PostGIS
PostgreSQL Spatial

Javier Pastorino

September 27th, 2017

BIG Data Management and Mining Laboratory
UNIVERSITY OF COLORADO DENVER | ANSCHUTZ MEDICAL CAMPUS

Outline

- PostgreSQL and Spatial Extensions (PostGIS)
- Server Installation
- Loading Spatial Data to the Database
- Spatial Data Types
- Spatial Functions
- Indexing Spatial Data
- Spatial Queries
- Visualizing the Results
  - Connecting QGIS to PostGIS
  - Using Views for QGIS Layers
  - Filtering Data in QGIS

PostgreSQL and Spatial Extensions

- What is PostgreSQL
  - PostgreSQL is an **object-relational database** management system (ORDBMS)
  - Developed at the **UC at Berkeley** Computer Science Department.
- What is PostGIS
  - PostGIS is a **spatial database extender** for PostgreSQL object-relational database.
  - It adds support for geographic objects allowing location queries to be run in SQL.
  - PostGIS was developed by **Refractions Research Inc**., as a spatial database technology research project
  - Refractions is a GIS and database consulting company in Victoria, British Columbia, Canada, specializing in data integration and custom software development.
  - PostGIS is delivered in PostgreSQL Extensions.
- What is a PostgreSQL Extension
  - An Extension in PostgreSQL is like a package in java.
  - Basically, is a SQL script which defines all the necessaries objects in the database to create the new functionality.
  - Objects can include data types, functions & procedures, tables, etc.

Server Installation

- PostGIS is available Windows/Mac/Linux.
- The installation process is different in each platform.
- For Ubuntu, it is included in the distribution packages and can be installed by **atp**
  - `sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt xenial-pgdg main" >> /etc/apt/sources.list'`
  - `wget --quiet -O - http://apt.postgresql.org/pub/repos/apt/ACCC4CF8.asc | sudo apt-key add -`
  - `sudo apt-get update`
  - `sudo apt-get install postgresql-9.6`
  - `sudo apt-get install postgresql-9.6-postgis-2.3 postgresql-contrib-9.6 postgresql-9.6-postgis-scripts`
  - `sudo apt-get install postgis  #to get the commandline tools shp2pgsql, raster2pgsql`
- Adding a PostGIS to a Database
  - `sudo -u postgres psql`
  - `CREATE EXTENSION adminpack;`
  - `CREATE DATABASE gisdb;`
  - `\connect gisdb;`
  - `CREATE SCHEMA postgis;`
  - `ALTER DATABASE gisdb SET search_path=public, postgis, contrib;`
  - `\connect gisdb;`
  - `CREATE EXTENSION postgis SCHEMA postgis;`

http://trac.osgeo.org/postgis/wiki/UsersWikiPostGIS23UbuntuPGSQL96Apt

Loading Spatial Data

- We can use external tools to fast load/extract data from shape files.
- **ogr2ogr**
  - Converts simple features data between file formats performing various operations during the process such as spatial or attribute selections, reducing the set of attributes, setting the output coordinate system or even re-projecting the features during translation
  - `ogr2ogr -f "PostgreSQL" PG:"host=<> port=5432 dbname=<> user=<> password=<>" <shpFolder>  -progress. --config PG_USE_COPY YES`
  - `ogr2ogr -f "PostgreSQL" PG:"host=localhost port=5432 dbname=gisdb user=postgres password=Postgres" tlgdb_2016_a_us_school.gdb -progress --config PG_USE_COPY YES`
- **shp2pgsql**
  - The **shp2pgsql** data loader converts ESRI SHP files into SQL suitable for insertion into a PostGIS database.
  - `shp2pgsql [<options>] <shapefile> [[<schema>.]<table>]`
  - `shp2pgsql tl_2016_us_state.shp boston.state > boston.state.ddl.sql`
  - `psql -U postgres -d gisdb -f boston.state.ddl.sql`
  - `shp2pgsql tl_2016_us_state.shp boston.state | psql -U $USER_NAME -d $DB`
- **pgsql2shp**
  - Extract a table/query result which contains geography to a ESRI SHP File
  - `pgsql2shp [OPTIONS] database [schema.]table`
  - `pgsql2shp [OPTIONS] database query`

