OpenStreetMap Case Study [SQL]

Student: Daniel de Carvalho Rust
Date: March 26th 2017

Map Region: Brazil, Rio de Janeiro Metropolitan Area
I've chosen to analyze an extract of Rio de Janeiro, Brazil (my hometown) using SQL.
The data was obtained from Mapzen. For code evaluation purposes, a sample file with every 10th element was created using the sample.py code.

1. Data Overview

The first step in our project is to analyze our file to check its structure and gather basic information. For example, which and how many tag types there are. In Openstreetmap case there's a dictionary available at their website, but that's not always the case. It's also necessary to check for unexpected tags which may need to be fixed.

1.1 File Size

Full datasets (not submitted):

rio-de-janeiro_brazil.osm ---------------------	353.2 MB
rio_de_janeiro.db -----------------------------	250.5 MB

Sample datasets:

rio-de-janeiro_brazil_sample.osm -------------	35.6 MB
rio_de_janeiro_sample.db ---------------------	24.9 MB

1.2 - Number of Unique Tags:

Source code: mapparser.py

```
<bounds>: 1
<member>: 25.291
<nd>: 2.015.603
<node>: 1.615.351
<osm>: 1
<relation>: 4.178
<tag>: 627.853
<way>: 193.769
```

1.3 - Patterns:

Source code: tags.py

The <tag> tags contain key/value pairs named 'k' and 'v' respectively. The aim here was to explore the 'k' value for patterns and potential problems. We used regular expressions to categorize the tags in four groups and count them.

```
'lower': 559.334 tags that contain only lowercase letters and are valid
'lower_colon': 45.851 valid tags with a colon in their names
'problemchars': 1 tag with problematic characters
'other': 22.667 tags that do not fall into the other three categories
```

1.4 - Number of Contributors:

Source code: users.py

There are 1,396 unique contributing users in our database.

2. Encountered Problems

2.1 - Street Type Inconsistencies:

Source code: audit.py

The most recurring problem found in the dataset was the way street types were written. To overcome it we used regular expressions (regex) at the first element of the string, to match Brazilian street type descriptions. We have found two types of errors:

1 - Typos:

> Rue / Ruas => Rua
> Praca => Praça

2 - Abbreviations:

> R / R. => Rua
> Est / Estr => Estrada
> Av / Av. => Avenida
> Trav => Travessa
> Rod / Rod. => Rodovia
> Pca / Pça => Praça

2.2 - ZIP Code Inconsistencies:

Source code: audit_zipcodes.py

We modified the previous street type code to check for inconsistencies in ZIP codes. The important aspects to keep in mind are:

- The first digit must be '2' for Rio de Janeiro.
- The ZIP code must be 8 numbers long, with a dash after the 5th digit (2XXXX-XXX).

The main problems found were:

- No minus sign '-', only integers. It's the most common problem found. It's understandable, many people write it that way.
- Odd characters, like backslashes, blank spaces and dots ('\', '.', ' ')
- Only 2 ZIP codes not starting with '2'
- ZIP codes with 5 numerical digits. The last 3 digits were added about 10 years ago and some sources still don't reflect the fact.

3. XML to SQL Import

After auditing our data, we need to import it to our SQL database:

> Step 1: Fix the problems found while transforming the OSM file into tabular CSV files. Source code: data.py
> Step 2: Import the CSV files to our SQL database. Source code: database.py

We've chosen SQLite as our database manager for being light and independent of a server, in spite of having a more limited set of functions. Connecting to our database:

```
import sqlite3
import pprint

db = sqlite3.connect("rio_de_janeiro.db")
cur = db.cursor()
```

4. Basic Descriptive Statistics and Sanity Checks

4.1 Number of Nodes and Ways

Do we have the same number of nodes and ways as in our original osm file?

```
nodes_count = cur.execute('SELECT count(*) FROM nodes')
print "number of nodes:", nodes_count.fetchone()[0]
```

number of nodes: 1,615,351

```
ways_count = cur.execute('SELECT count(*) FROM ways')
print "number of ways:", ways_count.fetchone()[0]
```

number of ways: 193,769

The import was successful: both OSM and SQL files have the same number of nodes and ways.

