Brief Specification

Downloading OS Open Data into a PostGIS database using a Python script

Document Version: 0.2

Compiled by K.Ardakova

for Astun Technology

Overview

Objective

This document provides a high level, functional description of how a Python script will provide required base functionality to meet the requirements for loading Ordnance Survey Open Data into a PostGIS database and how it will be structured to allow importing of 1 or more datasets.

Brief

The Python script will be created with the view of being used internally by Astun Technology as well as externally by clients. It will accommodate for the loading of the OS Open Data of different formats (currently shapefiles and csv/txt files only) using predefined shared folder structure on the hard drive as well as PostGIS database structure manually created before hand.

Assumption

Within this document a variety of assumptions have been made. Please provide any feedback regarding these assumptions as soon as possible.

System Outline

Overview

The specification may duplicate information from the brief, or even completely contradict the brief (if it contradicts the brief this should been shown in the footnotes for the brief)

The following diagram highlights the integration of the system along with the key components.

DIAGRAM

From the above diagram there are four main elements:

- PostGIS database design including schemas
- Loading different data formats into a PostGIS database
- Central raw data storage structure and applying updates
- Program structure overview

These elements are summarised in the system components section

System Workflow

The following is a simple representation of the workflow.

PostGIS database design including schemas

- Database design and relevant schemas allowing appropriate OS Open Data loading will be discussed.

Loading different data formats into a PostGIS database

- A table listing parameters to be used by the program for all the OS Open vector datasets, apart from Code Point (as the ONSPD contains a fuller information), will be discussed.

Central raw data storage structure and applying updates

- The structure of a shared Central Storage folder with OS Open Data downloaded from OS website is proposed and the procedure of updates is outlined
- A folder with Working Area is additionally created which will include newest datasets only. These will be used by the program to load data into a PostGIS database.

Program structure overview

- The pseudocode for a Python script automating the loading of various types of OS Open data from a Working Area into a PostGIS database is proposed.
- The Python script automating the loading of various types of OS Open data from a Working Area into a PostGIS database is proposed.

NB. We perform this here so that the solution will be extremely flexible in the future if any requirements change. As such the updates could be performed if desired.

System Components

PostGIS database design including schemas

Overview

A PostGIS database will need to be created to include 5 different schemas (currently) to hold OS Open Data downloaded: meridian2, strategi, vmd, bdryline and other.

Data Inputs:

The OS Open Data will be loaded from a Working Area folder into a PostGIS database using specific schemas, as follows:

OS OPEN DATASETS	SCHEMA USED TO IMPORT INTO POSTGIS
ONSPD, OS Locator, 1:50KGazetteer	other
Boundary-Line	bdryline
Strategi	strategi
Meridian 2	meridian2
VectorMap District (vector)	vmd

To create the schemas in a database, the SQL queries listed in an .sql file will need to be executed.

Loading different data formats into a PostGIS database

Overview

The resulting script aims to make it simple to load data of various formats (currently only shapefiles and CSV/TXT files), as well as make it simple to include new data in the future if it fits within the current formats. To achieve this a table listing datasets with parameters to be used by the program and a config file with input parameters will be used.

Data Inputs:

Table.csv

The Table.csv is created to list all OS Open datasets to be loaded into a PostGIS database, as well as the parameters to be used by the program to load data. The current structure of this table is as follows:

Dataset	**Location_folder**	**Schema**	**Source_Projection**	**VRT_filename**	**CSV_filename**	**Header**	**Delimiter**
ONSPD	ONSPD	other	EPSG:27700	ONSPD.vrt.	ONSPD_FEB_2011_UK_O.csv		,
OSLocator	OSLocator	other	EPSG:27700	OS_Locator2011.vrt	OS_Locator2011_1_Open.csv		:
Gazetteer	Gazetteer	other	EPSG:27700	Gazetteer.vrt	50kgaz2011.txt		:
BoundarLine	Boundary Line	bdryline	EPSG:27700				
Strategi	Strategi	strategi	EPSG:27700	Strategi_Gazetteer.vrt	STRATEGI_2011_GAZETTEER.TXT		*
Meridian2	Meridian2	meridian2	EPSG:27700				

VMDvector	VMDvector	vmd	EPSG:27700				

Note1: Strategi dataset also contains a Gazetteer.txt file
Note2: See Table.csv for full Header lines for all CSV/TXT files

