Diana Sujanto

Part II (40 points)

**YELP DATASET PROJECT**

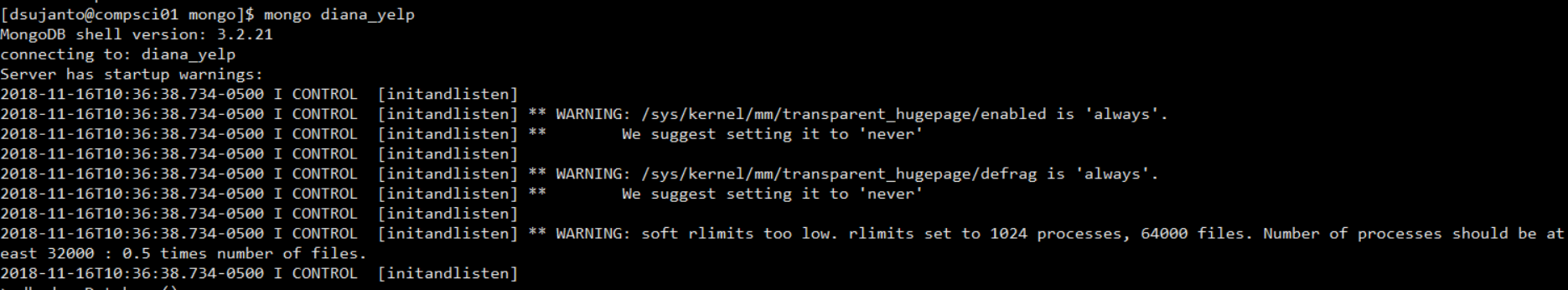
For the final project, I chose the Yelp Dataset because it looked interesting to me.

After I downloaded the Yelp Dataset Challenge data from the [website](http://www.yelp.com/dataset_challenge), a mongo database called diana\_yelp was created.

Since the purpose of this project was to help Yelp in adding a feature to its website to list the nearest business type from a given business of another type, I decided to use GeoSpatial Queries that are available for Mongo database.

**STEPS:**

**1. Create mongo db : mongo diana\_yelp**



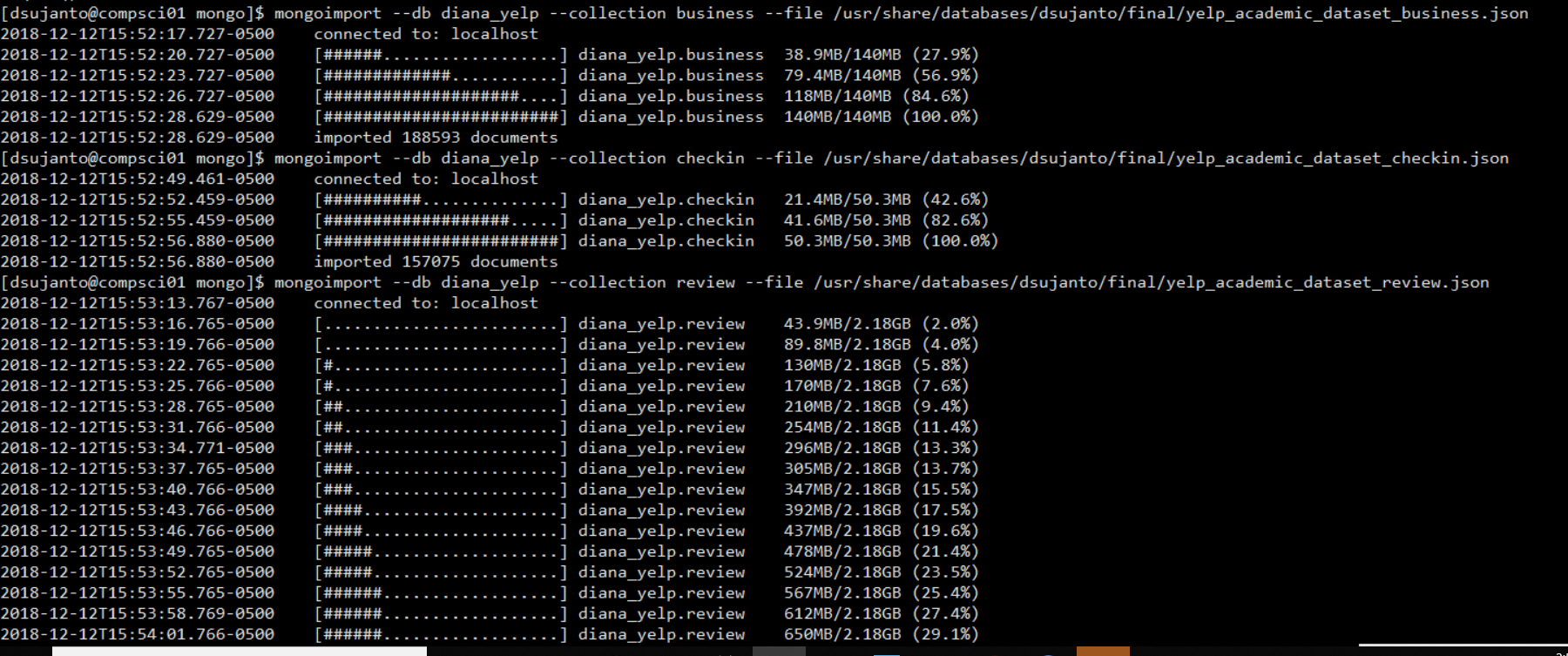
As you can see the mongo DB was created successfully. The next step would be to load .json files into the collections.