Spatial Data Type

- The OpenGIS specification defines two standard to express spatial objects
  - **WKB** – Well-Known Binary
  - **WKT** – Well-Known Text

```
bytea WKB = ST_AsBinary(geometry);
text WKT = ST_AsText(geometry);
geometry = ST_GeomFromWKB(bytea WKB, SRID);
geometry = ST_GeometryFromText(text WKT, SRID);
```

- POINT(0 0)
- LINESTRING(0 0,1 1,1 2)
- POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
- MULTIPOINT(0 0,1 2)
- MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))
- MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))
- GEOMETRYCOLLECTION(POINT(2 3),LINESTRING(2 3,3 4))

```
INSERT INTO geotable ( the_geom, the_name )
  VALUES ( ST_GeomFromText('POINT(-126.4 45.32)', 312), 'A Place');
```

Spatial Data Type (cont'd)

- PostGIS has also an **Extended** format for those **(EWKB & EWKT)**
  - **POINT**(0 0 0) -- XYZ
  - SRID=32632;POINT(0 0) -- XY with SRID
  - **POINTM**(0 0 0) -- XYM
  - **POINT**(0 0 0 0) -- XYZM
  - SRID=4326;**MULTIPOINTM**(0 0 0,1 2 1) -- XYM with SRID
  - **MULTILINESTRING**((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
  - **POLYGON**((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
  - **MULTIPOLYGON**(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))
  - **GEOMETRYCOLLECTIONM**(POINTM(2 3 9), LINESTRINGM(2 3 4, 3 4 5))
  - **MULTICURVE**((0 0, 5 5), CIRCULARSTRING(4 0, 4 4, 8 4))
  - **POLYHEDRALSURFACE**(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))
  - **TRIANGLE** ((0 0, 0 9, 9 0, 0 0))
  - **TIN**(((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 0 0 0)))

```
bytea EWKB = ST_AsEWKB(geometry);
text EWKT = ST_AsEWKT(geometry);
geometry = ST_GeomFromEWKB(bytea EWKB);
geometry = ST_GeomFromEWKT(text EWKT);
```

- A **Spatial Reference System Identifier** (**SRID**) is a unique value used to unambiguously identify projected, unprojected, and local spatial coordinate system definitions

Spatial Data Type (cont'd)

- **Geometry** Data Type
  - Stores spatial object using a single geometry coordinate system
  - The basis for the PostGIS geometry type is a plane. **The shortest path between two points on the plane is a straight line.**
  - That means calculations on geometries (areas, distances, lengths, intersections, etc.) can be calculated using Cartesian mathematics and straight line vectors.

```
CREATE TABLE gtest ( ID int4, NAME varchar(20) );
SELECT AddGeometryColumn('', 'gtest', 'geom', -1, 'LINESTRING', 2);

INSERT INTO gtest (ID, NAME, GEOM)
VALUES ( 1, 'First Geometry',
        ST_GeomFromText('LINESTRING(2 3,4 5,6 5,7 8)', -1)
        );
```

Spatial Data Type (cont'd)

- **Geography** Data Type
  - Stores spatial object a geodetic (lat/lon) coordinate system.
  - The basis for the PostGIS geographic type is a sphere. **The shortest path between two points on the sphere is a great circle arc.**
  - That means that calculations on geographies (areas, distances, lengths, intersections, etc.) must be calculated on the sphere. For more accurate measurements, the calculations must take the actual spheroidal shape of the world into account, and the mathematics becomes very complicated indeed.