Config file

This file contains input parameters to be used by the program, including path to the Working Area folder, a path to the Table.csv and PostGIS database parameters (e.g. database=dbname='OSOPEN' user='postgres' host='localhost' password='postgres')

Format specific details

Shapefiles

Every shapefile from a specific folder in a Working Area (Location_folder parameter taken from Table.csv) can be loaded into a PostGIS database using the ogr2ogr command.
The destination table names inside the PostGIS database will be created automatically corresponding with the names of shapefiles to be loaded. For the first loaded shapefile from a specified dataset the “ogr2ogr -overwrite” command will be used, and for the other shapefiles with identical name from the same dataset: “ogr2ogr -append”. The resulting table inside the relevant schema of the PostGIS database will include all features from all of the shapefiles with the same name.
The generic command that will be used to load all shapefiles within a specified dataset will look as follows:

```
ogr_cmd="ogr2ogr -%s -skipfailures -f PostgreSQL PG:'%s active_schema=%s'
-s_srs %s -a_srs EPSG:27700 -lco PRECISION=NO -nlt
GEOMETRY" %(appover,self.database,schema,source_projection)
```

Notes: appover will equal -overwrite or -append for reasons explained above; self.database refers to the PostGIS database parameters that will be taken from the config file; and schema and source_projection parameters will be taken by the script from Table.csv.
-lco PRECISION=NO allows for the full numeric fileds with geometry to be loaded into PostGIS
-nlt GEOMETRY allows for a mix of geometries to be imported into a PostGIS table from the same shapefile (e.g. polygons and multypolygons, linestrings and multilisnestrings, points and multipoints).

CSV/TXT files

As delimiters can vary, a column specifying the delimiter is added to the Table.csv. The script will then change the existing delimiter to ";" for the data to be loaded into a PostGIS database correctly. The header will also need to be added, hence a column specifying Header line is also added to the Table.csv from the Specifications provided by Ordnance Survey. The new edited file is then saved with "_new" appended to its name on the end.
For all of the CSV/TXT files a .vrt file needs to be also created which creates geometry by specifying point coordinates using specific fields in a .csv file (an edited version of a CSV/TXT file is used here as an input).
The .vrt file is then used in ogr2ogr command to load CSV data with a point geometry into a PostGIS database as follows.:

```
"ogr2ogr -overwrite  -skipfailures -f PostgreSQL PG:'%s active_schema=%s' -s_srs %s -a_srs
EPSG:27700" %(self.database,schema,source_projection)
```

Notes: self.database refers to the PostGIS database parameters that will be taken from config file; and schema and source_projection parameters will be taken by the script from Table.csv.

Central raw data storage structure and applying updates

Overview

The central data storage and the Working Area are separated with the intention to keep all historically downloaded data in a relevant folder of each OS dataset within the central data storage and only the newest dataset within the Working Area folder for processing. The central data storage is only updated after the data was successfully loaded from the Working Area folder into a PostGIS database.

Prerequisites

The folder structure within the Working Area folder will need to be prepared in advance prior to downloading data from OS. The user also needs to make sure that a PostGIS database is either created with 5 imported schemas (.sql file can be used for this) or already exists.

Data Inputs:

Updated OS data is downloaded from the OS website (http://parlvid.mysociety.org:81/os/) into a relevant Working Area folder for processing and loading into a PostGIS database. VRT files stored within a central data storage folder are also copied to a Working Area folder to be used by the data loading program (they need to be amended to include the newest file names). Additionally, a descriptive file for each dataset can be stored in a relevant folder within a central data storage folder.

Prior to running the script to load the data into a PostGIS database, the following input parameters will need to be specifically checked:
1. In **config file**: path to the Working Area folder, a path to the Table.csv and PostGIS database parameters
2. In **Table.csv**: all parameters, particularly file names.
3. In **VRT files**: make sure the name of the new edited CSV/TXT file is used (with "_new" at the end).

To run the script, e.g. type in command line: **python OSOpenLoader.py OSOpenLoader.config ONSPD**

When the import of data from a working folder into a PostGIS database was successful using a Python script (see below), it can be deleted from the Working Area folder and copied into a relevant folder within the Central Storage folder specifying time of download.

Program structure overview

Overview

The Python script is intended to flexibly load OS Open datasets of various formats (currently shapefiles and CSV/TXT files).

