POSTGIS SPATIAL TRICKS

REGINA OBE

lr@pcorp.us @reginaobe
Consulting

FIND N-CLOSEST PLACES (KNN)

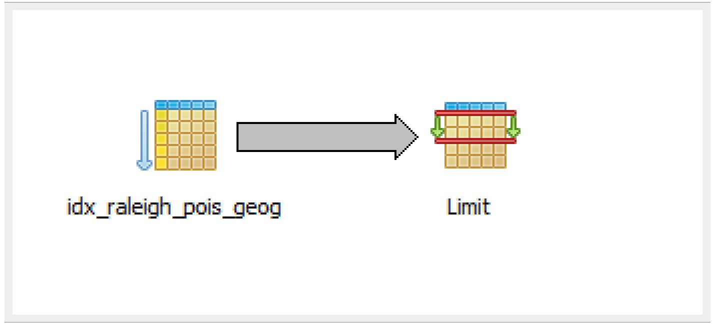
Given a location, find the N-Closest places. Geography and n-D geometry operator support new in PostGIS 2.2.

EXAMPLE N-CLOSEST USING GEOGRAPHY DATA TYPE

Closest 5 Indian restaurants to here

```
-- 51ms
SELECT name, other_tags->'amenity' As type,
    ST_Point(-78.64040,35.77627)::geography <-> geog As dist_m
FROM raleigh_pois As pois
WHERE other_tags @> 'cuisine=>indian'::hstore
ORDER BY dist_m
LIMIT 5;
```

```
            name             |    type    |       dist_m
-----------------------------+------------+------------------
 Blue Mango                  | restaurant | 1059.16153525522
 Kadhambam Spices            |            | 19087.6284119947
 Sitar                       | restaurant | 35408.8116290629
 Vimala's Curryblossom Cafe  | restaurant | 40860.2976504395
 Mint                        | restaurant | 40963.1102551244
(5 rows)
```

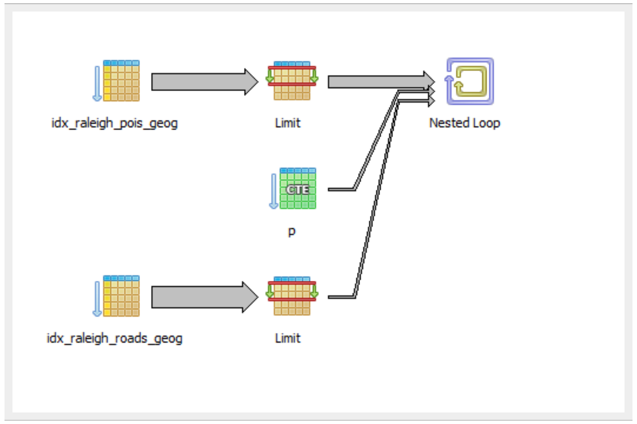


FIND 2 CLOSEST ROADS TO POINTS OF INTEREST WITH LATERAL

PostgreSQL 9.5+ and PostGIS 2.2+ for true distance.

```
WITH p AS (SELECT name, other_tags->'amenity' As type,
    ST_Point(-78.64040,35.77627)::geography <-> geog As dist_m, geog
FROM raleigh_pois As pois
WHERE other_tags @> 'cuisine=>indian'::hstore
ORDER BY dist_m LIMIT 4)
SELECT p.name, p.type, r.name As road,
 r.dist_m_road::numeric(10,2), p.dist_m::numeric(10,2)
FROM p, LATERAL (SELECT rr.name, rr.geog <-> p.geog AS dist_m_road
FROM raleigh_roads AS rr WHERE rr.name > ''
ORDER BY dist_m_road LIMIT 2) As r;
```

```
           name             |    type    |             road             | dist_m_road |  dist_m
----------------------------+------------+------------------------------+-------------+----------
 Blue Mango                 | restaurant | West Lane Street             |       14.64 |  1059.16
 Blue Mango                 | restaurant | Glenwood Avenue              |       16.62 |  1059.16
 Kadhambam Spices           |            | Hatchet Creek Greenway       |      111.25 | 19087.63
 Kadhambam Spices           |            | Hatchet Creek Greenway       |      112.37 | 19087.63
 Sitar                      | restaurant | Chapel Hill Blvd Service Road |       36.62 | 35408.81
:
(8 rows)
Time: 45.210 ms
```



WHAT PLACES ARE WITHIN X-DISTANCE

Limit results set by distance rather than number of records. Like KNN, geometry can be anything like distance from a road, a lake, or a point of interest.

EXAMPLE: GEOGRAPHY WITHIN 1000 METERS OF LOCATION

What are closest fast food joints within 1000 meters. This will work for PostGIS 1.5+

```
-- Time: 2.241 ms
SELECT name,  other_tags->'cuisine' As cuisine,
        ST_Distance(pois.geog,ref.geog) As dist_m
FROM raleigh_pois AS pois,
        (SELECT  ST_Point(-78.64040,35.77627)::geography) As ref(geog)
  WHERE other_tags @> 'amenity=>fast_food'::hstore
        AND ST_DWithin(pois.geog, ref.geog, 1000)
ORDER BY dist_m;
```