**2. Load business.json and checkin.json were loaded into the collections:**

mongoimport --db diana\_yelp --collection business --file /usr/share/databases/dsujanto/final/yelp\_academic\_dataset\_business.json

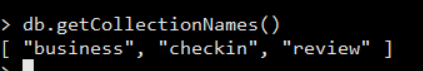
mongoimport --db diana\_yelp --collection checkin --file /usr/share/databases/dsujanto/final/yelp\_academic\_dataset\_checkin.json

mongoimport --db diana\_yelp --collection business --file /usr/share/databases/dsujanto/final/yelp\_academic\_dataset\_review.json



Three collections were created: business, review, and checkin. If I execute the following command, three collections will be displayed:

db.getCollectionNames()



After all collections have been populated, we needed to restructure the latitude and longitude documents to accommodate the usage of Mongo GeoSpatial Queries.

**3. Restructure the latitude and longitude documents by running the following function in order to utilize the 2d index for Mongo GeoSpatial Queries. The 2d index must be set on any two value fields in this order: {longitude: x, latitude: y}**

Here is the function:

db.business.find().forEach(

function (doc) {

doc.location = { latitude: doc.latitude, lon: doc.longitude };

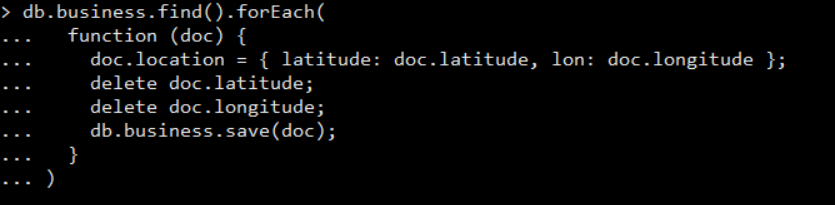
delete doc.latitude;

delete doc.longitude;

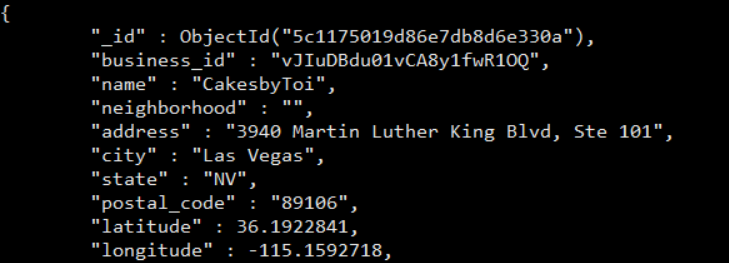
db.business.save(doc);

