Open Street Map Project

June 16, 2017

1 Open Street Map Data Wrangling Project

1.1 Location: Edinburgh Scotland

By complete chance my wife and two good friends ended up here on the same days in the same hotel. The best I can do is this map.

The data file for the whole city of Edinburgh was rather large so I took a manually selected subset of the city and exported it using the Overpass API.

1.2 Importing the schema to MySQL

The provided .sql had to be modified for use with MySQL because the column names **key** and **value** are reserved words in MySQL. They simply need to be escaped by surrounding them in back-ticks (e.g. **`key`** and **`value`**) in the .sql file.

From the command line:

```
mysql -u username -p database_name < schema_file.sql
```

StackOverflow

1.3 Importing the csv files to MySQL

From command line (repeat for each of the csv files):

```
mysqlimport --ignore-lines=1 --fields-terminated-by=, --verbose --local -u root -p open_street_
```

Import csv to MySQL

1.4 Data Issues and Questions

- Skipped nodes records on import to MySQL
- Predominance of node_tags k="source" and slight variations of the v="values"
- v attributes with commas were truncated when being loaded into the database

1.4.1 Skipped node records on import to MySQL

- Using the --verbose flag during import I was able to see that ***165,602*** out of ***195,524*** nodes records were skipped (i.e. not imported into the table). After some investigation of the raw nodes.csv file I noticed that the id values were up to 10 digits long. In the open_map_project_schema.sql file the id fields were set to INT. I then looked at the MySQL documentation and was able to determine that the INT type can only store values up to ***4,294,967,295*** assuming the field is UNSIGNED. Since the schema file only specified INTEGER the limit was ***2,147,483,647***. The largest node ids were larger than the unsigned limit. I changed all of the columns that were related to ids to the BIGINT type. After that change all nodes loaded successfully.

1.4.2 Node tags where k="source"

While exploring the nodes_tags table I noticed that **35,693** out of **159,704** or **22.35%** of the key values were source.

```
SELECT `key`, COUNT(*) as cnt
FROM nodes_tags
GROUP BY `key`
ORDER BY cnt DESC;
```

Looking at the distribution of values within the keys we see that the vast majority of values, **35,043** are **survey** (**19,901**) and **Bing** (**15,142**). In the remainder of the list there are some minor issues. For example, there are records with values of naptan_import;survey, naptan_import; survey, naptan_import/survey, and naptan_import;survet. Additionally, there were a few combinations of Bing;survey and NLS_OS_Edinburgh_map_YYYY;Bing;survey.

Show all keys with value == 'source'

```
SELECT `value`, COUNT(*) as cnt
FROM nodes_tags
WHERE `key`='source'
GROUP BY `value`
ORDER BY cnt DESC;
```

Show any value containing 'naptan_import' where key == 'source'

```
SELECT `value`, COUNT(`value`) AS cnt
FROM nodes_tags
WHERE `key` = 'source' AND `value` REGEXP "naptan_import"
GROUP BY `value`
ORDER BY cnt DESC;
```

Show any value containing 'Bing' where key == 'source'

```
SELECT `value`, COUNT(`value`) AS cnt
FROM nodes_tags
WHERE `key` = 'source' AND `value` REGEXP "Bing"
GROUP BY `value`
ORDER BY cnt DESC;
```

To handle these minor variances I decided to use only the first entry in any record with ';'. This change was made within the `populate_tags` helper function in `prep_for_database.py`

This code:

```
tag_detail['value'] = value
```

Was changed to this code:

```
val_strs = val_strs.split(';')
tag_detail['value'] = val_strs[0]
```

1.4.3 V Attributes with Commas

I noticed strange values in the nodes_tags table after importing the csv files. After digging in to some examples it became clear that the presence of commas was causing the field to be truncated when importing the csv to the database. To deal with this issue I used a simple regular expression to check for commas in the value string and then used the `string.replace()` method to replace all commas with blank. Alternatively, I could have escaped the commas, but removing them seemed simpler.

```
In [3]: %%html
        <style>
        table {float:left}
        </style>
```

```
<IPython.core.display.HTML object>
```

1.5 Data Overview

1.5.1 File Sizes

File	Size
endinburgh_scotland.osm	55 MB
nodes.csv	16 MB
nodes_tags.csv	5.3 MB
ways.csv	2.2 MB
ways_nodes.csv	7.4 MB
ways_tags.csv	3.6 MB

1.5.2 Unique Users

Since both the nodes table and ways table contain the field **uid** and there is no foreign key relationship the number of unique users will be a union of the unique users from each table. Using `UNION` automatically removes duplicates so all we need to do is count the number of records from the result of the union.

```
SELECT COUNT(*)
FROM (SELECT uid FROM nodes
UNION
SELECT uid FROM ways) as users;
```

There are **291** unique users in the dataset.

