POSTGIS, PGROUTING, AND OTHER SPATIAL EXTENSIONS

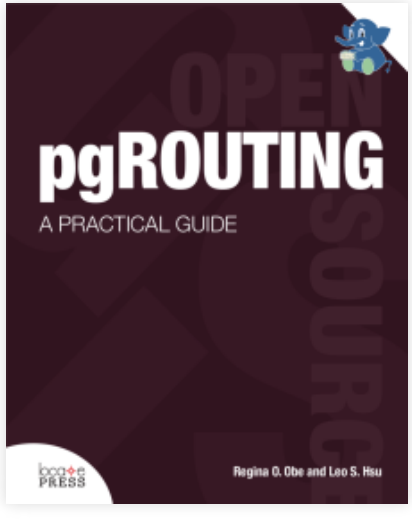
REGINA OBE AND LEO HSU

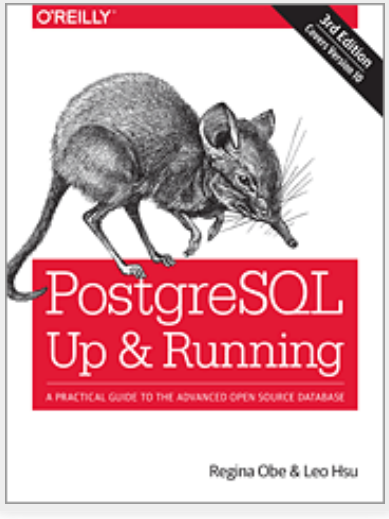
http://www.paragoncorporation.com

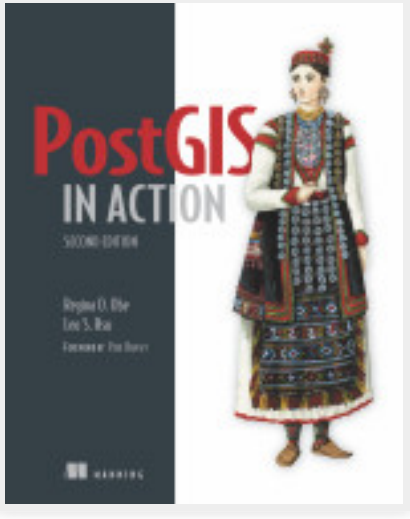
Buy our books! at http://postgis.us/page_buy_book