}

)

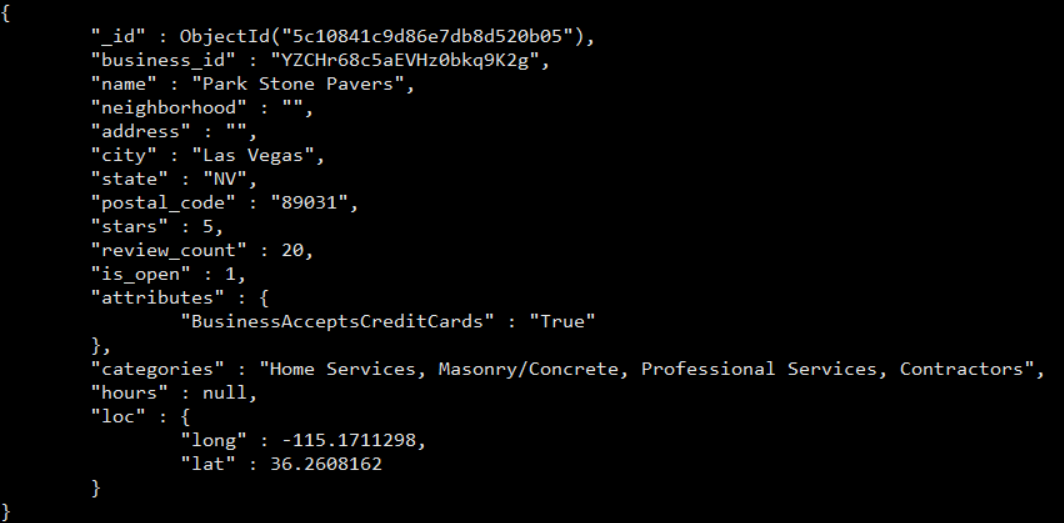


This was the original format of business collection where latitude and longitude were the documents:



Now, we have the following format, where latitude and longitude are parts of loc

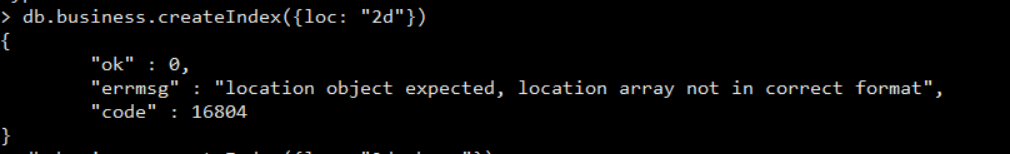
Document and fit to the criteria for 2d index: {longitude: x, latitude: y}



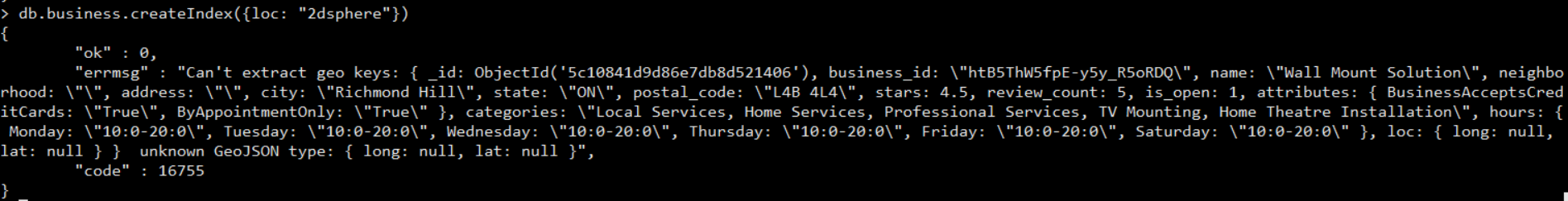
As you can see longitude(*lon*) and latitude(*lat*) are parts of “loc” document. We are ready to create 2d index.

