Joel Smith
5/31/2015
Data Wrangling with MongoDB

OpenStreetMaps - Data Wrangling with MongoDB

1. **Problems Encountered in the Map**
    a. Parsing large XML files with ElementTree
        i. I initially had issues parsing the large OpenStreetMap XML files. ElementTree's iterparse method struggles with large XML files because it still builds a data structure while iterating through serial chunks of the XML. However, this data structure can be cleared by calling the clear method on a cached reference to the root element of the tree. With this adjustment, large files can be serially processed with no large memory overhead and I no longer had issues.
        ii. Reference - http://effbot.org/zone/element-iterparse.htm

        Incremental parse code example -

```
# get an iterable
context = ET.iterparse(fileName, events=("start", "end"))

# turn it into an iterator
context = iter(context)

# get the root element
event, root = context.next()

for event, elem in context:
        # Process element

        # Clear ElementTree
        root.clear()
```

    b. Auditing the available data
        i. I wasn't initially sure of the depth of the OpenStreetMap data. To get a better understanding though, I audited all the key/value pairs that were nested within a node/way XML element. With this I had a set containing a count of all the keys. This assisted me in finding areas to examine along with additional tags that could be further cleaned and decomposed like the "address" keys. For example,

there is a “contact” field that appears often with different data for email, phone, facebook, etc.

Some fields of interest (small portion)

{ 'shop': 164, maxspeed': 679, 'wikipedia': 136, 'area': 149, 'drive_through': 10, 'voltage': 120, 'office': 6, 'usage': 124, 'leisure': 451 'historic': 6, 'motor_vehicle': 18, 'tourism': 56, 'sidewalk': 650, 'bicycle': 678, 'parking': 41, 'power': 5184, 'access': 843, 'religion': 691, 'highway': 37237, 'barrier': 241, 'electrified': 187, 'water': 39, 'route': 33, 'railway': 676, 'bridge': 737, 'horse': 665, ‘service': 2458, 'source': 801, 'website': 110, 'lanes': 1530, 'phone': 86, 'landuse': 621, 'amenity': 2390, surface': 382, 'waterway': 200, 'cables': 19, 'minspeed': 51, 'cuisine': 153, 'oneway': 2891, 'building': 1700, 'natural': 204, 'wheelchair': 7, 'outdoor_seating': 13, junction': 12 }

c. Redundant/Ambiguous street names and cities
    i. After examining the data a bit in MongoDB, I found a few street and city names with spelling discrepancies. To handle this, I made changes to my auditing and JSON converting code. In my auditing code, I dynamically built a set of the different street types and cities. This set was then used to build a mapping of the correct strings for these values that could be applied during the JSON conversion. For identifying the street type, I used the regular expression from the exercises.

```
cityAudit = []

cityMappings = {
        "Birmingham, Alabama": "Birmingham",
        "Brimingham": "Birmingham"
}

streetTypeAudit = []

streetMappings = {
        "St": "Street",
        "St.": "Street",
        "Ave": "Avenue",
        "Rd.": "Road",
        "Blvd": "Boulevard",
        "Pkwy": "Parkway",
        "Rd": "Road",
        "Dr": "Drive"
}
```

```
def audit_node_data(node):
    for key in node:
        if key not in nodeKeyCount:
            nodeKeyCount[key] = 0
        nodeKeyCount[key] = nodeKeyCount[key] + 1

    audit_streetType(node)
    audit_city(node)

def audit_city(node):
    if "address" in node and "city" in node['address']:
        if node['address']['city'] not in cityAudit:
            cityAudit.append(node['address']['city'])

def audit_streetType(node):
    if "address" in node and "street" in node['address']:
        streetType = street_type_re.search(node['address']['street']).group()
        if streetType not in streetTypeAudit:
            streetTypeAudit.append(streetType)

def clean_data(node):
    if "address" in node:
        if "street" in node['address']:
            node['address']['street'] = clean_street(node['address']['street'])

        if "city" in node['address']:
            node['address']['city'] = clean_city(node['address']['city'])

    return node

def clean_street(street):
    streetType = street_type_re.search(street).group()

    if streetType in streetMappings:
        street = street.replace(streetType, streetMappings[streetType])

    return street

def clean_city(city):
    if city in cityMappings:
        city = cityMappings[city]

    return city
```

