sm11326

Sai Nishanth Mettu

Homework 3 – Mongo DB

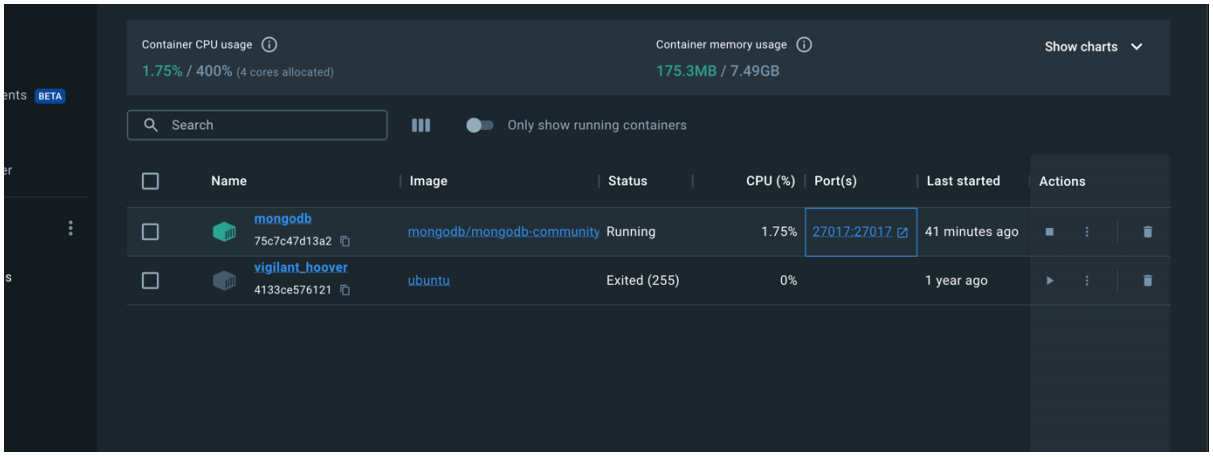
Pre-Requisite Mongo DB Setup

I am using a local instance of Mongo DB. Running on Docker.

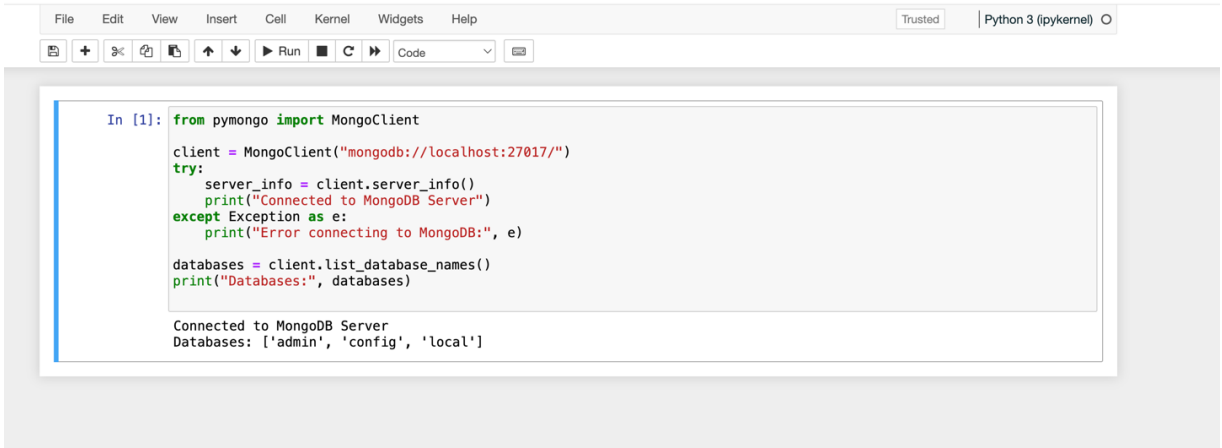
I have used the following commands to install MongoDB

https://www.mongodb.com/docs/manual/tutorial/install-mongodb-community-with-docker/

