OpenStreetMap Sample Project

Data Wrangling using MongoDB

Area of interest aarhus – Denmark

1- Area of interest:

Aarhus is the second-largest city in Denmark and the seat of Aarhus municipality. It is located on the east coast of the Jutland peninsula, in the geographical centre of Denmark, 187 kilometres (116 mi) northwest of Copenhagen and 289 kilometres (180 mi) north of Hamburg, Germany. The inner urban area contains 264,716 inhabitants (as of 1 January 2016) and the municipal population is 330,639 (as of 2016). Aarhus is the central city in the East Jutland metropolitan area, which had a total population of 1.378 million in 2016

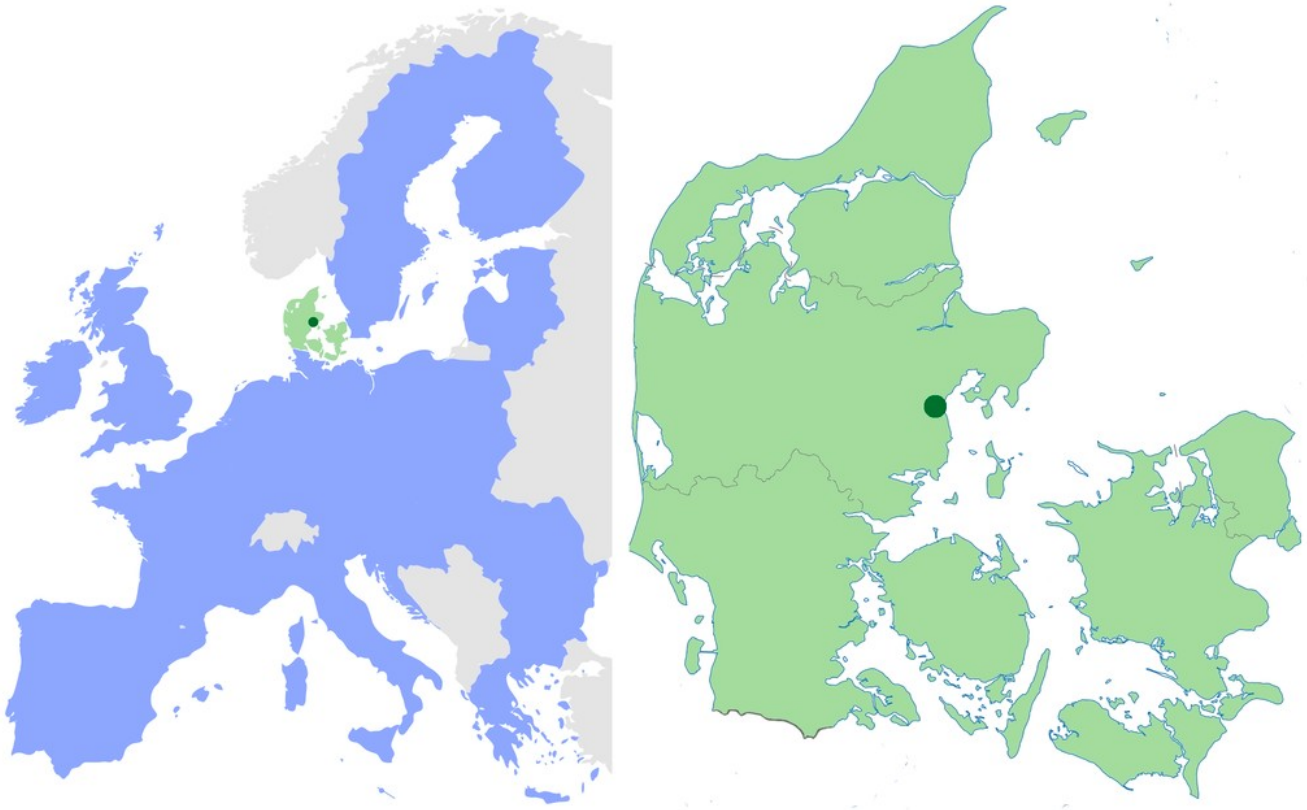

Illustration 1: Position of Aarhus in Denmark and in the E.U

I downloaded the OSM data of Aarhus from Map Zen https://mapzen.com/data/metro-extracts/metro/aarhus_denmark/ with a size of:

aarhus-denmark.osm ------- 135.6 MB

2- Problems encountered with the data:

Problematic characters:

the raw data needs to be extracted in the JSON format so that it can be later inserted in MongoDB so for that purpose the data will need to be cleaned from problematic characters like [=+/&<>;'"?%#$@\,\. \t\r\n] and since the name of the streets and nodes are in Danish I had to make sure note only the 26 alphabetical characters are taken in consideration but also the additional characters in the Danish Alphabet.

Another problem is that some tags included in address appear in this way : <tag k="addr:street" v="Bjørnholt"/> , this has been taken care of in the wrangling part to shape the elements in the JSON format.

Some tags have duplicate information in addr key and in osak which stands for *Officielle Standard Adresser og Koordinater* (Official Standard Addresses and Coordinates) in Denmark. So for this project I only consider data inside the addr tags

Cities Names :

* After loading the data in MongoDB which I will explain the procedure in the next part, I checked the name of the cities present in the data :

db.aarhus_nodes.distinct("address.city")
db.aarhus_ways.distinct("address.city")

Some names where duplicated since in the Danish Alphabet Aarhus is spelled "Århus" and there were also some “undefined” values which I will discard, for the duplicates in city names I will transform them into Latin Alphabet to unify them.

For the towns Viby and Tranbjerg, they both go by the name Tranbjerg J and Viby J, since they are the same in this region of Denmark I will take off the J at the end of them.

For Søften, it is a small town near Hinnerup only 3 km between them and shares the same postal code, so I will change it to Hinnerup.

These would be the cleaned names of the cities:

```
Cities={
        u"Århus C":"Aarhus C",
        u"Århus N":"Aarhus N",
        u"Århus V":"Aarhus V",
        "Tranbjerg J":"Tranbjerg",
        "Viby J":"Viby",
        u"Søften":"Hinnerup"
}
```

For the undefined values (actually there is only one in nodes) I will change it manually since the city does appear in the name of the node and there is no need to automate it.

Postal Codes:

In order to check the postal codes, I went to the Postal Code lookup site and got all the Zip codes of the Aarhus region, then checked them against the cleaned cities name in the address of each node and way before inserting them in the Database.

3- Wrangling Procedure:

The goal of this part is to explain what has been done to get the ready to insert data in mongoDB.

The input is the XML file from the OSM data of Aarhus the first outputs would be two JSON files: aarhus_nodes.json with the elements with the node tag are shaped in JSON format, and aarhus_ways.json which contains elements with way tag shaped into the same JSON format.

This first part of the data wrangling is done automatically in the file XMLtoJSON.py, here is an overview of what happens in it:

- the script opens the aarhus-denmark.osm file
- iterates over its element using xml.etree.cElementTree:
    - creates a dictionary for that element
    - checks if the element is a node or a way (by its first tag)
    - inserts the available properties as entries of the dictionary
        - loop through the addr: tags of the elements
        - filter the problematic characters from the address tags
        - group the elements of the address as a dictionary and add it as an entry to the element's dictionary
    - if the element is of type way group the references of its nodes “nd” in an array and add it as an entry to the element's dictionary
- writes the element in the JSON file for its type (nodes/ways)

the JSON output should be in this form :

```
{
"id": "2406124091",
"type: "node",
"visible":"true",
"created": {
        "version":"2",
        "changeset":"17206049",
        "timestamp":"2013-08-03T16:43:42Z",
        "user":"linuxUser16",
        "uid":"1219059"
      },
"pos": [41.9757030, -87.6921867],
```

"address": {
"housenumber": "5157",
"postcode": "60625",
"street": "North Lincoln Ave"
},
"amenity": "restaurant",
"cuisine": "mexican",
"name": "La Cabana De Don Luis",
"phone": "1 (773)-271-5176"
}

after generating the 2 json files, the script JSONtoMONGO.py inserts their content in the mongo Database.

4- Data Overview:

in this part I will present some statistics about the data, first the size of the files generated in the first part of the wrangling process:

aarhus_nodes.json → 119.9 MB
aarhus_ways.json → 18.4 MB

* Number of documents:

the Database named DAND3 contains 2 collections aarhus_nodes and aarhus_ways, here are the sizes of each:

db.aarhus_ways.find().count() → 55432
db.aarhus_nodes.find().count() → 441696

* Number of unique users :

for both Collections how many users there are :

db.aarhus_nodes.distinct(“created.user”) → 335
db.aarhus_ways.distinct(“created.user”) → 285

* Top contributing user:
db.aarhus_nodes.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}}, {"$sort": {"count":-1}}, {"$limit":1}]) → { "_id" : "AWSbot", "count" : 82996 }

I looked for it in the OSM wikis, this user is a bot used by the council of the city aarhus in Denmark to load data from different servers to OSM, it stands for Adresse Web Services

however for the ways another user takes the lead :

db.aarhus_ways.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":1}]) → { "_id" : "Flare", "count" : 11580 }

* Number of users appearing only once:
db.aarhus_nodes.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}}, {"$group":{"_id":"$count", "num_users":{"$sum":1}}}, {"$sort":{"_id":1}}, {"$limit":1}]) → 73

In this project I am more interested in nodes since they are the elements that have a geographic information

5 – Additional Ideas :

since the ways are represented as a group of nodes, and they have no geographic information, and as a GIS engineer I would like to consider extraction the position from the nodes and generate a geometry attribute for the ways.

This pipeline does the work for now, it uses $lookup to join the 2 collections on the nodes id and outputs the result in a new collection called nodes:

db.aarhus_ways.aggregate([{$lookup:{from: "aarhus_nodes",localField: "node_refs",foreignField: "id",as: "nodes"}},{"$project":{"id":1,"name":1,"highway":1,"nodes.pos":1}},{"$out":"nodes"}])

* Note that I ignored some of the data since here I'm only interested in the [id,name,highway] of the ways and the position from the nodes

There is another way to do it using a combination of $unwind , $lookup and $group stages of a pipeline:

first I unwind the ways collection on the node_refs table then I do a lookup to join it with the nodes collection and then group by the id of the way to get all nodes belonging to each way.

db.aarhus_ways.aggregate([{"$unwind":"$node_refs"},{$lookup:{from: "aarhus_nodes",localField: "node_refs",foreignField: "id",as: "nodes"}},{"$project":{"id":1,"name":1,"highway":1,"nodes.pos":1}},{"$group":{"_id":"$id"}}]).pretty()

Conclusion:

As I mentioned in the problems encountered part there were some duplicated information about the address, since Denmark used their own address system to feed the OSM data and left it there as additional tag values increasing the size of the files and taking unnecessary storage volume either in the servers of OSM or in the downloading mirrors. The data should be unified when the providers are few (using bots) and official, like OSAK.

In the last part of this project I extracted the position of the nodes composing the ways in order to get a geographic position of the ways, the best way to do that would be to create a geometry field and either

feed it by WKT (Well Known Text) geometry format, this allows to exchange geometry types easily through the Web and GIS softwares (Geographic Information System) in a string format.

It can be either be done by OSM (by programming it from their side) or by the (bulk) data providers who would be asked to provide this kind of geometry format with their data.