2. Data Overview

a. File sizes
    i. birmingham_alabama.osm - 90,884 kilobytes
    ii. birmingham_alabama_output.json - 101,716 kilobytes
b. Queries

Number of documents -

```
db.OSM.find().count()
```

445,279

Number of “nodes” -

```
db.OSM.find({"type":"node"}).count()
```

406,034

Number of “ways”

```
db.OSM.find({"type":"way"}).count()
```

39,229

Number of unique users -

```
len(db.OSM.distinct("created.user"))
```

343

Number of one time users -

```
numberOfOneTimeUsers = db.OSM.aggregate([
    { "$group" : { "_id" : "$created.user", "count" : { "$sum" : 1 } } },
    { "$match" : { "count" : 1 } }
])
numberOfOneTimeUsers = len(list(numberOfOneTimeUsers))
```

50

Top street names

```
topStreetNames = db.OSM.aggregate([
    { "$match" : { "address.street" : { "$exists" : True } } },
    { "$group" : { "_id" : "$address.street", "count" : { "$sum" : 1 } } },
    { "$sort" : { "count" : -1 } }, # Sort by count
    { "$limit" : 20 }
])
topStreetNames = list(topStreetNames)
```

[{u'_id': u'Myrtlewood Drive', u'count': 48},
{u'_id': u'Green Springs Highway', u'count': 20},
{u'_id': u'Fieldstown Road', u'count': 14},
{u'_id': u'Summit Boulevard', u'count': 12},
{u'_id': u'Healthy Way', u'count': 12},
{u'_id': u'Decatur Highway', u'count': 10},
{u'_id': u'Montgomery Highway', u'count': 10},
{u'_id': u'Scout Creek Drive', u'count': 7},

{u'_id': u'Cahaba Valley Road', u'count': 6},
{u'_id': u'Tom Williams Way', u'count': 5},
{u'_id': u'Center Point Parkway', u'count': 5},
{u'_id': u'Grandview Parkway', u'count': 4},
{u'_id': u'University Boulevard', u'count': 4},
{u'_id': u'Main Street', u'count': 4},
{u'_id': u'Alford Avenue', u'count': 4},
{u'_id': u'Pelham Parkway', u'count': 4},
{u'_id': u'Highland Avenue South', u'count': 4},
{u'_id': u'Mount Olive Road', u'count': 3},
{u'_id': u'Odum Road', u'count': 3},
{u'_id': u'Finley Boulevard', u'count': 3}]

Top city names

```
topCityNames= db.OSM.aggregate([
    { "$match" : { "address.city" : { "$exists" : True } } },
    { "$group" : { "_id" : "$address.city", "count" : { "$sum" : 1 } } },
    { "$sort" : { "count" : -1 } }, # Sort by count
    { "$limit" : 20 }
])
topStreetNames = list(topStreetNames)
```

[{u'_id': u'Birmingham', u'count': 65},
{u'_id': u'Gardendale', u'count': 28},
{u'_id': u'Vestavia Hills', u'count': 22},
{u'_id': u'Homewood', u'count': 8},
{u'_id': u'Irondale', u'count': 4},
{u'_id': u'Bessemer', u'count': 2},
{u'_id': u'Hoover', u'count': 2},
{u'_id': u'Mountain Brook', u'count': 1},
{u'_id': u'Trussville', u'count': 1},
{u'_id': u'Fairfield', u'count': 1},
{u'_id': u'Hueytown', u'count': 1},
{u'_id': u'Leeds', u'count': 1},
{u'_id': u'Odenville', u'count': 1},
{u'_id': u'Pelham', u'count': 1},
{u'_id': u'Indian Springs Village', u'count': 1},
{u'_id': u'Fultondale', u'count': 1},
{u'_id': u'Chelsea', u'count': 1}]

Top religions

```
topReligions = db.OSM.aggregate([
    { "$match" : { "religion" : { "$exists" : True } } },
    { "$group" : { "_id" : "$religion", "count" : { "$sum" : 1 } } },
```

```
        { "$sort" : { "count" : -1 } }, # Sort by count
        { "$limit" : 20 }
    ])
    topReligions = list(topReligions)
```