OUR LATEST BOOK

pgRouting: A Practical Guide http://locatepress.com/pgrouting

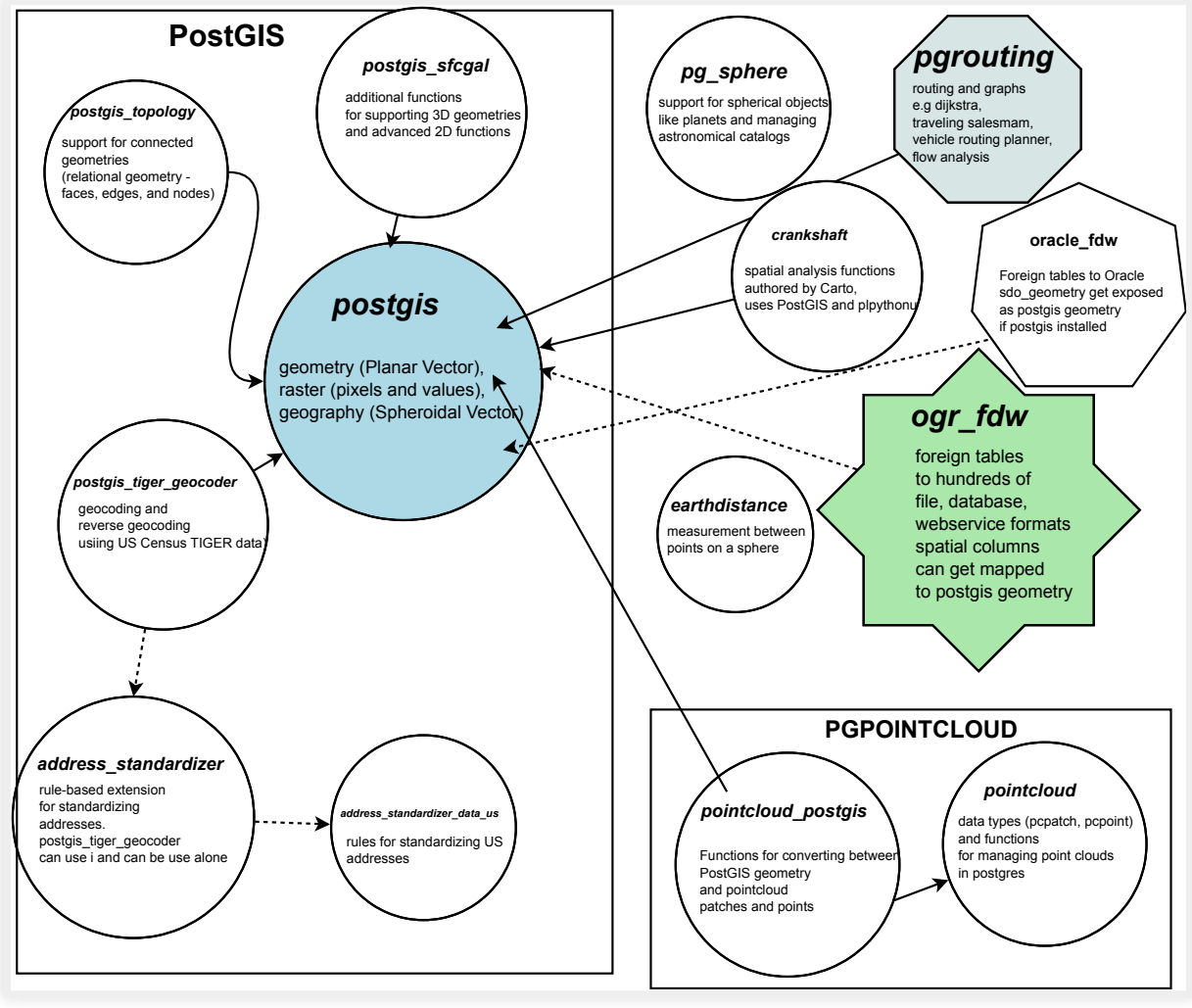






WORLD OF POSTGRESQL SPATIAL EXTENSIONS

CREATE EXTENSION ...;



WHAT IS POSTGIS?

Spatial types: geometry, geography, raster, topogeometry (via postgis_topology)

Lots of spatial functions: over 400 in core postgis extension for proximity, time proximity, linear referencing, spatial aggregation and other kinds of geometry processing, raster -> vector conversion, vector to raster conversion, raster map algebra and other raster analysis.

POSTGIS CAN TRANSFORM COORDINATE SYSTEMS

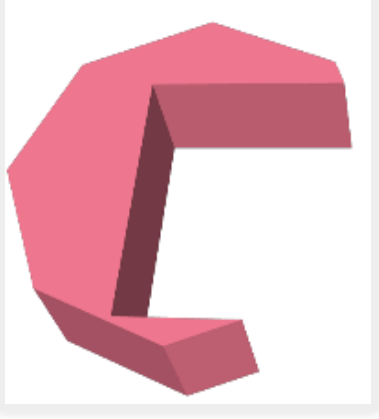
PostGIS allows you to transform between coordinate systems.

- **spatial_ref_sys** table - This is how PostGIS knows how to transform geometric coordinates from one spatial system to another.
- **ST_Transform** function - the function that transforms one set of coordinates to another spatial reference system coordinate space.
- ST_SRID, ST_SetSRID - sets the meta data on a geometry/raster
- geography type coordinate are always in degrees and to get measurements in meters, some planetary spheroid assumption is required looked up in spatial_ref_sys.