1. **docker pull mongodb/mongodb-community-server:latest**
2. **docker run --name mongodb -p 27017:27017 -d mongodb/mongodb-community-server:latest**



I have a Mongo DB Server running at port 27017:27017 (localhost)



Question 0) Load all the datasets into MongoDB

```
==================================================

Collection: durham_foreclosures
Number of Documents: 1948
{'_id': ObjectId('6749593179a7fce314ff3410'), 'datasetid': 'foreclosure-2006-2016', 'recordid': '629979c85b1cc68c1d
4ee8cc351050bfe3592c62', 'record_timestamp': '2017-03-06T12:41:48-05:00', 'parcel_number': '110138', 'geocode': [3
6.0013755, -78.8922549], 'address': '217 E CORPORATION ST', 'year': '2006', 'type': 'Point', 'coordinates': [-78.89
22549, 36.0013755]}

==================================================

Database: meteorites_db
Total Collections: 1
Collection: meteorites
Number of Documents: 1000
{'_id': ObjectId('6749593279a7fce314ff6e0a'), 'fall': 'Fell', 'id': '1', 'mass': '21', 'name': 'Aachen', 'nametyp
e': 'Valid', 'recclass': 'L5', 'reclat': '50.775000', 'reclong': '6.083330', 'year': '1880-01-01T00:00:00.000', 'ge
olocation.type': 'Point', 'geolocation.coordinates': [6.08333, 50.775], ':@computed_region_cbhk_fwbd': nan, ':@comp
uted_region_nnqa_25f4': nan}

==================================================

Database: restaurants_db
Total Collections: 1
Collection: restaurants
Number of Documents: 3772
{'_id': ObjectId('6749593179a7fce314ff1bb4'), 'address': {'building': '1007', 'coord': [-73.856077, 40.848447], 'st
reet': 'Morris Park Ave', 'zipcode': '10462'}, 'borough': 'Bronx', 'cuisine': 'Bakery', 'grades': [{'date': '2014-0
3-03T00:00:00', 'grade': 'A', 'score': 2}, {'date': '2013-09-11T00:00:00', 'grade': 'A', 'score': 6}, {'date': '201
3-01-24T00:00:00', 'grade': 'A', 'score': 10}, {'date': '2011-11-23T00:00:00', 'grade': 'A', 'score': 9}, {'date':
'2011-03-10T00:00:00', 'grade': 'B', 'score': 14}], 'name': 'Morris Park Bake Shop', 'restaurant_id': '30075445'}

==================================================

Database: worldcities_db
Total Collections: 1
Collection: cities
```

```
Database: durham_data
Total Collections: 2
Collection: durham_restaurants
Number of Documents: 2463
{'_id': ObjectId('6749593179a7fce314ff2a71'), 'ID': 56060, 'Premise_Name': 'WEST 94TH ST PUB', 'Premise_Address1':
'4711 HOPE VALLEY RD', 'Premise_Address2': 'SUITE 6C', 'Premise_City': 'DURHAM', 'Premise_State': 'NC', 'Premise_Zi
p': '27707', 'Premise_Phone': '(919) 403-0025', 'Hours_Of_Operation': None, 'Opening_Date': '1994-09-01', 'Closing_
Date': None, 'Seats': 60.0, 'Water': '5 - Municipal/Community', 'Sewage': '3 - Municipal/Community', 'Insp_Freq':
4, 'Est_Group_Desc': 'Full-Service Restaurant', 'Risk': 4, 'Smoking_Allowed': 'NO', 'Type_Description': '1 - Restau
rant', 'Rpt_Area_Desc': 'Food Service', 'Status': 'ACTIVE', 'Transitional_Type_Desc': 'FOOD', 'geolocation': '35.92
07272, -78.9573299'}

==================================================

Collection: durham_foreclosures
Number of Documents: 1948
{'_id': ObjectId('6749593179a7fce314ff3410'), 'datasetid': 'foreclosure-2006-2016', 'recordid': '629979c85b1cc68c1d
4ee8cc351050bfe3592c62', 'record_timestamp': '2017-03-06T12:41:48-05:00', 'parcel_number': '110138', 'geocode': [3
6.0013755, -78.8922549], 'address': '217 E CORPORATION ST', 'year': '2006', 'type': 'Point', 'coordinates': [-78.89
22549, 36.0013755]}
```

