DZone Refcardz

CONTENTS INCLUDE:

Essential PostGIS

By: Leo Hsu and Regina Obe

PostGIS is a free, open source spatial extender for the PostgreSQL open source database. Spatial extenders leverage existing database capabilities to handle geographical and image data. Common applications include proximity analysis for addressing questions of how far something is, geocoding for finding latitude and longitude of addresses, and web mapping for supplying data to various map servers.

PostGIS includes functionality to process raster data as well, enabling analysis of digital elevation data, weather, and other machine generated raster data. Raster support elevates PostGIS to a full-fledged image-processing tool with support for formats such as PNG, JPG, TIFF and many others. You can resize, overlay, colorize, and stitch together many images at a time by writing SQL.

PostGIS has features equal to, and often exceeding, popular commercial offerings such as Oracle Spatial and SQL Server. Like Oracle, it offers 2D and 3D support for geometry. PostGIS geography type mimics what you'll find in SQL Server.

This Refcard provides a quick reference for common tasks you perform when using PostGIS. We based our examples on PostgreSQL 9.1+ and PostGIS 2.1. For a thorough treatment of PostGIS, we encourage you to buy PostGIS In Action, 2nd Edition and to chomp the PostGIS manuals at http://postgis.net/documentation.

This Refcard assumes that you are reasonably familiar with PostgreSQL and have more or less heard of basic concepts required to work with geographical and image data. No experience with PostGIS is assumed. For more on PostgreSQL, check out our Essential PostgreSQL Refcard (http://refcardz.dzone.com/refcardz/essential-postgresql).

INSTALLATION

To find installer packages for most operating systems, visit http://postgis.net/install. To compile from source, visit http://postgis.net/source . Make sure you already have a working PostgreSQL server.

To create a new database in PostgreSQL, execute the following command in psql or use pgAdmin:

```
CREATE DATABASE mygisdb;
```

You then need to enable PostGIS for your new database. To do so, you first connect to your database with psql and then run one of the following:

Version	Install/Upgrade
PostgreSQL with extension support	`CREATE EXTENSION postgis;`

Lower versions of PostgreSQL without extension support (path dependent on distribution)	`cd /usr/pgsql/share/contrib/postgis-2.1`
	`\i postgis.sql`
	`\i spatial_ref_sys.sql`
	`\i postgis_comments.sql`
	`\i rtpostgis.sql`

Install topology	`CREATE EXTENSION postgis_topology;`
Install tiger geocoder	`CREATE EXTENSION postgis_tiger_geocoder;`

DATA TYPES

PostGIS introduces four families of spatial data types:

Version	Install/Upgrade
Geometry - Planar vector type. SRID determines the underlying coordinate system as well as the unit of measure.	Subtypes: point, linestring, polygon, triangle, curvepolygon, circularstring Collections: Each subtype has a corresponding collection type often prefixed with "multi" - subtypes with "multi" as in multipoint, multipolygon, etc. Others:TINZ, PolyhedralSurfaceZ, GeometryCollection
Geography - Geodetic type. Units of measurement always in meters and coordinates in degrees.	Subtypes: Has a subset of subtypes as geometry, except that spatial reference is always WGS 84 (SRID 4326). No curves or 3D advanced types. PolyhedralSurface,TIN Fewer functions available and slower than geometry, especially when computation encompasses large areas.
Raster - Pixel-based, multi-band type similar to common image types. Numerous statistical and processing functions. Georeferenced rasters have a SRID. SRID is irrelevant for non-georeferenced rasters.	Band pixel types supported are: 1BB (1-bit Boolean) Unsigned ints:,(2,4,16,32) BUI Signed ints: (8,16,) BSI Floats: (32,64) BF PostGIS places no limits on the number of bands that a raster can have. The pixel type of each band can also vary. Rasters, in addition to analysis, are useful for general image manipulation such as resizing, conversion between different formats.
TopoGeometry	Relational vector type. Lives in a network of connected nodes (points), edges (linestrings), and faces (polygons) . The topogeometry is made up of these elemental components and is referred to as punctal (if point), lineal (if linestring/multilinestring), areal (if polygon/multipolygon) Most useful to enforce consistency of edits and for simplification, ensuring connected edges still remain connected.
Geomval	Geomval is a compound type that arises when you intersect a geometry with a raster - for example, if you overlay a polygon or linestring atop a single-band raster. The raster neatly partitions your geometry into smaller geometries along pixel values. So for a linestring you may end up with a point or multilinestring cut where elevations change along the space. The combination of the geometry and the value forms a geomval.