- `CREATE TABLE testgeog(gid serial PRIMARY KEY, the_geog geography(POINT,4326) );`

- `CREATE TABLE testgeog(gid serial PRIMARY KEY, the_geog geography(POINTZ,4326));`

Spatial Data Type (cont'd)

ST_BdPolyFromText — Construct a Polygon given an arbitrary collection of closed linestrings as a MultiLineString Well-Known text representation.
ST_BdMPolyFromText — Construct a MultiPolygon given an arbitrary collection of closed linestrings as a MultiLineString text representation Well-Known text representat
ST_GeogFromText — Return a specified geography value from Well-Known Text representation or extended (WKT).
ST_GeographyFromText — Return a specified geography value from Well-Known Text representation or extended (WKT).
ST_GeogFromWKB — Creates a geography instance from a Well-Known Binary geometry representation (WKB) or extended Well Known Binary (EWKB).
ST_GeomCollFromText — Makes a collection Geometry from collection WKT with the given SRID. If SRID is not give, it defaults to -1.
ST_GeomFromEWKB — Return a specified ST_Geometry value from Extended Well-Known Binary representation (EWKB).
ST_GeomFromEWKT — Return a specified ST_Geometry value from Extended Well-Known Text representation (EWKT).
ST_GeometryFromText — Return a specified ST_Geometry value from Well-Known Text representation (WKT). This is an alias name for ST_GeomFromText
ST_GeomFromGML — Takes as input GML representation of geometry and outputs a PostGIS geometry object
ST_GeomFromGeoJSON — Takes as input a geojson representation of a geometry and outputs a PostGIS geometry object
ST_GeomFromKML — Takes as input KML representation of geometry and outputs a PostGIS geometry object
ST_GMLToSQL — Return a specified ST_Geometry value from GML representation. This is an alias name for ST_GeomFromGML
ST_GeomFromText — Return a specified ST_Geometry value from Well-Known Text representation (WKT).
ST_GeomFromWKB — Creates a geometry instance from a Well-Known Binary geometry representation (WKB) and optional SRID.
ST_LineFromMultiPoint — Creates a LineString from a MultiPoint geometry.
ST_LineFromText — Makes a Geometry from WKT representation with the given SRID. If SRID is not given, it defaults to -1.
ST_LineFromWKB — Makes a LINESTRING from WKB with the given SRID
ST_LinestringFromWKB — Makes a geometry from WKB with the given SRID.
ST_MakeBox2D — Creates a BOX2D defined by the given point geometries.
ST_3DMakeBox — Creates a BOX3D defined by the given 3d point geometries.
ST_MakeLine — Creates a Linestring from point or line geometries.
ST_MakeEnvelope — Creates a rectangular Polygon formed from the given minimums and maximums. Input values must be in SRS specified by the SRID.
ST_MakePolygon — Creates a Polygon formed by the given shell. Input geometries must be closed LINESTRINGS.
ST_MakePoint — Creates a 2D,3DZ or 4D point geometry.
ST_MakePointM — Creates a point geometry with an x y and m coordinate.
ST_MLineFromText — Return a specified ST_MultiLineString value from WKT representation.
ST_MPointFromText — Makes a Geometry from WKT with the given SRID. If SRID is not give, it defaults to -1.
ST_MPolyFromText — Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not give, it defaults to -1.
ST_Point — Returns an ST_Point with the given coordinate values. OGC alias for ST_MakePoint.
ST_PointFromText — Makes a point Geometry from WKT with the given SRID. If SRID is not given, it defaults to unknown.
ST_PointFromWKB — Makes a geometry from WKB with the given SRID
ST_Polygon — Returns a polygon built from the specified linestring and SRID.
ST_PolygonFromText — Makes a Geometry from WKT with the given SRID. If SRID is not give, it defaults to -1.
ST_WKBToSQL — Return a specified ST_Geometry value from Well-Known Binary representation (WKB). This is an alias name for ST_GeomFromWKB that takes no srid
ST_WKTToSQL — Return a specified ST_Geometry value from Well-Known Text representation (WKT). This is an alias name for ST_GeomFromText