Question 2)

```
2de2fd3e3b1375388cd925', 'record_timestamp': '2017-03-06T12:41:48-05:00', 'parcel_number': '111399', 'geocode': [3
5.993026, -78.888343], 'address': '721 LIBERTY ST', 'year': '2006', 'type': 'Point', 'coordinates': [-78.888343, 3
5.993026]}
{'_id': ObjectId('6749593179a7fce314ff3415'), 'datasetid': 'foreclosure-2006-2016', 'recordid': 'ae17ea44c5918fd2d
5e54144cc84956c830e009', 'record_timestamp': '2017-03-06T12:41:48-05:00', 'parcel_number': '111426', 'geocode': [3
5.99217, -78.888092], 'address': '729 HOPKINS ST', 'year': '2006', 'type': 'Point', 'coordinates': [-78.888092, 35
99217]}
{'_id': ObjectId('6749593179a7fce314ff3416'), 'datasetid': 'foreclosure-2006-2016', 'recordid': '33bcfab5aa69ed55e
dc6839da2c8a5ee989e185', 'record_timestamp': '2017-03-06T12:41:48-05:00', 'parcel_number': '112166', 'geocode': [3
5.9865799, -78.886681], 'address': '1302 E MAIN ST', 'year': '2006', 'type': 'Point', 'coordinates': [-78.886681,
5.9865799]}
{'_id': ObjectId('6749593179a7fce314ff3418'), 'datasetid': 'foreclosure-2006-2016', 'recordid': 'f88a039e24c4182df
c6caad4199ab23c1bfd0af', 'record_timestamp': '2017-03-06T12:41:48-05:00', 'parcel_number': '112934', 'geocode': [3
5.992438, -78.874621], 'address': '1516 LATHROP ST', 'year': '2006', 'type': 'Point', 'coordinates': [-78.874621,
```

```
In [36]: print(f"Number of foreclosures within the polygon: {foreclosure_count}")

Number of foreclosures within the polygon: 738
```

In my approach, I started by querying the restaurants_collection to find restaurants that are categorized under "Food Service" and have at least 100 seats. I then extracted the geolocation data (latitude and longitude) for each restaurant, ensuring that the coordinates were correctly parsed into numeric values. These coordinates were stored in a list (restaurant_coordinates) that would be used later for the geospatial query. To enable efficient geospatial operations, I created a GEOSPHERE index on the coordinates field in the foreclosures_collection.

Next, I constructed a polygon_query that used the coordinates from the restaurants to define a polygon. This query was then executed using the $geoWithin operator to find foreclosures within the defined polygon area. Finally, I iterated through the matching foreclosures, counting how many were found and printing the details of each. At the end, I printed the total number of foreclosures that fell within the polygon, providing insight into how many foreclosures exist in the region where the restaurants are located.