DATA ADMINISTRATION

Version	Install/Upgrade
Create a table with a point geometry column	`CREATE TABLE x (gid serial primary key , name varchar(150) , geom. geometry(point, 4326));`
Create a table with linestring geography columns	`CREATE TABLE x (gid serial primary key , name varchar(150) , geog geography(linestring, 4326));`
Create a table of rasters	`CREATE TABLE x (rid serial primary key, name varchar(150), rast raster);`
Add a spatial index to raster column	`CREATE INDEX idx_x_rast ON x USING gist( ST_ConvexHull(rast) );`
Change SRID of a geometry column	`ALTER TABLE x ALTER COLUMN geom TYPE geometry(point, 4269) USING ST_SetSRID( eom., 4269)`
Transform spatial reference	`ALTER TABLE x ALTER COLUMN geom TYPE geometry(point, 2163) USING ST_Transform( eom., 2163);`
Add new spatial geography column	`ALTER TABLE roads ADD COLUMN geog geography(MultiLineString);`
Add a spatial 2d index to column (geometry or geography)	`CREATE INDEX idx_x_geom ON x USING gist (geom);`
Add a spatial 3d index to column	`CREATE INDEX idx_x_geom ON x USING gist( geom. geometry_nd_ops)`
Add a spatial functional index on a table	`CREATE INDEX idx_x_geom_2163 ON x USING gist (ST_Transform(geom, 2163));`

IMPORTING DATA USING SHP2PGSQL

shp2pgsql is a command-line tool for importing ESRI shape files. Use it with psql. Shp2pgsql generates a SQL script that you then execute to perform the import itself

Common Options

Option	Description
	Type shp2pgsql all by itself to retrieve help
-s srid_from [:srid_to]	srid_from is the spatial reference system of the incoming data). Omitting srid_to will load the data without transformation.
-g	Specify name of the target geometry or geography column. If omitted, column names default to geom. and geog.
-G	Imports data in as geography type. Source SRID must be 4326 or you must transform to 4326 with `--s.`
-I	Create a 2D spatial index. If omitted, shp2pgsql addes no index.
-N	NULL geometries handling policy (insert, skip, abort).

Example

Generate script using (loads into geography format):

```
shp2pgsql -s 4326 -G -I shapefile.shp a_table > import.sql
```

Execute script using psql:

```
psql -d my_db -f import.sql
```

Load wgs84 long lat (4326) and transform to web Mercator (3857):

```
shp2pgsql -s 4326:3857 shapefile.shp a_table | psql -d my_db
```

IMPORTING RASTER USING RASTER2PGSQL

raster2pgsql is a command-line tool for importing raster data. Supports many formats of raster data depending on GDAL library.

Common Options

Option	Description
	Type raster2pgsql all by itself to retrieve help
-G	Prints/list of support raster formats
-s srid_from	srid_from is the spatial reference system of the incoming data.
-F	Create a filename field in new raster table. Useful if loading a folder of rasters where you need to cross-reference back to file.
-R	Only register raster as out of database. does not import
-I	Creates a 2D spatial index
-t widthxheight	Chop up rasters into widthxheight where width and height are specified in pixels (e.g. 512x512)
-a	Append to an existing table instead of creating new one
-C	Add raster constraints needed to properly display in raster_columns view
-e	Execute statements individually -- do not use transaction

Example

Generate sql load script for new project_pictures table (loads in all pictures in a folder):

```
raster2pgsql -e -F *.jpg project_pictures > pics.sql
```

Execute script using psql:

```
psql -d my_db -f pics.sql
```

Load aerial coverage tiling 512x512, index, constraints in one step:

```
raster2pgsql -I -e -C *.jpg aerials -t 512x512 | psql -d my_db
```

FUNCTIONS

Geometry, geography, and raster data types often share the same function, but not always. You can usually intuit which functions apply or don't apply to a specific data type. For instance, you'd never try to count the number of bands against a geometry type.

