DAND - Project 3

OpenStreetMap Data Wrangling with SQL

By Alex Mathew

18th September 2017

I have decided to choose New Delhi, the capital city of India where I reside as it had a large enough dataset and seemed interesting enough to analyse.

URL : https://mapzen.com/data/metro-extracts/metro/new-delhi_india/

First we make a smaller sample of the close to 1 Gb osm file using the mapparser procedure.

We will try all the stages on the smaller sample of size 1.24 Mb

1. Data Audit

Using the Map Parser Procedure (mapparser.py), we go through the dataset using the ElementTree and find out the count of the unique tags.

We get the following result :

node : 34163

nd : 42178

member : 308

tag : 8414

relation : 62

way : 6953

osm : 1

TAG EXPLORATION

Using the tags.py procedure, we use a few REGEX patterns to check for patterns.

The result we get are as follows :

Tags with only lowercase characters

lower : 8138

Tags with lowercase characters and colons

lower_colon : 267

Tags with problematic characters

problemchars : 0

Other category tags

other : 9

2. Problems with the Data

Now we will see the problems encountered with our data using audit.py procedure.

I assume that most of the problems we will encounter will be errors in the name of the street etc

There are quite a few fields with street names in Hindi, some of which were rectified and translated using the mapping. A few abbreviations were also completed. Some case consistencies were corrected.

Some had too much extra information which was corrected.

Few changes made include :

- Baazar -> Market
- Kala Pathar -> Black Stone
- Marg -> Road
- W-1 -> West 1
- Kala Pathar Road, Indirapuram -> Black Stone Road",
- Vaibhav Khand, Indirapuram - > Vaibhav Khand
- Ajmal Khan Road, Karol Bagh - >Ajmal Khan Road

3. Creating CSV file

We use the data.py procedure and the schema.py file (provided) in order to build our csv data.

We make sure to keep the columns in the csv file labels correctly aligned to the columns of the schema.

The result is the following 5 csv files =>

- nodes_csv : 2.8 MB
- nodes_tags.csv : 20.4 KB
- ways_csv : 419 KB
- ways_nodes.csv : 1.03 MB
- ways_tags.csv : 260 KB

CREATING THE DATABASE FROM THE CSV FILE

We use the createDB.py procedure to create a connection to the sqlite module and then making a cursor connection. We create the respective tables for the various CSV data and can now run our queries on the database.

RUNNING QUERIES ON THE DATABASE

Query.py also contains all these queries

Number of nodes

```
sqlite> SELECT COUNT(*) FROM nodes
```

34163

Number of ways

```
sqlite> SELECT COUNT(*) FROM ways
```

6953

Number of Unique Users

```
sqlite> SELECT COUNT(DISTINCT(elem.uid))
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) as elem;
```

423

Top ten Contributing Users in the DataBase

```
sqlite> SELECT elem.user, COUNT(*) as num
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) as elem
GROUP BY elem.user
ORDER BY num DESC
LIMIT 10;
```

Oberaffe | 2704
Premkumar | 1657
Saikumar | 1606
Naresh08 | 1376
Anushap | 1334
Sdivya | 1310
anthony1 | 1291

sathishshetty | 1231
himabindhu | 1214
Apreethi | 1153

Count of users contributing only once to the dataset

```
sqlite> SELECT COUNT(*)
FROM
(SELECT elem.user, COUNT(*) as num
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) as elem
GROUP BY elem.user
HAVING num=1) ;
```

165

ADDITIONAL ANALYSIS

Top Amenities in the city according to count (node)

```
sqlite> SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='amenity'
GROUP BY value
ORDER BY num DESC
LIMIT 10;
```

fuel|5
fast_food|3
atm|2
cafe|2
embassy|2
police|2
school|2
bank|1
bar|1
marketplace|1

Types of Buildings according to count (node)

```
sqlite> SELECT tags.value, COUNT(*) as count
FROM
( SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) as tags
WHERE tags.key='building'
GROUP BY tags.value
ORDER BY count DESC
LIMIT 10;
```

Yes – 5185
Residential – 7
House – 5
Commercial – 1

Hospital – 1
Office – 1

Top Amenities in the city according to count (ways)

```
sqlite> SELECT value, COUNT(*) as num
FROM ways_tags
WHERE key='amenity'
GROUP BY value
ORDER BY num DESC
LIMIT 10;
```

Schools – 7
Place of Worship – 5
Parking – 3
Hospital – 2
Police – 2
Bus Station – 1
Graveyard – 1
Kindergarten – 1
Theatre – 1
University – 1

5. CONCLUSION

The OpenStreetMap data of Delhi is in my opinion lacking in reasonable quality with quite a number of typo errors and misfeed caused by the human inputs. I have cleaned the data up to a limit though it's still far from a clean dataset. There are lots of improvement needed in the dataset. The dataset has negligible amount of useful information such as amenities, tourist attractions, restaurants and other classifications.

Also, I understand that the dataset contains very old information which is quite behind information from say google maps.

There needs to be a much better data entering procedure especially in order to help with proper feeding of information in the correct field.

Data in ways and nodes don't really sync and I believe that probably information is fed in separately.

SUGGESTIONS

1. Control the input of data by matching it through some pattern or regex checker which may help to filter out incorrect type or language.

2. Give more information like popularity index for certain places which may help people especially tourists choose their outing.

ANTICIPATED PROBLEMS

1. Developing Validation constraints for input considering ‘global data’ would be a difficult job as the constraints would be differing across the globe. Different fields would have different ‘range’ of values in different places.

2. Putting something like a language check would disable the possibility of ‘translated information’ which might have come in handy for tourists who don’t speak the same language.

3. Controlling input feed might have an adverse effect on the user experience.

FILES INCLUDED

- sample-new-delhi.osm : sample data of the bigger OSM file
- mapparser.py : find unique tags in the data
- tags.py : count multiple patterns in the tags
- audit.py : audit data
- data.py : build CSV files
- createDB.py : create db