Question 2) Homework 2 – Question 7 Version

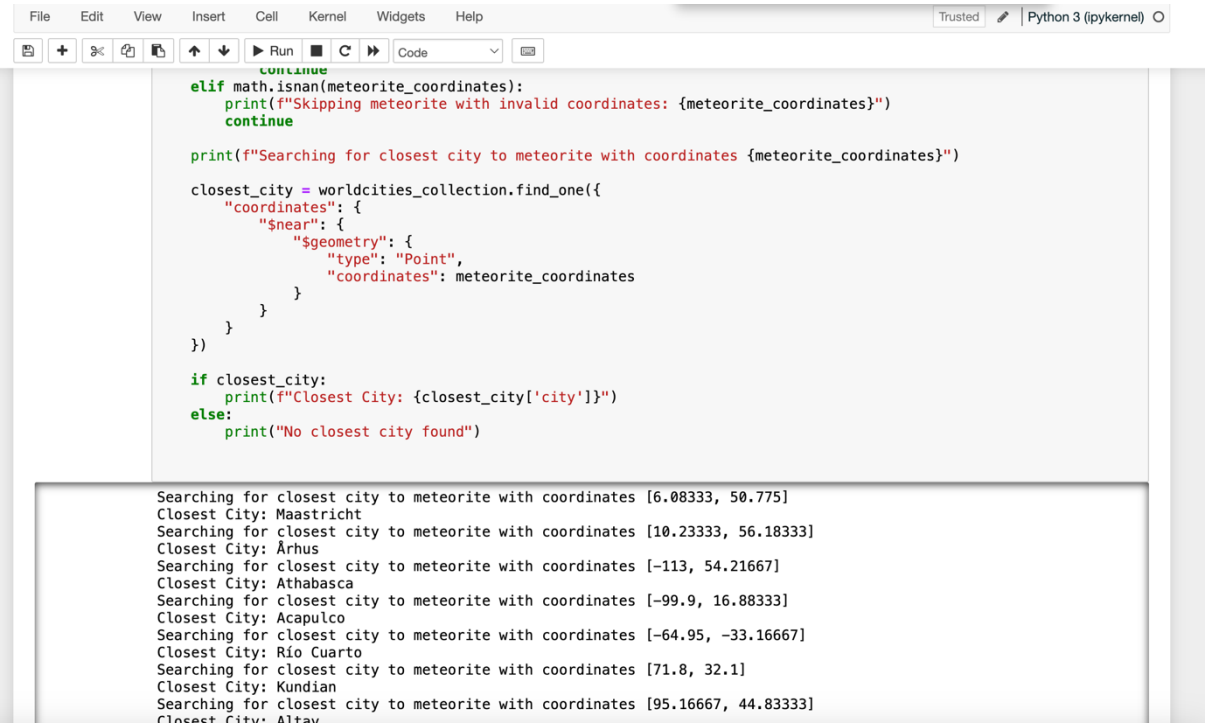
```
restaurants_within_circle = restaurants_collection.find({
    "Rpt_Area_Desc": "Food Service",
    "Status": "ACTIVE",
    "geolocation": {
        "$geoWithin": {
            "$polygon": circle_polygon
        }
    }
})

closest_restaurant = None
for restaurant in restaurants_within_circle:
    closest_restaurant = restaurant
    print("Closest Restaurant:", closest_restaurant)
    break

foreclosures_within_circle = foreclosures_collection.find({
    "coordinates": {
        "$geoWithin": {
            "$polygon": circle_polygon
        }
    }
})

foreclosure_count = 0
for foreclosure in foreclosures_within_circle:
    foreclosure_count += 1

print("Foreclosure Count within 1 mile:", foreclosure_count)
```

```
Foreclosure Count within 1 mile: 289
```

Approach

I first defined a target location (latitude and longitude) and a radius of 1 mile. To identify the area within this radius, I created a circle polygon using trigonometry. The circle was approximated by generating a polygon with 36 points around the target coordinates, taking into account the Earth's radius to convert miles into degrees of latitude and longitude. Using this polygon, I queried the restaurants_collection to find "Food Service" restaurants that are marked as "ACTIVE" and fall within the circle. I then retrieved the closest restaurant by iterating over the results. Similarly, I queried the foreclosures_collection to find foreclosures within the same circle and counted how many were within the area.

QUESTION 3) BONUS



Approach

I begin by iterating through all the cities in the worldcities_collection. For each city, I check if the latitude (lat) and longitude (lng) are available. If they are, I create a coordinates field, which stores the coordinates as an array in the format [longitude, latitude]. I then update each city document with this new field and proceed to create a 2dsphere index on the coordinates field to enable efficient geospatial queries.