Not all functions require parameters of the same data type. For instance, to see if a point lies within a raster, you'd pass a geometry type and a raster type to the ST_Intersects function.
PostGIS offers close to a thousand functions, each with multiple overloads,

but mastering only about a dozen or so for each data type will suffice. We cover the most commonly used functions below along with the most typical arguments. To explore the ever growing set of functions, visit http://postgis.net.

Functions and operators

Function	Description
ST_DWithin(X, Y, D)	Returns true if two objects are within a specified distance (D) of each other. X, Y spatial types can be geometry, geography or raster. X and Y must be of same spatial type. All inputs should have the same spatial reference, which will dictate the unit of measurement for the distance parameter. In case of geography D should be noted in meters.
ST_Intersects(X, Y)	Returns true if two objects share space. X, Y can be geometry, geography or raster.
ST_Intersection(X, Y)	Returns the spatially shared object or empty. Only available for geometry, geography, and raster. The raster version of the function that goes by the same name will return a new raster that is the intersection of another raster (nodata vals), or a set of geomvals that is the intersection of raster/geometry. If X is raster and Y is geometry, output is raster. If X is geometry and Y is raster, output is set of geomvals. If X is geometry and Y is geometry output is a geometry.
X && Y	Operator returns true if the bounding boxes of two objects intersect. X, Y can be geometry, geography, raster, or topogeometry.
X <-> Y	Returns distance between the bounding box centroid of one fixed against others using indexed near neighbor search. Use this operator in the ORDER BY clause to utilize geometry only index.
X <#> Y	Returns distance between the bounding boxes of one fixed geometry against others using indexed near neighbor search. Use this operator in the ORDER BY clause to utilize geometry only index.
ST_Transform(X, SRID)	Reprojects a raster or a geometry from one spatial reference system to another. X can be a raster or geometry and SRID must be one found in the spatial_ref_sys table.
ST_SetSRID(X,SRID)	Sets the spatial reference system of a geometry or raster to a known SRID (found in spatial_ref_sys). Unlike ST_Transform, this just sets the SRID meta data and does not change the coordinates.

Geometry and geography functions

Function	Description
ST_AsText, ST_AsGML, ST_AsGeoJSON, ST_AsKML, ST_AsSVG, ST_AsX3D	Outputs a geometry or geography in textual, GML, GeoJSON, KML, SVG, X3D formats.
ST_GeomFromText, ST_GeomFromGML, ST_GeomFromJSON, ST_GeomFromGML	Converts representation in textual, GML, JSON, KML to geometry. NB: Replace Geom with Geog in function name for geography.
ST_Point(X,Y)	Create a geometry point from X, Y coordinates. For longitude, latitude (WGS 84), X corresponds to longitude and Y corresponds to latitude.
ST_Buffer(X, D)	Returns a geometry or geography that defines the area within distance D from X. Often used in conjunction with ST_DWithin to visualize the enclosed area.
ST_Expand(geom.,num_units)	Returns a 2D geometry that is the bounding box rectangle of the geom expanding num_units in both directions. num_units is in units of the spatial reference system of the geometry.
ST_X(aPoint), ST_Y(aPoint)	For geometry points, returns the X and Y coordinates. If spatial reference is WGS 84 then ST_X returns longitude and ST_Y returns latitude.
ST_Extent	Aggregate function that returns bounding box of a set of geometries.
ST_Union	Aggregate function that returns spatial union from a set of geometries, effectively dissolving boundaries of overlapped regions.
ST_ClosestPoint (only geometry)	Returns closest point on one geometry to another.
ST_Distance(X, Y)	Returns minimum distance between X geometry and Y geometry.
ST_MaxDistance(X, Y)	Returns maximum straight line distance between X and Y. If X and Y are the same, this function finds and measures the farthest possible distance within the same geometry.
ST_ShortestLine (X, Y)	Outputs the shortest linestring between X and Y.
ST_LongestLine (X, Y)	Outputs the longest linestring between X and Y.
ST_3DIntersects, ST_3DDistance, ST_3DLongestLine, ST_3DShortestLine	3D companions to aforementioned functions.
ST_MakeLine	Aggregate function that creates a linestring from geometric points. Use ORDER BY to control order of the points.
ST_Equals(X,Y), ST_OrderingEquals(X,Y)	Returns true if geometries X and Y share the same 2D space. The ordering of vertices can be different and even represented by a different set of vertices. Similar related function is ST_OrderingEquals which tends to be faster, but requires directional order to be the same.
ST_MakeEnvelope(minx, miny,maxx,maxy,srid)	Creates a polygon rectangle geometry bounded by coordinates minx,miny,maxx,maxy. Although doesn't exist for geographay (for srid=4326) you can apply a cast to geography and others ST_Transform(..,4326)::geography

