Program 3

a. Execute query selectors (comparison selectors, logical selectors) and list out the results on any collection

b. Execute query selectors (Geospatial selectors, Bitwise selectors) and list out the results on any collection

Comparison Selectors: Comparison selectors are used to compare fields against specific values or other fields. Here are some common comparison selectors:

$eq - Matches values that are equal to a specified value.

$ne - Matches all values that are not equal to a specified value.

$gt - Matches values that are greater than a specified value.

$gte - Matches values that are greater than or equal to a specified value.

$lt - Matches values that are less than a specified value.

$lte - Matches values that are less than or equal to a specified value.

$in - Matches any of the values specified in an array.

$nin - Matches none of the values specified in an array.

Logical Selectors: Logical selectors are used to combine multiple conditions in a query. Here are some common logical selectors:

$and - Joins query clauses with a logical AND and requires that all conditions be true.

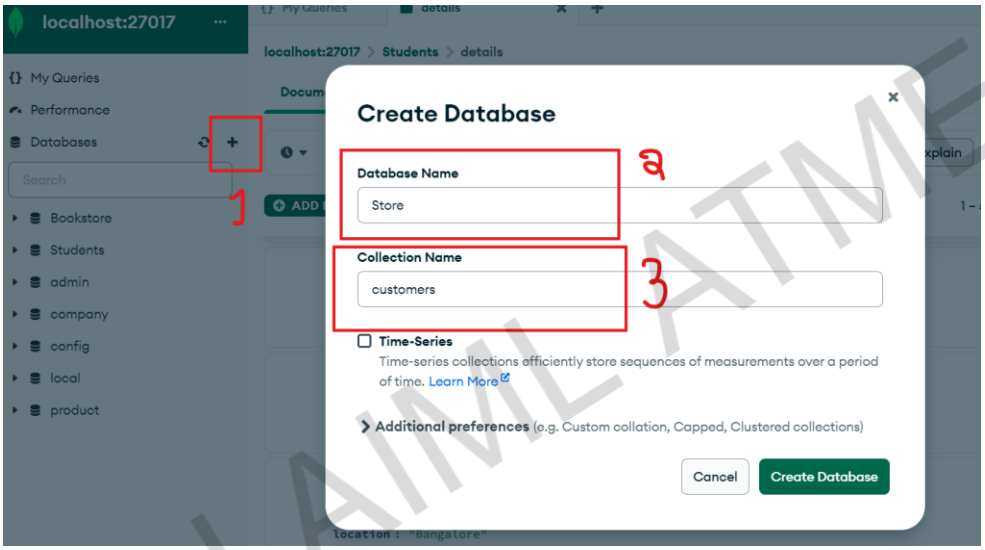
SN AIML ATME

$or - Joins query clauses with a logical OR and requires that at least one condition be true.

$not - Inverts the effect of a query expression and returns documents that do not match the query expression.

$nor - Joins query clauses with a logical NOR and requires that none of the conditions be true.

Create a database **Store** and collection **customers** in Mongo DB IDE**.**



In MongoDB Shell:

>use Store

> db.customers.insertMany([{ _id: 1, name: "Alice", age: 30, city: "New York" },

{ _id: 2, name: "Bob", age: 25, city: "San Francisco" },

{ _id: 3, name: "Charlie", age: 35, city: "Los Angeles" },

{ _id: 4, name: "David", age: 28, city: "Chicago" },

{_id: 5, name: "Eve", age: 32, city: "Miami" }])

SN AIML ATME

a. **Execute query selectors (comparison selectors, logical selectors) and list out the results on any collection.**

Using Comparison Selectors

1. **Find customers aged 28:**

   >db.customers.find({ "age": { "$eq": 28 } })

   Output:

```
< {
    _id: 4,
    name: 'David',
    age: 28,
    city: 'Chicago'
  }
```

2. **Find customers older than 30:**

   >db.customers.find({ "age": { "$gt": 30 } })

   Output:

```
< {
    _id: 3,
    name: 'Charlie',
    age: 35,
    city: 'Los Angeles'
  }
  {
    _id: 5,
    name: 'Eve',
    age: 32,
    city: 'Miami'
  }
```

SN AIML ATME

Using Logical Selectors

3. Find customers in city is New York OR city is Los Angeles:

```
>db.customers.find({
  $or: [
    { city: "New York" },
    { city: " Los Angeles"  }
  ] })
```

Output:

```
< {
    _id: 1,
    name: 'Alice',
    age: 30,
    city: 'New York'
  }
  {
    _id: 3,
    name: 'Charlie',
    age: 35,
    city: 'Los Angeles'
  }
```

4. Find customers age 30 and city New York

```
>db.customers.find({
  $and: [
    { age: 30 },
    { city:"New York"  }
  ] })
```