1.5.3 Number of Nodes and Ways

According the the schema, nodes are unique so each record in the nodes table has a unique id

```
SELECT COUNT(*)
FROM nodes;
```

There are **195,524** nodes.
Similarly, ways records are unique

```
SELECT COUNT(*)
FROM ways;
```

There are **37,167** ways.

1.5.4 Tree Friendly

There are **14,667** trees included in this subset of the Edinburgh data.

```
SELECT `key`, `value`, COUNT(*) as cnt
FROM nodes_tags
WHERE `key` = 'natural'
GROUP BY `key`, `value`
ORDER BY cnt DESC;
```

Key	Value	Count
natural	tree	14,467
natural	peak	7
natural	spring	2
natural	cliff	1
natural	mud	1

1.5.5 Bicycle Friendly too

The most frequently listed **amenity** is **bicycle_parking**.

```
SELECT `key`, `value`, COUNT(*) as cnt
FROM nodes_tags
WHERE `key` = 'amenity'
GROUP BY `key`, `value`
```

```
ORDER BY cnt DESC
LIMIT 10;
```

Key	Value	Count
amenity	bicycle_parking	394
amenity	restaurant	386
amenity	bench	298
amenity	cafe	284
amenity	fast_food	188
amenity	pub	168
amenity	motorcycle_parking	131
amenity	telephone	128
amenity	post_box	108
amenity	atm	103

1.5.6 Sources

I was initially surprised by the frequency of **survey** and **Bing** as source values in the nodes_tags, but after looking at the OpenStreetMap Wiki these listed among the most common values for **human mappers** in the **Usage Statistics** section.

```
SELECT `key`, `value`, `type`, COUNT(*) as cnt
FROM nodes_tags
GROUP BY `key`, `value`, `type`
ORDER BY cnt DESC
LIMIT 10;
```

Key	Value	Count
source	survey	19,901
city	Edinburgh	15,155
source	Bing	15,142
natural	tree	14,667
country	GB	14,176
denotation	urban	12,663
entrance	yes	1,158
barrier	gate	927
barrier	bollard	513
housenumber	1	509

1.5.7 Top Contributors

To get a view of what users were making the most contributions I combined all of the activity from nodes, nodes_tags, ways, ways_nodes, and ways_tags. Every entry in these tables is attributed to a user.

```
SELECT user, COUNT(*) as cnt
FROM
    ((SELECT n.user FROM nodes n
        JOIN
            (SELECT id FROM nodes
                UNION ALL
            SELECT id FROM nodes_tags) AS na
        ON n.id = na.id)
    UNION ALL
    (SELECT w.user FROM ways w
        JOIN
            (SELECT id FROM ways
                UNION ALL
            SELECT id FROM ways_nodes
                UNION ALL
            SELECT id FROM ways_tags) AS wa
        ON w.id = wa.id))
    AS all_activity
GROUP BY user
ORDER BY cnt DESC
LIMIT 10;
```

User	Records
eric_	284,268
sophiemccallum	262,818
eisa	90,274
leilaz	55,668
drnoble	23,543
saintam1	12,090
rob_michel	10,793
sairfeet	9,982
c0d0	6,801
Rostranimin	5,906

1.5.8 Ideas

The biggest issue with this dataset is that when a node has only a single node_tag it is a k="source" attribute, which refers to the source of the node information. This is different from additional node_tags which are descriptive of the location or item being identified (e.g. k="highway" v="traffic_signals"). Though both are node_tags these seem like fundamentally different information and any analysis should probably consider them as separate.

It would be interesting to understand why the records in the data set are so heavily concentrated in only two users. Do other areas have similarly skewed distributions of user contributions? I would also be curious to expand the map area and see how far these users contributions extend.

1.5.9 Data quality

In general, the data that is available seems reasonably clean. If I were to continue to make improvements I may fix minor issues such as keys with similar meanings (e.g. **url** and **website**). I would also dig further into the use of non-ascii characters. In this project I used a function to detect any strings with non-ascii characters and excluded them.

In [ ]: