

Data Wrangling with MongoDB - OpenStreetMap project

Stefan Dulman

February 22, 2016

In this project I selected as a dataset the map of my home town Iasi in Romania. The data was already made available in compressed form on the OpenStreetMap website at this address:

https://s3.amazonaws.com/metro-extracts.mapzen.com/iasi_romania.osm.bz2

The uncompressed XML file for the city of Iasi is 61MB long. The dataset is unfortunately far from perfect, so, in this project, I will focus on the following items:

- check and correct street types (with python/xml)
- check and correct phone numbers (with python/xml)
- detect missing building names (or any type of identifiers) leading to an unnamed icon being displayed (with python/xml)
- detect and combine duplicate entries for museum names (overlapping on the map) (with python/mongodb)

1 Data exploration

As a first step, I explored the xml file to get a better understanding of it. This section used several small python scripts, all of which are placed under a common file: `project_exploration.py`. In the following I will just include the output of these scripts and explain the results.

I started by looking at various elements in the xml document. This is what the script revealed:

```
tags encountered in the xml file:
count: 1      tag name: bounds
count: 1      tag name: osm
count: 307    tag name: relation
count: 6108   tag name: member
count: 42164  tag name: way
count: 106011 tag name: tag
count: 274869 tag name: node
count: 346194 tag name: nd
```

As expected, 'node' and 'way' are both present and the numbers show that they include several 'tag' and 'nd'. Some of the other tags allowed by the OpenStreetMap rules are also present in smaller numbers (first four elements in the listing above).

As instructed by the lesson, I am to create database entries in MongoDB with certain predefined keys ('pos', 'type', 'id', 'created'). Some of these keys might be overlapped by the key values in the 'tag' element in xml. Let's check:

```
overlapping keys with default tags:
duplicates: 0   for key id
duplicates: 0   for key pos
duplicates: 0   for key created
duplicates: 21  for key type
```

Zooming in:

```
"node" elements with tags named "type": 19
"way"  elements with tags named "type": 2
```

Indeed, 21 elements will lead to a conflict due to the key 'type' being used in the xml tags. As a solution, we will rename the key 'type' in keys with 'tag.type', avoiding the overlapping.

I was curious if any more keys from tag elements will overlap each other and did a check:

```
duplicate tags appear in:
duplicates: 0   in node
duplicates: 0   in way
```

Although possible, in the current data set we do not have any more overlapping.

2 Fixing street names

In Romanian language, the street type is always the first word in the name. There are obviously quite a number of types available - let's check the xml file by printing the first word in the street names and counting how many times it occurs (all the code in this section is also to be found in the file `project_exploration.py`):

```
count: 1      name Br\u0103tulenii
count: 1      name Palat
count: 1      name Miroslova-Valea
count: 1      name Frumoasa
count: 1      name Cet\u0103\u021uia
count: 1      name General
count: 1      name DN28
count: 1      name Pictorului
count: 1      name Comuna
count: 1      name Aroneanu
count: 1      name Alee
count: 1      name Fundac
count: 1      name Ciric
```

count: 1	name George
count: 3	name Parcul
count: 4	name Fundacul
count: 6	name Trec\u0103toarea
count: 8	name Fund\u0103tura
count: 20	name Calea
count: 31	name Splai
count: 32	name Pia\u021a
count: 67	name Stradela
count: 171	name \u0218oseaua
count: 233	name Aleea
count: 349	name Bulevardul
count: 1920	name Strada

This list shows that a few names need to be corrected. In particular:

- some names completely miss a street designation (like in the entries 'George'). We have several choices here. I opted for adding the general 'street' designation ('Strada' in Romanian) before the name.
- minor variations in the same name are present. For example 'Alee' and 'Aleea' mean both 'alley', the first one is missing a post article. I will correct for this kind of error, getting all the names to look the same.
- some types (such as 'Fundac') miss the post article and are grammatically incorrect - I will correct for this (resulting in this case in 'Fundacul').
- I am actually amazed that only such a small variation is found in the names, was expecting much more variety taking into account the amount of unicode characters with diacritics used in my language (it is common practice to spell without diacritics - for example use the letter 'S' for 'Ș').