Raster functions

Function	Description
ST_Clip(rast, aPolygon)	Returns a new raster cropped by the boundary of the geometry, usually a polygon.
ST_Value(rast, band, aPoint)	Returns a pixel value on a specific band at a specific geometry point.
ST_DumpAsPolygons(rast, band_num, exclude_no_data)	Outputs a raster as a set of geomvals. Exclude_nodata is optional and defaults to true if not specified.

Function	Description
ST_Union(rast, band, 'type') Type options: LAST (Default if omitted) FIRST MEAN SUM COUNT RANGE	Aggregate function works on rows of rasters, usually of the same dimension, band number, and value types. Output is a new raster. If you omit the band argument, union operation takes place across all bands in all rows.
ST_Tile(rast, width, height, padwithnodata)	This set-returning function chops up an input raster into smaller equally sized rasters. The padwithnodata argument is optional and defaults to false. When false, some resulting tiles may be smaller than others if the dimensions of your original raster cannot be evenly divisible by the width and height.
ST_DumpValues(rast, band) ST_DumpValues(rast)	Returns a band of a raster as a 2-dimensional array. If band is not specified, the function returns a table with each band taking up a row. The column names of the table are nband for the band number and valarray for the array.
ST_SetValue(rast, nband, aPoint, newvalue) ST_SetValue(rast, nband ,X, Y, newvalue)	Updates a pixel value at given band within a given geometry (usually a point). You may also explicitly specify the X and Y coordinate of the pixel.
ST_SetValues(rast,nband, arraygeomval)	Sets a set of pixel locations on a band given an array of geomvals or a matrix of values
ST_ColorMap(rast, colormap)	Creates 8BUI multiband raster from single band raster. Colormap takes on values such as "grayscale", "pseudocolor", or a colormap text string that defines mapping. Use this function to "colorize" your raster.
ST_AsPNG , ST_AsTiff, ST_AsJPEG	Outputs a raster in common formats. Mostly used for exporting outside the database to web pages or reporting tools.
ST_AsGDALRaster(rast, format)	Output a raster to other formats available via GDAL. To obtain list of available formats, use ST_GDALDrivers().
ST_FromGDALRaster(X)	Convert from other raster formats to PostGIS format (from type is automatically inferred).

Topology functions

Header	Description
CreateTopology(name, SRID)	Creates a new database schema to house topology faces, edges, nodes, and relations. Returns the id of new topology created.
DropTopology(name)	Drops a topology schema with name name and all topogeometry columns within it.
AddTopoGeometryColumn(topology,schema,table, column, feature_type)	Adds a topogeometry column to an existing table in a schema and returns the new topology layer id for the column
toTopoGeom(geom, topology_id, layer_id, tolerance)	Converts a geometry to a topogeometry and adds edges,faces, and nodes as needed to create the missing elementsin the topology. The layer_id is the identifier returned by AddTopoGeometryColumn
ST_GetFaceGeometry(topology_id, face_id)	Given a topology id and a face id returns the polygon corresponding to the face
Geometry(topogeom)	Casts a topogeometry to a geometry. Alternate syntax: topo::geometry.
Topogeo_AddLineString	Adds a linestring to a topology creating edges,faces, nodes in the process

LOCATION QUERIES

These are examples that showcase some PostGIS functions.

