Small Manual for PostGis

Installation of PostGis

1. Download the PostgreSQL Installer: http://www.enterprisedb.com/products/pgdownload.do#windows
2. Install PostgreSQL by running the installer
3. Install PostGis the following way:
   "Start -> Programs -> PostgreSQL 8.4 -> Application Stack Builder." (Alternatively you can select the „launch stack builder" option at the end of the installation.)
   „Select your PostgreSQL installation from the list and then click "Next". The latest version of PostGIS will appear under the "Spatial Extensions" category."
   (http://postgis.refractions.net/download/windows/)
4. Restart the computer

   Notes:
   a) Ensure that there are sufficient permission to the PostgreSQL data folder. Either use a new folder (outside C:\Program Files), that is read and writeable to the PostgreSQL-user (the PostgreSQL-user is created at installation), or grant permissions to the appropriate user.
   b) Remember the password of the PostgreSQL-user created at the PostgreSQL installation (2nd step), you will need this password later! Use this password in the 3rd step.
   c) Mirror selection in 3rd step: the first FTP-Mirror did not work with me (Krisztian Buza), but http-mirror was fine.

Creating a Spatial Table

Create a table with spatial data is done in two stages:

1. Create a normal non-spatial table.
   - CREATE TABLE parks (PARK_ID int4, PARK_NAME varchar(128), PARK_DATE date, PARK_TYPE varchar(2));
2. Add a spatial column to the table using the OpenGIS "AddGeometryColumn" function.
   - SELECT AddGeometryColumn('parks', 'parks_geom', -1, 'POLYGON', 2);

AddGeometryColumn(varchar, varchar, varchar, integer, varchar, integer)

Syntax: AddGeometryColumn(<schema_name>, <table_name>, <column_name>, <srid>, <type>, <dimension>). Adds a geometry column to an existing table of attributes. The schema_name is the name of the table schema (unused for pre-schema PostgreSQL installations). The srid must be an integer value reference to an entry in the SPATIAL_REF_SYS table. The type must be an uppercase string corresponding to the geometry type, eg, 'POLYGON' or 'MULTILINESTRING'.

(The OpenGIS specification requires that the internal storage format of spatial objects include a spatial referencing system identifier (SRID). The SRID is required when creating spatial objects for insertion into the database. Undefined SRID value is equal to -1)

Here is another example:

```
CREATE TABLE roads ( ROAD_ID int4, ROAD_NAME varchar(128) );
SELECT AddGeometryColumn( 'roads', 'roads_geom', -1, 'LINESTRING', 2 );
```

Loading GIS Data

```
BEGIN;
INSERT INTO roads (ROAD_ID, roads_geom, ROAD_NAME ) VALUES
(1,GeomFromText('LINESTRING(191232 243118,191108 243242)',-1),'Jeff Rd');
INSERT INTO roads (ROAD_ID, roads_geom, ROAD_NAME ) VALUES
(2,GeomFromText('LINESTRING(189141 244158,189265 244817)',-1),'Geordie Rd');
INSERT INTO roads (ROAD_ID, roads_geom, ROAD_NAME ) VALUES
(3,GeomFromText('LINESTRING(192783 228138,192612 229814)',-1),'Paul St');
INSERT INTO roads (ROAD_ID, roads_geom, ROAD_NAME ) VALUES
(4,GeomFromText('LINESTRING(189412 189412,189631 259122)',-1),'Graeme Ave');
INSERT INTO roads (ROAD_ID, roads_geom, ROAD_NAME ) VALUES
(5,GeomFromText('LINESTRING(190131 224148,190871 228134)',-1),'Phil Tce');
INSERT INTO roads (ROAD_ID, roads_geom, ROAD_NAME ) VALUES
(6,GeomFromText('LINESTRING(198231 263418,198213 268322)',-1),'Dave Cres');
COMMIT;
```

GeomFromText(text,[<srid>])

Makes a Geometry from Well-Known Text (WKT) with the given SRID.

Here is another example:

```
BEGIN;
INSERT INTO parks (PARK_ID, parks_geom, PARK_NAME ) VALUES (1,
GeomFromText('POLYGON((0 0,190000 0,190000 190000,0 190000,0 0))',-1),'Prince
George');
INSERT INTO parks (PARK_ID, parks_geom, PARK_NAME ) VALUES (1,
GeomFromText('POLYGON((60000 60000,1200000 60000,1200000 1200000,60000
1200000,60000 60000))',-1),'Central Park');
END;
```

Examples of the text representations (WKT) of the spatial objects of the features are as follows:

- POINT(0 0)
- LINESTRING(0 0,1 1,1 2)
- POLYGON((0 0,4 0,4 4,0 4,0 0))
- 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)))

Loading data from ESRI shapefiles

Insert path C:\Program Files\PostgreSQL\8.2\bin in your system’s path.