3 Correcting phone numbers

The last part of the file `project_exploration.py` contains the python code for checking and correcting the phone numbers in the xml file. An example of phone numbers encountered in the xml file:

count: 1	phone: +40-232-412121
count: 1	phone: +40 232 264266
count: 1	phone: +40232261846
count: 1	phone: 0747.499.406
count: 1	phone: +40 232-254 391
count: 1	phone: 0232/20.11.02, 0232/20.24.51
count: 1	phone: +40232 267719
count: 1	phone: 0232/275.979 (int. 145)
count: 1	phone: 0232/218.383
count: 1	phone: +40724-330818
count: 1	phone: +40 246 810 0232

```
count: 1      phone: +40 232 264271;+40 232 473020
count: 1      phone: 0232 246 160
```

There are hardly two entries that look the same. This is probably given by the fact that phone numbers evolved in the last twenty years from five digit phone numbers to ten digits in several stages. Also, for fixed land lines the first part of the number identifies the actual area in the country.

I will clean the phone number list in two steps. First, I will try to uniform the representation.

- Following python code, first I will remove the extension numbers (represented as '(int. xxx)'). This is just a choice, motivated by the fact that systems with an extension number usually have a start menu.
- Next, I will remove all the special characters, leaving only the numbers. My choice was to use python string operations as the re expressions are very tricky to debug later on. I spelled the specific characters one after another - a different way could have been to go through all the letters and remove the ones that were not a digit.
- Some institutions have more than one phone number listed, so I converted the string to an array (maximum two elements were observed in this dataset).
- As last step, I fixed the country code for all numbers, so all of them start with '+40'.

The second step is checking the validity of the numbers. The area code for Iasi is 232, so land line numbers always start with '+40232'. A local Romanian cell phone number always starts with '07' - leading to '+407' with country code. I will discard all numbers which do not follow this pattern as they are invalid. A preview of the results:

```
new phone: [None]      old phone: ['+40 332 441 419']
new phone: ['+40232266268'] old phone: ['+40 232 266268']
new phone: ['+40232246160'] old phone: ['0232 246 160']
new phone: ['+40232270914'] old phone: ['+40 232 270914']
new phone: ['+40749403820'] old phone: ['+40749403820']
new phone: ['+40788373798'] old phone: ['+40 788-373 798']
new phone: ['+40232264264', '+40755083635'] old phone: ['+40232-264 264', '+40755-083 635']
```

4 Missing building names

I am interested in detecting the places indicated as landmarks on the map which do not have any name information attached. Once detected, one may either choose to fill in the information or simply delete them (rather than showcasing an empty icon).

I will scan the file for all elements 'ways' which define buildings that fail to provide info to be used as a name and have additional tags. I will print the set of the additional tags that were encountered and their corresponding list of ids (see the file `project_missingnames.py`).

Below is a fragment of the listing output:

```

tags: [('amenity', 'school')]
ids: ['232257179', '239706046', '261825977']

tags: [('shop', 'bakery')]
ids: ['61421722', '206817772']

tags: [('amenity', 'place_of_worship'), ('religion', 'christian'), ('denomination', 'romanian_orthodox')]
ids: ['229889390']

tags: [('tourism', 'hostel')]
ids: ['213201639']

tags: [('amenity', 'library')]
ids: ['335340923']

tags: [('amenity', 'bar')]
ids: ['76832834', '165687894', '230288657', '285191983']

```

The code above returns a list of 93 items with issues. Once identified, these elements can be easily manually annotated to include also the names of the places (especially for representative landmarks as churches, hotels and libraries). 93 items is actually a pretty small number, taking into account Iasi is in top ten largest cities in the country, with roughly 400.000 inhabitants.

5 Handling duplicate museum names

I noticed that, at least for some landmarks (museums, churches, hospitals), data has also been entered automatically from several databases. The descriptions indicate a governmental institution database, a database of all museums, etc. The result is that for some institutions the name appears several times on top of the building. An example is shown in the image below.

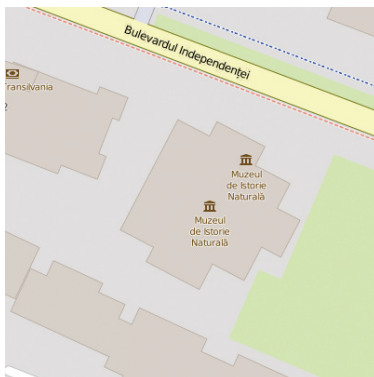


Figure 1: Example duplicate entries

First, I would like to detect these duplicates. For now, I will focus only on the museums, to keep things simple. I will convert the data to a json file as indicated in lesson 6 with the code in the file `project_data.py`. This code produces a large json file (70MB) which I load into a mongodb database with the command:

```
mongoimport -d osm -c test --file=iasi-romania.osm.json --host=127.0.0.1:27017
```

Then, I will launch a query which aggregates the following steps (file: `project_q-dupmuseums.py`):

