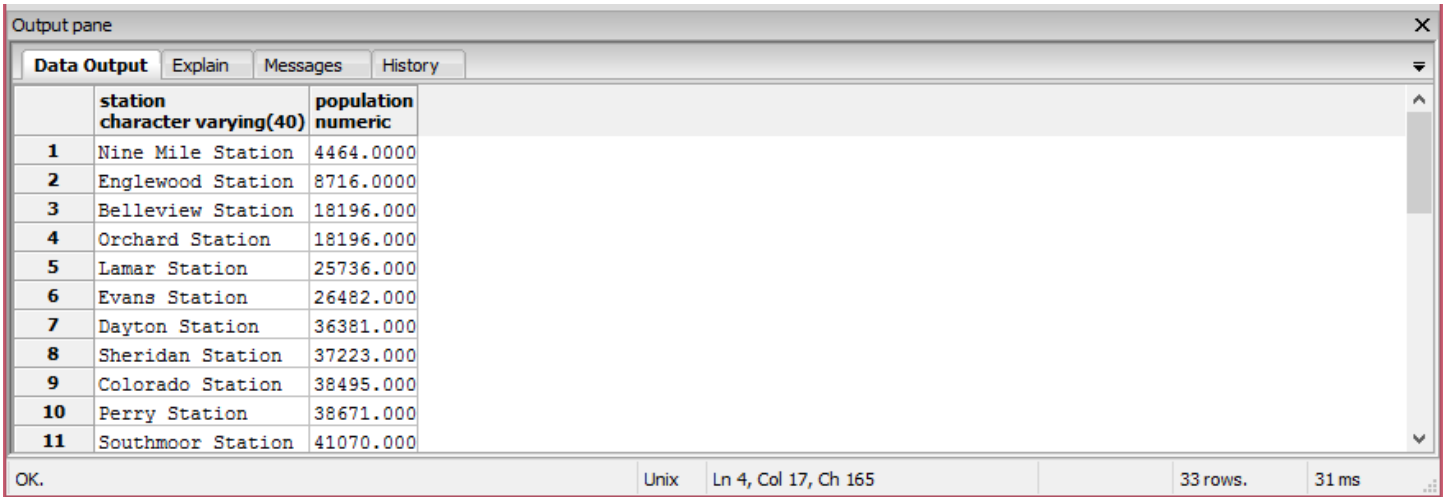
	station character varying(40)	neighborhoods character varying(254)
1	10th / Osage Stati	Union Station
2	10th / Osage Stati	CBD
3	10th / Osage Stati	Baker
4	10th / Osage Stati	Valverde
5	10th / Osage Stati	Lincoln Park
6	10th / Osage Stati	Civic Center
7	10th / Osage Stati	Sun Valley
8	10th / Osage Stati	Auraria
9	10th / Osage Stati	Jefferson Park
10	10th / Osage Stati	Capitol Hill
11	16th St / Californ	Union Station

OK. | Unix | Ln 1, Col 1, Ch 1 | 166 chars | 249 rows. | 31 ms

We can use the ST_DWithin function to analyze the total population on those neighborhoods within one mile from the stations. We just have to change a few things on our query.

```
select st.name as station, sum(n.population) as population
from neighborhoods as n, lightrailstations as st
where ST_DWithin(n.geom, st.geom, 5280)
group by st.name
order by sum(n.population) ASC
```

This time we just add the sum(n.population) part, which will return the population sum within one mile from the stations. We have to aggregate the returning table by name stations, and lastly we order our rows by population to facilitate our analysis. The final result is:



Output pane

Data Output | Explain | Messages | History

	station character varying(40)	population numeric
1	Nine Mile Station	4464.0000
2	Englewood Station	8716.0000
3	Belleview Station	18196.000
4	Orchard Station	18196.000
5	Lamar Station	25736.000
6	Evans Station	26482.000
7	Dayton Station	36381.000
8	Sheridan Station	37223.000
9	Colorado Station	38495.000
10	Perry Station	38671.000
11	Southmoor Station	41070.000

OK. | Unix | Ln 4, Col 17, Ch 165 | 33 rows. | 31 ms

PostGIS offers many more spatial functions that can be used to analyze data, the three functions presented here are just a glimpse of what can be done inside postgreSQL . These functions should offer to you a pretty solid start but if you feel that you need more tools check the postgis documentation at http://postgis.net/docs/reference.html#Spatial_Relationships_Measurements for a look at different functions.

On the next tutorial we will learn how to create new data from our analysis and how to back-up our database.