4.3.4 Utilities for PostGIS

PostGIS - Import and Export

import and export utilities shp2pgsql and pgsql2shp

import and export of shapefiles

Two utilities that come with PostGIS allow the import and export of shapefiles from and to PostGIS. Using shp2pgsql one can convert a shapefile into a SQL statement that can be loaded into PostGIS. Using pgsql2shp, a PostGIS layer can be exported and written to a shapefile.

Located in
c:/Program Files (x86)/PostgreSQL/9.4/bin/shp2pgsql.exe

For example to convert the shapefile theimport_2008.shp (projection is decimal degrees - SRID 4326) and write to a SQL file "theimport_2008.sql" use this command :

shp2pgsql -I -s 4326 theimport_2008.shp in_2008 > in_2008.sql

Notes on optional switches to use on the command line:

- **-I** *option is to create a spatial index –> this is absolutely essential to make queries perform fast*
- **-i** *option is to use short integers as data format for the new data base table*
- **-a** *append mode (to add to an existing data base table)*
- **-W latin1** *for encoding (use with attribute data in languages that have non-English alphabet characters*. In such cases you could get the *ERROR: invalid byte sequence* without this switch.

Now to load the data into PostGIS either copy the contents of in_2008.sql into the SQL window of pgAdmin and run or run the SQL from the command line:

psql -U postgres -d osgis -f thepath/in_2008.sql

Or you can perform the above import operation in one step using the pipe command '|' to import the 2008 Washington County data in a regional map projection (WA State Plain North in feet):

shp2pgsql -I -s 2285 counties2008.shp counties2008 | psql -U osgis -d osgis

importing formats other than shapefile

Note that the map projection used for the -s switch has to match the projection the shape file to import is in. In order to load GIS data sources other than shapefiles the ogr2ogr utility can be used (see the table in the resource section for supported data formats). Another option to import data into PostGIS is the **OGR Converter Tool**, a plug-in that you can install with QGIS). This is an example of loading a dataset from an ESRI Personal Geodatabase into PostGIS using ogr2ogr on the command line:

```
ogr2ogr -f "PostgreSQL" PG:"host=localhost user=postgres
port=5432 dbname=postgis_in_action password=mypassword"
gadm_v0dot9.mdb -lco GEOMETRY_NAME=the_geom
-where "ISO='USA'" -t_srs "EPSG:2163" -nln "us.admin_boundaries"
gadm
```

Loading data from a file into PostGIS and making them spatial

The following work flow is an example to create a table in PostGIS, to load data into the empty table and then converting it into a spatial data set.

```
-- Creating and empty table called locations via SQL
CREATE TABLE locations
(
id int,
name varchar(30),
latitude float8,
longitude float8,
gid serial);

-- make user osgis the owner of the table
ALTER TABLE locations OWNER TO osgis;


--Load data with the copy command
copy locations (id, name, latitude, longitude) from 'c:/class/data/
    layers/point/locations.csv' using delimiters ',' with null as '
    ';


-- Create spatial features from the new data:
-- To add a geometry column (only really needed for PostGIS 1.5 and
    earlier, but also can still be used in PostGIS 2.X !):
select AddGeometryColumn('','locations','geom',4326,'POINT',2);

-- when using PostGIS 2.0 and up one can use instead a column for the
     geometry in the table definition or add the column subsequently
    :
ALTER TABLE locations ADD COLUMN geom geometry(Point,4326);


-- Update new column with features generated by the Makepoint
    function, with EPSG (SRID) 4326 (that's decimal degrees)
update locations set geom = ST_SETSRID(st_Makepoint(longitude,
    latitude),4326);

--Create a spatial index (very IMPORTANT) on the geometry column:
create index locations_gidx on locations using gist(geom);

-- create unique id field
-- Add Primary Key on id
```

```
ALTER TABLE locations ADD CONSTRAINT pkey_id PRIMARY KEY(id);
```

And finally delete the records from the imported table where we could not build a geometry. We just delete entries where geometry is empty: delete from locations where geom is null;
Built internal query index for PostgreSQL using the vacuum command:

```
VACUUM ANALYZE;
```

Now finally the newly loaded data is usable as input for MapServer.