Find all places within 100 meters of a road

```
SELECT p.place_name, ST_Distance(p.geog, r.geog) As dist
FROM places AS p INNER JOIN roads As r
ON ST_DWithin(p.geog, r.geog, 100)
WHERE r.road_name = 'Some Road';
```

Find ten closest places to a given point

Using indexed nearest neighbor bounding box centroid operator. Only approximate because the check is approximate because we're treating the earth as planar instead of spherical.

```
SELECT postal_code
FROM postal
ORDER BY ST_SetSRID(ST_Point(-71.06, 42.36), 4326) <-> geom LIMIT 10;
```

For each place find the closest hospital

Using indexed distance operator) - approximate if data is long lat or if place and hospital are not both points:

```
SELECT p.place_name
  , (SELECT h.place_name
       FROM places As h
       WHERE h.type = 'hospital'
        ORDER BY  h.geom <-> p.geom LIMIT 1) AS closest_hospital
FROM places As p;
```

For each postal find the closest road

This example assumes our geometry is stored in WGS 84 long lat. Use indexed box operator <#> to find 100 closest roads by bounding box distance and then narrow down to true geometry proximity by doing a spherioidal distance check.

```
SELECT p.postal_code
  , ( SELECT road_name
        FROM
(SELECT r.road_name
 , ST_Distance(r.geom::geography
      , p.geom::geography) As dist_m
       FROM roads As r
        ORDER BY  r.geom <#> p.geom LIMIT 100) As span_1
ORDER BY dist_m LIMIT 1 ) AS closest_road
FROM postal As p WHERE place_name = 'Boston';
```

Number and limit records by distance order

For each road segment, find the 2 closest postals within 1000 meters of the road. If no postal within 1000 meters, then NULL is returned for postal_code.

```
SELECT rnum,road_name, postal_code
FROM (SELECT r.gid
 , ROW_NUMBER() OVER(PARTITION BY r.gid
       ORDER BY ST_Distance(p.geog,r.geog) ) As rnum
 , p.postal_code, r.road_name
FROM  roads As r LEFT JOIN  postal As p
   ON ST_Dwithin(p.geog, r.geog,1000) ) As x
WHERE rnum < 3
ORDER BY x.road_name, x.gid;
```

Rectangle box

Returns geometries within the WGS 84 planar rectangle defined by ST_MakeEnvelope(min_lon,min_lat,max_lon,max_lat,4326):

```
SELECT geom
FROM roads As r
WHERE r.geom && ST_MakeEnvelope(-71.06,42.36,-70.10,41.40,4326);
```

Returns geographies within the WGS 84 spheroid rectangle defined by ST_MakeEnvelope(min_lon,min_lat,max_lon,max_lat,4326):

```
SELECT geog
FROM roads As r
WHERE r.geog
 && ST_MakeEnvelope(-71.06,42.36,-70.10,41.40,4326)::geography;
```

ADDING & UPDATING GEOMETRY / GEOGRAPHY DATA

Insert geometry statement

```
INSERT INTO roads(road_name,geom)
VALUES ('Main St', ST_GeomFromText('LINESTRING(-71.06 42.36
,-71.50 42.50)', 4326));
```

Insert geography statement

```
INSERT INTO roads(road_name, geog)
VALUES ('Main St', ST_GeogFromText('LINESTRING(-71.06 42.36,
-71.50 42.50)'));
```

Update geometry point

Updates a geometry column with data from lon,lat column

```
 UPDATE postal SET geom = ST_SetSRID(ST_Point(lon,lat),4326);
```

Update geography point

Updates a geometry column with data from lon,lat column

```
 UPDATE postal SET geog = ST_Point(lon,lat)::geography;
```

Add missing roads

Insert statement, selects only those records from rnew table with road geometry not already present in roads:

```
INSERT INTO roads(road_name,geom)
SELECT rnew.road_name, rnew.geom
FROM rnew
  LEFT JOIN roads ON (ST_OrderingEquals(rnew.geom, roads.geom) )
WHERE roads.gid IS NULL;
```

Spatial update count

For each county populate a field with count of restaurants in county:

```
UPDATE counties
 SET num_restaurants = (SELECT COUNT(r.id)
     FROM restaurants As r
       WHERE ST_Intersects(counties.geom,r.geom) );
```

GEOMETRY PROCESSING QUERIES

Linestring path from GPS points

This example uses an ordered aggregate clause to ensure the linestring formed is in order of time travel. For each user, for each day will create linestring representing travel for that day:

```
SELECT user_name,travel_ts::date As travel_date
 , ST_MakeLine(geom ORDER BY travel_ts) AS geom
 FROM gps_points
GROUP BY user_name, travel_ts::date;
```

