

# OpenStreetMap Project Data Wrangling with SQL

Ju Yang

## Contents

1. Problems encountered in the map.....	1
1.1 FIXME or fixme .....	2
1.1.1 "FIXME": map the rest of series brass plaques .....	2
1.1.2 "fixme": estimated bicycle parking position.....	2
1.1.3 "fixme": store under reconstruction .....	2
1.1.4 "fixme": FIKA coffee shop position uncertain .....	2
1.1.5 "fixme": a grocery shop position uncertain.....	2
1.1.6 "fixme": continue .....	3
1.1.7 "fixme": missing address & hour information .....	3
1.1.8 "fixme": missing all details .....	3
"FIXME" or "fixme" auditing summary.....	3
1.2 Node keys containing problem chars .....	3
1.3 Node keys in other category.....	4
1.4 Audit street names of nodes and ways .....	4
1.4.1 addr:street contains more than street information .....	4
1.4.2 Variation of expected street names: abbreviation and typos .....	4
1.4.3 Numbered street name without "street" .....	4
1.4.4 End with direction "East", "West", "South", "North" abbreviation .....	4
1.5 Audit postcode inconsistency.....	5
2. Overview of the data .....	5
2.1 Export XML data to csv .....	5
2.2 Import csv to database .....	5
2.3 Dataset general information .....	5
2.4 Explore the database with SQL query .....	5
3. Other ideas about the datasets .....	6
3.1 Database diagram: a User Table.....	7
3.2 Who are the contributors? .....	7

## 1. Problems encountered in the map

In this study, I first group node tag keys into four tag categories in a dictionary:

- "lower", for tags that contain only lowercase letters and are valid
- "lower\_colon", for otherwise valid tags with a colon in their names
- "problemchars", for tags with problematic characters
- "other", for other tags that do not fall into the other three categories

```
lower = re.compile(r'^([a-z]|_)*$', re.IGNORECASE)
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$', re.IGNORECASE)
problemchars = re.compile(r'[=+/\&<>\'\"?%#$@\\.\ \t\r\n]', re.IGNORECASE)
```

Using `audit_node_tags.py`, we can tell

The number of keys for each type is:  
 {'lower': 280, 'lower colon': 176, 'others': 53, 'problemchars': 9, 'total': 518}

By auditing each tag category and values, I found the following problems

## 1.1 FIXME or fixme

These tags represent values that are either missing or uncertain.

`audit_prob_node_key.py`

### 1.1.1 “FIXME”: map the rest of series brass plaques

```
node id, user id, tags containing 'FIXME':
('3579300780', '139424', {'k': 'FIXME', 'v': 'series of brass plaques set into sidewalk; need to map the rest and add descriptions, please (and pics to Wikimedia Commons!)}))
('3579300781', '139424', {'k': 'FIXME', 'v': 'series of brass plaques set into sidewalk; need to map the rest and add descriptions, please (and pics to Wikimedia Commons!)}))
('3579300782', '139424', {'k': 'FIXME', 'v': 'series of brass plaques set into sidewalk; need to map the rest and add descriptions, please (and pics to Wikimedia Commons!)}))
```

These 3 nodes have consecutive ids and same user id. They are series of brass plaques set into sidewalk and need to map the rest and add descriptions. These nodes themselves are accurate, so I decide to include them in my data.

### 1.1.2 “fixme”: estimated bicycle parking position

Part of the output:

```
node id, user id, tags containing 'fixme'
('419360249', '158826', {'k': 'fixme', 'v': 'really located on the street ?'})
('419360607', '91056', {'k': 'fixme', 'v': 'position'})
('419361445', '345833', {'k': 'fixme', 'v': 'resurvey'})
```

All of these nodes represent bicycle\_parking. Some of them are cityracks. Fixme information suggests the location may not be accurate, on the street or by the street. The accuracy of these tags is in question, and the problem seems to be that the location coordinate is ON the street, not BY the street, which I think is okay for a general purpose mapping. So I decide to include them.

### 1.1.3 “fixme”: store under reconstruction

```
node id, user id, tags containing 'fixme'
('2818577524', '3779040', {'k': 'fixme', 'v': 'The former C-Town has been demolished along w/ the adjacent parking lot. Construction ongoing until Spring 2017, planned multi-story residential w/ new C-Town on ground floor.'})
```

Because the new store will be open in 2017, I will keep this node.