4.2 Correspondence between ways_nodes and nodes tables

Do all node_id in ways_nodes table have a correspondence in nodes table?

```
outer_join = cur.execute('''
SELECT ways_nodes.node_id AS missing_node_id, nodes.id AS null_id, count(*) AS qty
FROM ways_nodes LEFT JOIN nodes ON ways_nodes.node_id = nodes.id
WHERE null_id IS NULL
GROUP BY missing_node_id
''')

pprint.pprint(outer_join.fetchall())
```

OK! All node_id are filled with existing IDs in nodes table as expected.

4.3 Number of tag elements in node elements

```
count_distinct = cur.execute('''
SELECT count(DISTINCT nodes_tags.id) AS distinct_qty
FROM nodes_tags JOIN nodes ON nodes_tags.id = nodes.id
''')

pprint.pprint(count_distinct.fetchall())
```

Node elements with tag elements: 52,681
Only 3.2% of all node elements have tag elements.

4.4 Number of tag elements in way elements

```
count_distinct = cur.execute('''
SELECT count(DISTINCT ways_tags.id) AS distinct_qty
FROM ways_tags JOIN ways ON ways_tags.id = ways.id
''')

pprint.pprint(count_distinct.fetchall())
```

Way elements with tag elements: 189,054
98% of all way elements have tag elements.

5. Data Analysis

5.1 Top 10 Contributing Users

```
top_users = cur.execute('''
SELECT joined_tables.user, COUNT(*) AS count
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) joined_tables
GROUP BY joined_tables.user
ORDER BY count DESC
LIMIT 10
''')

pprint.pprint(top_users.fetchall())
```

```
[(u'Alexandrecw', 373474), (u'smaprs_import', 185068), (u'ThiagoPv', 184319), (u'AlNo', 163244), (u'Import Rio', 84927), (u'Geaquinto', 69218), (u'Nighto', 64816), (u'Ricardo Mitidieri', 58544), (u'Thundercel', 54138), (u'M\xe1rcio V\xedn\xedcius Pinheiro', 37055)]
```

5.2 Top 10 Amenities

```
top_ammenities = cur.execute('''
SELECT value, COUNT(*) as count
FROM nodes_tags
WHERE key="amenity"
GROUP BY value
ORDER BY count DESC
LIMIT 10
''')

pprint.pprint(top_ammenities.fetchall())
```

```
[(u'school', 1554), (u'bicycle_parking', 1475), (u'restaurant', 1040), (u'fast_food', 821), (u'bank', 498),
 (u'fuel', 455), (u'place_of_worship', 400), (u'pub', 391), (u'telephone', 374), (u'pharmacy', 353)]
```

5.3 Streets with Higher Concentration of Restaurants and Fast-Foods

```
restaurant_streets = cur.execute('''
SELECT ways_tags.key, ways_tags.value, COUNT(*) AS qty_restaurants
FROM ways_nodes JOIN ways_tags ON ways_tags.id = ways_nodes.id JOIN nodes_tags ON nodes_tags.id =
ways_nodes.node_id
WHERE ways_tags.key = "street" AND nodes_tags.key="amenity" AND (nodes_tags.value = 'restaurant' OR
nodes_tags.value = 'fast_food')
GROUP BY ways_tags.key, ways_tags.value
ORDER BY qty_restaurants DESC
LIMIT 10
''')

pprint.pprint(restaurant_streets.fetchall())
```

[(u'street', u'Rua Da Conceição', 9), (u'street', u'Rua Do Catete', 4), (u'street', u'Rua Barata Ribeiro', 3), (u'street', u'Rua Das Laranjeiras', 2), (u'street', u'Avenida Prado Junior', 1), (u'street', u'Rua Da Quitanda', 1), (u'street', u'Rua Professor Alvaro Ramos', 1), (u'street', u'Travessa Dos Tamoios', 1)]

This result is very disappointing. There are 2167 streets, 1040 restaurants and 821 fast-foods in Rio's OpenStreet map. Obviously there are many more streets with lots of restaurants, but they don't appear together with street names.