RASTER QUERIES

Return elevation at each point in a table of points

```
SELECT p.place_name, ST_Value(r.rast,1,p.geom) as elev
FROM places As p INNER JOIN dems As r
  ON ST_Intersects(p.geom, r.rast);
```

Bounding box raster filter

```
SELECT rast
FROM dem As r
WHERE r.rast && ST_MakeEnvelope(-71.06,42.36,-70.10,41.40,4326);
```

Return a raster clipped to within 100 unit box of a parcel

```
SELECT ST_Union(ST_Clip(a.rast,ST_Expand(p.geom,100)))
FROM aerials As a
   INNER JOIN
     parcels As p ON ST_DWithin(a.rast::geometry, p.geom, 100)
WHERE p.pid = '57-169E';
```

Resize and output to PNG

Creates a rast of 25% of original size using Lanczos resampling algorithm and then outputs to PNG format

```
SELECT ST_AsPNG(
     ST_Resize(rast, 0.25,0.25, 'Lanczos') ) As png
FROM project_pictures
```

Convert a geometry to a raster using reference raster

Rasters formed will be 1 band 8BUI value 1 (defaults when not specified) and pixel size will be the same and clipped to the extent of the dem rasters and return first.

```
SELECT ST_AsRaster(r.geom, d.rast) As rast
FROM roads r INNER JOIN dems d ON ST_Intersects(r.geom,d.rast)
WHERE r.road_name = 'Some Road' LIMIT 2;
```

Convert to geometry elevation range for particular bounding box

Raster tiles that intersect bounding box and of a particular elevation range, convert to polygon/multipolygon

```
SELECT ST_Union( (gval).geom)
FROM (
SELECT ST_DumpAsPolygons(rast,1, true) As gval
FROM usgs_srtm
WHERE ST_Intersects(rast
  , ST_MakeEnvelope(85.99958, 49.0004, 85.99959, 49.0005, 4326) )
  ) AS elev
WHERE (gval).val BETWEEN 2403 and 2405
```

Convert to postgis raster, resize, convert to PNG

Create 100px width png thumbnails of a JPEG or other supported raster formats:

```
WITH x AS
(SELECT ST_FromGDALRaster(pic_image) As rast
   FROM project_pictures
   WHERE project = 'Gulliver Redevelop'
)
SELECT ST_AsPNG(ST_Resize(rast, 100, (ST_Height(rast)/ST_
Width(rast)*1.0*100)::integer) ) As thumb
FROM x;
```

SPATIAL CATALOG

The spatial catalog tables provide an inventory of spatial columns you have in your database as well as additional properties about them often used by third-party tools and viewers. They are similar in concept to the standard information_schema.columns table of PostgreSQL proper.

Table	Description
spatial_ref_sys	Table that lists all spatial reference systems and proj4text projection information to convert between spatial reference systems.

Table	Description
geometry_columns	A view that lists all columns in your database that are of type geometry and what subclass of geometry they are, in addition to dimension, spatial reference. Pre-2.0 this was a table that required manual intervention.
geography_columns	A view that lists all columns that are of type geography. It is very similar to geometry_ columns in structure.
raster_columns *	A view that lists all columns of type raster and key properties: spatial reference, number of bands, pixel types, extent of raster coverage for each.
raster_overviews *	A view that lists all tables and raster columns that serve as lower res versions of higher res raster columns. In addition, it lists the main raster table/column and the pyramid overview_factor. Raster over view tables are created when you use the -l option when loading.
topology.topology *	A table that lists all topology schemas in database, spatial reference system, and dimension of each.
topology.layer *	A table that lists all topogeometry columns and what topology they belong to.

USEFUL EXTENSIONS

These are just a few of the most popular PostGIS extensions.

pgrouting

http://pgrouting.org FOSS PostgreSQL extension that extends PostGIS to provide geospatial routing functionality. pgRouting 2.0+ can now be installed with

```
CREATE EXTENSION pgrouting;
```

PL/R

http://www.joeconway.com/plr
PostgreSQL language extension that allows writing stored functions in R.

```
CREATE EXTENSION plr;
```

hstore

Key/value column type that allows storing and quering multiple ad-hoc key-values in a single field with very fast indexing support using PostgreSQL gist index. Comes packaged with all binary distributions of PostgreSQL.

```
CREATE EXTENSION hstore;
```

INDEXES

All spatial indexes use the PostgreSQL gist index type for indexing, but have a different operator class that controls what operations can be used on them and how those operations behave.