- select all entries that are museums and have a name field
- group the entries by name and count occurrences
- sort in decreasing order
- hold only entries with two or more occurrences

The query looks like this:

```
# send a query to find the duplicate museum entries
cursor = db.test.aggregate([
    # the type of entry needs to be a museum and have a name
    {"$match": {'tourism': 'museum',
                'name': {'$exists': 'true'}}},
    # group the entries by name and count how many occurrences are
    {"$group": {"_id": "$name",
                "count": {"$sum": 1}}},
    # sort in a decreasing order
    {"$sort": {"count": -1}},
    # hold only the entries that are actual duplicates
    {"$match": {'count': {'$gte': 2}}}
])
```

Because I chose only the museums, there are only two results:

count: 2	name: Muzeul de Istorie Naturala
count: 2	name: Muzeul Unirii

Next, I will modify these entries. For each entry which occurs several times, there is a 'way' document describing the geometry of the building and several 'node' documents which add meta information on top. I chose to strip the 'node' documents of all info apart from the basic one ('id', 'type', 'created', 'pos', '_id') and move this info into the 'way' document. As I chose to do this operation on the documents in the mongodb database, overlapping may obviously occur. If done directly on the xml file, then all 'tag' entries would find themselves under the same 'way' document. The code achieving this is to be found in the second half of the file: `project_q-dupmuseums.py` (please note that the code to update the database is commented out).

As examples, the output for a 'node' document, before and after the modification, is:

```
—original node—
{u'LMI': u'IS-II-m-B-03917',
```

```

u'SIRUTA_code': u'95079',
u'_id': ObjectId('56c49d110f349ff2b11db4c6'),
u'address': {'u'city': u'Ia\u00219i', u'housenumber': u'16', u'postcode': u'700101',
             u'street': u'Bulevardul Independen\u0021bei'},
u'codul_entit\u00103\u0021bii_muzeale': u'7412100.0',
u'created': {'u'changeset': u'22634321', u'timestamp': u'2014-05-30T09:32:51Z',
             u'uid': u'2095993', u'user': u'Cosmo28', u'version': u'4'},
u'email': u'grigore@uaic.ro',
u'id': u'2634652849',
u'museum_type': u'Natural Sciences - Geology, Zoology',
u'name': u'Muzeul de Istorie Natural\u00103',
u'name_en': u'Natural History Museum',
u'opening_hours': u'Tu,Th,Sa 08:00-15:00; We,Fr,Su 09:00-16:00; Mo closed',
u'phone': u'0232/201.339',
u'pos': [47.166908, 27.58423],
u'source': u'INP',
u'source_link': u'http://date.gov.ro/dataset/ghidul-muzeelor-din-romania',
u'tourism': u'museum',
u'tourism_survey': u'10.07.2012 16:44:53',
u'type': u'node',
u'website': u'http://www.bio.uaic.ro/muzeu/muzeu.html'}

```

—new node—

```

{'u'_id': ObjectId('56c49d110f349ff2b11db4c6'),
 u'created': {'u'changeset': u'22634321', u'timestamp': u'2014-05-30T09:32:51Z',
              u'uid': u'2095993', u'user': u'Cosmo28', u'version': u'4'},
 u'id': u'2634652849',
 u'pos': [47.166908, 27.58423],
 u'type': u'node'}

```

6 Database statistics

In this section I will perform a few queries to identify the global characteristics of the data entered in the mongodb database. The code for this section is in the file `project_statistics.py`.

First, let's take a look at the number of documents in the database. The output of the first part of the code is:

```

number of documents in the database 317033
number of "node" documents 274869
number of "way" documents 42164

```

Next, we look at the statistics of users who contributed documents. The last piece of code in the file `project_statistics.py` does that: query the database for the number of documents each user contributed, transform the data into a numpy array and plot the histogram. The result is below:

In order to get better understanding of the distribution, the number of contributions were first transformed via `log10` function. Thus, '1' on the x axes means 10 contributions, while

