Experiment:3

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

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

SELECTORS:

In MongoDB, selectors are query expressions used to specify criteria for selecting documents within a collection. They allow you to define conditions that documents must meet to be included in the query results, using key-value pairs and various operators.

For example, a selector like **`{ age: { $gt: 18 } }`** finds all documents where the `age` field is greater than 18.

COMPARISON gt lt:

```
db> db.students.find({age:{$gt:20}});
[
  {
    _id: ObjectId('6649bb89b51b15a423b44ad0'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44ad1'),
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44ad2'),
    name: 'Student 305',
    age: 24,
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']",
    gpa: 3.4,
    home_city: 'City 6',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44ad5'),
    name: 'Student 440',
    age: 21,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 2.56,
    home_city: 'City 10',
    blood_group: 'O-',
    is_hotel_resident: true
```

The provided MongoDB query retrieves documents from the `students` collection, **excluding the `_id` field, and sorts them in descending order by their `_id`.** This is achieved using the **`sort` method with `{_id: -1}`** and **limits the result set to the first 5 documents**.

The result displays the most recently inserted or updated students, showing their names, ages, courses, GPAs, home cities, blood groups, and hotel residency status. This approach is useful for viewing the latest entries in a collection, helping users quickly access and analyze recent data.

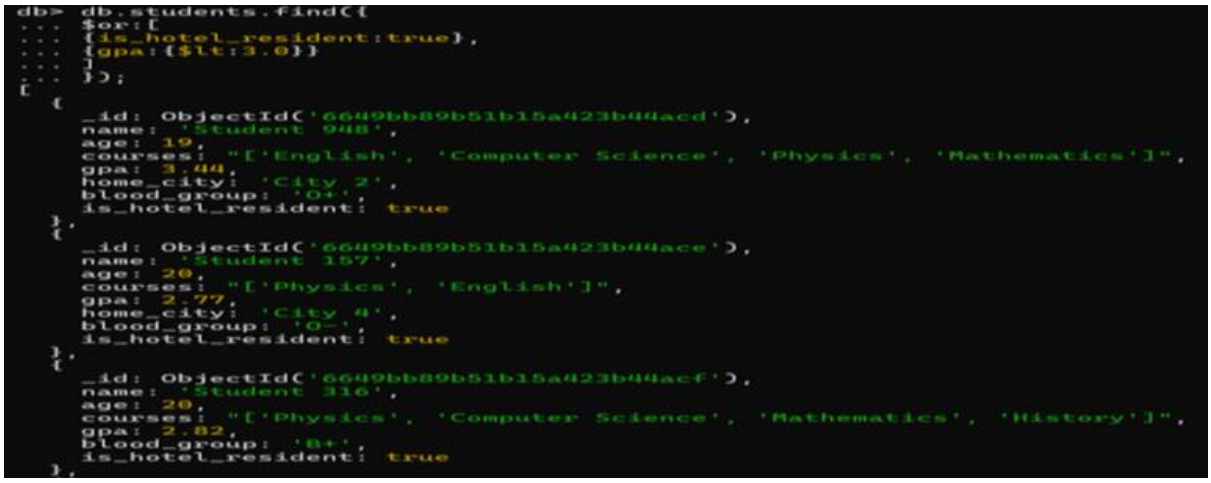
AND OPERATOR:

```
db> db.students.find({
... $and:[
... {home_city:"City 2"},
... {blood_group:"B+"}
... ]
... });
[
  {
    _id: ObjectId('6649bb89b51b15a423b44ae5'),
    name: 'Student 504',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
    gpa: 2.92,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44c93'),
    name: 'Student 872',
    age: 24,
    courses: "['English', 'Mathematics', 'History']",
    gpa: 3.36,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  }
]
db>
```

The MongoDB query uses the **`find` method with an `$and` operator** to retrieve documents from the `students` collection where both conditions `home_city` equal to "City 2" and `blood_group` equal to "B+" are met.

The query returns documents that match these criteria, displaying the `_id`, `name`, `age`, `courses`, `gpa`, `home_city`, `blood_group`, and `is_hotel_resident` fields for the matching students. In this case, the result includes two students who live in "City 2" and have a blood group of "B+", providing a way to filter and obtain specific data based on multiple conditions.

OR OPERATIONS:



The MongoDB query uses the **`find` method with an `$or` operator to retrieve documents** from the `students` collection where at least one of the two conditions is met: `is_hotel_resident` is `true` or `gpa` is less than `3.0`.

This query fetches all students who are either hotel residents or have a GPA below 3.0. The result includes multiple students with various attributes such as `_id`, `name`, `age`, `courses`, `gpa`, `home_city`, `blood_group`, and `is_hotel_resident`.

The students in the result set either reside in the hotel or have a GPA below 3.0, indicating that the query successfully combined the two conditions using the `$or` operator to provide a list of students meeting at least one of the specified criteria.