Index Type/Op	Description
gist_geometry_ops_2d	2-dimensional spatial index and is the default for geometry, raster, and topology. It's best to just allow the default. In pre-2.0 versions, this was called `geometry_ops`. If loading data from PostGIS 1.5 or below that explicitly stated the old operator class, you may need to run the legacy_gist.sql packaged with 2.0.2+ which includes the old op name before you can restore this data.
gist_geometry_ops_nd	n-Dimensional index used for 3D and 4D geometry columns.
gist_geography_ops	The only and the default class for geography type data.

TOOLS

PostGIS comes bundled with several tools for loading and exporting spatial data. There are many other FOSS tools that are commonly used with PostGIS as well. Listed below is a common subset.

Tool	Description
shp2pgsql	Command-line packaged with PostGIS for loading data from ESRI shape files.
pgsql2shp	Command-line packaged with PostGIS for exporting data to ESRI shape file or DBF.
raster2pgsql	Command-line tool packaged with PostGIS for loading various formats of raster data.
shp2pgsql-gui	A graphical tool for both loading and exporting spatial data. Versions prior to 2.0 only supported import. It is packaged with some distributions of PostGIS, for example windows distributions of PostGIS have it packaged.
pgAdmin III	Popular graphical user interface packaged with some distributions of PostgreSQL and standalone source and binaries, available from http://pgadmin.org
pg_restore	PostgreSQL packaged Command-line tool for restoring compressed or tar or directory backups created with pg_dump
pg_dump	PostgreSQL packaged Command-line tool for doing backups. Great for automated backups. Can also output in SQL or copy mode format (plain text)
osm2pgsql	Separately available command-line tool for loading OpenStreetMap (OSM) data into PostGIS. Not packaged with PostGIS. http://wiki.openstreetmap.org/wiki/Osm2pgsql
impOSM	Another tool for importing OSM data (not packaged with PostGIS) http://imposm.org. Newer than osm2pgsql and some say faster but not as many OS supported.
GDAL/OGR toolkit	A free open source command-line toolkit useful for loading hundreds of different kinds of spatial data formats into PostGIS - .e.g. has driver to transfer data between SQL Server and PostGIS (or any other supported GDAL/OGR). The core library is also embedded in PostGIS and the engine that powers much of raster functionality. MIT License http://gdal.org/
QGIS	Desktop viewer/editor for all types of PostGIS data and other spatial formats. Also comes with QGIS server that allows turning a QGIS mapworkspace into a mapping webservice. http://qgis.org GPLv2
OpenJump	Desktop Editor / Viewer / Ad-hoc spatial query tool for PostGIS and other formats http://www.openjump.org/
gvSig	Another desktop viewer / editor with support for PostGIS. http://www.gvsig.org (GPLv2)
Mapserver	Web Mapping server, supports PostGIS among many other formats. More suited for those who like to edit configuration scripts directly http://mapserver.org/ (MIT License)

Tool	Description
MapGuide OpenSource	Web mapping server. Layers can be edited with a free desktop edtior (Majestro) or AutoDesk MapGuide Editor. Originally developed by AutoDesk http://mapguide.osgeo.org/ . Supports standard web mapping protocols. Supports both windows and Linux (Centos, Ubuntu). LGPL
GeoServer	Another mapping server commonly used with PostGIS - sports a user-friendly GUI. (LGPL)
OpenGeo Suite	A suite that include PostGIS, GeoServer, its own webserver, javascript mapping api and examples (including web spatial data editor). Built on Java Servlets. Comes in free community edition flavor and enterprise (enterprise is much like community except includes technical support contract) http://opengeo.org/products/suite/register/ (mix of open source licenses)
OSGeo Live	http://live.osgeo.org self-contained Xubuntu (comes in bootable form or as a VMWare, VirtualBox, KVM Virtual machines) that includes PostGIS, MapServer,QGIS, gvSig and other FOSS GIS tools. Great for teaching and experimenting.

