TOP 10 PROBLEMS SOLVED BY POSTGIS

LEO HSU AND REGINA OBE

lr@pcorp.us Consulting

Buy our books! at http://www.postgis.us/page_buy_book

BOOK IN PROGRESS: PGROUTING: A PRACTICAL GUIDE HTTP://LOCATEPRESS.COM/PGROUTING

locate PRESS

Open Source "Geo" Books & Training

1. FIND N-CLOSEST PLACES (KNN)

Given a location, find the N-Closest places.

EXAMPLE N-CLOSEST USING GEOGRAPHY DATA TYPE

Closest 5 Indian restaurants to here

```
SELECT name, other_tags->'amenity' As type,
    ST_Point(-73.988697, 40.69384)::geography <-> geog As dist
FROM brooklyn_pois As pois
WHERE other_tags @> 'cuisine=>indian'::hstore
ORDER BY dist
LIMIT 5;
```

```
           name             |    type    |       dist
----------------------------+------------+------------------
 Asya                       | restaurant |  704.78880769187
 Desi Express (food truck)  | fast_food  | 2071.71309417315
 Joy Indian Restaurant      | restaurant | 2108.03043091333
 Bombay Cuisine             | restaurant | 2170.82610386014
 Diwanekhaas                | restaurant | 2407.92883192109
(5 rows)
```

2. 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

Things within 1000 meters from a location. This will work for PostGIS 1.5+

```
SELECT name, other_tags->'amenity' As type,
        ST_Distance(pois.geog,ref.geog) As dist_m
FROM brooklyn_pois AS pois,
        (SELECT ST_Point(-73.988697, 40.69384)::geography) As ref(geog)
WHERE other_tags @> 'cuisine=>indian'::hstore
    AND  ST_DWithin(pois.geog, ref.geog, 1000)
ORDER BY dist_m;
```

```
           name            |    type    |    dist_m
---------------------------+------------+--------------
 Asya                      | restaurant |  704.31393886
(1 row)
```

3. CONTAINMENT

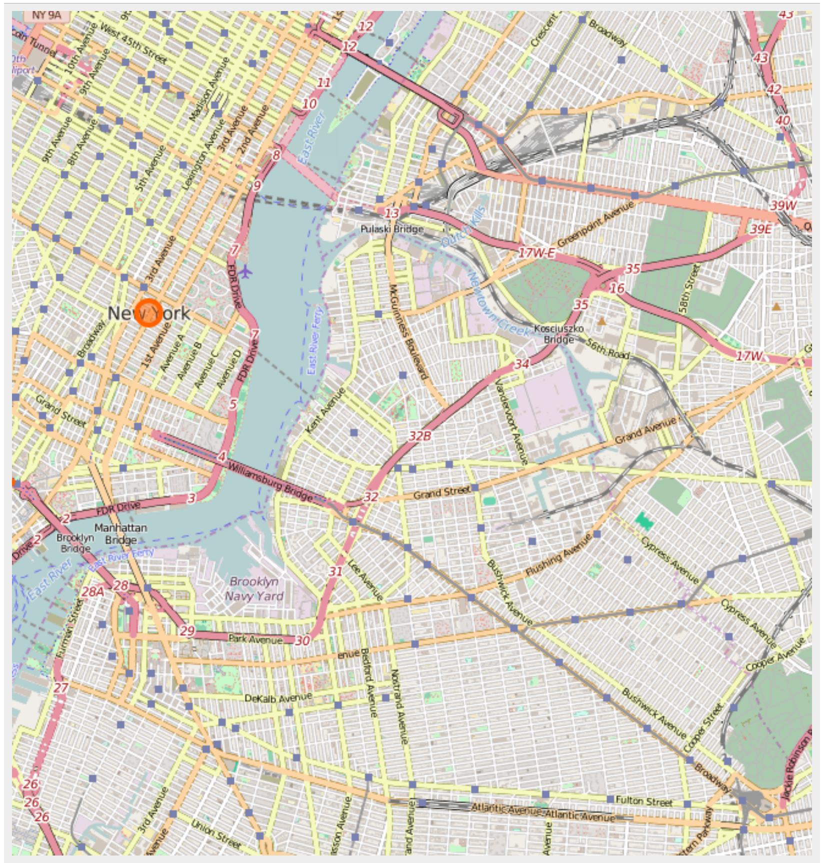
Commonly used for political districting and aggregating other pertinent facts. E.g. How many people gave to political campaigns in 2013 and what was the total per boro ordering by most money.

```
SELECT c.boro_name, COUNT(*) As num, SUM(amount) As total_contrib
FROM ny_campaign_contributions As m INNER JOIN nyc_boros As c ON ST_Covers(c.geom
GROUP BY c.boro_name
ORDER BY total_contrib DESC;
```

```
  boro_name    | num  | total_contrib
---------------+------+---------------
 Manhattan     | 4872 |    4313803.55
 Queens        | 3751 |    1262684.36
 Brooklyn      | 2578 |    1245226.04
 Staten Island |  813 |     248284.47
 Bronx         |  999 |     219805.02
(5 rows)
```