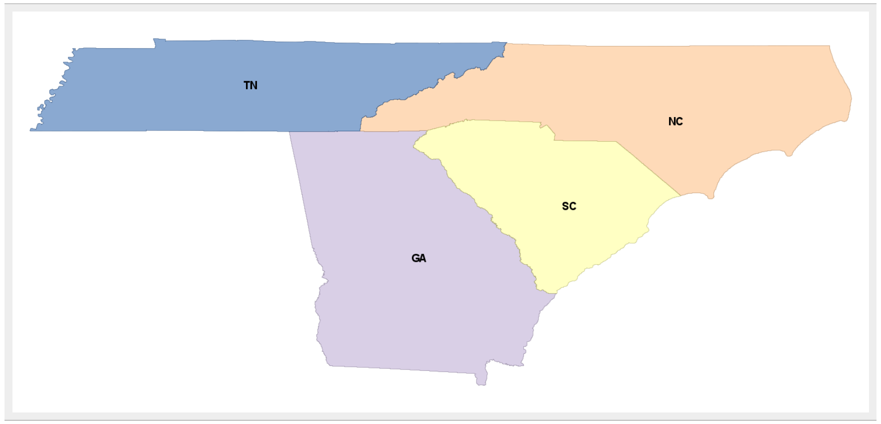
```
           name              | cuisine  |    dist_m
-----------------------------+----------+--------------
 Chick-fil-A                 | chicken  | 115.15429719
 Quiznos                     | sandwich |  208.9641767
 zpizza                      | pizza    | 246.56944119
 Snoopy's                    |          | 851.75116195
 Quiznos Sandwich Restaurant | sandwich | 890.35270577
 Char Grill                  | burger   | 906.69076176
 Bruger's Bagels             |          |  997.0456652
(7 rows)
```

DIVIDE LARGE GEOMETRIES INTO SMALLER ONES WITH ST_SUBDIVIDE

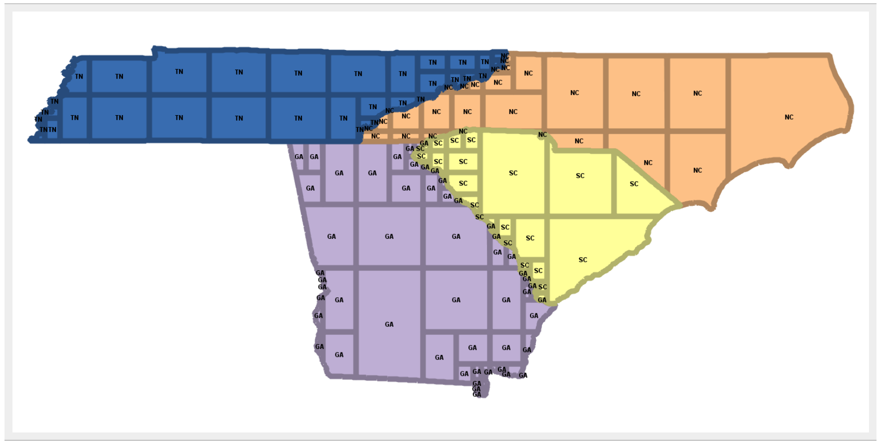
New in PostGIS 2.2. Works for non-point geometries (only 2D). Second arg is max number of points to allow per divide.

```
SELECT stusps, ST_SubDivide(geom, 1000) AS geom
FROM states
WHERE stusps IN('TN', 'NC', 'SC', 'GA');
```

Before had 4 rows



After have 186 rows

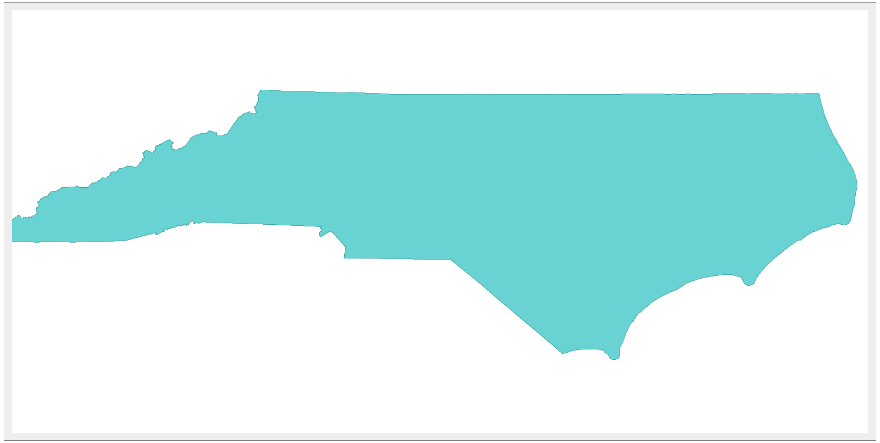


NEW POSTGIS 2.3 ST_GENERATEPOINTS

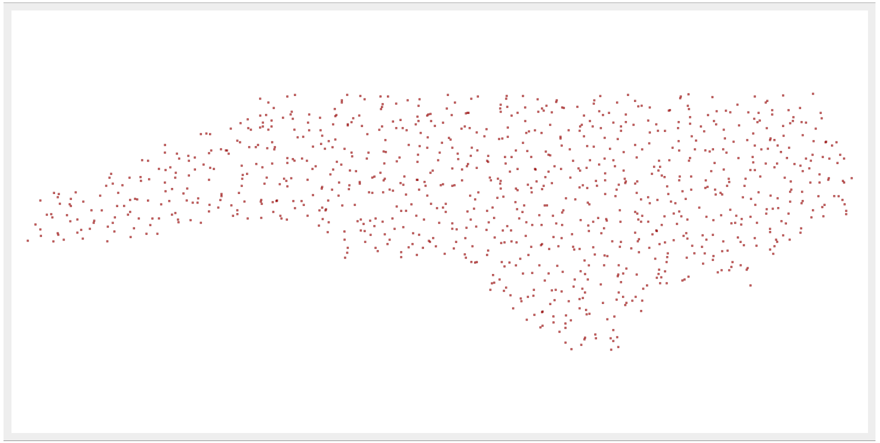
Converts multipolygon/polygon to multpoint - random space filling the area

```
SELECT stusps, ST_GeneratePoints(geom, 1000) AS geom
FROM states
WHERE stusps =  'NC';
```