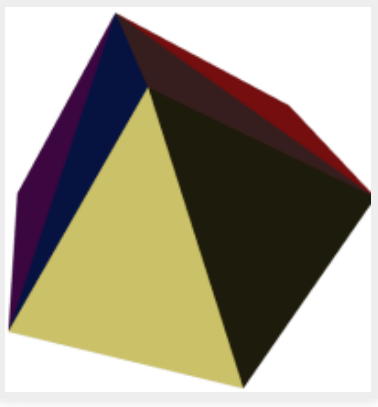
POSTGIS GEOMETRY TYPE

The flat space model. The world is a cartesian grid. Supports drawing of linestrings, polygons, 3D polygons, points, 3d points, collections of polygons, points, linestrings, Polyhedral Surfaces, and TINS

Basic geometric types



Polyhedral Surface



Triangulated Irregular Network (TIN)

POSTGIS GEOGRAPHY TYPE

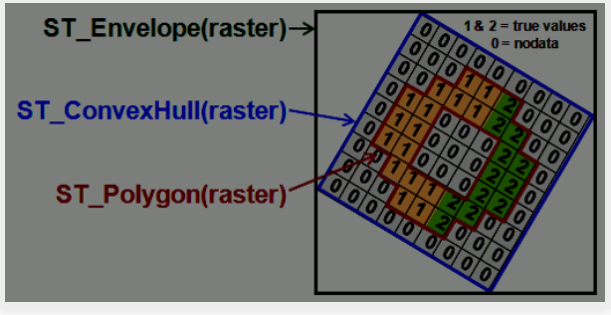
Model of space as spheroid. Takes into consideration the earth or any given planet whose spatial reference is defined in *spatial_ref_sys* table.

GEODETIC (GEOGRAPHY) 4326 (WGS 84 LON LAT) IN GEOGRAPHY

POSTGIS RASTER TYPE

Model of space as a numeric matrix (with cells (called pixels) that have values (on) or don't have values (off))

- Elevation
- Soil
- Weather
- Fire



RASTERS HAVE BANDS

Bands / Channels -- correspond to the matrices in raster. For example an RGB picture has 3 matrices.

Band pixel value types: 1BB (boolean), [2, 4, 8, 16, 32]BUI (bit unsigned integer), [8, 16, 32]BSI (bit signed integer), [32,64] BF (bit float)

Original raster

ST_Band(rast, '{2,1,1}'::int[])

ST_Band(rast,3)

RASTER TYPE AND GEOMETRY TYPE INTEROPERATE

```
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)

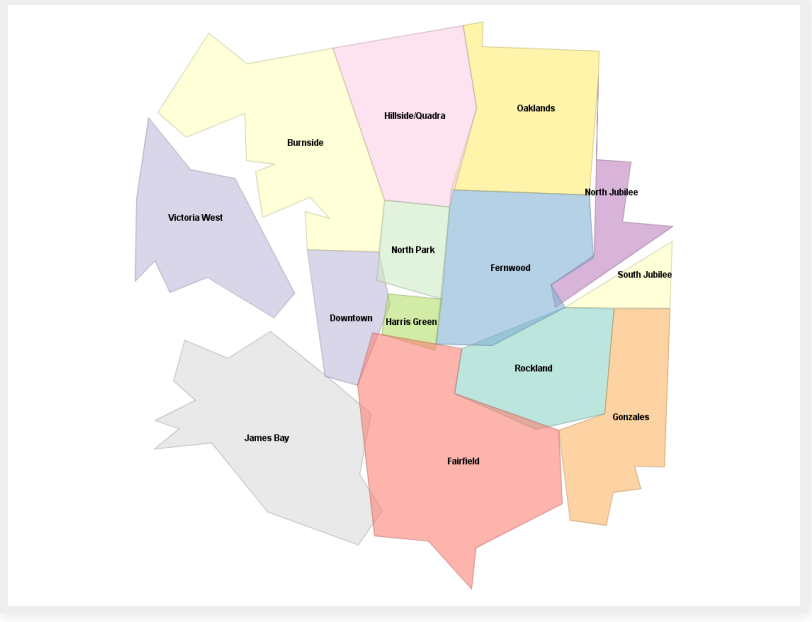
Using o_4_aerials resize 0.2, 2000 ft - 5.7 secs

POSTGIS TOPOGEOMETRY TYPE

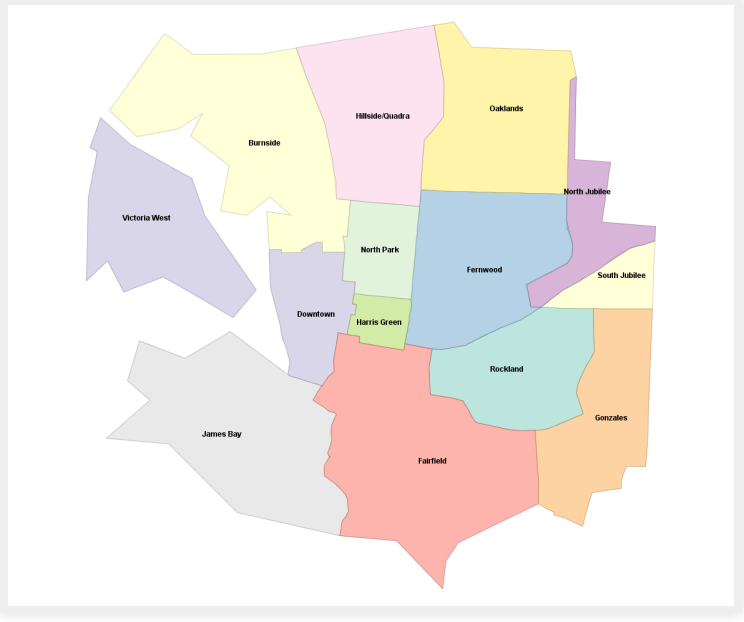
The topogeometry is how topology represents a geometry. It is a reference to a bunch of edges, nodes, and faces that make up the geometry.

