OpenStreetMap Sample Project

Data Wrangling with MongoDB

Xiaolong Yang

CONTENTS

1. Problem encountered in the Map

Using the data.py file to transfer the open-street map information of Tianjin, China, I found three main problems among the data. We would discuss it as below:

1st, Data written in Chinese

Here is an example with the data written in Chinese in the OSM:

<node **id="3904051392"** lat="39.3540266" lon="116.5455919" version="2" timestamp="2015-12-22T07:52:55Z" changeset="36099838" uid="510465" user="轩辕 CN"><tag k="addr:city" v="永清县"/><tag k="addr:housenumber" v="1 号"/><tag k="addr:street" v="恒山北路"/><tag k="name" v="永清台湾工业新城管委会"/><tag k="office" v="government"/></node>

Use the following query to search the message with the ID of *"3904051392"*:

```
def get_db(db_name):
    from pymongo import MongoClient
    client = MongoClient('localhost:27017')
    db = client[db_name]
    return db
```

```
def find(db, query):
    return db.tianjin.find(query)

tag_query = {"id":"3904051392"}
tag_results = find(db, tag_query)
```

Here is the result after the data record in the MongoDB:

{u'_id': ObjectId('568262e78dbf0341d891a09d'), u'address': {
u'city': u'\u6c38\u6e05\u53bf', u'housenumber': u'1\u53f7', u'street': u'\u6052\u5c71\u5317\u8def'},
u'created': {u'changeset': u'36099838', u'timestamp': u'2015-12-22T07:52:55Z', u'uid': u'510465',
u'user': u'\u8f69\u8f7eCN', u'version': u'2'}, u'id': u'**3904051392**',
u'name': u'\u6c38\u6e05\u53f0\u6e7e\u5de5\u4e1a\u65b0\u57ce\u7ba1\u59d4\u4f1a',
u'pos': [39.3540266, 116.5455919], u'type': u'node', u'visible': None}

Another example would be searching all the street name.

Use the following query to search the data with the “street” in their address:

```
road_pipeline = [{"$group":{"_id": "$address.street", "count":{"$sum":1 }}},{ "$sort":{"count" : -1}}]
```

```
road_result = aggregate(db, road_pipeline)

pprint.pprint(road_result)
```

Here is the result after the data record in the MongoDB:

[{u'_id': None, u'count': 315517},
{u'_id': u'\u6f37\u9a6c\u8def', u'count': 5},
{u'_id': u'\u9a6c\u9a79\u6865', u'count': 4},
{u'_id': u'HONG QI Road', u'count': 1},
{u'_id': u'Xinhua East Road', u'count': 1},
*{u'_id': u'Tianta **Dao**', u'count': 1},*
{u'_id': u'Yanhe Road', u'count': 1},
{u'_id': u'\u7ecf\u4e09\u8def', u'count': 1},
{u'_id': u'\u6606\u7eac\u8def', u'count': 1},
{u'_id': u'Xinhua Dong Dao', u'count': 1},
{u'_id': u'Xuefu Road', u'count': 1},
{u'_id': u'Rongyuan Road, Huayuan Industrial Park', u'count': 1},
{u'_id': u'\u5eb7\u5b9a\u8857', u'count': 1},
{u'_id': u'\u5357\u4eac\u8def', u'count': 1},
{u'_id': u'\u6052\u5c71\u5317\u8def', u'count': 1},
{u'_id': u'Jinbei Road', u'count': 1},
{u'_id': u'\u548c\u5e73\u533a\u8d35\u5dde\u8def', u'count': 1},
{u'_id': u'\u5bbe\u9986\u897f\u8def', u'count': 1},
{u'_id': u'\u56f4\u5824\u9053', u'count': 1},
{u'_id': u'\u65b0\u5f00\u8def', u'count': 1},
{u'_id': u'Guangrong Avenue', u'count': 1},
{u'_id': u'Binshui West Street', u'count': 1},
*{u'_id': u'Dong Man Zhong **Lu**', u'count': 1},*
{u'_id': u'\u534e\u5174\u9053', u'count': 1},
{u'_id': u'\u65b0\u73af\u8def', u'count': 1},
{u'_id': u'\u6b23\u82d1\u8def', u'count': 1}]

From here, we can find there are many data was written in Chinese, and the MongoDB cannot read that message out. This Chinese message was written in UTF-8 format, understood by MongoDB and passing the Regex, but it cannot read out as a normal format by python.