Before: 1 Multi-Polygon



After: 1 multipoint of 1000 points



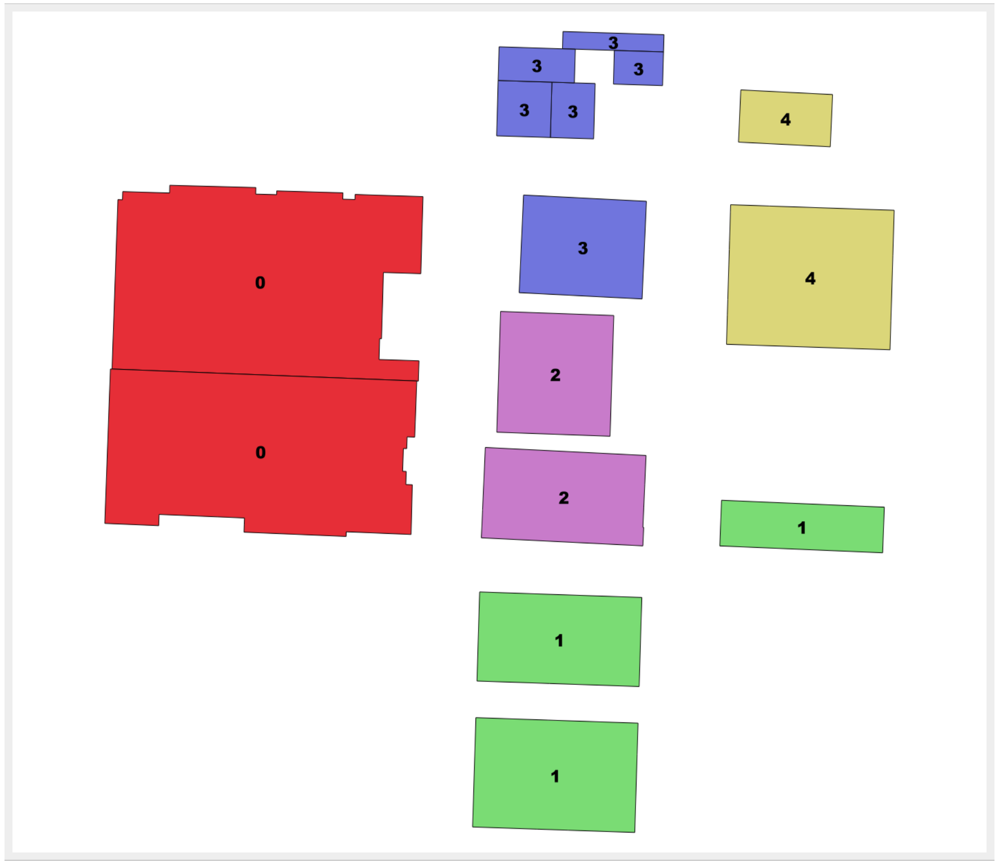
CLUSTERING GEOMETRIES USING WINDOW FUNCTIONS: COMING POSTGIS 2.3

- 2.3: ST_ClusterKMeans
- 2.3: ST_ClusterDbSCAN

ST_CLUSTERKMEANS: NUMBER BUILDINGS FROM 0-4 BY PROXIMITY TO EACH OTHER

```
SELECT name, ST_ClusterKMeans(geom, 5) OVER() AS bucket
FROM raleigh_polys
WHERE name > '' AND building > ''
AND ST_DWithin(geom, 'SRID=2264;POINT(2106664 737626)'::geometry,500)
ORDER BY bucket;
```

Need to add geom column to view



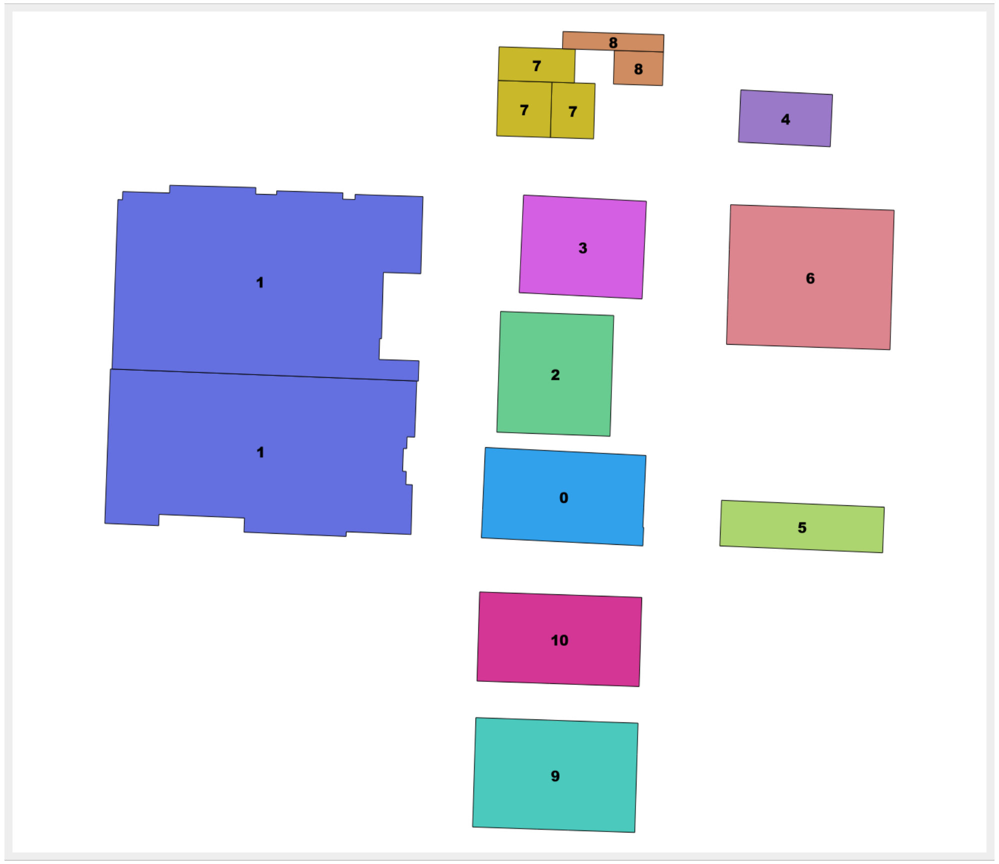
```
               name                | bucket
-----------------------------------+--------
 Wake County Justice Center        |      0
 Wake County Public Safety Center  |      0
 Sir Walter Raleigh Hotel          |      1
 Capital Bank Plaza                |      1
 Sheraton Raleigh Hotel            |      1
 Wake County Office Building       |      2
 Wake County Courthouse            |      2
 Kings                             |      3
 The Mahler                        |      3
 Capital Club 16                   |      3
 CrossFit Invoke                   |      3
 Federal Building                  |      3
 North State Bank                  |      3
 PNC Plaza                         |      4
 First Citizens Bank               |      4
(15 rows)
Time: 1.228 ms
```