Prerequisites

This component requires a structured database and working area folder with newly downloaded OS Open Data as well as VRT files copied from the Central Storage folder (must be located in the same folder with CSV/TXT data they refer to). And importantly, all parameters in config file, Table.csv and VRT files have to be entered correctly.

Pseudocode

```
Input parameters:
        Config_file name
        requested_dataset name (to be loaded into a PostGIS database)

From config file set the following:  WorkingArea path, Table.csv path and PostGIS database
parameters

If Config_file exists:
        Build a dictionary of configuration
        Run loader with Config_file

If WorkingArea path does not exist, output error message: 'Filepath does not exist'
Test the connection to  PostGIS database, and if no database found, output error
message: "Unable to connect to the database"

Open table with datasets available and parameters used by this script (Table.csv)
Read the header line and get the important field indices
Figure out positions of all parameters in the table by creating Index for each parameter in
header
Loop through each line in the table, splitting it with delimiter ";"
Create variables for all needed field items based on index position

According to requested_dataset go to the row with the same Dataset name in Table.csv and get
all parameters, i.e. If requested_dataset name equals to Dataset name in Table.csv:
        folder_path = path to Location_folder
        If folder_path exists:
```

```
        num_shapefiles_processed = 0
        Create empty list with seen_files
        Parse through all filenames in folder_path:
            If filename ends with ".shp":
                Get file_path
                Print "Processing” file_path
                    Use ‘-append’ with ogr2ogr command below
                If not:
                    Use ‘-overwrite’ with ogr2ogr command below
                    Append filename to a seen_files list
                If "text" or “Text ” are in filename:
                     Set encoding to "LATIN1"
                Otherwise:
                    Set encoding to "UTF8"
                Run ogr2ogr command to load data to PostgreSQL format using
PostGIS database parameters from config file, and schema and source projection from
Table.csv (set Precision to No to avoid numeric field overflow for geometries and Geometry to
any geometry to allow loading of shapefiles with mixed geometries, e.g. polygons and
multipolygons)
                Increase num_shapefiles_processed by 1

            If filename equals to csv_filename in Table.csv:
                Get original file_path
                Create new csv file_path with “_new” added to it on the end
                If original file_path exists:                                          Prin
                    Replace delimiter from Table.csv to “;”  for new csv file
                Else:
                    Print "No correct CSV file found"
                    Exit
                Get VRTfile_path
                If VRTfile_path exists:
                    Print "Loading csv data using VRTfile_path”
                    If requested_dataset equals to "Gazetteer"
                or "GAZETTEER" is in filename:
                        Set encoding to "LATIN1"
                    Run ogr2ogr command to load data to PostgreSQL format
                    using PostGIS database parameters from config file, and
                    schema and source projection from Table.csv
                Else:
                    Print "No correct VRT file found. Cannot load ‘dataset’
data"                   Exit
```

```
                                Print "Successfully loaded CSV data from ‘dataset’”

                      Print "Loaded num_shapefiles_processed shapefiles"


            Else if folder_path does not exist:
                      Print "Path to folder is not correct"                                    Exit
```

