OpenStreetMap Project Data Wrangling with MongoDB

Map Area: Saint Petersburg, Russia; Sochi, Russia; Moscow, Russia (part)

At first, I picked Moscow as it's where I live. The full map for the city was enormous (100+ MB zipped) so I took only the city center. I tried to investigate the subway stations and the supplementary information about them on the map. I faced a lot of problems distinguishing what nodes on the map were metro stations as not everyone followed the convention of tagging them. Some of them were tagged as:

```
<tag k="railway" v="station"/>
<tag k="subway" v="yes"/>
```

Others were tagged as:

```
<tag k="station" v="subway"/>
```

… or:

```
<tag k="transport" v="subway"/>
```

The other problem was with the tag labels. Many of them had incompatible format. For example, some of the closed metro stations were tagged with the "was_railway" instead of "was:railway". Moreover, some of the tags contained serialized array values which were sometimes separated by a colon instead of semicolon.

Proceeding by trials and errors I extracted the data I wanted using a lot of nested "if's". But, unfortunately, I found nothing interesting in it. The result dataset contained only about a hundred entries.

So I took Saint Petersburg, second largest Russian city. I downloaded the map on mapzen.com, zipped file was about 62 MB, 800 MB unzipped. I wanted to get all the data about Saint Petersburg's park. I ran a script iterating through all the nodes or ways. Function that shaped elements only returned an output if the element had a tag `leisure : park` as it is a conventional way to mark parks in OSM.

```
for tag in element.iter('tag'):
    if tag.attrib['k'] == 'leisure' and tag.attrib['v'] == 'park':
        return { "id": element.attrib['id'],
                 "uid": element.attrib['uid'],
                 "tags": [tag.attrib for tag in element.iter('tag')] }
...
```

Unfortunately, I didn't manage to finish this script because it always got killed as the process ran out of memory after append approximately 1200 lines to the output file.

```
Jan 31 20:05:15 laptop kernel: [18201.406132] Out of memory: Kill
process 13015 (python) score 599 or sacrifice child

Jan 31 20:05:15 laptop kernel: [18201.406133] Killed process 13015
(python) total-vm:9838480kB, anon-rss:6523228kB, file-rss:252kB
...
```

I spent a lot of time profiling the script with a memory_profiler module and PyCharm's built-in profiler. I also read some articles on the topic and posts on StackOverflow. Most of them pointed to one article on IBM developersWorks. The article stated that for extra large files one must clear the element and all its previous siblings after being parsed. I tried to call `.clear()` method of the element but it had no effect. Actually the author of the article used `lxml` parser and its `iterparse` method which is an extension of the ElementTree API. The approach introduced in the article worked for the big dataset. In fact, I managed to parse the whole dataset on a dedicated server of the company where I work. But, unfortunately, I still can't understand why `cElementTree's iterparse` devours the RAM of my laptop completely.

I didn't want to extend the task further so I just picked a smaller dataset. I chose Sochi, a Russian resort city that hosted the latest Winter Olympic Games.

Overview of the data

Dataset

The dataset has 720,709 entries added by 390 users.

```
> db.nodes.find().count();
720709

> db.nodes.distinct('uid').length
390
```

There are 619,098 nodes and 101,611 ways in the dataset.

```
> db.nodes.aggregate([
        { $group: { _id: "$type",
                    count: { "$sum": 1 } } },
        { $sort: { count: -1 } }
```

```
    ])

{
    "result" : [
        {
            "_id" : "node",
            "count" : 619098
        },
        {
            "_id" : "way",
            "count" : 101611
        }
    ],
    "ok" : 1
}
```

Amenities

I took a small subset of nodes of the map which consisted of amenities only. According to OSM Wiki, “Amenity” tag is used “for describing useful and important facilities for visitors and residents.”

There are 1850 amenities in total.

```
> db.amenities.find().count()
1850
```

This is an example of the document stored in the amenities collections. I kept uid as a unique user identifier for further studies as well as all the tags.

```
> db.amenities.findOne();
{
    "_id" : ObjectId("56ae43d60147664e2ca37dc4"),
    "tags" : {
        "amenity" : "cinema",
        "name" : "К и н о т е а т р  \"К о м с о м о л е ц\""
    },
    "id" : "292203399",
    "uid" : "233125"
}
```

There are different types of amenities. 5 most popular amenities are: parking, cafe, pharmacy, restaurant, school.

```
> db.amenities.aggregate([
        { $group: { _id: "$tags.amenity",
                    count: { "$sum": 1 } } },
        { $sort: { count: -1 } },
        { $limit: 5 }
    ])
{
    "result" : [
        {
            "_id" : "parking",
            "count" : 281
        },
        {
            "_id" : "cafe",
            "count" : 223
        },
        {
            "_id" : "pharmacy",
            "count" : 109
        },
        {
            "_id" : "restaurant",
            "count" : 100
        },
        {
            "_id" : "school",
            "count" : 89
        }
    ],
    "ok" : 1
}
```

Let's take a closer look at parking. Many of the "parking amenities" have a tag "parking" which indicates the type of the parking. Most common parking is a surface parking. However, almost the same amount of "parking amenities" are not marked with a parking tag.

```
> db.amenities.aggregate([
        { $match: { "tags.amenity": "parking", }},
        { $group: { _id: "$tags.parking",
                    count: { "$sum": 1 } } }
    ])

{
```

```
      "result" : [
            {
                  "_id" : "surface",
                  "count" : 146
            },
            {
                  "_id" : null,
                  "count" : 125
            },
            {
                  "_id" : "multi-storey",
                  "count" : 8
            },
            {
                  "_id" : "underground",
                  "count" : 2
            }
      ],
      "ok" : 1
}
```


Contacts

For the purpose of the study I extracted all the contact data for all nodes.