ST_CLUSTERDBSCAN: SIMILAR TO KMEANS, BUT USES DESIRED DISTANCE AND MINIMUM NUMBER ITEMS

Cluster together buildings that intersect each other.

```
SELECT name, ST_ClusterDBSCAN(geom, 0, 2) OVER() AS bucket, geom
FROM raleigh_polys
WHERE name > '' AND building > ''
AND ST_DWithin(geom, 'SRID=2264;POINT(2106664 737626)'::geometry,500)
ORDER BY bucket;
```

Need to add geom column to view



```
                 name                 | bucket
--------------------------------------+---------
 Wake County Office Building          |      0
 Wake County Public Safety Center     |      1
 Wake County Justice Center           |      1
 Wake County Courthouse               |      2
 Federal Building                     |      3
 First Citizens Bank                  |      4
 Capital Bank Plaza                   |      5
 PNC Plaza                            |      6
 Kings                                |      7
 Capital Club 16                      |      7
 CrossFit Invoke                      |      7
 North State Bank                     |      8
 The Mahler                           |      8
 Sheraton Raleigh Hotel               |      9
 Sir Walter Raleigh Hotel             |     10
(15 rows)


Time: 1.046 ms
```

PARALLELIZATION OF SPATIAL JOINS AND FUNCTIONS

Will require PostgreSQL 9.6+ and PostGIS 2.3+. Read more: http://blog.cleverelephant.ca/2016/03/parallel-postgis-joins.html

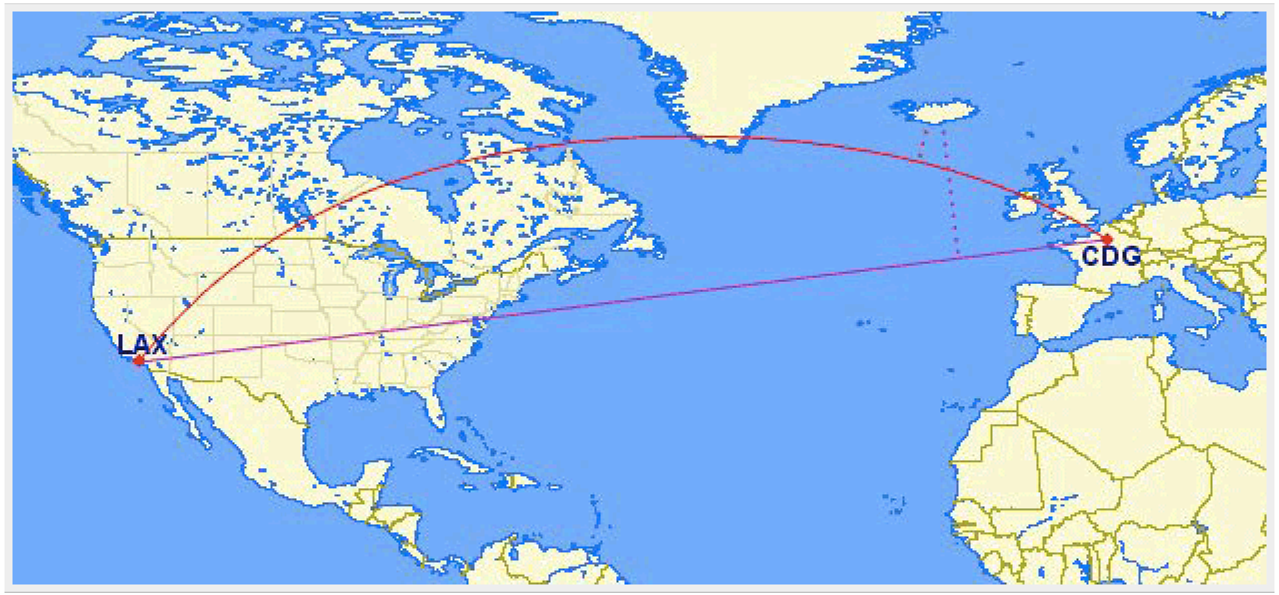
Not yet committed to PostGIS repo, go here - https://github.com/pramsey/postgis/tree/parallel

```
set parallel_tuple_cost=0.01;
SET max_parallel_degree=2;
```

SEGMENTIZE A LINESTRING IN GEOGRAPHY

PostGIS 2.1+ ST_Segmentize(geography) can create great circles

GEOGRAPHY SEGMENTIZE VS. GEOMETRY SEGMENTIZE ON A MAP



From BoundlessGeo docs

SEGMENTIZE IN GEOGRAPHY OUTPUT AS GEOMETRY WKT

```
SELECT ST_AsText(
 ST_Segmentize('LINESTRING(-118.4079 33.9434, 2.5559 49.0083)'::geography,
  10000) );
```

```
LINESTRING(-118.4079 33.9434,-118.365191634689 33.9946750650617,
 -118.322351004015 34.0460320153076,
 ...,2.48756947085441 49.0516183725212,2.5559 49.0083)
```

SEGMENTIZE AND OUTPUT AS GOOGLE ENCODED LINE

PostGIS 2.2 we have ST_AsEncodedPolyline useful for drawing on google maps and use in Leaflet. ST_LineFromEncodedPolyline for getting back a geometry.

```
SELECT ST_AsEncodedPolyline(
  ST_Segmentize(
    'LINESTRING(-118.4079 33.9434, 2.5559 49.0083)'::geography,
        10000)::geometry,
   4);
```

```
gqdnEjpuqUo_I}iG}_IwjGo`IqkG_aImlGoaIgmG..~mGskLvmGajL
```

ADDRESS STANDARDIZATION / GEOCODING / REVERSE GEOCODING

PostGIS 2.2 comes with extension address_standardizer. Also included since PostGIS 2.0 is postgis_tiger_geocoder (only useful for US).