```
CREATE EXTENSION postgis_topology;
```

Geometry simplification



Topogeometry simplification



ADDRESS STANDARDIZATION / GEOCODING / REVERSE GEOCODING

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

```
CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION address_standardizer;
CREATE EXTENSION address_standardizer_data_us;
```

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'
, 'One Seaport Lane',
  'Boston, Massachusetts 02210' )))
WHERE value > '';
```

```
    key     |     value
------------+---------------
 house_num  | 1
 name       | SEAPORT
 suftype    | LANE
 city       | BOSTON
 state      | MASSACHUSETTS
 postcode   | 02210
(6 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'
, 'One Seaport Lane',
  'Boston, Massachusetts 02210' )))
WHERE value > '';
```

```
    key     |  value
------------+---------
 house_num | 1
 name      | SEAPORT
 suftype   | LN
 city      | BOSTON
 state     | MA
 postcode  | 02210
(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('1 Seaport Lane, Boston, MA 02210',1);
```

```
            address              |        lon         |       lat         | rating
---------------------------------+--------------------+-------------------+--------
 1 Seaport Ln, Boston, MA 02210 | -71.0411493412951 | 42.3497520198983 |      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(-71.04115,42.34975)) AS r
    , unnest(r.addy) As addrs;
```

```
             padd               | cross_streets
--------------------------------+---------------
 Northern Ave, Boston, MA       | Seaport Ln
 5 Seaport Ln, Boston, MA 02210 | Seaport Ln
(2 rows)
```

WHAT ARE POINT CLOUDS?

A cloud of points where each point can have many numeric attribute associated with it. It is collected by satellites, drones, and planes and used to develop the other forms of data raster and vector. Most popular format is LiDAR (Light detection and Ranging).

HOW TO STORE POINTCLOUDS IN POSTGRESQL

PostGIS Bundle for windows (EDB) includes this extension. Will install both pointcloud and pointcloud_postgis as well as postgis if it isn't already installed.

```
CREATE EXTENSION pointcloud SCHEMA postgis;
CREATE EXTENSION pointcloud_postgis SCHEMA postgis;
```

POSTGRESQL + GDAL (OGR) ~ POSTGIS = OGR_FDW POSTGRESQL FOREIGN DATA WRAPPER

Doesn't require PostGIS to use, but will expose spatial columns as PostGIS geometry if PostGIS is installed.

