Rishab Vaishya

Class - CSE 8392 Special Topics (Advanced Application Programming)

Quest 6 - Mongo DB -part 2

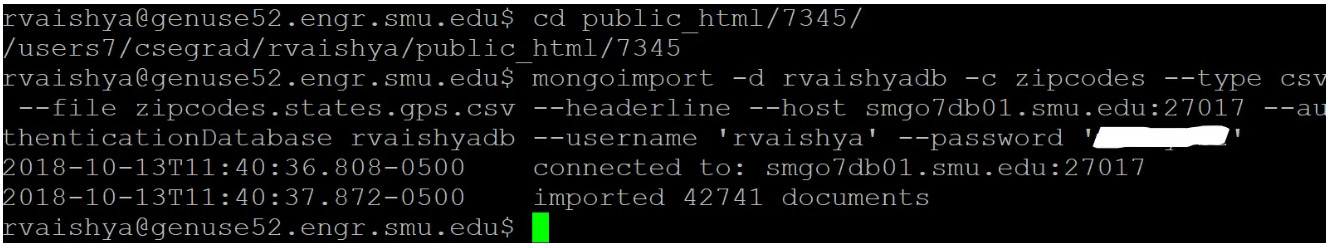
SMU ID – 47505527

Part A - Do Mongo Indexes Matter

In [28]:

```
import pymongo
from pymongo import MongoClient
```

Mongo-Import



```
rvaishya@genuse52.engr.smu.edu$ cd public_html/7345/
/users7/csegrad/rvaishya/public_html/7345
rvaishya@genuse52.engr.smu.edu$ mongoimport -d rvaishyadb -c zipcodes --type csv
 --file zipcodes.states.gps.csv --headerline --host smgo7db01.smu.edu:27017 --au
thenticationDatabase rvaishyadb --username 'rvaishya' --password '[redacted]'
2018-10-13T11:40:36.808-0500    connected to: smgo7db01.smu.edu:27017
2018-10-13T11:40:37.872-0500    imported 42741 documents
rvaishya@genuse52.engr.smu.edu$ 
```

connect to mongoCLient

In [29]:

```
#create the python client - replace with your userid:password below
client = MongoClient('mongodb://rvaishya:mc46yCs2@smgo7db01.smu.edu:27017/rvaishyadb')

# Choose the database to use as 'db'
db = client.rvaishyadb
```

drop index on column zip_code if already present

In [30]:

```
db.zipcodes.drop_index([("zip_code", pymongo.ASCENDING)])
```

Query without Index

In [31]:

```
query  = {'zip_code':10463}
projections = {"_id": 0,"city":1,"state":1}

print ("City & state for zip_code : 10463\n ")

# pretty print all the documents in the collection
cursor =  db.zipcodes.find(query,projections)
for p in cursor:
    print(p)

stats_dict1 = cursor.explain()["executionStats"] # returns a dict not JSON
```

```
print ("\nTotal Time in MilliSeconds (without index) : ", stats_dict1["executionTimeMillis"])
print ("Total Docs Examined (without index) : ", stats_dict1["totalDocsExamined"])
```

```
City & state for zip_code : 10463

{'city': 'Bronx', 'state': 'NY'}

Total Time in MilliSeconds (without index) :  39
Total Docs Examined (without index) :  42741
```

Create Index on column Zip_code

In [32]:

```
db.zipcodes.create_index([("zip_code", pymongo.ASCENDING)])
```

Out[32]:

```
'zip_code_1'
```

Query with index

In [33]:

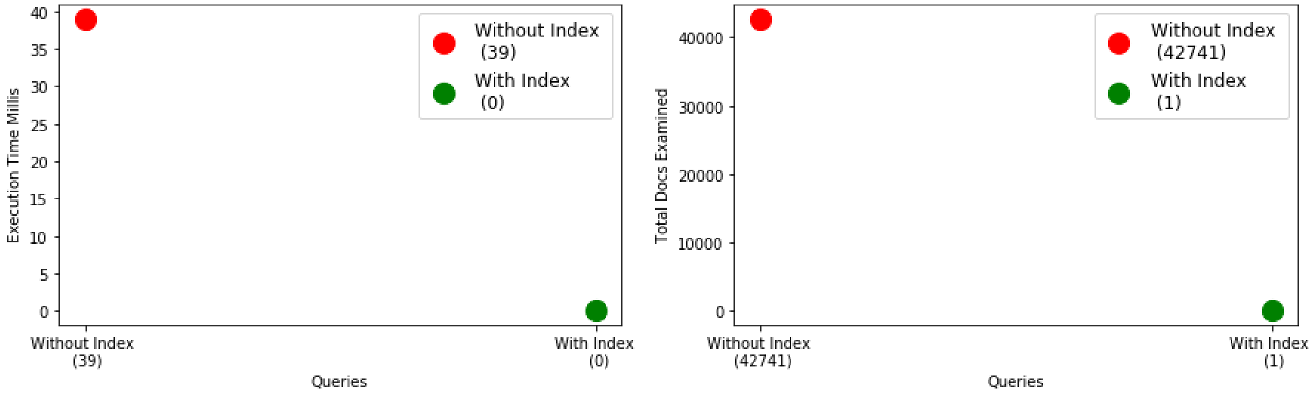
```
print ("City & state for zip_code : 10463\n ")

# pretty print all the documents in the collection
cursor =  db.zipcodes.find(query,projections)
for p in cursor:
    print(p)

stats_dict2 = cursor.explain()["executionStats"] # returns a dict not JSON
print ("\nTotal Time in MilliSeconds (with index) : ", stats_dict2["executionTimeMillis"])
print ("Total Docs Examined (with index) : ", stats_dict2["totalDocsExamined"])
```

```
City & state for zip_code : 10463

{'city': 'Bronx', 'state': 'NY'}

Total Time in MilliSeconds (with index) :  0
Total Docs Examined (with index) :  1
```