5.4 Biggest Religions by Number of Temples

```
biggest_religion = cur.execute('''
SELECT nodes_tags.value, COUNT(*) as count
FROM nodes_tags JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value="place_of_worship") i ON
nodes_tags.id=i.id
WHERE nodes_tags.key="religion"
GROUP BY nodes_tags.value
ORDER BY count DESC
LIMIT 3
''')

pprint.pprint(biggest_religion.fetchall())
```

[(u'christian', 343), (u'spiritualist', 6), (u'jewish', 5)]. As expected, Christianity is the biggest religion in Brazil, which is reflected in the number of Christian temples.

5.5 Popular Cuisines

```
popular_cuisines = cur.execute('''
SELECT nodes_tags.value, COUNT(*) as count
FROM nodes_tags JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value="restaurant") i ON nodes_tags.id=i.id
WHERE nodes_tags.key="cuisine"
GROUP BY nodes_tags.value
ORDER BY count DESC
LIMIT 10
''')

pprint.pprint(popular_cuisines.fetchall())
```

[(u'pizza', 82), (u'regional', 75), (u'italian', 38), (u'japanese', 36), (u'brazilian', 20), (u'steak_house', 16), (u'barbecue', 13), (u'international', 11), (u'chinese', 8), (u'seafood', 8)]. Still very disappointing number of restaurants on the map.

6. Conclusion

Data about Rio de Janeiro encompasses a larger area beyond its political limits, known as "Rio de Janeiro Metropolitan Area".

The existing data is of a fair quality for an open source and often manually inputted data, although some 'static' elements, which should fit well in `<node>`elements, were found in `<way>` elements. To a reasonable extent the work done in this project to clean the data was well performed, but there should be a way to prevent users from inserting the most common typos.

A particular problem found during the wrangling phase could not be fixed: the lack of information about `<node>` elements, with only 3% of them containing `<tag>` elements. For a successful fix, external data with lat/long coordinates should be used to feed our map. Unfortunately, there isn't such database freely available at the present time.

The lack of information on `<node>` elements could be associated to the fact that many of them only exist as reference to `<way>` elements, indicating the orderly "dots" (static locations) along a path. But this wouldn't explain the lack of information about restaurants in a city with 6 Million inhabitants. The largest concentration points to only 9 restaurants on the same street, while the second position has only 4 of them. There is clearly a problem of poor POI (points of interest) coverage.

Suggestions and Ideas

The opportunities to improve the map are endless. Below are a few of them:

1 - Facilitate user collaboration

It should be simple and straightforward for users to collaborate on the go, making it a live interaction. With thousands or millions of users reporting outdated, missing or misplaced information as they use the map, it can be up to date more often than not. The more the project depends on "super users" to upload information, the greatest are the chances of the map being misleading. The more accurate, the more users it will attract, creating a virtuous cycle.

2 - GPS Data from Public or Partner Sources

Batch importing public GPS information into the map would significantly improve it, bringing more people to visit and contribute to it. Another possibility is to find private partners willing to further develop the solution by providing data in exchange for some sort of advanced privileges or rights. Examples of such sources are parcel services and government bodies related to transport, business or tourism sectors.

3 - Predefined Values

To avoid typos, lowercase/uppercase, language and other problems, the 'k' tags could be globally curated and get it's own id. Those not in the list (which could also not be fixed into one of the predefined categories by wrangling) would go into an 'other' category. This way tag names would be standardized from the start, not allowing users to freely input whatever name they wish. Those contributors willing to improve the map would focus on the 'other' category or on new data.

4 - Language Synonyms

Language synonyms could be placed in another tag to be created, using a new 'id' reference. This is easy after standardizing the 'k' tag values. This way, tourists would be able to find landmarks and ways using their mother language.

5 - Automated cleaning of existing records

There should be an effort to batch clean existing map data using a bot, while also avoiding the input of typos and other errors by users (for example by the use of predefined values, as explained above).

With these simple measures, OpenStreetMap could better compete with solutions offered by Google (Google Maps) and Microsoft (Bing), which are at the present more complete and user friendly.