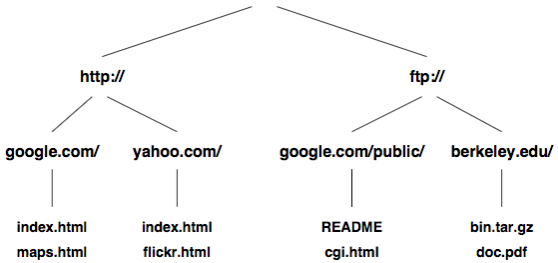
Spatial Functions

ST_Area — Returns the area of the surface if it is a polygon or multi-polygon. For "geometry" type area is in SRID units. For "geography" area is in square meters.
ST_Azimuth — Returns the angle in radians from the horizontal of the vector defined by pointA and pointB. Angle is computed clockwise from down-to-up: on the clock: 12=0; 3=PI/2; 6=PI; 9=3PI/2.
ST_Centroid — Returns the geometric center of a geometry.
ST_ClosestPoint — Returns the 2-dimensional point on g1 that is closest to g2. This is the first point of the shortest line.
ST_Contains — Returns true if and only if no points of B lie in the exterior of A, and at least one point of the interior of B lies in the interior of A.
ST_ContainsProperly — Returns true if B intersects the interior of A but not the boundary (or exterior). A does not contain properly itself, but does contain itself.
ST_Covers — Returns 1 (TRUE) if no point in Geometry B is outside Geometry A
ST_CoveredBy — Returns 1 (TRUE) if no point in Geometry/Geography A is outside Geometry/Geography B
ST_Crosses — Returns `TRUE` if the supplied geometries have some, but not all, interior points in common.
ST_LineCrossingDirection — Given 2 linestrings, returns a number between -3 and 3 denoting what kind of crossing behavior. 0 is no crossing.
ST_Disjoint — Returns TRUE if the Geometries do not "spatially intersect" - if they do not share any space together.
ST_Distance — For geometry type Returns the 2-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units. For geography type defaults to return spheroidal minimum distance between two geographies in meters.
ST_HausdorffDistance — Returns the Hausdorff distance between two geometries. Basically a measure of how similar or dissimilar 2 geometries are. Units are in the units of the spatial reference system of the geometries.
ST_MaxDistance — Returns the 2-dimensional largest distance between two geometries in projected units.
ST_Distance_Sphere — Returns minimum distance in meters between two lon/lat geometries. Uses a spherical earth and radius of 6370986 meters. Faster than ST_Distance_Spheroid ST_Distance_Spheroid, but less accurate. PostGIS versions prior to 1.5 only implemented for points.
ST_Distance_Spheroid — Returns the minimum distance between two lon/lat geometries given a particular spheroid. PostGIS versions prior to 1.5 only support points.
ST_DFullyWithin — Returns true if all of the geometries are within the specified distance of one another
ST_DWithin — Returns true if the geometries are within the specified distance of one another. For geometry units are in those of spatial reference and For geography units are in meters and measurement is defaulted to use_spheroid=true (measure around spheroid), for faster check, use_spheroid=false to measure along sphere.
ST_Equals — Returns true if the given geometries represent the same geometry. Directionality is ignored.
ST_HasArc — Returns true if a geometry or geometry collection contains a circular string
ST_Intersects — Returns TRUE if the Geometries/Geography "spatially intersect in 2D" - (share any portion of space) and FALSE if they don't (they are Disjoint). For geography -- tolerance is 0.00001 meters (so any points that close are considered to intersect)
ST_Length — Returns the 2d length of the geometry if it is a linestring or multilinestring. geometry are in units of spatial reference and geography are in meters (default spheroid)
ST_Length2D — Returns the 2-dimensional length of the geometry if it is a linestring or multi-linestring. This is an alias for `ST_Length`
ST_3DLength — Returns the 3-dimensional or 2-dimensional length of the geometry if it is a linestring or multi-linestring.
ST_Length_Spheroid — Calculates the 2D or 3D length of a linestring/multilinestring on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection.
ST_Length2D_Spheroid — Calculates the 2D length of a linestring/multilinestring on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection.
ST_3DLength_Spheroid — Calculates the length of a geometry on an ellipsoid, taking the elevation into account. This is just an alias for ST_Length_Spheroid.
ST_LongestLine — Returns the 2-dimensional longest line points of two geometries. The function will only return the first longest line if more than one, that the function finds. The line returned will always start in g1 and end in g2. The length of the line this function returns will always be the same as st_maxdistance returns for g1 and g2.
ST_OrderingEquals — Returns true if the given geometries represent the same geometry and points are in the same directional order.
ST_Overlaps — Returns TRUE if the Geometries share space, are of the same dimension, but are not completely contained by each other.