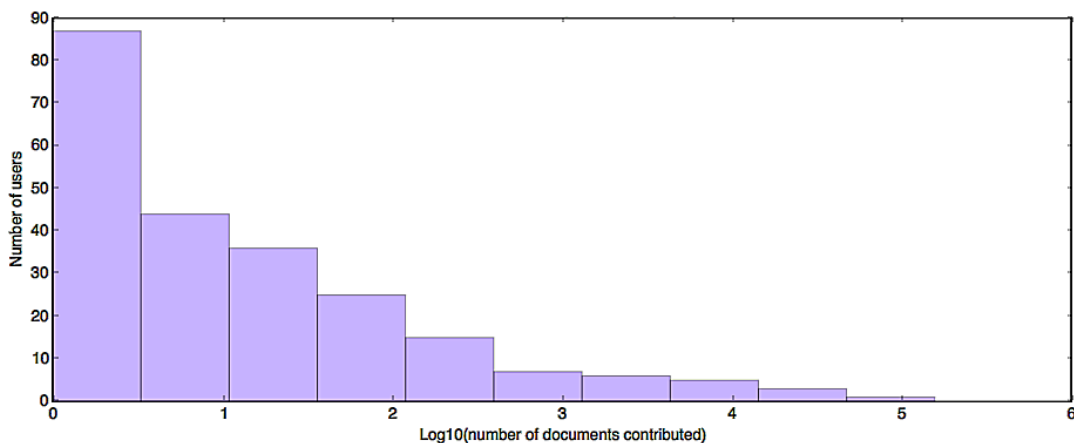


Figure 2: Histogram of users - document contributions

'6' means over 100000. As expected, the distribution is heavily skewed, with just a few users contributing the large majority of documents, while most of users contributed less than 10 documents.

The python file also outputs the top contributors, confirming our results. The first contributor added more than half of the documents while the first three contributors added more than two thirds of all the entries.

```
{u'count': 157527, u'_id': u'Laurentiu IS'}
```

```
{u'count': 41094, u'_id': u'sorein'}
```

```
{u'count': 27615, u'_id': u'LuluMOB'}
```

```
{u'count': 18737, u'_id': u'Romania CLC import bot'}
```

```
{u'count': 12740, u'_id': u"Cyrille '97"}
```

```
{u'count': 10858, u'_id': u'igerula'}
```

```
{u'count': 9836, u'_id': u'Cosmo28'}
```

```
{u'count': 5507, u'_id': u'Cristian Haulica'}
```

```
{u'count': 4442, u'_id': u'Cristian Draghici'}
```

```
{u'count': 4210, u'_id': u'razor74'}
```

7 Future work

Studying this dataset led to several questions. The first one refers on how to improve open-streetmap further? I can think of several ways of answering this one: refine the duplicate mechanism and automatically feed the corrected data back to the system, building a similar mechanism for detecting areas based on collections of buildings having the same name and adding those records, typechecking the spellings of street names and objects against a dictionary of Romanian names, etc. All these mechanisms add little information, they just clean up the erroneous data in the system.

The second question refers to how to add new data into the system automatically. The

easiest answer would be to look for public databases and import data from there. I guess the top contributors for this map already picked the low hanging fruits, sorting out the names of all streets, the shapes of the buildings, the names of public places, etc.

Similarly to their efforts I was thinking of updating the map with the names of the companies and their addresses. By looking at a familiar area of the town I noticed that many shops and businesses are not present in the dataset. Further investigating the problem led me to the Romanian Chamber of Commerce site, where I made an account. They allow me to query their database for the name of a company and its city (names of companies are enforced to be unique) and shows me the status of the company (failed, active, etc.) for free. Any other information costs money. I also noticed that there are other companies who have a similar business case as the Chamber of Commerce - they sell information about existing economic agents. <http://www.listaфирme.ro/> seems to be such a site, and they provide for free a preview of the companies in a certain city, with the name and address of company (see image below). An idea would be to scrap their website for this info, and check the company names against the Chamber of Commerce engine to figure out if the company is still active and then add it or remove it from the map. Both websites have a lot of small print copyright information so further investigation into the legal ownership of data is needed before writing any code.













CHV CONTAINER RO SRL Str. Ramei 9 A Jud. IASI, Loc. IASI	   		★★★★☆
HTR SPEDITION SRL Str. Dealul Bucium 26 Jud. IASI, Loc. IASI	   		★★★★☆
MIRAD TEHNIC SRL Str. Margareta Baciu 11 Jud. IASI, Loc. IASI	   		★★★★☆
FRAROM INTERNATIONAL EST SRL Sos. Bucium 93-95 Jud. IASI, Loc. IASI	   		★★★★☆
QUARTZ MATRIX SRL Bdul. Carol I 5 Jud. IASI, Loc. IASI	   		★★★★☆
EUROTECH SRL Str. Splai Bahlui-mal Stang 25 B Jud. IASI, Loc. IASI	   		★★★★☆

Figure 3: Example of active company listings

The last question is about building applications on top of this database. I noticed that for my home city, museums have a tag giving the open hours/dates. This data can be easily scrapped and used to suggest a visiting itinerary for tourists, especially if they have restrictions on the amount of time they spend in the city.