### 1.1.4 “fixme”: FIKA coffee shop position uncertain

Part of the output

```
node id, user id, tags containing 'fixme'
('3792495211', '3435069', {'k': 'fixme', 'v': 'Check position'})
```

An example node:

```
<node id="3792495211" lat="40.7384625" lon="-73.9962727" version="2" timestamp="2015-12-05T04:29:58Z" changeset="35762084" uid="3435069" user="Mytle Ott">
  <tag k="name" v="FIKA 6th Avenue"/>
  <tag k="fixme" v="Check position"/>
  ...
</node>
```

I checked on Google map with the given coordinates and name of the store, and found the position is correct or within 100 ft of the coffee shop.

### 1.1.5 “fixme”: a grocery shop position uncertain

```
node id, user id, tags containing 'fixme'
```

```
('3819506465', '336460', {'k': 'fixme', 'v': 'position'})
```

An example node:

```
<node id="3819506465" lat="40.7226087" lon="-73.9830098" version="2" timestamp="2016-02-10T09:11:27Z" changeset=
"37119476" uid="336460" user="robgeb">
  <tag k="name" v="Ben's Deli and Groceries"/>
  <tag k="shop" v="supermarket"/>
  <tag k="fixme" v="position"/>
  ...
</node>
```

The position is actually correct (<https://www.yelp.com/biz/bens-deli-new-york>) I will include this node.

### 1.1.6 “fixme”: continue

Part of the output:

```
node id, user id, tags containing 'fixme'
('1721287290', '238370', {'k': 'fixme', 'v': 'continue'})
```

Nodes that constitute the final known point in a way. These suggests these nodes may need to include 'noexit' in the tag key, and 'yes' as a value. <http://wiki.openstreetmap.org/wiki/Key:noexit>

I will keep these nodes and add to these nodes a new tag:

```
<tag k="noexit" v="yes"/>
```

### 1.1.7 “fixme”: missing address & hour information

```
node id, user id, tags containing 'fixme'
('4431954764', '1190762', {'k': 'fixme', 'v': 'address & hours'})
```

After checking Google map(<https://www.google.com/maps/place/Dunkin'+Donuts/@40.6768651,-73.9182797,17z/data=!4m1!1m12!4m11!1m6!1m2!1s0x89c25c689b2d2961:0x7ffba5dd29409a4d!2sDunkin'+Donuts,+1993+Atlantic+Avenue,+Brooklyn,+NY+11233!2m2!1d-73.9160848!2d40.6770271!1m3!2m2!1d-73.9160551!2d40.6769856!3m4!1s0x89c25c689b2d2961:0x7ffba5dd29409a4d!8m2!3d40.6770271!4d-73.9160848>), I found that it is open 24 hours and address is 1993 Atlantic Ave, Brooklyn, NY 11233. I will keep this node and add the following tags

```
<tag k="addr:city" v="Brooklyn"/>
<tag k="addr:country" v="US"/>
<tag k="addr:housenum" v="1993"/>
<tag k="addr:postcode" v="11233"/>
<tag k="addr:state" v="NY"/>
<tag k="addr:street" v="Atlantic Ave "/>
<tag k="opening_hours" v="24/7"/>
```

### 1.1.8 “fixme”: missing all details

Part of the output

```
node id, user id, tags containing 'fixme'
('4431954765', '1190762', {'k': 'fixme', 'v': 'all details'})
```

These nodes have consecutive ids and same user id. They seem to be a series of shops added by the same user. I will include these nodes in my data.

### “FIXME” or “fixme” auditing summary

Of keys that contain "FIXME" or "fixme", most of the values are in fact correct except 1.1.8 where there is missing information. Node tags for 1.1.6 and 1.1.7 could be updated.

## 1.2 Node keys containing problem chars

```
node tag keys with problem chars
set(['Rehearsal space',
    'cityracks.housenum',
    'cityracks.installed',
    'cityracks.large',
    'cityracks.rackid',
    'cityracks.small',
    'cityracks.street',
    'geographical feature',
    'service area'])
```

These keys contain white space or '.' To make the data consistent, it may be better to convert white space to underscore and convert "." to colon.

- "Rehearsal space" --> "Rehearsal\_space"
- "cityracks.housenum" --> "cityracks:housenum"

The conversion function `convert_key` is defined in `audit_node_tags.py`.

