Experiment 3:

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

Comparison Selectors:

$gt (Greater Than)

Find students older than 22.

```
db.students.find({ age: { $gt: 22 } })
```

$lt (Less Than)

Find students younger than 23.

```
db.students.find({ age: { $lt: 23 } })
```

$gte (Greater Than or Equal To)

Find students aged 22 or older.

```
db.students.find({ age: { $gte: 22 } })
```

$lte (Less Than or Equal To)

Find students aged 23 or younger.

```
db.students.find({ age: { $lte: 23 } })
```

$eq (Equal To)

Find students aged 22.

```
db.students.find({ age: { $eq: 22 } })
```

$ne (Not Equal To)

Find students not from the "AIML" department.

```
db.students.find({ department: { $ne: "AIML" } })
```

Logical Selectors:

$and (Both Conditions Must Be True)

Find students from CSE and older than 22.

```
db.students.find({
  $and: [
    { department: "CSE" },
    { age: { $gt: 22 } }
  ]
```

```
})
```

$or (At Least One Condition Must Be True)

Find students who are either from AIML or IT.

```
db.students.find({
  $or: [
    { department: "AIML" },
    { department: "IT" }
  ]
})
```

$not (Negates a Condition)

Find students who are not from the IT department.

```
db.students.find({
  department: { $not: { $eq: "IT" } }
})
```

$nor (Neither Condition is True)

Find students who are neither from IT nor from CSE.

```
db.students.find({
  $nor: [
    { department: "IT" },
    { department: "CSE" }
  ]
})
```

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

Geospatial Selectors

Geospatial query selectors ($near, $geoWithin, $geoIntersects) to work with location-based data stored in GeoJSON format.

Modifying students Collection to Include Location Data

Students have a location field representing their latitude and longitude.

GeoJSON Type Usage Example

Point A single location "coordinates": [77.5946, 12.9716]

LineString A **path** or **route** "coordinates": [[Bangalore], [Hyderabad], [Delhi]]

Polygon A **region** or **boundary** "coordinates": [[City1, City2, City3, City4, City1]]

Point:

```
db.students.insertMany([
{ _id: 30, name: "Esha", age: 22, department: "AIML", location: { type: "Point", coordinates: [77.5946, 12.9716] } }, // Bangalore
{ _id: 31, name: "Farhan", age: 23, department: "CSE", location: { type: "Point", coordinates: [78.4867, 17.3850] } }, // Hyderabad
{ _id: 32, name: "Gaurav", age: 24, department: "ECE", location: { type: "Point", coordinates: [72.8777, 19.0760] } }, // Mumbai
{ _id: 33, name: "Himani", age: 21, department: "IT", location: { type: "Point", coordinates: [77.2090, 28.6139] } }  // Delhi
])
```

Creating a 2dsphere Index

To perform geospatial queries, we need to create an index on the location field.

```
db.students.createIndex({ location: "2dsphere" })
```

$near (Find Students Near a Location)

Find students within a 50 km radius of Bangalore (77.5946, 12.9716).

```
db.students.find({
location: {
```

```
$near: {
$geometry: { type: "Point", coordinates: [77.5946, 12.9716] },
$maxDistance: 50000 // 50 km in meters
}
}
})
```

$geoWithin (Find Students Inside a Specific Area)

Find students within a defined polygon (e.g., Delhi, Mumbai, and Hyderabad).

```
db.students.find({
location: {
$geoWithin: {
$geometry: {
        type: "Polygon",
        coordinates: [[
        [77.2090, 28.6139], // Delhi
        [72.8777, 19.0760], // Mumbai
        [78.4867, 17.3850], // Hyderabad
        [77.2090, 28.6139]  // Closing loop (Delhi)
        ]]
}
}
}
})
```

$geoIntersects (Find Students Who Fall on a Specific Geospatial Path)

Find students who fall exactly on a given line (example: a route from Delhi to Mumbai).

```
db.students.find({
location: {
$geoIntersects: {
```

```
$geometry: {
          type: "LineString",
          coordinates: [
          [77.2090, 28.6139], // Delhi
          [72.8777, 19.0760]  // Mumbai
          ]
}
}
}
})
```

LineString:

Inserting Students with Travel Routes (LineString)

```
db.students.insertMany([
  {
    _id: 50,
    name: "Aarav",
    age: 22,
    department: "AIML",
    travel_route: {
      type: "LineString",
      coordinates: [
        [77.5946, 12.9716],  // Home (Bangalore)
        [78.4867, 17.3850],  // Transit (Hyderabad)
        [77.2090, 28.6139]   // University (Delhi)
      ]
    }
  },
  {
    _id: 51,
```

```
    name: "Sanya",
    age: 23,
    department: "CSE",
    travel_route: {
      type: "LineString",
      coordinates: [
        [72.8777, 19.0760],  // Home (Mumbai)
        [77.2090, 28.6139],  // Transit (Delhi)
        [78.4867, 17.3850]   // University (Hyderabad)
      ]
    }
  }
])
```

Find students whose **travel route intersects with Hyderabad (78.4867, 17.3850).**

```
db.students.find({
  travel_route: {
    $geoIntersects: {
      $geometry: { type: "Point", coordinates: [78.4867, 17.3850] }
    }
  }
})
```

Polygon:

Inserting Students with Home Areas (Polygon)

```
db.students.insertMany([
  {
    _id: 52,
    name: "Rohan",
    age: 24,
```

```
    department: "IT",
    home_area: {
      type: "Polygon",
      coordinates: [[
        [77.5946, 12.9716],  // Point 1 (Bangalore)
        [78.4867, 17.3850],  // Point 2 (Hyderabad)
        [77.2090, 28.6139],  // Point 3 (Delhi)
        [72.8777, 19.0760],  // Point 4 (Mumbai)
        [77.5946, 12.9716]   // Closing loop (Bangalore)
      ]]
    }
  },
  {
    _id: 53,
    name: "Kavya",
    age: 21,
    department: "ECE",
    home_area: {
      type: "Polygon",
      coordinates: [[
        [80.2785, 13.0827],  // Chennai
        [81.8463, 16.5062],  // Rajahmundry
        [83.2185, 17.6868],  // Visakhapatnam
        [80.2785, 13.0827]   // Closing loop (Chennai)
      ]]
    }
  }
])
```

Find students **who live inside the Bangalore-Hyderabad-Delhi-Mumbai zone**.

```
db.students.find({
  location: {
    $geoWithin: {
      $geometry: {
        type: "Polygon",
        coordinates: [[
          [77.5946, 12.9716],  // Bangalore
          [78.4867, 17.3850],  // Hyderabad
          [77.2090, 28.6139],  // Delhi
          [72.8777, 19.0760],  // Mumbai
          [77.5946, 12.9716]   // Closing loop
        ]]
      }
    }
  }
})
```

Bitwise Selectors

Add a permissions field where each bit represents access rights (e.g., 1 = Read, 2 = Write, 4 = Execute).

```
db.students.insertMany([
{ _id: 20, name: "Amit", age: 22, department: "AIML", permissions: 5 },  // Binary: 101 (Read & Execute)
{ _id: 21, name: "Bhavya", age: 23, department: "CSE", permissions: 3 }, // Binary: 011 (Read & Write)
{ _id: 22, name: "Chirag", age: 24, department: "ECE", permissions: 6 }, // Binary: 110 (Write & Execute)
{ _id: 23, name: "Diya", age: 21, department: "IT", permissions: 7 }    // Binary: 111 (Read, Write, Execute)
])
```

$bitsAllSet (Match Documents Where All Specified Bits are Set)

Find students who have both Read (001) and Execute (100) permissions.

```
db.students.find({ permissions: { $bitsAllSet: 5 } }) // Binary 101
```

$bitsAnySet (Match If At Least One Specified Bit is Set)

Find students who have either Write (010) or Execute (100) permissions.

```
db.students.find({ permissions: { $bitsAnySet: 6 } }) // Binary 110
```

$bitsAllClear (Match If None of the Specified Bits Are Set)

Find students who do NOT have Execute (100) permissions.

```
db.students.find({ permissions: { $bitsAllClear: 4 } }) // Binary 100
```

$bitsAnyClear (Match If At Least One of the Specified Bits is Not Set)

Find students who are missing either Read (001) or Write (010) permissions.

```
db.students.find({ permissions: { $bitsAnyClear: 3 } }) // Binary 011
```