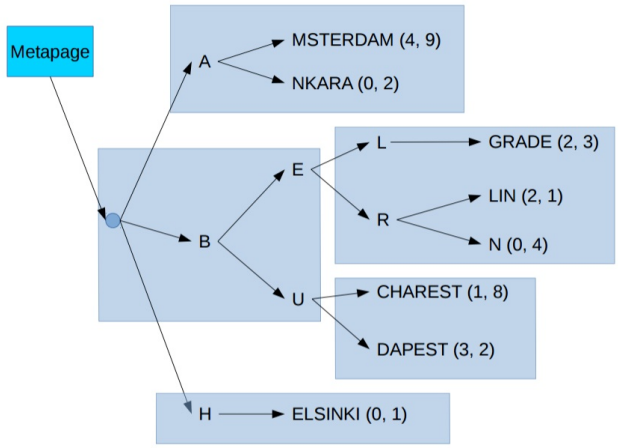
Just a sample

Indexing Spatial Data

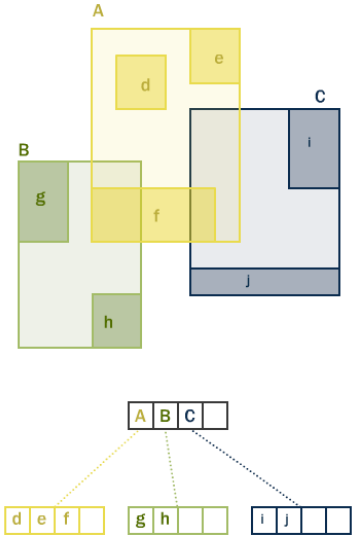
- Postgres Available Indexes:
  - **B-Trees**. Require linear arrangement. Not available for Spatial Data.
  - **R-Trees**. Divides Space into rectangles and sub-rectangles to index the data by location.
  - **GiST**: Generalized Search Trees.
    - GIST range type indexing uses large ranges at the top level of the index, with ranges decreasing in size at lower levels, just like how r-tree bounding boxes are indexed.
    - box_ops, circle_ops, point_ops, poly_ops
    - it supports “contains”, “left of”, “overlaps”, etc.
    - Gist allows KNN-search



SP-GiST page layout



R-tree Hierarchy



Indexing Spatial Data (cont'd)

```
CREATE INDEX [indexname] ON [tablename]
USING GIST ( [geometryfield] );

CREATE INDEX [indexname] ON [tablename]
USING GIST ([geometryfield] gist_geometry_ops_nd);
```

- GiST indexes have two advantages over R-Tree indexes in PostgreSQL:
  - GiST indexes are "null safe", meaning they can index columns which include null values.
  - GiST indexes support the concept of "lossiness" which is important when dealing with GIS objects larger than the PostgreSQL 8K page size.

- Lossiness allows PostgreSQL to store only the "important" part of an object in an index; in the case of GIS objects, just the bounding box. GIS objects larger than 8K will cause R-Tree indexes to fail in the process of being built.

Spatial Queries with PostGis

- Spatial queries are merged within the normal SQL query.
- This means that we can query spatial tables as any other table.
- We can use the spatial functions (ST_***) with the Geometry or Geography attributes
- This functions can be used in the **Where** clause or in the **Join** clauses.
- I.E. We can join tables based on the result of the ST functions or the spatial operators

Spatial Operators:

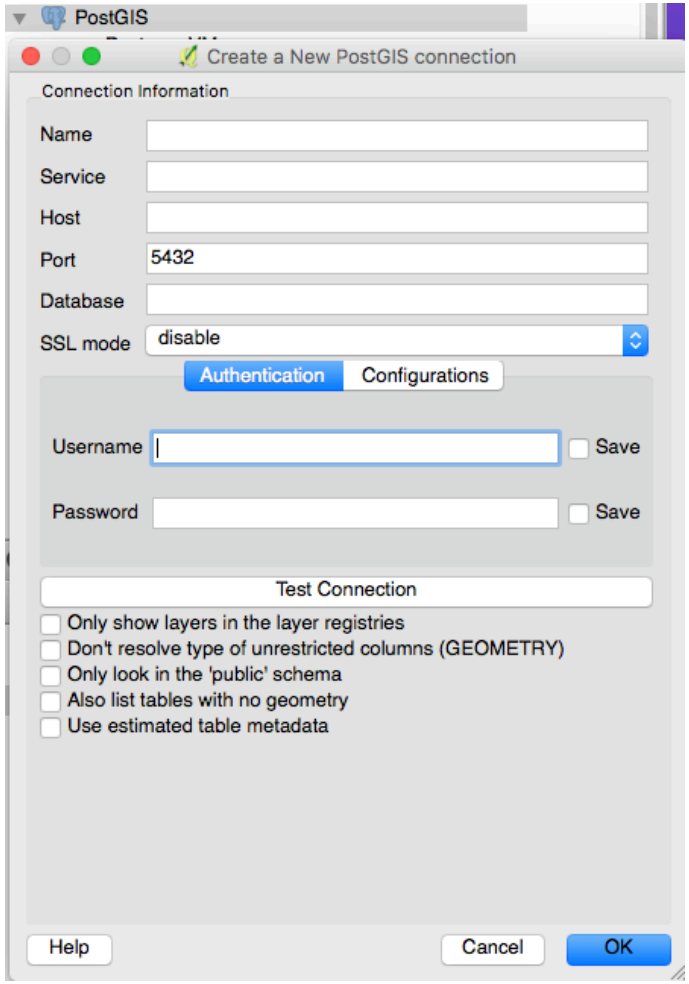
&& — Returns TRUE if A's 2D bounding box intersects B's 2D bounding box.
&&& — Returns TRUE if A's 3D bounding box intersects B's 3D bounding box.
&< — Returns TRUE if A's bounding box overlaps or is to the left of B's.
&<| — Returns TRUE if A's bounding box overlaps or is below B's.
&> — Returns TRUE if A' bounding box overlaps or is to the right of B's.
<< — Returns TRUE if A's bounding box is strictly to the left of B's.
<<| — Returns TRUE if A's bounding box is strictly below B's.
= — Returns TRUE if A's bounding box is the same as B's. Uses double precision
>> — Returns TRUE if A's bounding box is strictly to the right of B's.
@ — Returns TRUE if A's bounding box is contained by B's.
|&> — Returns TRUE if A's bounding box overlaps or is above B's.
|>> — Returns TRUE if A's bounding box is strictly above B's.
~ — Returns TRUE if A's bounding box contains B's.
~= — Returns TRUE if A's bounding box is the same as B's.
<-> — Returns the distance between two points. For point / point checks it use
types the distance between the floating point bounding box centroids is returne
<#> — Returns the distance between bounding box of 2 geometries. For point ,
and geometries are double precision). Useful for doing distance ordering and ne

Spatial Queries with PostGis (cont'd)

```
SELECT st_area(c.the_geom) area, st_centroid(c.the_geom) dist
FROM   tiger_data.county_all c, tiger_data.state_all s
WHERE  s.name='Colorado'
       and ST_intersects(s.the_geom,c.the_geom);

SELECT c.*
FROM   tiger_data.county_all c, tiger_data.state_all s
WHERE  s.name='Colorado'
       and ST_Contains (s.the_geom,c.the_geom);

SELECT h.*
FROM   tiger_data.state_all s, public.areal_hydrography h
WHERE  s.name='Colorado'
       and ST_Contains (s.the_geom,h.wkb_geometry);

SELECT h.*
FROM   tiger_data.county_all c, tiger_data.state_all s, public.areal_hydrography h
WHERE  s.name='Colorado'
       and c.name='Denver'
       and ST_Contains(s.the_geom,c.the_geom)
       and ST_Contains (c.the_geom,h.wkb_geometry);
```

Visualizing Data in QGIS

- We can connect QGIS to postgres.
- To get data from PostGIS we can access the whole table or create Views



Visualizing Data in QGIS (cont'd)

- One thing about views and QGIS….

- QGIS require that the views have an integer unique key. If that is non-existent… we won't be able to load the data in QGIS... So we need to get one..