OpenStreetMap Project
Data Wrangling with MongoDB

Map area: San Jose, CA, US

https://www.openstreetmap.org/export#map=17/37.40219/-121.93171 - 971 kb
https://www.openstreetmap.org/export#map=15/37.4068/-121.9241 - 6.1 mb
https://s3.amazonaws.com/metro-extracts.mapzen.com/san-jose_california.osm.bz2 - 282.7 mb

Data description:

For this project I choose San Jose, CA, the largest city within Bay Area and the largest city in Northern California (according to https://en.wikipedia.org/wiki/San_Jose,_California). Also it's the city where I'm living, so I've decided that it will be more interesting to investigate and analyze this data, also to study something new about this area.
Links above can be used to download data of chosen area: starting from the the smallest file that contains only several neighbourhoods and finally the biggest file that contains all OpenStreetMap data about San Jose and closest cities.

Problems encountered in data:

After downloading and analysing street data I've found several problems with data that can cause difficulties in future during querying data from database:

1. Problems with street types

- As soon as data was added manually it usually happens that there is a variety for naming street types, e.g. *"Rd."* instead of *"Road"* or *"Ave"* instead *"Avenue"*. That will cause a huge queries just for looking for one specific street.
- To solve this problem I've created a function for auditing street types according all existed street types in this area (see audit.py/audit_street(name)). Also I've created *street_mapping* dictionary to fix most common abbreviations: *street_mapping = {'St': 'Street', 'Ave': 'Avenue'...}*
- As result the street name became more standardized:
  - Linwood Dr >>>>>> Linwood Drive
  - N McCarthy Blvd >>>>>> N McCarthy Boulevard

2. Street names that contains the whole address
    - Sometimes instead of street name the *“addr:street”* field contains the whole address, e.g. *“West Evelyn Avenue Suite #114”*. These addresses can also contain the abbreviation of the street type inside (*“Zanker Rd., San Jose, CA”*).
    - To fix this problem I’ve added additional check to audit_street function to parse street name, fix it, and return only street name and type.
    - As result all street names became more brief and consistent:
        - Zanker Road, San Jose, CA >>>>>> Zanker Road
        - Linwood Dr >>>>>> Linwood Drive
        - West Evelyn Avenue Suite #114 >>>>>> West Evelyn Avenue
        - Stewart Drive Suite #1 >>>>>> Stewart Drive
        - Zanker Rd., San Jose, CA >>>>>> Zanker Road

3. Mistakes in city names
    - As soon as this territory was discovered by Spanish there are still names that contains foreign symbols (e.g. *“San José”*). Also there are a lot of mistakes in city names.
    - To fix these problems I’ve created function audit_city(name) fo fix city names and problems with spelling.
    - As result such problems were fixed:
        - San José >>> San Jose
        - Los Gatos, CA >>> Los Gatos
        - sunnyvale >>> Sunnyvale
        - SUnnyvale >>> Sunnyvale
        - campbell >>> Campbell
    - Cities by count, descending:

```
> db.san_jose.aggregate([{'$match': {'address.city': {'$exists': 1}}}, {'$group': {'_id': '$address.city', 'count': {'$sum': 1}}}, {'$sort': {'count': -1}}])
{ "_id" : "Sunnyvale", "count" : 3392 }
{ "_id" : "San Jose", "count" : 781 }
{ "_id" : "Morgan Hill", "count" : 373 }
{ "_id" : "Santa Clara", "count" : 288 }
{ "_id" : "Saratoga", "count" : 230 }
{ "_id" : "Milpitas", "count" : 89 }
{ "_id" : "Cupertino", "count" : 57 }
{ "_id" : "Campbell", "count" : 51 }
{ "_id" : "Los Gatos", "count" : 50 }
{ "_id" : "Mountain View", "count" : 7 }
{ "_id" : "Alviso", "count" : 7 }
{ "_id" : "Redwood Estates", "count" : 1 }
{ "_id" : "Felton", "count" : 1 }
{ "_id" : "Moffett Field", "count" : 1 }
{ "_id" : "Mount Hamilton", "count" : 1 }
```

4. Grouping data:

After analysing all data tags that data can contain I've discovered that there are 733 different tags that can be used to describe node in data. The full list and statistics of tags can be found in *tags_statistics.scv*. I've noticed that there are several types of tags, I've decided to highlight the following:

1. **"addr:"** - all these tags contains address information, so they were added to separate dictionary "address", so any data about address of the object can be found just in one place.