OSOpenLoader.py

```
from __future__ import with_statement
import os,os.path,sys,glob
import shlex, subprocess
import string, csv
import psycopg2

config_file = sys.argv[1]
requested_dataset = sys.argv[2]

class OSOpenLoader:
   def __init__(self):
         pass
   def run(self, config):
         self.read_config(config)
         self.load()
   def read_config(self, config):
         self.config = config
         self.src_dir = config['src_dir']
         self.csv_table = config['csv_table']
         self.database = config['database']
         if not os.path.exists(self.src_dir):
               print 'Filepath does not exist'

    def load (self):
          try:
               conn=psycopg2.connect(self.database)
          except:
               print "Unable to connect to the database"
          # Open table with datasets available and parameters used by this script
          table = open(self.csv_table)
          # Read the header line and get the important field indices
          headerLine = table.readline()
          headerLine = headerLine.rstrip("\n")
          segmentedHeaderLine = headerLine.split(";")
```

```
# Figure out positions of all parameters in the table
datasetIndex = segmentedHeaderLine.index("Dataset")
location_folderIndex = segmentedHeaderLine.index("Location_folder")
schemaIndex = segmentedHeaderLine.index("Schema")
source_ProjectionIndex = segmentedHeaderLine.index("Source_Projection")
vrt_filenameIndex = segmentedHeaderLine.index("VRT_filename")
csv_filenameIndex = segmentedHeaderLine.index("CSV_filename")
headerIndex = segmentedHeaderLine.index("Header")
delimiterIndex = segmentedHeaderLine.index("Delimiter")


 # Loop through each line in the table, splitting it with delimiter ";"
for line in table.readlines():
     line = line.rstrip("\n")
    segmentedLine = line.split(";")

    # Create variables for all needed field items based on index position
    dataset = segmentedLine[datasetIndex]
    location_folder = segmentedLine[location_folderIndex]
    schema = segmentedLine[schemaIndex]
    source_projection = segmentedLine[source_ProjectionIndex]
    vrt_filename = segmentedLine[vrt_filenameIndex]
    csv_filename = segmentedLine[csv_filenameIndex]
    header = segmentedLine[headerIndex]
    delimiter = segmentedLine[delimiterIndex]

    # Find requested dataset name in the Table.csv and load files using parameters
from it
    if dataset==requested_dataset:
        for folder in glob.glob(os.path.join(self.src_dir, location_folder)):
            folder_path=os.path.join(self.src_dir, location_folder)
            if os.path.exists(folder_path):
                num_files = 0
                seen_files = []
                for root, dirs, files in os.walk(folder_path):

                    for name in files:

                        if name.endswith(".shp"):
                            file_path=os.path.join(folder_path,root,name)
                            print "Processing: %s" % file_path
                            if name in seen_files:
                                appover = "append"
                            else:
                                appover = "overwrite"
                                seen_files.append(name)
                                 os.environ["PGCLIENTENCODING"] = "UTF8"
```

```
                                            ogr_cmd="ogr2ogr -%s -skipfailures -f PostgreSQL PG:'%s
active_schema=%s' -s_srs %s -a_srs EPSG:27700 -lco PRECISION=NO -nlt
GEOMETRY" %(appover,self.database,schema,source_projection)
                                            args = shlex.split(ogr_cmd)
                                            args.append(file_path)
                                            rtn = subprocess.call(args)
                                           num_files += 1
                                      if name==csv_filename:
                                           new_csv=os.path.join(folder_path,root,name[:-4]+"_new.csv")
                                           if os.path.exists(original_csv):
                                               print "Editing %s and creating %s" %
(original_csv,new_csv)
                                               header_string= header.split()
                                               with open(original_csv, 'r') as original_csv:
                                                   with open(new_csv, 'w') as new_csv:
                                                       row = header_string
                                                       header = csv.writer(new_csv,delimiter=';')

                                                       for row in original_csv:
                                                           new_delimiter = row.replace(delimiter,";")
                                                           new_csv.write(new_delimiter)
                                               print "No correct CSV file found"
                                               sys.exit()


                                            vrtfile_path=os.path.join(folder_path,root,vrt_filename)
                                            if os.path.exists(vrtfile_path):
                                               print "Loading csv data using %s" % vrtfile_path
                                               if requested_dataset=="Gazetteer" or "GAZETTEER" in
name:
                                                        ogr_cmd="ogr2ogr -overwrite  -skipfailures -f
PostgreSQL PG:'%s active_schema=%s' -s_srs %s -a_srs
EPSG:27700" %(self.database,schema,source_projection)
                                                         args = shlex.split(ogr_cmd)
                                                         args.append(vrtfile_path)
                                                         rtn = subprocess.call(args)
                                           else:
                                               print "No correct VRT file found. Cannot load %s data" %
dataset

                                               sys.exit()
                                           print "Successfully loaded CSV data from %s" % dataset


                            print "Loaded %s shapefiles" % num_files

                        if not os.path.exists(folder_path):
```

```
                        print "Path to folder is not correct"
                        sys.exit()

def main():
        if os.path.exists(config_file):
        # Build a dict of configuration
        with open(config_file, 'r') as f:
        config = dict([line.replace('\n','').split('=',1) for line in f.readlines() if len(line.replace('\n',''))
and line[0:1] != '#'])
        loader = OSOpenLoader()
        loader.run(config)
        else:
        print 'Could not find config file:', config_file



if __name__ == '__main__':
        main()
```

Data Outputs

The outputs of running the Python script will be the newly created tables in a PostGIS database in a relevant to a specific dataset schema. These should replace any previously created tables with the same name if they already existed as well. For shapefiles from one dataset: one table in a relevant schema will contain records from all of the shapefiles with the same name from this dataset.