In works improved address standardizer and worldly useful geocoder - refer to: https://github.com/woodbri/address-standardizer

ADDRESS STANDARDIZATION

Need to install address_standardizer, address_standardizer_data_us extensions (both packaged with PostGIS 2.2+). Using json to better show non-empty fields

```
SELECT *
FROM json_each_text(to_json(standardize_address('us_lex', 'us_gaz','us_rules'
, '300 S. Salisbury St',
  'Raleigh, NC  27601' )))
WHERE value > '';
```

```
    key     |     value
-----------+----------------
 house_num | 300
 predir    | SOUTH
 name      | SALISBURY
 suftype   | STREET
 city      | RALEIGH
 state     | NORTH CAROLINA
 postcode  | 27601
(7 rows)
```

Same exercise using the packaged postgis_tiger_geocoder tables that standardize to abbreviated instead of full name

```
SELECT *
FROM json_each_text( to_json(
        standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz','tiger.pagc_rules'
, '300 S. Salisbury St',
  'Raleigh, NC  27601' )))
WHERE value > '';
```

```
    key     |   value
------------+-----------
 house_num  | 300
 predir     | S
 name       | SALISBURY
 suftype    | ST
 city       | RALEIGH
 state      | NC
 postcode   | 27601
(7 rows)
```

GEOCODING USING POSTGIS TIGER GEOCODER

Given a textual location, ascribe a longitude/latitude. Uses postgis_tiger_geocoder extension requires loading of US Census Tiger data.

```
SELECT pprint_addy(addy) As address,
    ST_X(geomout) AS lon, ST_Y(geomout) As lat, rating
    FROM geocode('300 S. Salisbury St, Raleigh, NC  27601',1);
```

```
                address                |        lon        |       lat        | rating
---------------------------------------+-------------------+------------------+--------
 300 S Salisbury St, Raleigh, NC 27601 | -78.6404024546499 | 35.7762672906178 |      0
(1 row)
```

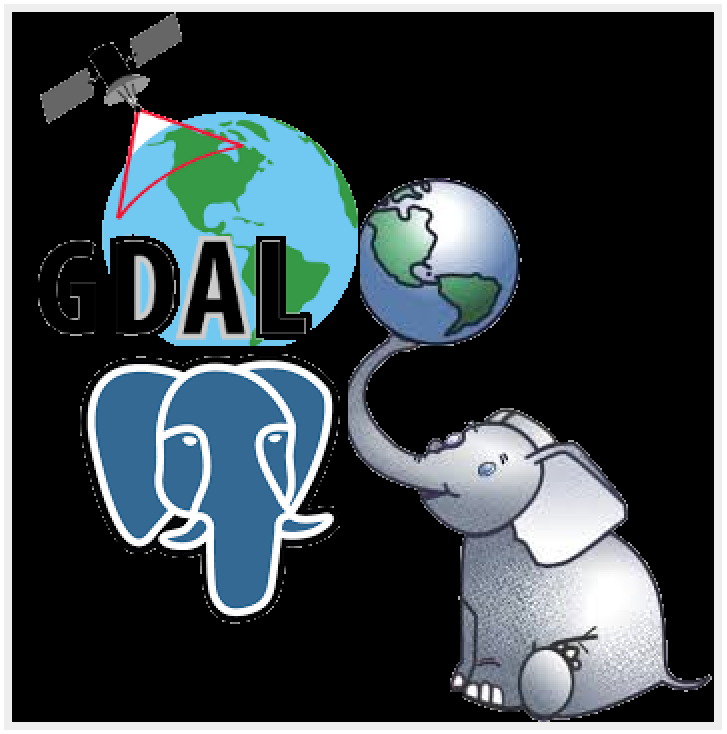
REVERSE GEOCODING

Given a longitude/latitude or GeoHash, give a textual description of where that is. Using postgis_tiger_geocoder reverse_geocode function

```
SELECT pprint_addy(addrs) AS padd,
  array_to_string(r.street,',') AS cross_streets
FROM reverse_geocode(ST_Point(-78.6404045,35.7762673)) AS r
    , unnest(r.addy) As addrs;
```

```
                padd                  | cross_streets
--------------------------------------+---------------
 304 S Salisbury St, Raleigh, NC 27601 | W Davie St
(1 row)
```

GDAL CONJOINS WITH POSTGIS AND POSTGRESQL



- Scene 1: PostGIS Raster
- Scene 2: OGR_FDW Foreign Data Wrapper

SCENE 1: POSTGIS + GDAL = POSTGIS RASTER

A long time ago, a crazy man named Pierre Racine had a very crazy idea: https://trac.osgeo.org/postgis/wiki/WKTRaster and he got others Bborie Park, Sandro Santilli, Mateusz Loskot, David Zwarg and others to help implement his crazy scheme.

REGISTER YOUR RASTERS WITH THE DATABASE: OUT OF DB

You could with raster2pgsql the -R means just register, keep outside of database. Without the -R the data is stored in Db

```
raster2pgsql -I -C -R C:/data/nc_aerials/*.tif -F aerials | psql
```

OR (useful especially if you are on windows to force recursion of folders). Requires PostgreSQL 9.3+ PostGIS 2.1+

```
CREATE TABLE dir_list(file_name text);
COPY dir_list FROM PROGRAM 'dir C:\data\nc_aerials\*.tif /b /S'
        WITH (format 'csv');

CREATE TABLE aerials( rid serial PRIMARY KEY,rast raster, filename text);
INSERT INTO aerials(rast, filename)
SELECT
        ST_AddBand(
                NULL::raster,
                d.file_name, NULL::int[]
        ), d.file_name
FROM dir_list AS d;

SELECT AddRasterConstraints('aerials', 'rast');
--verify constraints
SELECT srid, scale_x, scale_y, blocksize_x As width,
blocksize_y As height, pixel_types, out_db
FROM raster_columns
WHERE r_table_name = 'aerials';
```

```
 srid | scale_x | scale_y | width | height |    pixel_types     | out_db
------+---------+---------+-------+--------+--------------------+---------
 2264 |     0.5 |    -0.5 | 10000 |  10000 | {8BUI,8BUI,8BUI} | {t,t,t}
(1 row)
```

```
CREATE INDEX idx_aerials_rast ON aerials USING gist(ST_ConvexHull(rast));
analyze aerials;
```