4. MAP TILE GENERATION

Common favorite for generation tiles from OpenStreetMap data. Check out TileMill which reads PostGIS vector data and can generate tiles. Various loaders to get that OSM data into your PostGIS database: osm2pgsql, imposm, GDAL.



5. FEED DATA TO MAPS IN VARIOUS VECTOR FORMATS

GeoJSON, KML, SVG, and TWB (a new light-weight binary form in PostGIS 2.2). GeoJSON commonly used with Javascript Map frameworks like OpenLayers and Leaflet.

```
SELECT row_to_json(fc)
FROM ( SELECT 'FeatureCollection' As type,
array_to_json(array_agg(f)) As features
FROM (SELECT 'Feature' As type
, ST_AsGeoJSON(ST_Transform(lg.geom,4326))::json As geometry
, row_to_json((SELECT l
FROM (SELECT route_shor As route, route_long As name) As l
  )) As properties
FROM nyc_subway As lg   ) As f )  As fc;
```

6. 3D VISUALIZATION FOR SIMULATION

X3D useful for rendering PolyhedralSurfaces and Triangular Irregulated Networks (TINS), PolyHedralSufaces for things like buildings. TINS for Terrain

Checkout https://github.com/robe2/node_postgis_express built using NodeJS and http://www.x3dom.org (X3D in html 5)

Use 3D bounding box &&& operator and form a 3D box filter

```
SELECT string_agg('<Shape><Appearance>
  <ImageTexture url=''"images/'  ||
    use
   || '.jpg"'' />
  </Appearance>'
    || ST_AsX3D(geom)  || '</Shape>', '')
FROM data.boston_3dbuildings
WHERE geom &&& ST_Expand(
    ST_Force3D(
        ST_Transform(
            ST_SetSRID(
                ST_Point(-71.0596787, 42.3581945),4326),2249)
            ),
               1000);
```

X3Dom with texture

7. 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 hstore also to show fields

```
SELECT *
FROM each(hstore(standardize_address('us_lex', 'us_gaz','us_rules'
, '29 Fort Greene Pl',
  'Brooklyn, NY 11217' )))
WHERE value > '';
```

```
    key     |    value
-----------+-------------
 city       | BROOKLYN
 name       | FORT GREENE
 state      | NEW YORK
 suftype    | PLACE
 postcode   | 11217
 house_num  | 29
(6 rows)
```

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

```
SELECT *
FROM each(hstore(standardize_address('tiger.pagc_lex',
        'tiger.pagc_gaz',
        'tiger.pagc_rules', '29 Fort Greene Pl',
  'Brooklyn, NY 11217' )))
WHERE value > '';
```

```
   key     |    value
-----------+-------------
 city      | BROOKLYN
 name      | FORT GREENE
 state     | NY
 suftype   | PL
 postcode  | 11217
 house_num | 29
(6 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('29 Fort Greene Pl, Brooklyn, NY 11217',1);
```

```
              address                |       lon        |       lat        | rating
-------------------------------------+------------------+------------------+--------
 29 Fort Greene Pl, New York, NY 11217 | -73.976819945824 | 40.6889624828967 |      8
(1 row)
```

