

# OpenStreetMap Project

## Data Wrangling with MongoDB

---

Chun Zhu

Map Area: San Jose, California, United States

<https://www.openstreetmap.org/relation/112143>

[https://s3.amazonaws.com/metro-extracts.mapzen.com/san-jose\\_california.osm.bz2](https://s3.amazonaws.com/metro-extracts.mapzen.com/san-jose_california.osm.bz2)

## Section 1: Problems Encountered in the Map

---

### Unexpected tags

`mapparser.py` was used to count occurrences of each tag, with a result:

- `bounds`: 1
- `member`: 6468
- `nd`: 944840
- `node`: 801525
- `osm`: 1
- `relation`: 662
- `tag`: 468411
- `way`: 100334

Additional functionality was added to `mapparser.py` to examine the keys stored in each tag child- element of 'node' and 'way', in the `k` attribute.

Unexpectedly, I found the tag keys 'type' and 'address':

```
('type', 29), ('address', 4)
```

Compared to the total number of node and way, the tag key 'type' and 'address' appeared quite fewer. Since these 2 tag keys conflict with the keys I am about to use in the dataset, I will remit these 2 tags from populating into the database.

## Multiple Zip Codes

Zip codes are common search criteria. For node, zip code was presented in tag 'addr:postcode'; For way, zip codes were presented in the data under various permutations of tiger:zip\_left, tiger:zip\_right. The zip\_left and right are for the left and right side of the road, if you are driving along it in the direction of the way.

For node zip code, there are several formats: '95037-4209', 'CA 94035', '95070', 'CA 94088-3453'. To clean the data, I decide to use 5-digit zip code as a standard rule, which requires stripping all leading and trailing characters before and after the main 5-digit zip code. After cleaning, the postcode will be stored in node['address']['postcode'].

For way zip code, tiger:zip\_left and tiger:zip\_right are defined as semicolon delimited lists or colon delimited ranges, like: '94538; 95035', '94085', '94538:95035'. Since way usually contains more than one zip code, I thought that it would be a good idea to collect and serialize all zipcodes from sources into a single array, and populate this into the base of the node under zipcodes, or node['zipcodes'].

## Phone Numbers Inconsistency

Phone numbers were formatted inconsistently. For example,

```
'+1 408-782-8201'  
+1 (408) 376-3516'  
'+1 408 739 7717'  
'+ 408 980 6400'  
'4084507990'  
'+1.408.559.6900'  
'(408) 277-4625'  
u'+1 408-500-3000 \u200e'
```

I used the Python module `phonenumbers` to parse all phone numbers and re-format them to the standard (123) 456-7890.

However, for '+ 408 980 6400', I cannot use `phonenumbers` module to re-format it. So I'll use python regular expression to find this kind of phone number and re-format it.

Additionally, there are 2 tag keys for phone number: 'phone' and 'contact:phone'. As part of the cleaning process, I'll store all phone numbers in 'phone' key in the dataset.

## Abbreviated Street Names

audit.py is used to find out street name abbreviations. I updated all substrings in problematic address strings, such that “1425 E Dunne Ave” becomes “1425 E Dunne Avenue”.

## Section 2: Data Overview

---

This section contains basic statistics about the dataset and the MongoDB queries used to gather them.

File sizes:

- san-jose\_california.osm: 173 MB
- san-jose\_california.osm.json: 198 MB

Number of documents:

```
> db.sanjose.count()
901859
```

Number of nodes and ways:

```
> db.sanjose.find({'type': 'node'}).count()
801525
> db.sanjose.find({'type': 'way'}).count()
100334
```

Number of unique users:

```
db.sanjose.distinct("created.user").length
955
```

Top contributing user:

```
> db.sanjose.aggregate([
...     {'$group': {
...         '_id': '$created.user',
...         'count': {
...             '$sum': 1
...         }
...     }},
...     {'$sort': {
...         'count': -1
...     }},
...     {'$limit': 1
...     }])
```

```
...     ]])
{ "_id" : "nmixer", "count" : 207683 }
```

