OPEN STREET MAPS DATA WRANGLING

Final project

6 MARCH 2015

ERNI DURDEVIC

Problems encountered on the Map

I had some problems in reading the XML to count the number of nodes because the xml.etree.ElementTree iterparse method was building the XML tree incrementally and filling out the RAM (8 GB) very quickly. Even by calling the node.clear() method after processing each node was too RAM demanding because the tree was continuing to have a lot of references to empty elements.

Finally, I followed the instructions on this document by clearing the root node after each processed node and the RAM usage dropped down to few MB. I had to apply this pattern to all Lesson 6 scripts in order to let them run on my data.

Analysing street denominations

In Italy, the first word represents the street denomination; therefore, I changed the regular expression to match the first word of the address: *r'^\S+\.?'*

After adding the most common street types to the "Expected" list; I run the "audit_street_type" program and extracted the more location-specific ones. I also found some typos and case problems and fixed them using a mapping and the update_name method.

I ended up with the following expected and mapped street types:

```
expected = ["Via", "Borgo", "Campo", "Piazza", "Viale", "Largo",
"Calle",
"Piazzale", "Piazzetta", "Corte", "Lungomare", "Strada",
"Galleria", "Riva", "Salita", "Isola", "Ponte", "Campiello",
"Vicolo", "Riviera", "Passaggio", "Salizada", "Corso", "Villaggio",
"Fondamenta", "Monte", "Quartiere"]

mapping = {"via": "Via",
"VIa": "Via",
"VIA": "Via",
"V.le": "Viale",
"calle": "Calle",
"P.zza": "Piazza",
"piazza": "Piazza",
"viale": "Viale",
"Lugomare": "Lungomare"}
```

Importing XML into MongoDB

I started by running the tag types program in order to find out the possibly problematic tags attributes to map into JSON

```
{'lower': 1880451, 'lower_colon': 65512, 'other': 4299,
'problemchars': 243}
```

After that, I used the "Prepare for Database" script in order to build the JSON to be imported and imported it into MongoDB.

Querying MongoDB

I started with some basic query to gather some information about the amenities, but even simple queries were very slow. I created an index to improve search performance.

Performance overview:

Query	Before index created	After index created
db.nodes.find({"amenity": "school"}).explain()	'cursor': '**BasicCursor**', 'millis': **4013**, 'n': 1221, 'nscannedObjects': **8277917**, ...	'cursor': '**BtreeCursor** amenity_1', 'millis': **8**, 'n': 1221, 'nscannedObjects': **1221**, ...

The performance boost was huge; the query execution dropped from 4000 msec to 8 msec.

Data Overview

Top contributing users

```
> top_ten_users =
     [{"$group": {"_id": "$created.user", "count": {"$sum": 1}}},
     {"$sort": {"count": -1}},
     {"$limit": 10}]
> pprint.pprint(db.nodes.aggregate(top_ten_users))

{u'ok': 1.0,
 u'result': [{u'_id': u'DarkSwan_Import', u'count': 3773021},
             {u'_id': u'Al3xius', u'count': 1200412},
             {u'_id': u'GatoSelvadego', u'count': 562277},
             {u'_id': u'bellazambo', u'count': 421011},
             {u'_id': u'Geograficamente', u'count': 334667},
             {u'_id': u'Tizianos', u'count': 267045},
             {u'_id': u'voschix', u'count': 182971},
             {u'_id': u'SldrHartman', u'count': 153304},
             {u'_id': u'remix_tj', u'count': 145047},
             {u'_id': u'Bigshot', u'count': 136554}]}
```

Total unique users

```
> print len(db.nodes.distinct("created.user"))
1740
```

Total elements count

```
> elements_count = [{"$group": {"_id": "$type", "count": {"$sum": 1}}},
                    {"$sort": {"count": -1}},
                    {"$limit": 3}]
> pprint.pprint(db.nodes.aggregate(elements_count))

{u'ok': 1.0,
 u'result': [{u'_id': u'node', u'count': 7154368},
             {u'_id': u'way', u'count': 1123314},
             {u'_id': u'water', u'count': 77}]}
```

Additional ideas and data exploration

Amenity types

I started by investigating the most numerous amenity types

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

{u'ok': 1.0,
 u'result': [{u'_id': u'parking', u'count': 4852},
             {u'_id': u'place_of_worship', u'count': 1499},
             {u'_id': u'school', u'count': 1221},
             {u'_id': u'restaurant', u'count': 1195},
             {u'_id': u'fuel', u'count': 748},
             {u'_id': u'bar', u'count': 743},
             {u'_id': u'bench', u'count': 556},
             {u'_id': u'bank', u'count': 466},
             {u'_id': u'cafe', u'count': 406},
             {u'_id': u'drinking_water', u'count': 316}]}
```

Parkings

Venice is car-free, only boats can enter the city. Parkings are still numerous in the surrounding area.

```
> print db.nodes.find({"amenity": "parking"}).count()
4852

> print db.nodes.find({"amenity": "parking", "fee": "yes"}).count()
70

> print db.nodes.find({"amenity": "parking", "fee": "no"}).count()
348
```

There are more free than toll parkings, but the majority are still unclassified. To improve parking fee classification a phone app could be built. The app would serve the user by finding the closest parking to his destination. If the parking the user reaches is unclassified, the app would ask the user to check-in into the parking and input the parking fee (if any) in order to calculate the amount to be paid on checkout. The gathered data could then be shared by the user in order to improve the map.

Closest amenities to the Venice Train station

What are the first amenities you find when you come to Venice? To answer this question I had to create a geospatial index and query for amenities nearby the Venice train station.

```
db.nodes.create_index([("pos", pymongo.GEOSPHERE)])
u'pos_2dsphere'
```

I used it to find out amenities nearby the central Venice train station:

```
> earth_radius_km = 6371
> distance = 0.2 #km
> radians = distance / earth_radius_km
> train_station_pos = [45.4412123, 12.3216705]
```

```
> amenities_near_train_station_qry = {"amenity": {"$exists": 1},
  "pos": {"$nearSphere": train_station_pos,
          "$maxDistance": radians}}

> print db.nodes.find(amenities_near_train_station_qry).count()
19

> nodes_nearby = db.nodes.find(amenities_near_train_station_qry)

> for n in nodes_nearby:
      pprint.pprint(n["amenity"])
'post_box'
'post_box'
'telephone'
'telephone'
'telephone'
'telephone'
'bar'
'ferry_terminal'
'restaurant'
'restaurant'
'restaurant'
'ferry_terminal'
'ferry_terminal'
'restaurant'
'bank'
'ferry_terminal'
'toilets'
'ferry_terminal'
'pharmacy'
```