Two most popular types were ‘phone’ and ‘website’. So first, I needed to extract that data, then clean it and save. Moreover, I want to preserve all other contact types like social media accounts. Not all ‘contact’ tags were marked conventionally. Some had ‘contact:’ prefix, others didn’t. So I used a regular expression to filter out the ‘contact’ tags. As a result, each node had the following fields:

- id - OSM id
- uid - user identifier
- type - node or way
- contacts - dictionary of lists of contact types
    - website (if exists) - list of websites
    - phone (if exists) - list of phone
    - …
- tags - dictionary of other tags except for the contact tags

I chose to make a list for each contact type as there might be more than one phone or website for a single node. First, some tag consisted of multiple values separated by a comma or a semicolon. Second, there were tags like ‘contact:website2’.

I had the following problems while cleaning data:

1. There were incorrectly formatted phone numbers

    a. Didn't start with +
    b. Started with 8 instead of 7
    c. Literal prefixes
2. Phone number format was not unified
    a. With parentheses
    b. With spaces
    c. With dashes
    d. Just digits
3. Websites were not normalized

After cleaning data, I ran a few queries against MongoDB.

```
> db.nodes.findOne({
        "contacts.website": { $exists: true },
        "contacts.phone": { $exists: true },
        "tags.amenity": { $exists: true }
    })

{
     "_id" : ObjectId("56ca463db85f834bd22df403"),
     "contacts" : {
          "website" : [
               "http://www.russianpost.ru"
          ],
          "phone" : [
               "+7 862 264-16-21"
          ]
     },
     "type" : "node",
     "id" : "643533178",
     "tags" : {
          "addr:housenumber" : "1",
          "amenity" : "post_office",
          "name" : "О т д е л е н и е  с в я з и  №354000",
          "addr:postcode" : "354000",
          "atm" : "yes",
          "opening_hours" : "Mo-Fr 08:00-22:00; Sa-Su 09:00-18:00",
          "operator" : "П о ч т а  Р о с с и и",
          "ref" : "354000",
          "addr:country" : "RU",
          "addr:street" : "у л и ц а  В о р о в с к о г о"
     },
     "uid" : "157864"
}
```

```
> db.nodes.find({
        "contacts.website": { $exists: true },
        "contacts.phone": { $exists: true },
        "tags.amenity": { $exists: true }
    }).count()

48
```

```
> db.nodes.find("this.contacts.phone && this.contacts.phone.length >
1").count()
9
```

Additional Ideas

Contributors

Let's take a look at combined contribution of the users. The most active users added 71% of the total nodes and ways. Combined Top 10 and Top 100 users contribution is 93% and 99% respectively.

```
> db.nodes.aggregate([
            { $group: { _id: "$uid",
                        count: { "$sum": 1 } } },
            { $sort: { count: -1 } },
            { $limit: 1 },
            { $group: { _id: "",
                        total: { "$sum": "$count" } } },
            { $project: { _id: 0, total: "$total", fraction: { $divide:
[ "$total", db.nodes.find().count() ] } } }
        ])
{
    "result" : [
        {
            "total" : 510371,
            "fraction" : 0.7081512788101717
        }
    ],
    "ok" : 1
}

> db.nodes.aggregate([
```

```
        { $group: { _id: "$uid",
                    count: { "$sum": 1 } } },
        { $sort: { count: -1 } },
        { $limit: 10 },
        { $group: { _id: "",
                    total: { "$sum": "$count" } } },
        { $project: { _id: 0, total: "$total", fraction: { $divide: [
"$total", db.nodes.find().count() ] } } }
    ])

{
    "result" : [
        {
            "total" : 669164,
            "fraction" : 0.9284801494084297
        }
    ],
    "ok" : 1
}

> db.nodes.aggregate([
        { $group: { _id: "$uid",
                    count: { "$sum": 1 } } },
        { $sort: { count: -1 } },
        { $limit: 100 },
        { $group: { _id: "",
                    total: { "$sum": "$count" } } },
        { $project: { _id: 0, total: "$total", fraction: { $divide: [
"$total", db.nodes.find().count() ] } } }
    ])

{
    "result" : [
        {
            "total" : 716726,
            "fraction" : 0.9944734976252552
        }
    ],
    "ok" : 1
}
```

Mischievous users

Some of the contact tags were formatted poorly and there were not many of them. For example, I met tags with the ‘contact:website2’ or ‘contact:phone2’ only a couple of times so

we actually can track the only user who did that and do something with him. Also, just a few people use a comma as a separator for the list values in the contact tag.

Further study

Further study might include more precise analysis of amenities in Sochi. For example, there are a lot of cafes and restaurants in the city. And most of them are named and there is a lot more information about them than there is on parking. For example, one can make use of `$near` operator to find the area with the greatest amount of cafes. Having the contact info, it's possible to filter out all amenities with at least one contact type available and find the amount of amenities without any contact information.