REVERSE GEOCODING

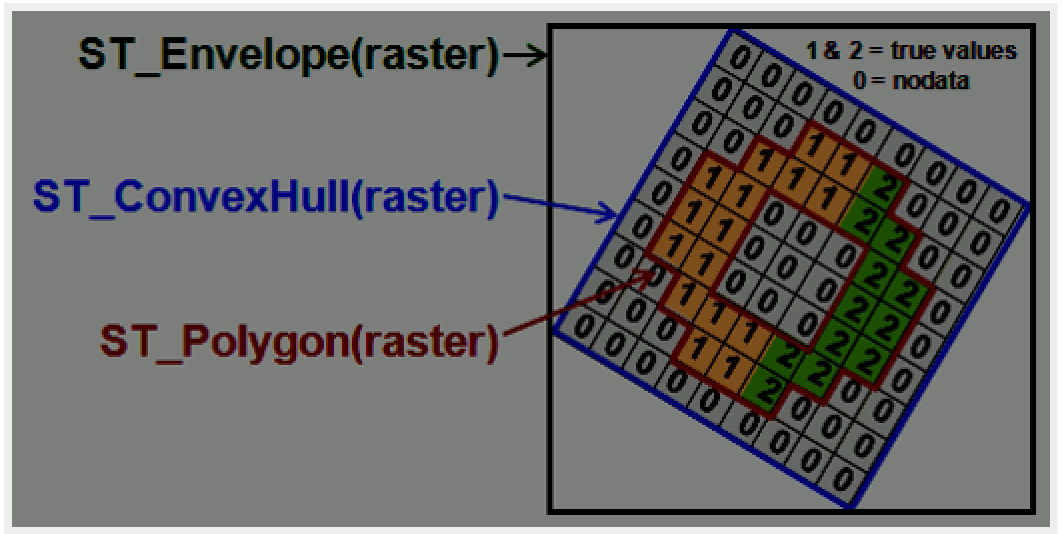
Given a longitude/latitude or GeoHash, give a textual description of where that is. Using postgis_tiger_geocoder reverse_geocode function

```
SELECT pprint_addy(addrs) AS padd,
  array_to_string(r.street,',') AS cross_streets
FROM reverse_geocode(ST_Point(-73.9768,40.689)) AS r
    , unnest(r.addy) As addrs;
```

```
                padd                  |    cross_streets
--------------------------------------+----------------------
 29 Fort Greene Pl, New York, NY 11217 | Dekalb Ave,Fulton St
(1 row)
```

8. RASTER: ANALYZE ENVIRONMENT

- Elevation
- Soil
- Weather



GIVE ME ELEVATION, TEMPERATURE, POLLUTION LEVEL AT SOME LOCATION

```
SELECT
  ST_Value(rast, 1, geom) As elev
FROM dems CROSS JOIN
   ST_Transform(
     ST_SetSRID(
      ST_Point(-71.09453, 42.36006)
         ,4326)
     ,26986) As geom
WHERE ST_Intersects(rast, 1, geom);
```

DID YOU KNOW POSTGIS IS NOT JUST A GEOGRAPHIC TOOL?

9. ANAYLZE AND CHANGE YOUR PICTURES WITH SQL

Pictures are rasters. You can manipulate them in SQL using the power of PostGIS.

READING PICTURES STORED OUTSIDE OF THE DATABASE: REQUIREMENT

new in 2.2 GUCS generally set on DATABASE or system level using ALTER DATABASE SET or ALTER SYSTEM. In PostGIS 2.1 and 2.0 needed to set these as Server environment variables.

```
SET postgis.enable_outdb_rasters TO true;
SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

REGISTER YOUR PICTURES WITH THE DATABASE: OUT OF DB

You could with raster2pgsql the -R means just register, keep outside of database:

```
raster2pgsql -R c:/pics/*.jpg -F pics | psql
```

OR

```
CREATE TABLE pics(id serial primary key, rast raster, file_name text);