LET'S TILE THE RASTER TO 200X200 CHUNKS RIGHT IN DB

Requires PostGIS 2.1+. ST_Tile, if working on out-db keeps out-db and very fast.

```
CREATE TABLE aerials_200_200(rid serial primary key, rast raster, filename text);
INSERT INTO aerials_200_200(rast,filename)
SELECT ST_Tile(rast,200,200) As rast, filename
FROM aerials;
SELECT AddRasterConstraints('aerials_200_200', 'rast');
--verify constraints
SELECT srid, scale_x, scale_y, blocksize_x As width,
 blocksize_y As height, pixel_types, out_db
FROM raster_columns
WHERE r_table_name = 'aerials_200_200';
```

```
 srid | scale_x | scale_y | width | height |   pixel_types    | out_db
------+---------+---------+-------+--------+------------------+---------
 2264 |     0.5 |    -0.5 |   200 |    200 | {8BUI,8BUI,8BUI} | {t,t,t}
(1 row)
```

```
CREATE INDEX idx_aerials_200_200_rast ON aerials_200_200 USING gist(ST_ConvexHull
analyze aerials_200_200;
```

CREATE OVERVIEWS RIGHT IN DB

Requires PostGIS 2.2+. This will make in-db raster from out-db so might take a while. Took 8 minutes for my aerials table that had 30 10000x10000 raster links.

```
SELECT ST_CreateOverview('aerials'::regclass, 'rast', 4);
```

```
st_createoverview
-------------------
o_4_aerials
```

```
CREATE INDEX idx_o_4_aerials_rast ON o_4_aerials USING gist(ST_ConvexHull(rast));
```

```
SELECT srid, scale_x, scale_y, blocksize_x As width,
 blocksize_y As height, pixel_types, out_db
FROM raster_columns
WHERE r_table_name = 'o_4_aerials';

 srid | scale_x | scale_y | width | height |   pixel_types    | out_db
------+---------+---------+-------+--------+------------------+---------
 2264 |       2 |      -2 | 10000 |  10000 | {8BUI,8BUI,8BUI} | {f,f,f}
(1 row)
```

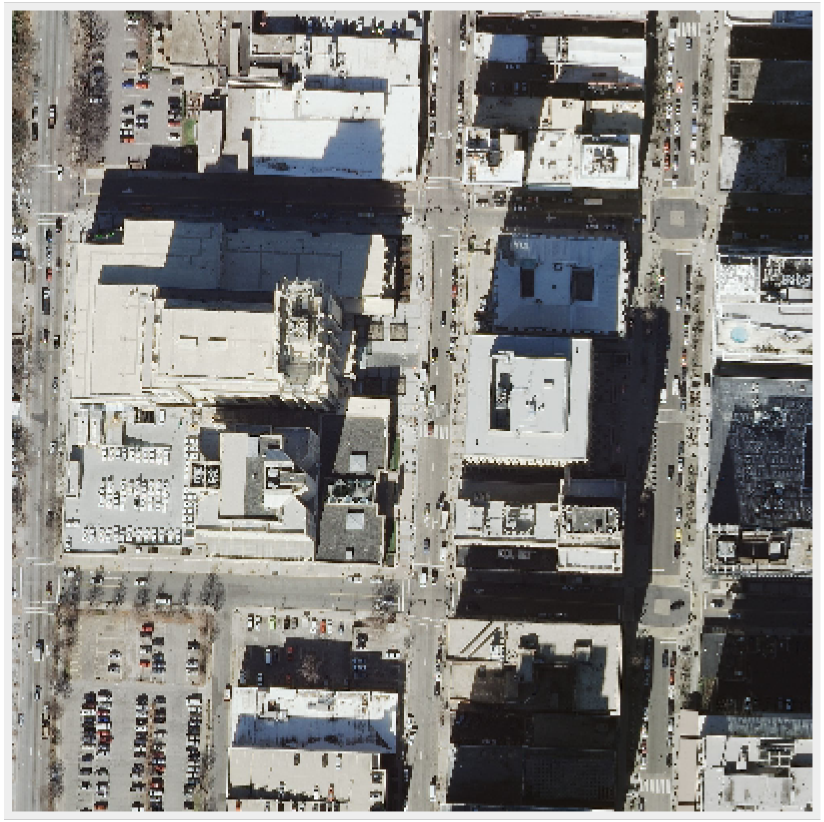
RETURN AN AREA: 500 FEET AROUND US

Project to same spatial ref as raster (2264 NC State Plane ft)