EXPORTING DATA WITH PGSQL2SHP

USAGE:

```
pgsql2shp [<options>] <database> [<schema>.]<table>
pgsql2shp [<options>] <database> <query>
```

Some common options:

Option	Description
-f <filename>	File name to output as
-h <host>	Server host
-p <port>	Port number
-P <password>	Password
-u <user>	User
-g <column>	Column name geography or geometry. If there are multiple and this is not specified, will output the first one found.
-m <filename>	Allows you to specify alternative names for columns using a name mapping file

Get help:

```
pgsql2shp
```

Dump out a table:

```
pgsql2shp -f ca_roads -g geog gisdb ca.roads
```

Dump out a query and reproject:

```
pgsql2shp -f roads_merc gisdb "SELECT road_name, ST_
Transform(geog::geometry, 3857) AS geom FROM ca.roads"
```

ADMIN TASKS

Backup and Restore

Below are common backup and restore statements. Note: backing up a spatial database is not different from backing up any other database.

Create a compressed backup:

```
pg_dump -h localhost -p 5432 -U postgres -F c -b -v -f "/
somepath/gisdb.backup" gisdb
```

Restore compressed backup without upgrade:

```
psql -h localhost -d postgres -U postgres -c "CREATE DATABASE
gisdb"
```

Upgrading

You'll need to do a hard upgrade if upgrading from major to new major: e.g. 1.5 -> 2.0 requires backup and restore, but 2.0->2.1 just requires standard upgrade with ALTER EXTENSION. You need to install the legacy script if you have applications that only support 1.5

```
psql -h localhost -d gisdb -U someuser -c "CREATE EXTENSION
postgis;"
psql -h localhost -d gisdb -U someuser -f legacy.sql
```

Optimization

After a large load or delete, to ensure good planner stats:

```
vacuum analyze verbose sometable;
```

Uninstalling PostGIS

Should you ever need to uninstall, run:

```
DROP EXTENSION postgis;
```

To both uninstall and drop all spatial columns, stored data, and dependent extensions:

```
DROP EXTENSION postgis CASCADE;
```

ABOUT THE AUTHORS

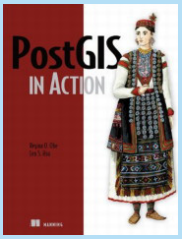
The wife and husband team of **Leo Hsu and Regina Obe** founded Paragon Corporation in 1997 specializing in database technology. Paragon Corporation works with numerous organizations to design, develop and maintain database and web applications. In 2002, Leo and Regina started to dabble in the growing field of spatial analysis and have become active participants in the on-going development of PostGIS which is a spatial extension of PostgreSQL. Regina is a member of the PostGIS core development team and Project Steering Committee.

Through Paragon Corporation, Leo and Regina have helped many clients using PostgreSQL, SQL Server and MySQL. Paragon Corporation takes on database projects in a wide range of industries and advocates database-driven development.

Leo and Regina met at MIT and both graduated with engineering degrees. Leo went on to obtain a master's degree from Stanford University in Engineering of Economic Systems. They maintain two sites: http://www.postgresonline.com -- provides tips and tricks for using PostgreSQL and http://www.bostongis.com - provides tips and tricks for using PostGIS, SQL Server 2008 Spatial and other open source and open GIS tools.

Email contact: lr@pcorp.us
URL: http://www.paragoncorporation.com

RECOMMENDED BOOK



PostGIS in Action is the first book devoted entirely to PostGIS. It will help both new and experienced users write spatial queries to solve real-world problems. For those with experience in more traditional relational databases, this book provides a background in vector-based GIS so you can quickly move to analyzing, viewing, and mapping data. Advanced users will learn how to optimize queries for maximum speed, simplify geometries for greater efficiency, and create custom functions suited specifically to their applications. It also discusses the new features available in PostgreSQL 8.4 and provides tutorials on using additional open source GIS tools in conjunction with PostGIS.

BUY NOW

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, blogs, feature articles, source code and more. "'DZone is a developer's dream',' says PC Magazine.

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-80-6
ISBN-10: 1-936502-80-1
50795
9 781936 502806
$7.95

Version 1.0