Compare the 2 queries using scatter plot

In [34]:

```
import matplotlib.pyplot as plt

Y1 = [stats_dict1["executionTimeMillis"], stats_dict2["executionTimeMillis"]]
Y2 = [stats_dict1["totalDocsExamined"], stats_dict2["totalDocsExamined"]]
labels = ['Without Index \n ({})', 'With Index \n ({})']

colors = [ 'red', 'green']

plt.rcParams["figure.figsize"] = [15,4]
plt.subplot(1,2,1)
for i in range(2):
    plt.scatter(labels[i].format(Y1[i]), Y1[i], c=colors[i], s=200, label=labels[i].format(Y1[i]))

plt.legend(prop={'size': 12},bbox_to_anchor=(1.0,1.0))
plt.ylabel(" Execution Time Millis")
plt.xlabel("Queries")

plt.subplot(1,2,2)
for i in range(2):
    plt.scatter(labels[i].format(Y2[i]), Y2[i], c=colors[i], s=200, label=labels[i].format(Y2[i]))

plt.legend(prop={'size': 12},bbox_to_anchor=(1.0,1.0))
plt.ylabel(" Total Docs Examined")
plt.xlabel("Queries")

plt.show()
```



we see that using index has huge benifits, we directly read the document which has that perticular latlong instead of going through entire list. time is nearly 0 milli-seconds & documents examined is 1

Part B - Find Location of Zip Codes

read file and store it in zip_code_List

In [35]:

```
file = open("zipcodes.txt",'r')
zip_code_List = file.readlines()[0].split(",")
projection={'_id':0,'city': 1, 'state': 1 }

print ('Zip_codes from text file = {}\n'.format(zip_code_List))
```

```
Zip_codes from text file = ['10463', ' 06520', ' 00603', ' 75225', ' 90210', ' 99999']
```

display result table

In [36]:

```
#print the column name
print ('{:<6}\t{:<17}\t{:<5}'.format('ZIP_CODE','CITY','STATE'))

for i in range(len(zip_code_List)):

    # the following line will cast string to int , stripping the leading zeros
    query = {'zip_code':int(zip_code_List[i])}

    # if valid zip_code
    if(db.zipcodes.find_one(query,projections)) :
        result = db.zipcodes.find_one(query,projections)
        # prints the result as a string without stripping zeros
        print ('{:<6}\t\t{:<17}\t{:<5}'.format(zip_code_List[i].strip(),result['city'], result['sta
te']))
    # zip_code not valid
    else :
        # prints the result as a string without stripping zeros
        print ("{:<6}\t\t<Not Valid Zip>".format(zip_code_List[i].strip()))
```

```
ZIP_CODE CITY                STATE
10463    Bronx              NY
06520    New Haven          CT
00603    Aguadilla          PR
75225    Dallas             TX
90210    Beverly Hills      CA
99999    <Not Valid Zip>
```

Part C -Find nearest zip code to mystery GPS coordinates

define the haversine function

In [37]:

```
from math import radians, cos, sin, asin, sqrt
def haversine(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    # Radius of earth in kilometers is 6371
    km = 6371* c
    return km
```

read file & display content

In [38]:

```
file = open("mysteryLatLong.txt",'r')
mystery_list = file.readlines()[0].split(",")
print ('Mystry Zip_codes from text file = {}\n'.format(mystery_list))
```

```
Mystry Zip_codes from text file = ['40.8276', ' -73.92614', ' 41.94887', ' -87.65778', '
32.7582813', ' -97.1105791']
```

calculate closest lat long for the mystry zip_codes & store it in 'distList' array

In [39]:

```
#setting distance to max for comparing later
distance = [float("inf"),float("inf"),float("inf")]

# stores closest lat-long data
dictList = [{},{},{}]

for p in db.zipcodes.find({}):
    lon = p["longitude"]
    lat = p["latitude"]

    #check if lon and lat are not null
    if(lat and lon):
        for i in range(3) :
            current_distance = haversine(float(mystery_list[i*2+1]),float(mystery_list[i*2]),lon,la
t)
            if (current_distance < distance[i]) :
                # stores closest lat-long in dictList
                dictList[i] = p

                # current distance becomes the closest distance
                distance[i] = current_distance
```

display result table

In [40]:

```
#print the column names
print ("{:<25}\t{:<10}\t{:<10}\t{:<10}".format('GPS-COORDINATES','CITY','STATE','ZIP_CODES'))
```

```
#print the column values
for i in range(3) :
        print ("{}, {:<10}\t\t{:<10}\t{:<10}\t{}"
               .format(mystery_list[i*2].strip(),mystery_list[i*2].strip(),
                       dictList[i]["city"],
                       dictList[i]["state"],
                       dictList[i]["zip_code"]))
```

```
GPS-COORDINATES           CITY       STATE      ZIP_CODES
40.8276, 40.8276     Bronx      NY         10451
41.94887, 41.94887    Chicago    IL         60659
32.7582813, 32.7582813  Arlington  TX         76012
```

What do these mystery zip codes have in common?

we can see that all three mystry lat-longs represent a US city, that's what is common between them