2nd, Synonym road name (Spelling of mandarin)

We can find the street message above, ending with not only “Road”, “Street”, “Avenue”, but also “**Dao**” and “**Lu**”. Chinese use “Dao”, a spelling of mandarin to show the meaning of “Avenue”, same to “Lu”, with the meaning of “road”.

I use the following coding in “edit_name.py” to make a change:

```
    expected = ["Street", "Avenue", "Road"]
    mapping = { "Dao": "Avenue", "Lu": "Road"}
    for old, renew in mapping.iteritems():
            m = street_type_re.search(name)
            street_type = m.group()
            if street_type in old:
                name = name.replace(old, renew)
return name
```

```
    for st_type, ways in st_types.iteritems():
            for name in ways:
                better_name =
update_name(name, mapping)
                print name, "=>", better_name
```

The result is following:

```
{'Dao': set(['Tianta Dao', 'Xinhua Dong Dao']),
 'Lu': set(['Dong Man Zhong Lu']),
 'Park': set(['Rongyuan Road, Huayuan Industrial Park'])}
Xinhua Dong Dao => Xinhua Dong Avenue
Tianta Dao => Tianta Avenue
Rongyuan Road, Huayuan Industrial Park => Rongyuan Road, Huayuan Industrial Park
Dong Man Zhong Lu => Dong Man Zhong Road
```

We can find the Chinese saying “dao” and “lu” was replaced.

3rd, Incorrect postal codes

We can find all the postcode in China is a series with 6 numbers. Tianjin’s postcode has the same format in 6 number, started with 3, such as “3XXXXX”.

Use the following query to search the data with the “postcode” in their address:

```
post_pipeline = [{"$group":{"_id": "$address.postcode", "count":{"$sum":1 }}},{ "$sort":{"count" : -1}}]
post_result = aggregate(db, post_pipeline)
print post_result
```

Here is the result after the data record in the MongoDB:

[{u'count': 315539, u'_id': None}, {u'count': 1, u'_id': u'300130'}, {u'count': 1, u'_id': u'300000'}, {u'count': 1, u'_id': u'063000'}, {u'count': 1, u'_id': u'300084'}, {u'count': 1, u'_id': u'300050'}, {u'count': 1, u'_id': u'300300'}, {u'count': 1, u'_id': u'300051'}, {u'count': 1, u'_id': u'300387'}, {u'count': 1, u'_id': u'100176'}, {u'count': 1, u'_id': u'300467'}]
[{u'_id': None, u'count': 315517},

Most of the results follow the format mentioned above, however, there are 2 postcode "100176" and "063000" did not follow the format. After searching online, we can find this two represent some location on the boundary of Tianjin, our target data.

Since the MongoDB don't allow to convert data type, from string into integrator, in the pipeline, I had to change the methods of storage in the "data.py" as follow:

```
if add_key == "postcode":
    if value != None:
        addr[add_key] = int(value)
```

And then edit the postcode result in the query.py:

```
# build the pipeline to change the postcode
re_post_pipeline = [{"$match" : {"address.postcode": {"$type": "int"},"address.postcode": {"$gt" : 300000}}},{"$group":{"_id": "$address.postcode", "count":{"$sum":1 }}},{ "$sort":{"count" : -1}}]
re_post_result = aggregate(db, re_post_pipeline)
print re_post_result
```

Earn the results as follow:

[{u'count': 1, u'_id': 300467}, {u'count': 1, u'_id': 300387}, {u'count': 1, u'_id': 300130}, {u'count': 1, u'_id': 300051}, {u'count': 1, u'_id': 300300}, {u'count': 1, u'_id': 300084}, {u'count': 1, u'_id': 300050}]

It now obey the rules of Tianjin that zipcode greater than 300000.

2. Data Overview

This section contains basic statistic about the dataset of Tianjin.