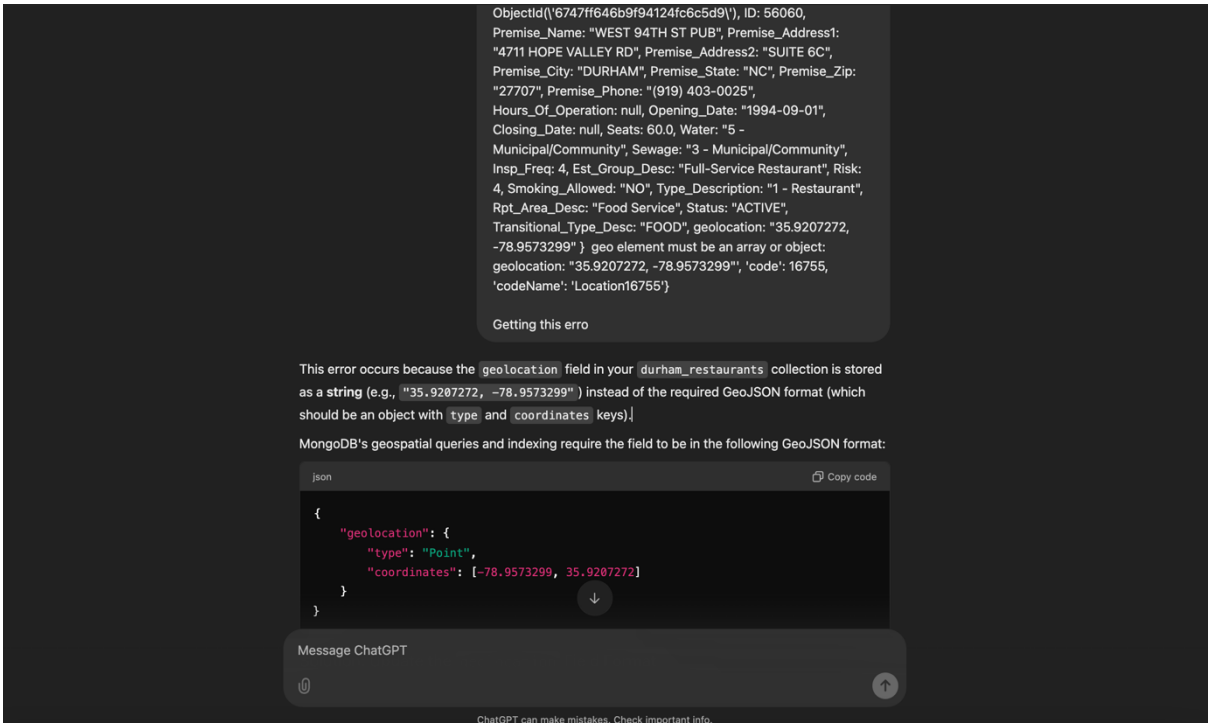
Next, I query for the closest city to a given meteorite's coordinates by using the $near operator with the 2dsphere index. This allows me to find the city closest to the meteorite's location. I print the closest city if found.

To ensure valid coordinates for meteorites, I extract the coordinates from each meteorite document and check whether they are valid by ensuring they aren't NaN. If any coordinate is invalid, I skip that meteorite. Otherwise, I proceed to search for the closest city to that meteorite's coordinates in the same way as before.

The core of the approach is using MongoDB's geospatial features to efficiently find the closest cities to meteorites based on their coordinates.

AI Acknowledgement

1- I have used AI – ChatGPT to understand the queries, and specially the GeoSpatial Queries in MongoDB for Question 2 & 3 as it was completely new for me.
2- I was getting a lot of errors, while constructing the polygon, while inserting the data, while restructuring the JSON and so on…hence I used GPT help here and there to fix errors and primarily help with GeoJSON, GeoDesic & GeoSphere questions.



3- The approach-text mentioned in this document, the text, I wrote my explanation and then I have AI Generated it for better language.

Thank You

Sai Nishanth Mettu.