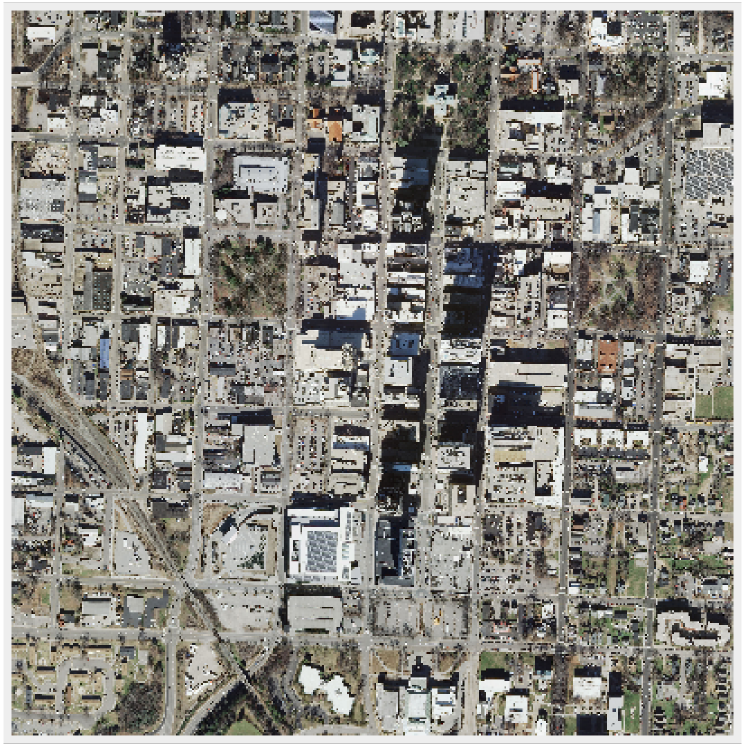
```
SELECT ST_AsPNG(ST_Resize(ST_Union(ST_Clip(rast, geom)), 0.20,0.20)), count(*)
FROM aerials_200_200 AS a,
         ST_Expand(
                 ST_Transform(ST_SetSRID(ST_Point(-78.6404,35.77627),4326),
                         2264),500) As geom
WHERE ST_Intersects(a.rast,geom);
```

Using aerials: 4 secs (1 row), aerials_200_200: 5.9 sec (120 rows)

o_4_aerials resize 0.5 (980ms 1 row)



Using o_4_aerials resize 0.2, 2000 ft - 5.7 secs



SCENE 2: POSTGRESQL + GDAL ~ POSTGIS = OGR_FDW POSTGRESQL FOREIGN DATA WRAPPER

5 years ago I asked, https://trac.osgeo.org/postgis/ticket/974 and someone finally did it. It was slicker than I ever imagined.

DATA WRANGLING WITH OGR_FDW

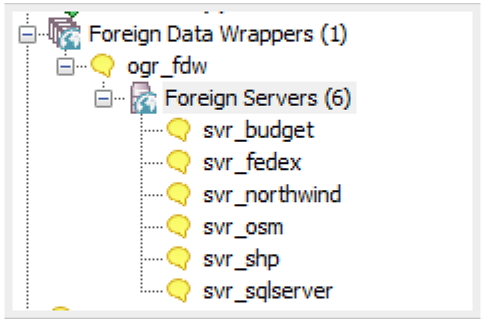
If you have all sorts of data of both a spatial and non-spatial flavor to tame, make sure you have ogr_fdw foreign data wrapper in your tool belt.

- For windows users, it's part of PostGIS 2.2 bundle on application stackbuilder.
- For CentOS/Red Hat/Scientific etc, it's available via yum.postgresql.org
- For others, if you have PostGIS with GDAL support, just need postgresql dev package to compile. Download the source https://github.com/pramsey/pgsql-ogr-fdw

WHY IS OGR_FDW SO SEXY?

You have the combined power of GDAL and PostgreSQL working seamlessly together. So many kinds of data you can query and take advantage of PostgreSQL functions and any extension functions and types such as PostGIS, hstore, built-in json.

- Spreadsheets
- ODBC datasources
- OSM files (OSM, PBF)
- ESRI Shapefiles
- Many more


```
Foreign Data Wrappers (1)
  ogr_fdw
    Foreign Servers (6)
      svr_budget
      svr_fedex
      svr_northwind
      svr_osm
      svr_shp
      svr_sqlserver
```

ENABLE IT IN YOUR DATABASE

```
CREATE EXTENSION ogr_fdw;
```

LINK IN A WHOLE FOLDER OF ESRI SHAPEFILES AND DBASE FILES

```
CREATE SERVER svr_shp FOREIGN DATA WRAPPER ogr_fdw
OPTIONS (datasource 'C:/fdw_data/gisdata',
 format 'ESRI Shapefile'
);
CREATE SCHEMA shps;
-- this is a PostgreSQL 9.5 feature
IMPORT FOREIGN SCHEMA ogr_all
FROM SERVER svr_shp INTO shps;
```

```
\dE shps.*
```

```
             List of relations
 Schema |   Name   |     Type      |  Owner
--------+----------+---------------+----------
 shps   | airports | foreign table | postgres
 shps   | nbi      | foreign table | postgres
(2 rows)
```

QUERY YOUR SHAPE FILES LIKE REGULAR OLD TABLES

```sql
SELECT locid, ST_AsText(geom) AS wkt
FROM shps.airports
WHERE locid  ='JFK';
```

```
 locid |                    wkt
-------+------------------------------------------
 JFK   | POINT(-73.7789255555556 40.6397511111111)
(1 row)
```

OGR_FDW NOW UPDATEABLE IF GDAL DRIVER ALLOWS WRITE

Version 1.0.1 brought IMPORT FOREIGN SCHEMA, latest in master branch supports updating, ability to include subset of columns, and detect srid

Check out the code and test: Download the source https://github.com/pramsey/pgsql-ogr-fdw

Windows users, winnie builds whenever master changes - http://postgis.net/windows_downloads/, look in extras folder for your PostgreSQL version - e.g 9.5 64-bit would be in pg9.5 extras folder and called ogrfdw-pg95-binaries-1.0w64gcc48.zip

DO AN UPDATE/INSERT/DELETE TO SHAPE FILE AS IF IT WERE A LOCAL TABLE

```
UPDATE shps.airports
    SET geom = ST_SnapToGrid(geom,0.00001)
WHERE locid = 'JFK' RETURNING locid, ST_AsText(geom) As wkt;
```

```
 locid |            wkt
-------+----------------------------
 JFK   | POINT(-73.77893 40.63975)
(1 row)


UPDATE 1
```

```
INSERT INTO shps.airports(locid, geom)
SELECT 'ROO', geom
FROM shps.airports
WHERE locid = 'BOS' RETURNING locid,ST_AsText(geom);
```

```
locid |                  st_astext
-------+------------------------------------------
 ROO   | POINT(-71.0064166666667 42.3629722222222)


INSERT 1
```

```
DELETE FROM shps.airports
WHERE locid = 'ROO';
```