File size

tianjin_china.osm……56.4MB

tianjin_china.json……92.4MB

Number of documents: 315549

```
print db.tianjin.find().count()
```

Number of nodes: 280075

```
def query(a):
# using query to find numbers of node.
query = {"type" : a}
return query
```

```
def find(db, query):
    return db.tianjin.find(query)

node_query = query("node")
node_results = find(db, node_query)
print node_results.count()
```

Number of ways: 35474

```
way_query = query("way")
way_results = find(db, way_query)
print way_results.count()
```

Number of unique users: 248

```
print len(db.tianjin.distinct("created.user"))
```

Top 1 contributing user: [{u'count': 53126, u'_id': u'XBear'}]

```
top_user_pipeline = [{"$group":{"_id": "$created.user", "count":{"$sum":1 }}},

{ "$sort":{"count" : -1}}, {"$limit":1}]

top_user_result = aggregate(db, top_user_pipeline)

print top_user_result
```

Number of users appearing only once：[{u'num_users': 31, u'_id': 1}]

```
user_once_pipeline = [{"$group":{"_id": "$created.user", "count":{"$sum":1 }}},

{"$group":{"_id": "$count", "num_users":{"$sum":1 }}}, { "$sort":{"_id" : 1}}, {"$limit":1}]

user_once_result = aggregate(db, user_once_pipeline)

print user_once_result
```

3. Additional ideas

1st Contributor statistics and reward suggestion

The contributor statistics of users seems a little bit skewed, here are some user percentage statistics:

- Top user contribution percentage Xbear 16.84%
- Contribution of top 2 Xbear & Chen Jia 28.78%
- Contribution of top 10 -- 72.79%
- Contributors who make up less than 1% of posts 14 45.16%

Those information tell us the data is contributed mostly by top 10 people, 30% people do the 70% work. This distribution shows that others may do more work in editing instead of submitting data, since most of the work had been done by the top 10 people. In order to encourage more people do the work, a reward system demand to build.

2nd Additional data exploration using MongoDB queries

Top 10 amenities:

```
amenity_pipeline = [{"$match":{"amenity":{"$exists":1}}},
                    {"$group":{"_id":"$amenity","count":{"$sum":1}}},
                    {"$sort":{"count":-1}},{"$limit":10}]
pprint.pprint(aggregate(db, amenity_pipeline))
```

Here is the result:

[{u'count': 79, u'_id': u'parking'}, {u'count': 55, u'_id': u'school'}, {u'count': 32, u'_id': u'hospital'}, {u'count': 26, u'_id': u'university'}, {u'count': 23, u'_id': u'fuel'}, {u'count': 14, u'_id': u'bank'}, {u'count': 14, u'_id': u'restaurant'}, {u'count': 7, u'_id': u'kindergarten'}, {u'count': 5, u'_id': u'police'}, {u'count': 4, u'_id': u'library'}]

Most popular cuisines:

```
cuisine_pipeline = [{"$match":{"amenity":{"$exists":1}, "amenity":"restaurant"}},
                    {"$group":{"_id":"$cuisine","count":{"$sum":1}}},
                    {"$sort":{"count":1}},{"$limit":5}]
pprint.pprint(aggregate(db, cuisine_pipeline)
```

Here is the result:

{u'_id': u'Human_Food', u'count': 1},
{u'_id': u'chinese', u'count': 2},
{u'_id': None, u'count': 11}]

Except the unknown restaurant, the most popular cuisines is the Chinese Food.

3rd Additional methods can be used to improve the data

One of the additional methods could be a translator between the Python and MongoDB. Although MongoDB build on the coding format of UTF-8, support the most widely used language, and Python can use the #coding utf-8 in the script, but the bridge between each other, the cmd.exe, hard to implement the appropriate language format.

Besides, even when we successfully apply the language format, it is still hard to get what

the Chinese characters mean for most of global data user. If we can translate the Chinese characters into spelling name, people from the world can definitely understand what each place mean.

However huge benefit the translator would bring, it is still a big challenge to implement. We need a dictionary to collect all Chinese word, translate into spelling words, and finally build a package in python for future import. The translation work need to be finished in the OSM file, and then follow the process of we had done.

4. Conclusion

After the review, we can find the dataset for Tianjin is incomplete, but clean enough for this project after the use of data.py. The second problem solved with a replacement python file “edit_name.py” mentioned in the course, and the third use an “>300000” operator in “query.py” to avoid the unrelated data submitted in the database.

If we want to improve the performance more, besides the python file mentioned above, we at least need a translator to solve the first problem.