### 1.3 Node keys in other category

- Some keys contain number such as 'name\_1'
- Some keys contain more than 1 colon such as "railway:maxspeed:diverging"
- The key 'alt\_name:1' does not seem to be consistent with 'name\_1' to represent different names. It may be better to convert it to 'alt\_name\_1'

### 1.4 Audit street names of nodes and ways

`audit_street_names.py`

```
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road",  
"Trail", "Parkway", "Alley", "Plaza", "Commons", "Broadway", "Expressway", "Terrace", "Center", "Circle", "Crescent", "High  
way", "Way"]
```

#### 1.4.1 addr:street contains more than street information

Example output:

```
'11217': set(['305 Schermerhorn St., Brooklyn, NY 11217']),
```

An example code:

```
<node id="3875234457" lat="40.6876305" lon="-73.9819376" version="1" timestamp="2015-12-03T21:43:30Z" changeset=  
"35736421" uid="3430001" user="DrDanielleSheypuk">  
  <tag k="name" v="Govinda&#39;s Vegetarian"/>  
  <tag k="amenity" v="restaurant"/>  
  <tag k="wheelchair" v="no"/>  
  <tag k="addr:street" v="305 Schermerhorn St., Brooklyn, NY 11217"/>  
  <tag k="wheelchair:description" v="My colleagues frequent this restaurant in a Brooklyn ashram known for  
  their delicious eggplant lasagna. But, there is a HUGE step to get in. So easy to put a ramp- shame on  
  you Hare Krishna!"/>  
</node>
```

When I check this node, I found there is no other addr: information except the addr:street. One idea is to convert the address to different addr tags.

#### 1.4.2 Variation of expected street names: abbreviation and typos

There are different variations and typos of these expected street names, and I will convert variations to the expected names using `update_name` function.

- Avenue: Ave, Ave., Avene, Aveneu, ave, avenue
- Boulevard: Blv., Blvd, blvd
- Broadway: Broadway.
- Center: Ctr
- Parkway: Pkwy:
- Plaze: Plz:
- Road: Rd
- Street: ST, St, St., Steet, Streeet, st, street

#### 1.4.3 Numbered street name without "street"

```
'27th': set(['W 27th']),  
'42nd': set(['West 42nd']),
```

Because there are many streets named by numbers, the addr:street may only contain the number, without "street". I decide add "Street" to these addr:street values ending with "st", "nd", "rd", "th" and not containing "street" or "Street", using `update_number_street` function.

#### 1.4.4 End with direction "East", "West", "South", "North" abbreviation

```
'Williamsburg Street East'
```

```
'Central Park North'
'Central Park South'
'Central Park West'
```

I notice there is abbreviation of “South” to “S” at the end of the street name

```
'S': set(['Central Park S', 'Park Avenue S', 'Van Cortlandt Park S']),
```

I will convert direction to its full name using `full_direction` function.

## 1.5 Audit postcode inconsistency

```
audit_postcode.py
postcode_re = re.compile(r'^\d{5}([\-\]?\d{4})?$',)
```

unexpected postcode values:

```
'(718) 778-0140',
    wrong value, should be the phone number
```

```
'100014',
    typo, should be 10014
```

```
'11201;11231',
    a range of postcodes of ways
```

```
'320', '83',
    number, not sure what it means, could be housenumber
```

```
'NY 10002',
    contain both state and postcode, may need to add a new tag for state
```

```
'New York, NY 10065'
    contain both city, state, and postcode, may need to add new tags
```

## 2. Overview of the data

### 2.1 Export XML data to csv

```
export_to_csv_schema.py
```

```
skip_csv_header.py
```

Export XML file to csv files with defined schema in `myschema.py` using `export_to_csv_schema.py`

Because in `nodes_tags.csv`, each node id may have multiple tags, and there is no unique primary unique key in the file, I decide to append one column called `id` to assign a unique INTEGER id to each row using function. Same approach is used for `ways_tags.csv` and `ways_nodes.csv`. I also remove header from csv files to import the data to database. I use the following function to create new csv files without header.

### 2.2 Import csv to database

I use SQLite command `linr` to create database, as recorded in `create_db.txt`

### 2.3 Dataset general information

filename	Size (KB)	size (MB)
Manhattan.osm	467,193 KB	456 MB
Manhattan.db	291,981 KB	285 MB
sample.osm (k = 50)	9,466 KB	9.24 MB
nodes.csv	165,124 KB	161 MB
ways.csv	20,671 KB	20.1 MB
ways_nodes.csv	62,776 KB	61.3 MB
nodes_tags.csv	10,247 KB	10.0 MB
ways_tags.csv	54,203 KB	52.9 MB

### 2.4 Explore the database with SQL query

1) How many nodes and ways are there?

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

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

There are 1,834,648 nodes and 313,418 ways in the dataset.