[{u'_id': u'christian', u'count': 687}, {u'_id': u'jewish', u'count': 4}]

Top amenities

```
    topAmenities= db.OSM.aggregate([
        { "$match" : { "amenity" : { "$exists" : True } } },
        { "$group" : { "_id" : "$amenity", "count" : { "$sum" : 1 } } },
        { "$sort" : { "count" : -1 } }, # Sort by count
        { "$limit" : 20 }
    ])
    topAmenities= list(topAmenities)
```

[{u'_id': u'place_of_worship', u'count': 707},
{u'_id': u'parking', u'count': 566},
{u'_id': u'school', u'count': 411},
{u'_id': u'grave_yard', u'count': 127},
{u'_id': u'fuel', u'count': 116},
{u'_id': u'restaurant', u'count': 104},
{u'_id': u'fast_food', u'count': 95},
{u'_id': u'hospital', u'count': 34},
{u'_id': u'townhall', u'count': 26},
{u'_id': u'fire_station', u'count': 24},
{u'_id': u'post_office', u'count': 23},
{u'_id': u'bank', u'count': 22},
{u'_id': u'library', u'count': 19},
{u'_id': u'public_building', u'count': 18},
{u'_id': u'pharmacy', u'count': 11},
{u'_id': u'toilets', u'count': 10},
{u'_id': u'fountain', u'count': 9},
{u'_id': u'swimming_pool', u'count': 6},
{u'_id': u'courthouse', u'count': 5},
{u'_id': u'bar', u'count': 4}]

Top shops

```
    topShops= db.OSM.aggregate([
        { "$match" : { "shop" : { "$exists" : True } } },
        { "$group" : { "_id" : "$shop", "count" : { "$sum" : 1 } } },
        { "$sort" : { "count" : -1 } }, # Sort by count
        { "$limit" : 20 }
    ])
    topShops= list(topShops)
```

[{u'_id': u'supermarket', u'count': 31},

{u'_id': u'clothes', u'count': 15},
{u'_id': u'car_repair', u'count': 12},
{u'_id': u'car', u'count': 12},
{u'_id': u'doityourself', u'count': 10},
{u'_id': u'mall', u'count': 10},
{u'_id': u'convenience', u'count': 7},
{u'_id': u'department_store', u'count': 6},
{u'_id': u'furniture', u'count': 5},
{u'_id': u'garden_centre', u'count': 5},
{u'_id': u'car_parts', u'count': 5},
{u'_id': u'yes', u'count': 5},
{u'_id': u'bicycle', u'count': 4},
{u'_id': u'newsagent', u'count': 3},
{u'_id': u'books', u'count': 3},
{u'_id': u'electronics', u'count': 3},
{u'_id': u'hardware', u'count': 3},
{u'_id': u'gas', u'count': 3},
{u'_id': u'hifi', u'count': 2},
{u'_id': u'pet', u'count': 2}]

3. Additional Ideas

a. Clean the data by comparing it to another data source
    i. Mapping APIs like Google maps would be great for cross validating and correcting the OpenStreetMap data. There would potentially be limitations with how many calls could be made to an API service. However, over time or just with certain problem areas an external data source would be great for improving the dataset.

b. Data visualizations
    i. All of the different shop, amenity, religion, and other values would make for great geographical visualizations showing the distribution of all these types of different places. These maps would be good for finding and identifying cultural and economic characteristics of an area. For example, you could examine the distribution of different types of churches and their religion or amenities by type using the latitude and longitude of the positional data.

c. Consumer applications
    i. Depending on the validity and fullness of the data, the OpenStreetMap dataset would be tremendously helpful in building all kinds of consumer applications if it could easily be queried. For example, it could be used to build applications that looked up destinations for a consumer depending on search metrics like

finding the nearest restauraunts, parking, historic site, etc. The GPS and address data could be used with a map/routing API like Google Maps to assist in traveling.

4. **References**
    a. **How to install and run MongoDB on Windows**
        i. **http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/**
        ii. **http://www.informit.com/articles/article.aspx?p=2246943**
    b. **How to process large XML files with incremental parsing and ElementTree**
        i. **http://effbot.org/zone/element-iterparse.htm**