Number of users contributing only once:

```
> db.sanjose.aggregate([
...     '$group': {
...         '_id': '$created.user',
...         'count': {
...             '$sum': 1
...         }
...     }, {
...         '$group': {
...             '_id': '$count',
...             'num_users': {
...                 '$sum': 1
...             }
...         }, {
...             '$sort': {
...                 '_id': 1
...             }
...         }, {
...             '$limit': 1
...         }
...     ]])
{ "_id" : 1, "num_users" : 190 }
```

Popular cuisines in San Jose:

```
db.sanjose.aggregate([
...     '$match': {
...         'cuisine': {
...             '$exists': 1
...         },
...         'amenity': 'restaurant'
...     }, {
...         '$group': {
...             '_id': '$cuisine',
...             'count': {
...                 '$sum': 1
...             }
...         }, {
...             '$sort': {
...                 'count': -1
...             }
...         }, {
...             '$limit': 3
...         }
...     ]])
{u'_id': u'mexican', u'count': 67},
{u'_id': u'chinese', u'count': 58},
{u'_id': u'pizza', u'count': 43},
```

Zip codes in San Jose:

```
db.sanjose.aggregate([
...     '$match': {
...         'zipcodes': {
...             '$exists': 1
...         }
...     }, {
...         '$unwind': '$zipcodes'
...     }, {
...         '$group': {
...             '_id': '$zipcodes'
...         }
...     }, {
...         '$group': {
...             '_id': 'Zip Codes in San Jose',
...             'count': {
...                 '$sum': 1
...             },
...             'zipcodes': {
...                 '$push': '$_id'
...             },
...         }
...     })
{ "_id" : "Zip Codes in San Jose", "count" : 1108, "zipcodes": [ "94041",
... # truncated
"94770", "94466", "94100", "94955", "94615"] }
```

Most common building types/entries:

```
> db.sanjose.aggregate([
...     '$match': {
...         'building': {
...             '$exists': 1
...         }
...     }, {
...         '$group': {
...             '_id': '$building',
...             'count': {
...                 '$sum': 1
...             }
...         }
...     }, {
...         '$sort': {
...             'count': -1
...         }
...     }, {
...         '$limit': 10
...     })
{u'_id': u'yes', u'count': 33803},
{u'_id': u'house', u'count': 3783},
{u'_id': u'residential', u'count': 3633},
```

```
{u'_id': u'apartments', u'count': 294},
{u'_id': u'roof', u'count': 288},
{u'_id': u'school', u'count': 177},
{u'_id': u'commercial', u'count': 158},
{u'_id': u'office', u'count': 144},
{u'_id': u'retail', u'count': 92},
{u'_id': u'garage', u'count': 72}
```

Most common street address:

```
db.sanjose.aggregate([
...     '$match': {
...         'address.street': {
...             '$exists': 1
...         }
...     }, {
...         '$group': {
...             '_id': '$address.street',
...             'count': {
...                 '$sum': 1
...             }
...         }
...     }, {
...         '$sort': {
...             'count': -1
...         }
...     }, {
...         '$limit': 1
...     }
... ])
{ "_id" : "Hollenbeck Avenue", "count" : 172 }
```

Nodes without addresses:

```
> db.sanjose.aggregate([
...     '$match': {
...         'type': 'node',
...         'address': {
...             '$exists': 0
...         }
...     }, {
...         '$group': {
...             '_id': 'Nodes without addresses',
...             'count': {
...                 '$sum': 1
...             }
...         }
...     }
... ])
{ "_id" : "Nodes without addresses", "count" : 796562 }
```

## Section 3: Conclusion

---

### Additional Ideas

It is interesting that an overwhelming number of nodes (88.3%) do not include addresses and a large number (944840) of `nd` reference tags in the map. This reminds me that we can compress the data by removing `way` tags and waypoint-like nodes, which could reduce the database size dramatically.

There are still several opportunities for cleaning and validation that I left unexplored. The 'building' tag definitely needs cleaning, since the top 1 building type is 'yes', which is weird.

### Comments

While there are many additional opportunities for cleaning and validation, I believe the data set was well-cleaned for the purposes of this exercise.

## References

---

<http://docs.mongodb.org/manual/reference/operator/>

Lesson 6 from Udacity course, "Data Wrangling with MongoDB"