   ```
   > db.san_jose.find({'address': {'$exists': 1}}).count()
   22353
   ```

2. **"tiger:"** - these fields contains information from Tiger database containing features such as roads, railroads, rivers, as well as legal and statistical geographic areas. Distinguishing these data into separate dictionary can be useful for specific queries. Aslo "tiger:" fields has versioning (e.g. 'tiger:zip_right', 'tiger:zip_right_1', 'tiger:zip_right_3'...) and it's important to keep all versions on values in one place, I've stored all values in lists. For details please see shape_data.py/shape_data(element) function.
   Example of object:

   ```
   > db.san_jose.findOne({'tiger.zip_right': {'$exists': 1}})
   {
           "_id" : ObjectId("57b7863088299743fd16e7b5"),
           "maxspeed" : "25 mph",
           "node_refs" : [
                   "65468975",
                   "3729899732",
                   "83061230",
                   "615953142",
                   "1124888512",
                   "26065103"
           ],
           "name" : "Soquel Way",
           "created" : {
                   "changeset" : "33805213",
                   "user" : "matthieun",
                   "version" : "16",
                   "uid" : "595221",
                   "timestamp" : "2015-09-04T22:11:02Z"
           },
           "source:maxspeed" : "sign",
           "surface" : "asphalt",
           "tiger" : {
                   "name_base" : [
                           "Soquel"
                   ],
                   "zip_left" : [
                           "94086",
                           "94085"
                   ],
                   "cfcc" : "A41",
                   "county" : "Santa Clara, CA",
                   "name_type" : "Way",
                   "zip_right" : [
                           "94085"
   ```

```
                ]
            },
            "oneway" : "no",
            "type" : "way",
            "id" : "5005992",
            "highway" : "residential"
}
```

Data overview:

This section contains basic statistics about the dataset and db queries:

- Files sizes:
    - san-jose_california.osm - 282.7 mb
      san-jose_california.osm.json - 411 mb
- Time processing file (2.5 GHz Intel Core i5, 8 GB 1600 MHz DDR3):
    - 4 min 17 sec
- Number of documents:
    - > db.san_jose.find().count()
      1447511
- Number of unique users:
    - > db.san_jose.distinct('created.user').length
      1193
- Number on <node>s:
    - > db.san_jose.find({'type': 'node'}).count()
      1276556
- Number of <way>s:
    - > db.san_jose.find({'type': 'way'}).count()
      169678
- Number of ways containing tiger data:
    - > db.san_jose.find({'tiger': {'$exists': 1}}).count()
    - 31091
- Number of ways containing full address:
    - > db.san_jose.find({'address.country': {'$exists': 1}, 'address.state': {'$exists': 1}, 'address.city': {'$exists': 1}, 'address.street': {'$exists': 1}, 'address.housenumber': {'$exists': 1}}).count()
      570

Additional data exploration:

- Top 5 amenities:
  > db.san_jose.aggregate([{"$match":{"amenity":{"$exists":1}}}, {"$group":{"_id":"$amenity", "count":{"$sum":1}}}, {"$sort":{"count": -1}}, {"$limit":5}])

    - { "_id" : "parking", "count" : 1798 }
      { "_id" : "restaurant", "count" : 907 }
      { "_id" : "school", "count" : 540 }
      { "_id" : "fast_food", "count" : 454 }
      { "_id" : "place_of_worship", "count" : 343 }
- Secondary highways with highest speed limit (50 mph):
    - > db.san_jose.aggregate([{'$match': {'maxspeed': '50 mph', 'highway': 'secondary'}}, {'$group': {'_id': '$name', 'count': {'$sum': 1}}}])
        - { "_id" : "McCarthy Boulevard", "count" : 37 }
        - { "_id" : "Bailey Avenue", "count" : 7 }
    - Only two streets. There are much more with lower speed limits.
- Caltrain stations in this region:
    - > db.san_jose.aggregate([{'$match': {'railway': 'station', 'name': {'$exists': 1}}}, {'$group': {'_id': '$name'}}])
      { "_id" : "San Jose Diridon" }
      { "_id" : "Lawrence" }
      { "_id" : "Santa Clara/University" }
      { "_id" : "Morgan Hill" }
      { "_id" : "Blossom Hill" }
      { "_id" : "Tamien" }
      { "_id" : "Sunnyvale" }
      { "_id" : "Capitol" }

Additional ideas:

After analysing OpenStreetMap data I've discovered that mistakes and freestyle abbreviations are not the worst problems that this data contains. As it was discussed above the data for this area contains 733 unique tags. Looking at complete list of all these tags it can be noticed that there are a lot of tags that duplicate information ("name", "palce_name", "name:", etc.) or tags that are not consistent (found only once or twice). And that creates a lot of difficulties for querying in this dataset. If fact as soon as users contribute any data in any format it makes dataset excess and heterogeneous.
There are several ways to solve the problem. The first is very local. In case if these data is accepted as suitable and relevant for some purposes it can be thoroughly cleaned programmatically: some fields can be removed (that duplicates each other), replaced or grouped (another value for the same field, e.g. "addr:housenumber_5") to escape redundancy. This improvements I've added to my final dataset where I've grouped some of data to improve future querying (e.g. tiger data).
Another (more difficult) way is to create more constraints for data contributors. But making restrictions in adding data can cause difficulties for users that will cause improvement in data quality but also reduce data volume.

Also it is noticed that data is incomplete. A lot of nodes doesn’t have fields that they suppose to have, such as address. Creating specific constraints for such fields can also improve data quality.

Conclusion

Working with OpenStreetMap data sometimes can be pretty challenging because data volume can be pretty vast. Converting all data into database makes all processes a lot more faster. But before this we need to make sure that this data is consistent and clean. After improving the process of gathering data and thoughtful data auditing we can create a powerful tool for making interesting analytical conclusions and statistics.