INSERT INTO pics(rast, file_name)
VALUES (
        ST_AddBand(
                NULL::raster,
                'C:/pics/pggroup.jpg'::text, NULL::int[]
        ), 'pgroup' ),
  (
        ST_AddBand(
                NULL::raster,
                'C:/pics/monasmall.jpg'::text, NULL::int[]
        ), 'mona' ),
         (
        ST_AddBand(
                NULL::raster,
                'C:/pics/osgeo_paris.jpg'::text, NULL::int[]
        ), 'osgeo_paris' );
```

CHECK BASIC INFO

```
SELECT file_name, ST_Width(rast) As width, ST_Height(rast) As height,
    ST_NumBands(rast) AS nbands
FROM pics;
```

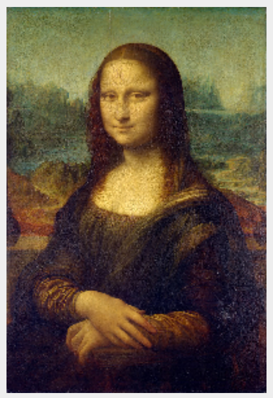
```
  file_name   | width | height | nbands
-------------+-------+--------+--------
 pgroup      |  1920 |   1277 |      3
 mona        |   800 |   1192 |      3
 osgeo_paris |  2048 |   1365 |      3
(3 rows)
```

RESIZE THEM AND DUMP THEM BACK OUT

This uses PostgreSQL large object support for exporting. Each picture will result in 4 pictures of 25%, 50%, 75%, 100% of original size

```
DROP TABLE IF EXISTS tmp_out ;
-- write to lob and store the resulting oids of lobs in new table
CREATE TABLE tmp_out AS
SELECT loid, lowrite(lo_open(loid, 131072),png) As num_bytes, file_name, p
FROM (
SELECT file_name, lo_create(0) AS loid,
ST_AsPNG(ST_Resize(rast, p*0.25, p*0.25)) AS png, p
FROM pics , generate_series(1,4) As p ) As f;

-- export to file system
SELECT lo_export(loid, 'C:/temp/' || file_name || '-' || p::text || '.png')
FROM tmp_out;

--delete lobs
SELECT lo_unlink(loid)
FROM tmp_out;
```

25% resized images



mona-1.png

osgeo_paris-1.png

pggroup-1.png

DO TONS OF OPERATIONS IN ONE SQL

This will do lots of crazy combo stuff using raster and geometry functions that merges all pictures into one. 12 secs

```sql
SELECT string_agg(file_name,'-') As file_name,
  ST_AsJPEG(
 ST_Resize(ST_Union(
  ST_SetUpperLeft(
    ST_Clip(rast,
        ST_Buffer(ST_Centroid(rast::Geometry), 1000) ),0,0), 'MAX'),
    0.5,0.5) ) AS jpg
FROM pics;
```

THE RESULT IS A BIT GHOSTLY

CREATE A NEW CURRENCY

```
SELECT
    ST_AsPNG(
         ST_Aspect( ST_Resize(
                ST_Clip(rast,
        ST_Buffer(ST_Centroid(rast::Geometry), (ST_Width(rast)/3)::integer  )
                ),
    0.8,0.8, algorithm := 'Lanczos')
        , 1,   '8BUI'   )

    ) AS png
FROM pics;
```

PostGIS OSGeo **PostgreSQL Group**

10. MANAGE DISCONTINUOUS DATE TIME RANGES WITH POSTGIS

A linestring can be used to represent a continous time range (using just X axis). A multi-linestring can be used to represent a related list of discontinous time ranges. PostGIS has hundreds of functions to work with linestrings and multilinestrings.

HELPER FUNCTION FOR CASTING LINESTRING TO DATE RANGES

```
CREATE FUNCTION to_daterange(x geometry)
  RETURNS daterange AS
$$
DECLARE

    y daterange;
    x1 date;
    x2 date;

BEGIN

    x1 = CASE WHEN ST_X(ST_StartPoint(x)) = 2415021 THEN '-infinity' ELSE 'J' || ST_X(ST_StartPoint(x
    x2 = CASE WHEN ST_X(ST_ENDPoint(x)) = 2488070 THEN 'infinity' ELSE 'J' || ST_X(ST_EndPoint(x)) EN

    y = daterange(x1, x2, '[)');