LET'S TAKE NEW DATASET: New students_permission dataset Explanation: Collection name: students_permission name: Student's name (string) age: Student's age (number)

Bitwise Value

- In our example its a 32 bit each bit representing different things
- Bitwise value 7 means all access 7 -> 111

Bit 3	Bit 2	Bit 1
cafe	campus	lobby

Bitwise Types:

Bitwise

Name	Description
`$bitsAllClear`	Matches numeric or binary values in which a set of bit positions *all* have a value of `0`.
`$bitsAllSet`	Matches numeric or binary values in which a set of bit positions *all* have a value of `1`.
`$bitsAnyClear`	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of `0`.
`$bitsAnySet`	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of `1`.

QUERY:

In MongoDB, a query is a command used to retrieve documents from a collection that match certain criteria. Queries are written in MongoDB's query language and can include conditions, projections, and other modifiers to specify exactly which documents to retrieve. MongoDB queries can be complex, allowing for precise filtering, sorting, and aggregation of data.

```
db> const LOBBY_PERMISSION=1;

db> const CAMPUS_PERMISSION=2;

db> db.students_permission.find({
... permissions:{$bitsAllSet:[LOBBY_PERMISSION,CAMPUS_PERMISSION]}
... });
[
  {
    _id: ObjectId('66635182d29d811170a4e560'),
    name: 'George',
    age: 21,
    permissions: 6
  },
  {
    _id: ObjectId('66635182d29d811170a4e561'),
    name: 'Henry',
    age: 27,
    permissions: 7
  },
  {
    _id: ObjectId('66635182d29d811170a4e562'),
    name: 'Isla',
    age: 18,
    permissions: 6
  }
]
db>
```

The MongoDB query utilizes the `find` method with the **`$bitsAllSe**t` operator to retrieve documents from the **`students_permission`** collection where specific bits in the `permissions` field are set. The **`$bitsAllSet`** operator checks if all specified bit positions are set to `1` in the binary representation of the `permissions` field. In this query, the bit

positions represented by `LOBBY_PERMISSION` and `CAMPUS_PERMISSION` are checked. These constants represent specific permission bits in a

binary format. The query returns documents where both of these bits are set in the `permissions` field. The result includes the following documents: 1. **George**: A 21-year-old with a `permissions` value of 6. 2. **Henry**: A 27-year-old with a `permissions` value of 7. 3. **Isla**: An 18-year-old with a `permissions` value of 6. These results indicate that for George and Isla, the `permissions` value of 6 (binary `110`) has both `LOBBY_PERMISSION` and `CAMPUS_PERMISSION` bits set. For Henry, the `permissions` value of 7 (binary `111`) also satisfies the condition as both required bits are set along with an additional bit. This query effectively filters the documents to find students who have both the lobby and campus permissions enabled.

LET'S TAKE NEW DATASET: New locations dataset.

GEOSPATIAL:

In MongoDB, geospatial refers to the capability to store and query data based on its geographic location. MongoDB supports various geospatial queries, including finding points within a specified distance of a location, finding objects within a specified polygon, and finding the nearest objects to a location. Geospatial indexes can be created to efficiently perform these types of queries on geospatial data.

GEOSPATIAL QUERY:

In MongoDB, a geospatial query is used to retrieve documents based on their geographical location. These queries utilize special operators like **$geoNear, $geoWithin**, and **$near** to find documents near a specific point, within a specified area, or based on proximity. Geospatial queries are particularly useful for applications that require location-based

searches, such asmapping or 9 location-based services.

```
db> db.locations.find({
... location:{
... $geoWithin:{
... $centerSphere:[[-74.005,40.712],0.00621376]
... }
... }
... });
[
  {
    _id: 1,
    name: 'Coffee Shop A',
    location: { type: 'Point', coordinates: [ -73.985, 40.748 ] }
  },
  {
    _id: 2,
    name: 'Restaurant B',
    location: { type: 'Point', coordinates: [ -74.009, 40.712 ] }
  },
  {
    _id: 5,
    name: 'Park E',
    location: { type: 'Point', coordinates: [ -74.006, 40.705 ] }
  }
]
db>
```

The given MongoDB query is using the **$geoWithin** operator with **$centerSphere** to find locations within a specified radius from a central point. In this case, it's searching for locations within approximately 500 meters (0.00621376 radians) from the coordinates [-

74.005, 40.712], which is a point in New York City. The result includes three locations: Coffee Shop A, Restaurant B, and Park E, along with their respective coordinates.

DATATYPES:

In MongoDB, the data types include String, Number, Boolean, Object, Array, Null, and Date, among others.

Point: A point is a GeoJSON object representing a single geographic coordinate.

LineString: **A LineString is a GeoJSON object** representing a sequence of connected line segments.

Polygon: A Polygon is a GeoJSON object representing a closed, two-dimensional shape with three or more vertices.

Data types and Operations:

Name	Description
`$geoIntersects`	Selects geometries that intersect with a GeoJSON geometry. The `2dsphere` index supports `$geoIntersects`.
`$geoWithin`	Selects geometries within a bounding GeoJSON geometry. The `2dsphere` and `2d` indexes support `$geoWithin`.
`$near`	Returns geospatial objects in proximity to a point. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$near`.
`$nearSphere`	Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$nearSphere`.