THE UNIVERSITY OF SYDNEY

School of Computer Science
Dr. Ying Zhou

COMP5338: Advanced Data Models **Sem. 2/2020**

Week 9: Spatial Query

27.10.2020

In this tutorial, we will practice the spatial query feature of MongoDB to understand the following key concepts in spatial data model and query:

- spatial object representation
- common spatial operations
- the filter and refine stage in spatial query.

Spatial Sample Objects

The following section will use the sample spatial objects as shown in figure 1. It shows 3 points representing points of interest each, one triangle representing a park and one square representing a large industry complex in a two dimensional space. The default coordinate reference system uses WGS84 datum, which calculates geometry over an Earth-like sphere. Figure 1 shows longitude on X-axis and latitude on Y-axis.

Question 1: Representing Spatial Objects

MongoDB uses GeoJSON format to represent spatial objects. A document representing geometry object should have a type field specifying the GeoJSON object type and a coordinates field specifying the object's coordinates. With the default coordinate reference system, the coordinates should always be listed in longitude, latitude order. MongoDB supports GeoJSON object types such as `Point`, `LineString`, `Polygon` and more. The geometry shape in figure 1 are examples of `Point` and `Polygon` types.

a) Run the following commands to store the five locations in a collection called `places`.

```
db.places.insert({name: "I", placeType: "shop",
    loc: { type: "Point", coordinates: [ 0.02, 0.04 ]}})

db.places.insert({name:"ABCD", placeType: "industry",
    loc:{ type: "Polygon",
        coordinates:[[[0.03,0.05],[0.05,0.05],[0.05,0.03],
            [0.03,0.03],[0.03,0.05]]] }})
```

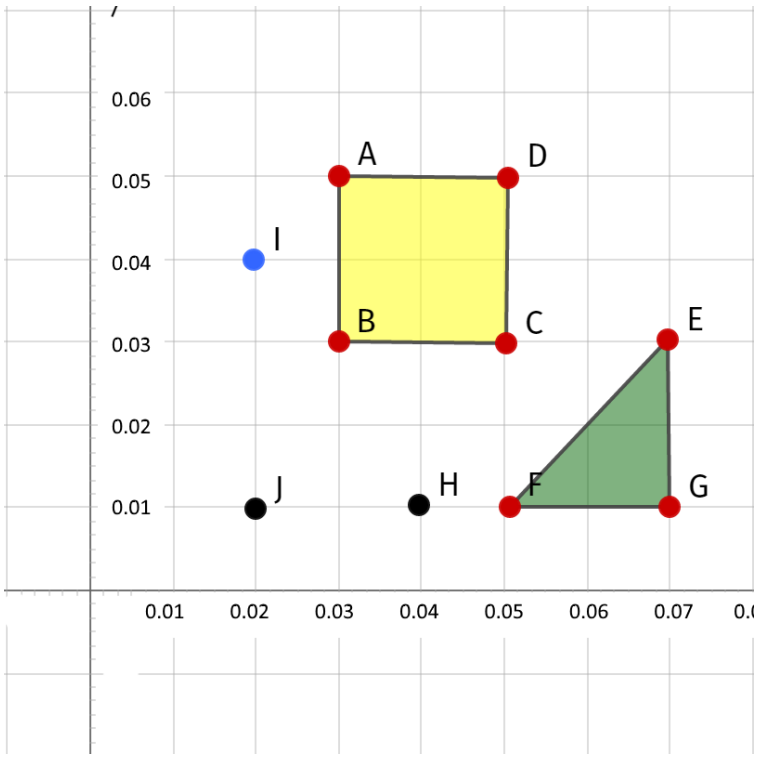

Figure 1: Example Spatial Data

```
db.places.insert({name: "J", placeType: "restaurant",
    loc: { type: "Point", coordinates: [ 0.02, 0.01 ]}})

db.places.insert({name: "H", placeType: "restaurant",
    loc: { type: "Point", coordinates: [ 0.04, 0.01 ]}})

db.places.insert({name: "EFG", placeType: "park",
    loc: { type: "Polygon",
        coordinates:[[[0.05,0.01],[0.07,0.03],
           [ 0.07, 0.01 ],[0.05,0.01]]]}})
```

b) Run the following command to set up geospatial index

```
db.places.createIndex({loc:"2dsphere"});
```

c) Run the following query to find the closet place to a location (0.02,0.02)

```
db.places.find({loc: {$near:{
    $geometry: {
        type: "Point",
        coordinates:[0.02,0.02]
    }
  }}}).limit(1)
```

This query demonstrates the basic use of the operator `$near`. The query point `(0.02,0.02)` is constructed using the `$geometry` operator followed by a GeoJSON object representing a point. The result is automatically sorted in ascending order of the distance. We use the `limit` qualifier to get the first result, which is the closest to the query point.

d) Run the following query to find any restaurant within 2km of location (0.02,0.025)

```
db.places.find({loc: {$near:{
    $geometry: {
        type: "Point",
        coordinates:[0.02,0.025]
    },
    $maxDistance: 2000
    }}, placeType :"restaurant"})
```

This query uses the operator `$near` as well. It specifies a maximum distance as well as a condition on the `placeType` field. The `$maxDistance` is measured in meter hence 2km is represented as 2000

e) Now write a query to find any point of interest within 2km of location (0.03, 0.03)

```
db.places.find({loc: {$near:{
    $geometry: { type: "Point", coordinates:[0.03,0.03] },
$maxDistance: 2000 }}})
```

Question 2: Spatial index usage

We will use the `explain` tool to inspect the use of spatial index

a) Run the following query to find all places contained inside a query polygon

```
db.places.find({ loc : {$geoWithin: {$geometry:
{type: "Polygon", coordinates:[[[0.01,0.01],[0.01,0.02],
[0.06,0.02],[0.06,0.01],[0.01,0.01]]]} }}})
```

This query demonstrates the use of the `$geoWithin` operator. It creates a geometry object with GeoJSON document and save it to a variable `shape`. This variable is referenced inside the `find` query as the query rectangle. The result should include the two points representing the two restaurants.

*Now use the **explain** command to see how many documents were returned initially by the index scan. If the index returns more documents than those in the results, which other documents are returned and why?*

b) MongoDB is able to handle shape with holes in side. Now run the following query to find the results; then inspect the index usage.

```
db.places.find({ loc : {$geoWithin:
    {$geometry: {
        type: "Polygon",
        coordinates:[
        [[0.01,0.005],[0.02,0.06],[0.06,0.005],[0.01,0.005]],
        [[0.035, 0.006],[0.035, 0.015],[0.045,0.015],[0.045,0.006],[0.035,0.006]]]
        }}}})
```

What does the query shape look like? How many documents are returned in the filter stage and how many are returned as the result

c) Write your own query to find all places intersect with a polygon represented by the following coordinates: [[[0.01,0.01],[0.01,0.02],[0.06,0.02],[0.06,0.01],[0.01,0.01]]]. Use the `explain` tool to see how index is used.

d) The `$near` query also uses spatial index, in slightly different way. The execution iteratively expands its search in distance intervals with respect to the query point. It checks index cells in that interval to find documents satisfy the distance requirements. If the the query point is close to any point in the dataset, the query can find the result in a relatively small interval and does not need to inspect a lot of index cells. Otherwise, it needs to expand to large interval and inspect many index cells. This feature can be examined by inspecting the index usage of different query point in the question `1.c`. For query point `[`0.02,0.02], 38 keys are examined. For query point [0.035,0.01], only 16 keys are examined. This is because [0.035,0.01] is very close to point `H` in the data set.