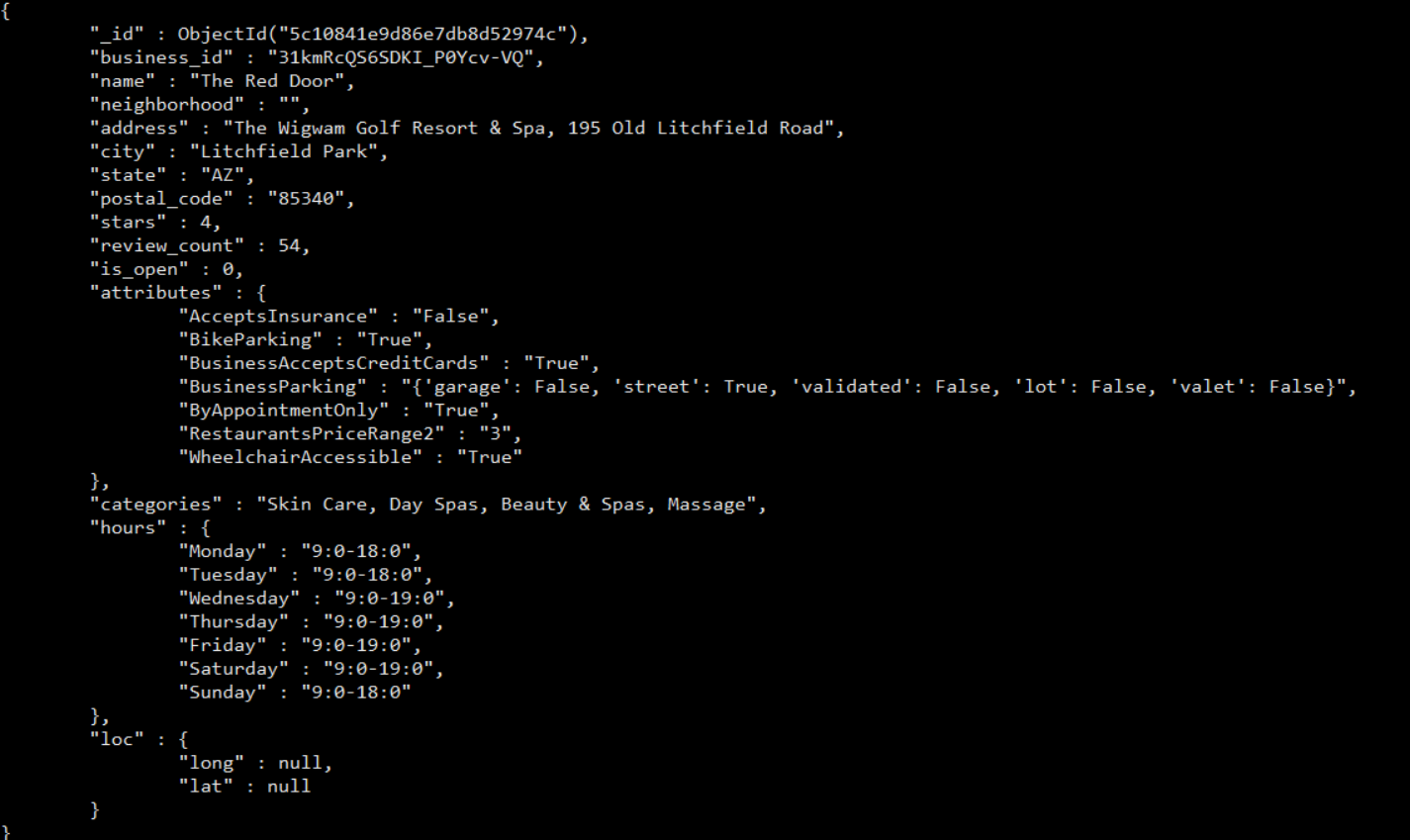
**4. Create 2d index for GeoSpatial Queries:**

db.business.createIndex({loc:”2d”})



I got an ERROR! It took a while for me to realize that there were some records that did have null latitude or longitude and were marked as null:



More null values: 

There were 6 records with null values on either latitude or longitude that needed to be fixed:



These procedures updated the null latitude/longitude to 0 (for data correction purposes):

db.business.find({"loc.lat": {$eq: null}}).forEach(function(doc){

db.business.update({\_id: doc.\_id},

{$set: {"loc.lon": 0, "loc.lat": 0}})

})

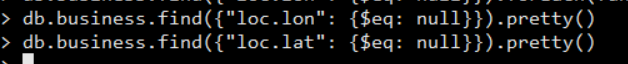
db.business.find({"loc.lon": {$eq: null}}).forEach(function(doc){

db.business.update({\_id: doc.\_id},

{$set: {"loc.lon": 0, "loc.lat": 0}})

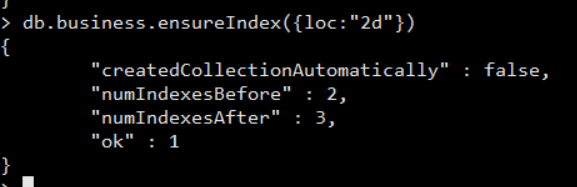
})

Double check to make sure there are no more null values:



Now, we are ready to create the index as there are no more null values for latitude or longitude. (The query returned “zero” results)

An attempt to create the “2d” index for the second time:



Now, it is successful!

**5. Start exploring with queries**

The following query returned too many results for the nearest businesses (188593 records)



By adding a distance, we can narrow down the results in close proximity:

db.business.aggregate([{$geoNear: {near:[-58.47, -34.50], distanceField: 'dist'}}, {$project:{\_id: 0, name:1, dist:1}}])

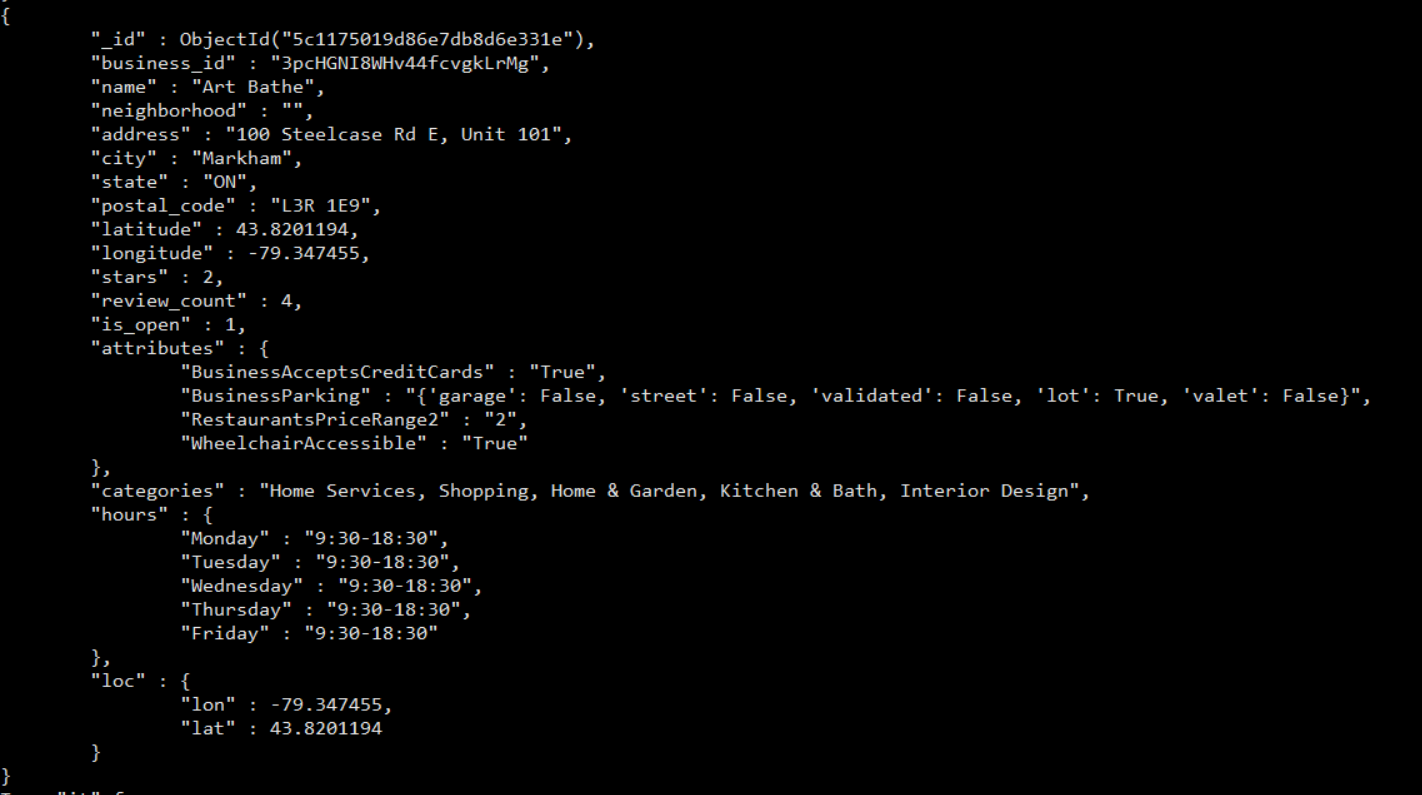
Another addition is to limit the maximum distance to 10 meters:

*Please note that this distance was purely used for illustration purposes, and would most likely not be applicable to a real world scenario. Obviously, a distance from one business to another would be much greater than 10 meters. Furthermore, you would change the “maxDistance” to a much more practical number in the query, such as 1000 meters.*

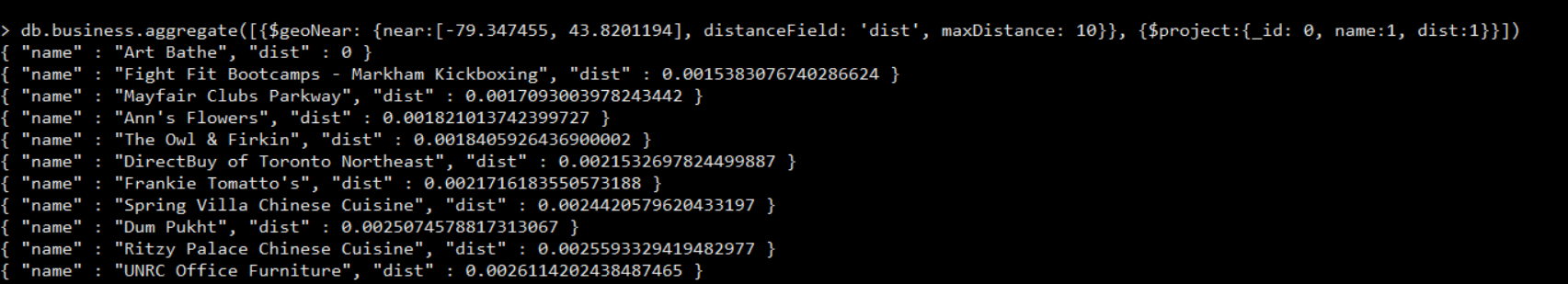
db.business.aggregate([{$geoNear: {near:[-79.347455, 43.8201194], distanceField: 'dist', maxDistance: 10}}, {$project:{\_id: 0, name:1, city:1, dist:1}}])

Let’s test the query for this business: “Art Bathe” located in Markham, ON, with the latitude and longitude coordinate as follows:

(-79.347455, 43.8201194)



These were the examples of the results returned by the query:



I checked “Fight Fit Bootcamps” on Google Maps to see if it was close in proximity to “Art Bathe”. Hence, it is very close to Art Bathe as shown on the map below:

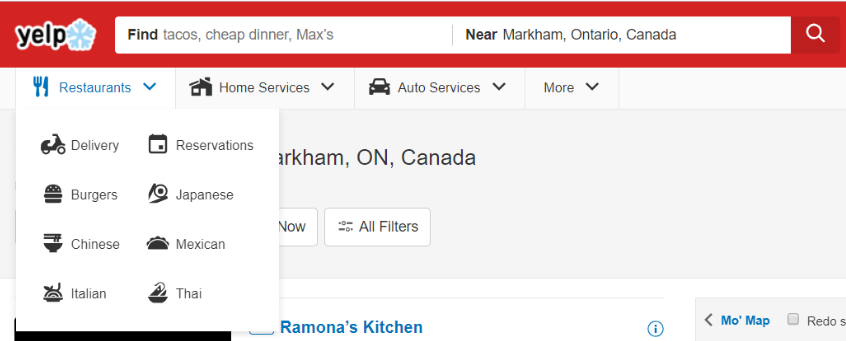


Other businesses that are within close proximity to Art Bathe and listed on Yelp are Frankie Tomatto’s and The Owl: A Firkin Pub. All of these businesses were shown in my initial query, thus proving that it was effective in locating nearby businesses.

**HOW DOES THIS SOLUTION HELP?**

The Geospatial query can be embedded to the code that serves the website by using different parameters, depending on categories, attributes (ie. Whether or not a restaurant offers delivery) or distance (in this case the distance is set to 10 m). For example, when a user is looking for a nearest restaurant that offers delivery after visiting Art Bathe, as he/she navigates on the “Restaurants” menu bar and click on the Nearest Restaurant option, the website will display a list of restaurants, located within 10 meters, all of which can deliver.

Below is the schema to show how the new addition can be implemented on Yelp’s website:



**Nearest Restaurants**

**List of Restaurants**

**that Offer Delivery**

**Imbiss Restaurant**

**Wing Wing Restaurant**

**Oriental City Restaurant**

**Ice Queen Restaurant**

**Pizzaville**

**And more …**

**WEB APPLICATION SERVER**



**FrontEnd:**

1. Adding an option for the user to choose the *nearest* business categories, in this case, it is the nearest restaurants that offer delivery.

2. Listing all restaurants that meet the above criteria.

**BackEnd:**

The application server runs the query to retrieve data from mongo DB.

db.business.aggregate([{$geoNear: {near:[-79.347455, 43.8201194], distanceField: 'dist', maxDistance: 10,

query: {"categories": "Restaurants", "attributes.RestaurantsDelivery": "True"}}}

{$project:{\_id: 0, name:1, city:1, dist:1}}])

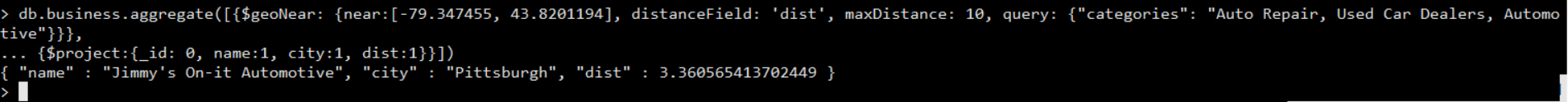
As a user click the Nearest Restaurants option, the application server will process the query and retrieve data from the database. Then, it will be displayed on the website for the end-user to view.

**Another Scenario:**

What if the user wants to select “Auto Services” that are close-by? Maybe, as the user finished shopping at Art Bathe, his/her car broke down. In this scenario, the query can be modified by changing the categories to “Auto Repair, Used Car Dealers, Automotive” as follows:

db.business.aggregate([{$geoNear: {near:[-79.347455, 43.8201194], distanceField: 'dist', maxDistance: 10, query: {"categories": "Auto Repair, Used Car Dealers, Automotive"}}},

{$project:{\_id: 0, name:1, city:1, dist:1}}])



The query returned “Jimmy’s On-it Automotive” with an approximate distance of 3.3 meters.

**CONCLUSION:**

GeoSpatial Queries are very effective when it comes to measuring the distance between two locations and providing the solution for ad-hoc queries. With this query embedded into the website’s code; Yelp can have this very useful feature, to find the closest business to another business (of a different type) that was chosen by a user. It may prove to be even more useful for the consumer when this is added to the Yelp app- ready to be used anytime, anywhere.