USE OGR_FDW EXTENSION

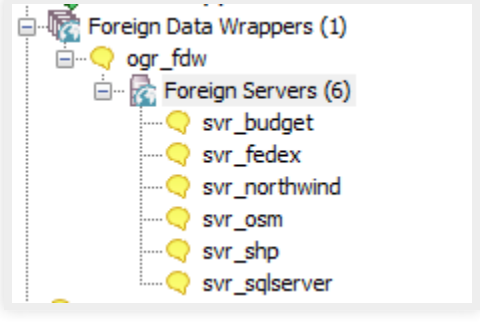
If you have all sorts of data of both a spatial and non-spatial flavor to tame, make sure you have ogr_fdw foreign data wrapper in your tool belt.

- For windows users using EDB distribution, it's part of PostGIS bundle (versions 2.2 and up) on application stackbuilder.
- For windows/linux/mac desktop users, it's part of the BigSQL PostGIS package.
- 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 GREAT?

You have the combined power of Geospatial Data Abstraction Layer (GDAL), PostgreSQL, and any PostgreSQL extension you want (including PostGIS) 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/jsonb to tame your data.

- Spreadsheets
- ODBC datasources
- Other relational
- OSM files (OSM, PBF)
- Dbase files
- ESRI Shapefiles
- Spatial web services
- Many more



ENABLE IT IN YOUR DATABASE

```
CREATE EXTENSION ogr_fdw;
```

OTHER RELATIONAL DATABASES

Format for SQL Server ODBC '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)
```

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/massgis/shps',
 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   | biketrails_arc       | foreign table | postgres
 shps   | towns_arc            | foreign table | postgres
 shps   | towns_poly           | foreign table | postgres
 shps   | towns_poly_areacode  | foreign table | postgres
 shps   | towns_polym          | foreign table | postgres
 shps   | towns_pop            | foreign table | postgres
 shps   | zipcodes_nt_poly     | foreign table | postgres
(7 rows)
```

SPREADSHEETS

Each workbook is considered a server and each sheet a table

```
CREATE SERVER svr_currency_rates
   FOREIGN DATA WRAPPER ogr_fdw
  OPTIONS (datasource '/fdw_data/ExchangeRates.xlsx',format 'XLSX',
    config_options 'OGR_XLSX_HEADERS=FORCE');
CREATE SCHEMA staging;
-- link only 2 spreadsheets preserve headers
IMPORT FOREIGN SCHEMA ogr_all  LIMIT TO (EUR, USD)
        FROM SERVER svr_currency_rates INTO staging
        OPTIONS (launder_column_names 'false');
```

Before `SELECT * FROM staging.usd;`

```
 fid |    Date    |   AED   |   ARS   |   AUD   |   BGN   |   GBP    |   CAD   |  CNY   |
-----+------------+---------+---------+---------+---------+----------+---------+--------+---
   2 | 2017-11-27 |  3.6729 | 17.3012 |  1.3147 |  1.6418 | 0.749935 | 1.27446 | 6.5989 | 0
   3 | 2017-11-28 |  3.6728 | 17.3421 | 1.31255 |  1.6462 | 0.755895 | 1.28042 | 6.6074 |
   4 | 2017-11-29 |  3.6729 | 17.4461 | 1.32052 |  1.6489 |  0.74437 |  1.2855 | 6.6145 | 0
   5 | 2017-11-30 | 3.67295 | 17.3674 | 1.32005 | 1.64275 |  0.74061 | 1.28896 | 6.6133 | 0
   6 | 2017-12-01 |   3.673 | 17.2453 | 1.31105 |  1.6413 | 0.739945 |  1.2702 | 6.6162 | 0
:
```

```
-- unpivot
SELECT f."Date" As date, 'USD' AS  from_cur,
    j.key AS to_cur, j.val::numeric As cur_rate
FROM staging.usd AS f, jsonb_each_text(to_jsonb(f)) AS j(key,val)
WHERE j.key NOT IN('fid', 'Date');
```

After

```
   date    | from_cur | to_cur | cur_rate
-----------+----------+--------+----------
2017-11-27 | USD      | AED    |   3.6729
2017-11-27 | USD      | ARS    |  17.3012
2017-11-27 | USD      | AUD    |   1.3147
2017-11-27 | USD      | BGN    |   1.6418
:
```

PGROUTING LIVE DEMOS

POSTGIS GROWS WITH POSTGRESQL

Memorable recent moments of change in PostgreSQL that elevated PostGIS.