## 2) How many nodes are historic? How many nodes are shops?

```
SELECT COUNT(*)
FROM Node_tag
WHERE Key = "historic";
```

```
SELECT COUNT(*)
FROM Node_tag
WHERE Key = "shop";
```

There are 283 historic nodes, 2042 shops in the dataset.

## 3) How many distinct street names? What are the top 5 common street names?

```
SELECT COUNT(DISTINCT(Value))
FROM Node_tag
WHERE Key = "street";
```

```
SELECT Value, COUNT(*) as num
FROM Node_tag
WHERE Key = "street"
GROUP BY Value
ORDER BY num DESC
LIMIT 5;
```

There are 2306 unique street names in the dataset and the top 5 common street names are

```
Broadway|2062
3rd Avenue|825
Amsterdam Avenue|610
Lexington Avenue|587
5th Avenue|577
```

Broadway, which runs north-to-south through Manhattan island, is the most common street name. Other major avenues follow.

## 4) How many unique users?

```
SELECT COUNT(DISTINCT(subq.UserId))
FROM (SELECT UserId FROM Node UNION ALL SELECT UserId FROM Way) AS subq;
```

There are 2114 unique users.

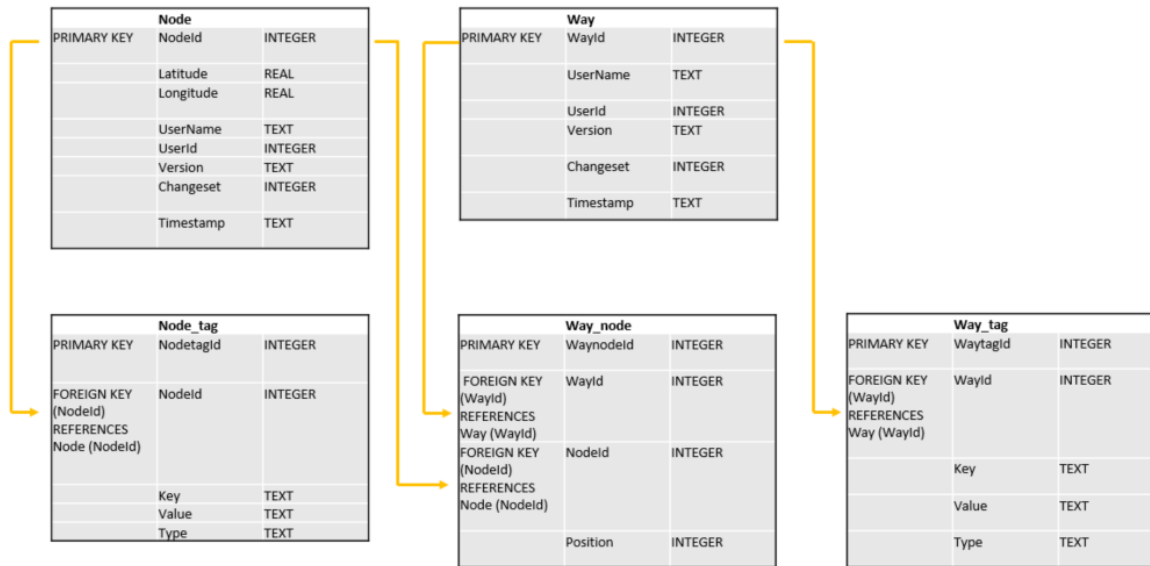
## 5) Who are the top contributors?

```
SELECT subq.UserName, COUNT(*) AS num
FROM (SELECT UserName FROM Node UNION ALL SELECT UserName FROM Way) AS subq
GROUP BY subq.UserName
ORDER BY num DESC
LIMIT 5;
```

```
Rub21_nycbuildings|1112566
ediyen_nycbuildings|124625
ingalls_nycbuildings|111472
smlevine|104442
lxbarth_nycbuildings|90579
```

## 3. Other ideas about the datasets

### 3.1 Database diagram: a User Table



The table Node and Way contains information of users (UserId and UserName). While users contribute to the data, user information should be in a different table and linked to other tables with UserId. This will make query on users more easily and we don't need to do subquery as in section 2.

### 3.2 Who are the contributors?

- 1) Top contributors' user names contain nycbuildings, suggesting that the data is likely from an official group NYC Department of Buildings
- 2) Out of 2114 users, 673 users that contribute once, 85 users contribute more than 1000 times and only 14 users contribute more than 10000 times. This suggests that most users are occasional contributor and this may explain why the data format is not uniform. As more data is imported and modified, it may be beneficial to have an OSM format standard guideline to improve the completeness and uniformity of the data.