    RETURN y;
END;
$$
  LANGUAGE plpgsql IMMUTABLE;
```

HELPER FUNCTION FOR CASTING DATE RANGE TO LINESTRING

```
CREATE FUNCTION to_linestring(x daterange)
  RETURNS geometry AS
$$
DECLARE
    y geometry(linestring);
    x1 bigint;
    x2 bigint;

BEGIN

    x1 = to_char(CASE WHEN lower(x) = '-infinity' THEN '1900-1-1' ELSE lower(x) END, 'J')::bigint;
    x2 = to_char(CASE WHEN upper(x) = 'infinity' THEN '2100-1-1' ELSE upper(x) END, 'J')::bigint;

    y = ST_GeomFromText('LINESTRING(' || x1 || ' 0,' || x2 || ' 0)');

    RETURN y;
END;
$$
  LANGUAGE plpgsql IMMUTABLE;
```

COLLAPSING OVERLAPPING DATE RANGES

Result is single linestring which maps to date range

```
SELECT id,
 to_daterange(
        (ST_Dump(
                ST_Simplify(ST_LineMerge(ST_Union(to_linestring(period))),0))
        ).geom)
FROM (

    VALUES
        (1,daterange('1970-11-5'::date,'1980-1-1','[)')),
        (1,daterange('1990-11-5'::date,'infinity','[)')),
        (1,daterange('1975-11-5'::date,'1995-1-1','[)'))
) x (id,  period)

GROUP BY id;
```

```
 id |      to_daterange
----+------------------------
  1 | [1970-11-05,infinity)
(1 row)
```

COLLAPSING DISCONTINUOUS/OVERLAPPING DATE RANGES

Result is a multi-linestring which we dump out to get individual date ranges

```
SELECT id,
 to_daterange(
        (ST_Dump(
                ST_Simplify(ST_LineMerge(ST_Union(to_linestring(period))),0))
        ).geom)
FROM (

    VALUES
        (1,daterange('1970-11-5'::date,'1975-1-1','[)')),
        (1,daterange('1980-1-5'::date,'infinity','[)')),
        (1,daterange('1975-11-5'::date,'1995-1-1','[)'))
) x (id, period)
GROUP BY id;
```

```
 id |      to_daterange
----+------------------------
  1 | [1970-11-05,1975-01-01)
  1 | [1975-11-05,infinity)
(2 rows)
```

COLLAPSING CONTIGUOUS DATE RANGES

Result is a linestring which we dump out to get individual date range

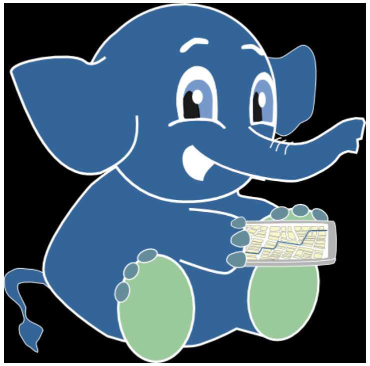
```
SELECT id,
 to_daterange(
        (ST_Dump(
                ST_Simplify(ST_LineMerge(ST_Union(to_linestring(period))),0))
        ).geom)
FROM (

    VALUES
        (1,daterange('1970-11-5'::date,'1975-1-1','[)')),
        (1,daterange('1975-1-1'::date,'1980-12-31','[)')),
        (1,daterange('1980-12-31'::date,'1995-1-1','[)'))
) x (id,  period)
GROUP BY id;
```

```
 id |      to_daterange
----+-------------------------
  1 | [1970-11-05,1995-01-01)
(1 row)
```

BONUS: ROUTING WITH PGROUTING

Finding least costly route along constrained paths like roads, airline routes, the vehicles you have in hand, pick-up / drop-off constraints.

Buy our upcoming book (Preview Edition available) pgRouting: A Practical Guide http://locatepress.com/pgrouting



to find out more

LINKS OF INTEREST

- PostGIS
- Planet PostGIS

THE END

THANK YOU. BUY OUR BOOKS

HTTP://WWW.POSTGIS.US