SNAIML ATME

Output:

```
{
  _id: 1,
  name: 'Alice',
  age: 30,
  city: 'New York'
}
```

Using Both Comparison and Logical Selectors

5. Find customers greater than or equal to 18, less than 35, in city New York or Miami

```
>db.customers.find({$and: [
    { age: { $gte: 18 } },        // age greater than or equal to 18
    { age: { $lt: 35 } },         // age less than 35
    { city: { $in: ["New York", "Miami"] } }  // city is either "New York" or "Miami"
]})
```

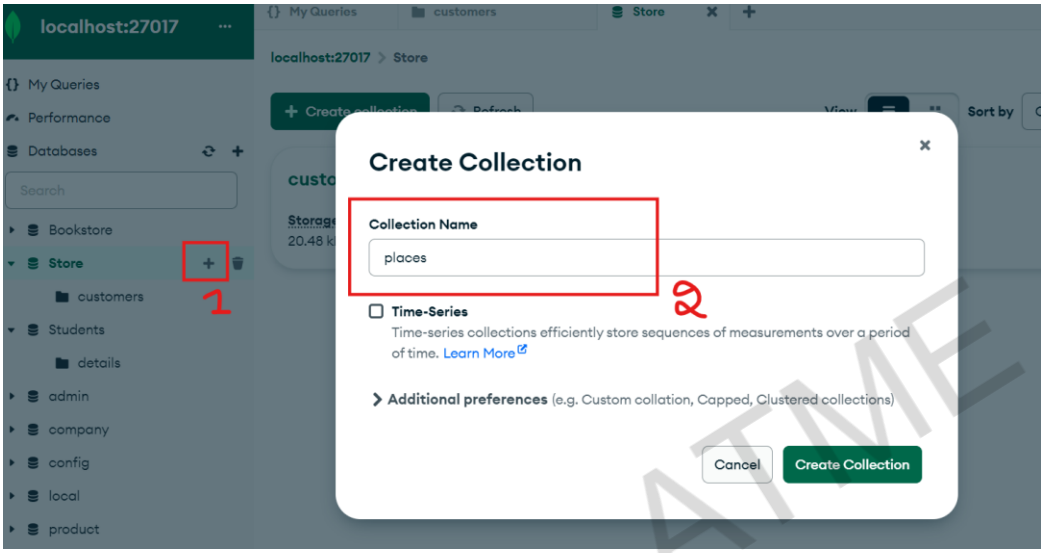
Output:

```
{
  _id: 1,
  name: 'Alice',
  age: 30,
  city: 'New York'
}
{
  _id: 5,
  name: 'Eve',
  age: 32,
  city: 'Miami'
}
```

SN AIML ATME

b. Execute query selectors (Geospatial selectors, Bitwise selectors) and list out the results on any collection

Under database **Store**, create a collection **places** in Mongo DB IDE**.**



Geospatial Selectors: MongoDB supports geospatial queries for geospatial data. It provides two types of geospatial indexes: 2d indexes and 2d sphere indexes.

Add the following documents in the **places collection** in MongoDB Shell.

>db.places.insertMany([

{ id: 1, name: "Place A", location: { type: "Point", coordinates: [-73.97, 40.77] } }, // New York

{ id: 2, name: "Place B", location: { type: "Point", coordinates: [-122.43, 37.77] } }, // San Francisco

{ id: 3, name: "Place C", location: { type: "Point", coordinates: [-118.25, 34.05] } }, // Los Angeles

{ id: 4, name: "Place D", location: { type: "Point", coordinates: [-87.63, 41.88] } }, // Chicago

SN AIML ATME

{ id: 5, name: "Place E", location: { type: "Point", coordinates: [-80.19, 25.77] } } // Miami

])

Create a 2dsphere Index on location:

>db.places.createIndex({ location: "2dsphere" })

Geospatial Query (Find places within 10km of a given point):

>db.places.find({ location: {

$near: {

$geometry: {

type: "Point",

coordinates: [-73.97, 40.77]