- Aggregate ORDER BY clause introduced in PostgreSQL 9.0
- Extension model in PostgreSQL 9.1
- LATERAL support in PostgreSQL 9.3 added
- Lots more things depend on PostGIS
- True K-Nearest Neighbor (KNN) support for GIST in PostgreSQL 9.5/ PostGIS 2.2
- Parallelism got added in PostgreSQL 9.6, functions such as ST_Intersects took advantage in 2.4, coming in PostGIS 2.5 ST_AsMVT aggregate function revised to allow parallel splitting.
- JIT Support coming in PostgreSQL 11, PostGIS 2.5.0alpha has JIT support.

POSTGIS AGGREGATES GET ELEVATED WITH ORDER BY IN 9.0

Before aggregate ORDER BY

```
SELECT vehicle, ST_MakeLine(geom) As path
FROM (SELECT geom, vehicle FROM gps_points ORDER BY vehicle, gps_time) AS f
GROUP BY vehicle;
```

After aggregate ORDER BY

```
SELECT vehicle, ST_MakeLine(geom ORDER BY gps_time) As path
FROM gps_points
GROUP BY vehicle;
```

POSTGIS EMBRACES THE EXTENSION MODEL IN 9.1

Before CREATE EXTENSION

Find where your package install put the PostGIS scripts.

After CREATE EXTENSION

Install in your database

```
CREATE EXTENSION postgis;
```

Upgrade

```
ALTER EXTENSION postgis UPDATE;
```

LATERAL SIMPLIFIES POSTGIS QUERIES

Before LATERAL

```
SELECT p1.name, (gp).geom,
(gp).path[1] As line_index, (gp).path[2] As pt_index
FROM (SELECT name, ST_DumpPoints(geom) AS gp
FROM roads) As p1;
```

After LATERAL

```
SELECT p1.name, gp.geom,
gp.path[1] As line_index, gp.path[2] As pt_index
FROM roads As p1, ST_DumpPoints(p1.geom) AS gp;
```

LOTS OF THINGS DEPEND ON POSTGIS

Restores fail.

Partial Solution: PostGIS changes from being a relocatable extension to being non-relocatable in 2.3

- Indexes and Constraints depend on PostGIS, restore of these fail, which makes data restore fail
- Materialized views require PostGIS, restore fails
- Other Extensions require PostGIS, restore fails
- Logical decoding requires PostGIS, logical decoding fails
- Foreign tables require PostGIS

TRUE KNN SIMPLIFIES POSTGIS QUERIES

Before 9.5 / PostGIS 2.2 (no true KNN)

2 closest roads to each point of interest

```
WITH c AS ( SELECT p.name AS p_name, r.name AS r_name, ST_Distance(p.geom, r.geom
FROM pois AS p INNER JOIN roads AS r ON ST_DWithin(p.geom, r.geom, 50) ),
c2 AS (SELECT p_name, r_name, dist, ROW_NUMBER() OVER(PARTITION BY p_name ORDER B
FROM c2)
SELECT p_name, r_name, dist
FROM c2 WHERE rn < 3;
```

After True KNN with LATERAL

```
SELECT p.name AS p_name, r.name AS r_name, r.dist
FROM pois AS p,
    LATERAL (SELECT r.name, r.geom <-> p.geom AS dist
        FROM roads ORDER BY dist LIMIT 2) AS r;
```

POSTGRESQL 9.6 ADDS PARALLEL SUPPORT

Queries didn't change, but some got a lot faster. Like ST_Intersects checks could be done in parallel. PostGIS 2.5.0alpha recently released with first parallel aggregate ST_AsMVT. Can build Mapbox Vector Tiles in parallel mode and recombine the final output.

POSTGRESQL 11 ADDS JIT, POSTGIS 2.5 ADDS JIT

Checkout PostGIS 2.5.0 alpha released recently. Still some code to be committed.

https://trac.osgeo.org/postgis/ticket/4060

FIN

BUY OUR BOOKS

HTTP://WWW.POSTGIS.US