Unzip the shapefile (produces 3 files: shp, shx, dbf).

You can view the files with the ShapeViewer.exe.

In command prompt, give:

```
shp2pgsql -c [shapefile_name] public.[tablename] > [tablename].sql
```

Open pgAdminIII, open Execute SQL window, open the created sql file and run it. For large files you may have to wait some moments. After you press refresh in pgAdminIII, you can see the created table. For large files (e.g., blk00) you may consider the option to go to command prompt and give the command: psql -d my_db -f [tablename].sql

Retrieving GIS Data

The simplest operation is to view all rows:

```
SELECT road_id, AsText(roads_geom) AS geom, road_name FROM roads;
```

AsText(geometry)

Return the Well-Known Text (WTK) representation of the geometry.

What is the total length of all roads, expressed in kilometers?

```
SELECT sum(length(roads_geom))/1000 AS km_roads
FROM roads;
```

How large is the park Prince George, in hectares?

```
SELECT area(parks_geom)/10000 AS hectares
FROM parks
WHERE park_name = 'Prince George';
```

Which is the largest park, by area?

```
SELECT park_name, area(parks_geom)/10000 AS hectares
FROM parks
ORDER BY hectares DESC
LIMIT 1;
```

…or equivalently

```
SELECT park_name, area(parks_geom)/10000 AS hectares
FROM parks
WHERE area(parks_geom) = (SELECT MAX(area(parks_geom)) FROM parks);
```

Find the names of all roads intersecting park Prince George

```
SELECT r.road_name
FROM roads AS r, parks AS p
WHERE intersects(r.roads_geom, p.parks_geom)
AND p.park_name = 'Prince George';
```

Find the name of the park whose center is closest to road Graeme Ave

```
SELECT p.park_name
FROM roads AS r, parks AS p
WHERE r.road_name = 'Graeme Ave'
ORDER BY distance(r.roads_geom, centroid(p.parks_geom))
LIMIT 1;
```

Creating and using indexes

To create a spatial index on the geometry column of a table, give:

```
create index [index_name] on [table_name]
using GIST ([name_of_spatial_column])
```

Next, update the statistics for the optimizer:

```
VACUUM ANALYZE [table_name]
```

To use the index:

- For range queries, apply the && operator and BOX3D function
- For spatial joins queries with intersection, use the && operator
- For othr types of queries (spatial join with containement, directional) find the appropriate operators

When not using these operators, then the index is inactive.

REFERENCE OF FUNCTIONS USED

Geometry Relationship Functions

Distance(geometry, geometry)

Return the cartesian distance between two geometries in projected units.

Equals(geometry, geometry)

Returns 1 (TRUE) if the given Geometries are "spatially equal". Use this for a 'better' answer than '='. equals('LINESTRING(0 0, 10 10)','LINESTRING(0 0, 5 5, 10 10)') is true.

Disjoint(geometry, geometry)

Returns 1 (TRUE) if the Geometries are "spatially disjoint".

Intersects(geometry, geometry)

Returns 1 (TRUE) if the Geometries "spatially intersect".

Intersects(g1, g2) --> Not (Disjoint(g1, g2))

Touches(geometry, geometry)

Returns 1 (TRUE) if the Geometries "spatially touch".

a.Touches(b) -> (I(a) intersection I(b) = {empty set}) and (a intersection b) not empty

Crosses(geometry, geometry)

Returns 1 (TRUE) if the Geometries "spatially cross".

Within(geometry A, geometry B)

Returns 1 (TRUE) if Geometry A is "spatially within" Geometry B.

Overlaps(geometry, geometry)

Returns 1 (TRUE) if the Geometries "spatially overlap".

Contains(geometry A, geometry B)

Returns 1 (TRUE) if Geometry A "spatially contains" Geometry B.

same as within(geometry B, geometry A)

Relate(geometry, geometry, intersectionPatternMatrix)

Returns 1 (TRUE) if this Geometry is spatially related to anotherGeometry, by testing for intersections between the Interior, Boundary and Exterior of the two geometries as specified by the values in the intersectionPatternMatrix.

Relate(geometry, geometry)

returns the DE-9IM (dimensionally extended nine-intersection matrix)

Geometry Processing Functions

Centroid(geometry)

Returns the centroid of the geometry as a point.

Area(geometry)

Returns the area of the geometry if it is a polygon or multi-polygon.

Length(geometry)

The length of this Curve in its associated spatial reference.

PointOnSurface(geometry)

Return a Point guaranteed to lie on the surface

“Real manual” for Postgis: http://postgis.refractions.net/docs/