OSM FILES

```
-- data from https://mapzen.com/data/metro-extracts/
CREATE SERVER svr_osm
   FOREIGN DATA WRAPPER ogr_fdw
  OPTIONS (datasource 'C:/fdw_data/raleigh_north-carolina.osm.pbf',format 'OSM');
  CREATE SCHEMA IF NOT EXISTS osm;
IMPORT FOREIGN SCHEMA ogr_all
FROM SERVER svr_osm INTO osm;

\dE osm.*
```

```
                 List of relations
 Schema |       Name        |     Type      |  Owner
--------+-------------------+---------------+----------
 osm    | lines             | foreign table | postgres
 osm    | multilinestrings  | foreign table | postgres
 osm    | multipolygons     | foreign table | postgres
 osm    | other_relations   | foreign table | postgres
 osm    | points            | foreign table | postgres
(5 rows)
```

```
-- requires CREATE EXTENSION hstore;
CREATE TABLE raleigh_pois AS
SELECT osm_id, name, geom::geography As geog, is_in,
    place, other_tags::hstore
FROM osm.points;

CREATE TABLE raleigh_roads AS
SELECT osm_id, name, geom::geography As geog,
    other_tags::hstore
FROM osm.lines
WHERE highway > '';
```

LINK EVEN NON-SPATIAL LIKE MS ACCESS DATABASE TABLES AND QUERIES

```
CREATE SERVER svr_northwind FOREIGN DATA WRAPPER ogr_fdw
OPTIONS (datasource 'C:/fdw_data/northwind.mdb',
 format 'ODBC'
);
CREATE SCHEMA IF NOT EXISTS northwind;
-- will link in all tables and queries starting with Ord
IMPORT FOREIGN SCHEMA "Ord"
FROM SERVER svr_northwind INTO northwind;
```

```
\dE northind.*
```

```
                       List of relations
  Schema   |          Name          |     Type      |  Owner
-----------+------------------------+---------------+----------
 northwind | order_details          | foreign table | postgres
 northwind | order_details_extended | foreign table | postgres
 northwind | order_subtotals        | foreign table | postgres
 northwind | orders                 | foreign table | postgres
 northwind | orders_qry             | foreign table | postgres
(5 rows)
```

The schema part is case sensitive, has to match source

EVEN SPREADSHEETS

Each workbook is considered a server and each sheet a table

```
CREATE SERVER svr_fedex FOREIGN DATA WRAPPER ogr_fdw
OPTIONS (datasource 'C:/fdw_data/Fedex2016.xls',
 format 'XLS'
);
-- link only 1 spreadsheet preserve headers
IMPORT FOREIGN SCHEMA ogr_all  LIMIT TO (Fedex_Rates_IP)
        FROM SERVER svr_fedex INTO public OPTIONS (launder_column_names 'false');

SELECT * FROM fedex_rates_ip;
```

Before

```
 fid |      Type      | Weight |  Zone A  |  Zone B  |  Zone C  |  Zone D  |  Zone E  |  Zone F  | Z
-----+----------------+--------+----------+----------+----------+----------+----------+----------+--
   2 | IntlPriority |      0 |   40.25 |    41.5 |      43 |   54.75 |   116.5 |      52 |
   3 | IntlPriority |     -1 |   66.25 |   67.75 |   62.25 |   74.25 |     132 |      68 |
   4 | IntlPriority |     -2 |   70.25 |    73.5 |   65.75 |   77.25 |  156.25 |      73 |
```

```
-- unpivot a subset of columns and keep others (requires CREATE EXTENSION hstore;
WITH fkv AS (
SELECT f."Type" As type, f."Weight" As weight,
        each(hstore(f) - '{fid,Type,Weight}'::text[]) AS kv
from fedex_rates_ip AS f)
SELECT type, weight, (kv).key AS zone, (kv).value::numeric As price
FROM fkv;
```

After

```
     type      | weight |        zone        |  price
---------------+--------+--------------------+---------
 IntlPriority |      0 | Zone A             |   40.25
 IntlPriority |      0 | Zone B             |    41.5
 IntlPriority |      0 | Zone C             |      43
 IntlPriority |      0 | Zone D             |   54.75
:
```

EVEN CSV FILES

You can point at a single CSV file or a whole folder of CSV files. Each file is considered a table.

Folder of CSV files

```
CREATE SERVER svr_census FOREIGN DATA WRAPPER ogr_fdw
OPTIONS (datasource 'C:/fdw_data/census',
 format 'CSV'
);

IMPORT FOREIGN SCHEMA ogr_all
FROM SERVER svr_census INTO public;
```

Single file

```
CREATE SERVER svr_census_income FOREIGN DATA WRAPPER ogr_fdw
OPTIONS (datasource 'C:/fdw_data/census/income.csv',
 format 'CSV'
);

IMPORT FOREIGN SCHEMA ogr_all
FROM SERVER svr_census_income INTO public;
```

EVEN OTHER RELATIONAL DATABASES

Format for SQL Server 'ODBC:your_user/your_password@yourDSN,table1,table2'. ODBC can be slow with a lot of tables (more than 150) so filter list if you have over 200 tables

```
CREATE SERVER svr_sqlserver FOREIGN DATA WRAPPER ogr_fdw
OPTIONS (datasource 'ODBC:pguser/whatever@MSSQLTest,dbo.IssueLog,dbo.IssueNotes',
 format 'ODBC'
);
CREATE SCHEMA IF NOT EXISTS ss;
IMPORT FOREIGN SCHEMA "dbo."
        FROM SERVER svr_sqlserver INTO ss;
```

```
\dE ss.*
```

```
                  List of relations
 Schema |       Name       |      Type      |  Owner
--------+------------------+----------------+----------
 ss     | dbo_issuelog     | foreign table  | postgres
 ss     | dbo_issuenotes   | foreign table  | postgres
(2 rows)
```

FIN

BUY OUR BOOKS

HTTP://WWW.POSTGIS.US