}, $maxDistance: 10000 // 10km in meters

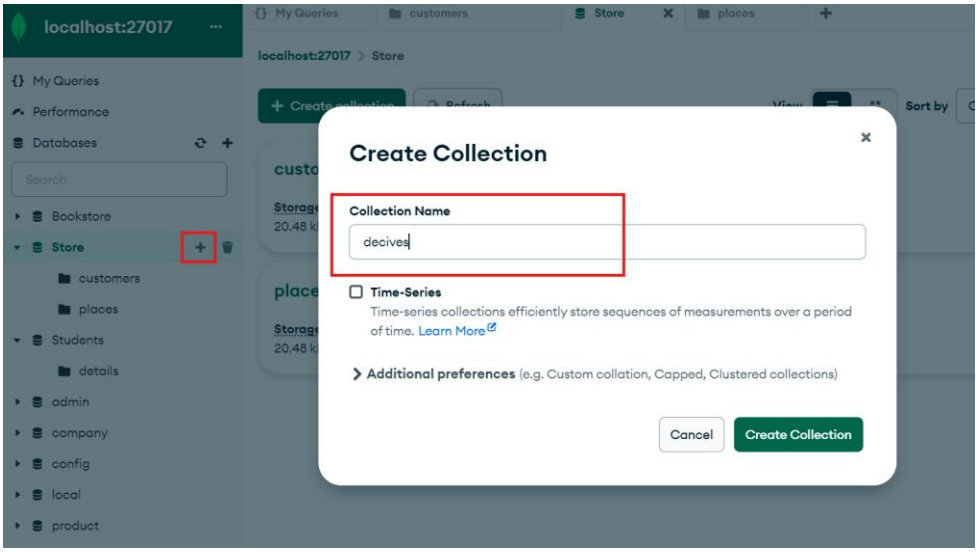
}})

```
{
  _id: ObjectId('66582607f7e76d265c992a3f'),
  id: 1,
  name: 'Place A',
  location: {
    type: 'Point',
    coordinates: [
      -73.97,
      40.77
    ]
  }
}
```

2. Bitwise Selectors

Under database **Store**, create a collection **devices** in Mongo DB IDE**.**

SNJAIML ATME



We'll use a collection devices with fields id, name, and status (where status is a bitwise flag).

>db.devices.insertMany([

{ id: 1, name: "Device A", status: 5 }, // 0101 in binary

{ id: 2, name: "Device B", status: 3 }, // 0011 in binary

{ id: 3, name: "Device C", status: 6 }, // 0110 in binary

{ id: 4, name: "Device D", status: 12 }, // 1100 in binary

{ id: 5, name: "Device E", status: 7 } // 0111 in binary

])

Bitwise AND Query (Find devices where the 2nd bit is set):

>db.devices.find({ status: { $bitsAllSet: 2 } })

SN AIML ATME

Bitwise OR Query (Find devices where any bit in 0101 is set):

To find all devices where any of the bits at positions 0 or 3 are set (i.e., either the least significant bit or the fourth bit is set), you can use the `$bitsAnySet` operator as follows:

> **db.devices.find({ "status": { "$bitsAnySet": [0, 3] } })**

SN AIML ATME

```
{
  _id: ObjectId('66583668f7e76d265c992a44'),
  id: 1,
  name: 'Device A',
  status: 5
}
{
  _id: ObjectId('66583668f7e76d265c992a45'),
  id: 2,
  name: 'Device B',
  status: 3
}
{
  _id: ObjectId('66583668f7e76d265c992a47'),
  id: 4,
  name: 'Device D',
  status: 12
}
```

```
{
  _id: ObjectId('66583668f7e76d265c992a48'),
  id: 5,
  name: 'Device E',
  status: 7
}
```

SIVAMLATME

In MongoDB, the main geospatial query operators include:

1. **$geoWithin**: Finds documents within a specified geometry (e.g., a polygon).
2. **$geoIntersects**: Finds documents that intersect with a specified geometry.
3. **$near**: Finds documents near a specified point, using a 2dsphere index.
4. **$nearSphere**: Similar to $near, but calculates distances using spherical geometry.

5. **$center**: Finds documents within a circular area (used with legacy coordinate pairs).
6. **$centerSphere**: Finds documents within a circular area on a sphere (used with legacy coordinate pairs).
7. **$box**: Finds documents within a rectangular area (used with legacy coordinate pairs).
8. **$polygon**: Finds documents within a polygon defined by multiple points (used with legacy coordinate pairs).

In MongoDB, the main bitwise query operators include:

1. **$bitsAllClear**: Matches documents where all of the given bit positions are clear (i.e., 0).
2. **$bitsAllSet**: Matches documents where all of the given bit positions are set (i.e., 1).
3. **$bitsAnyClear**: Matches documents where any of the given bit positions are clear (i.e., 0).
4. **$bitsAnySet**: Matches documents where any of the given bit positions are set (i.e., 1).

SN AIML ATME

Explanation

- **Geospatial Selector**:
  - **$near**: Finds documents near a specified point. Requires a `2dsphere` index on the location field.
  - **$geometry**: Specifies the reference point as a GeoJSON object.
  - **$maxDistance**: Limits the distance from the reference point (in meters).
- **Bitwise Selector**:
  - **$bitsAllSet**: Matches documents where all of the given bit positions are 1.
  - **$bitsAnySet**: Matches documents where any of the given bit positions are 1.

By executing these queries, you can filter documents based on geospatial proximity and bitwise conditions.

SN AIML ATME