# Experiment:3

## Comparison gt lt

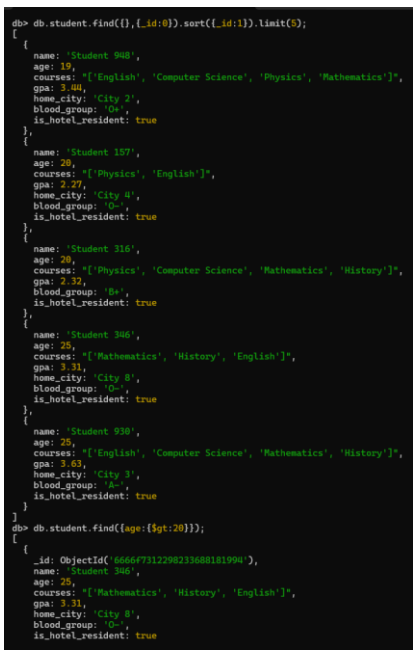
```
// Find all students with age greater than 20
db.students.find({ age: { $gt: 20 } });
```

**db.students.find({ age: { $gt: 20 } })** : This line uses the `.find` method of the `students` collection to retrieve documents from the database that match the specified filter criteria.

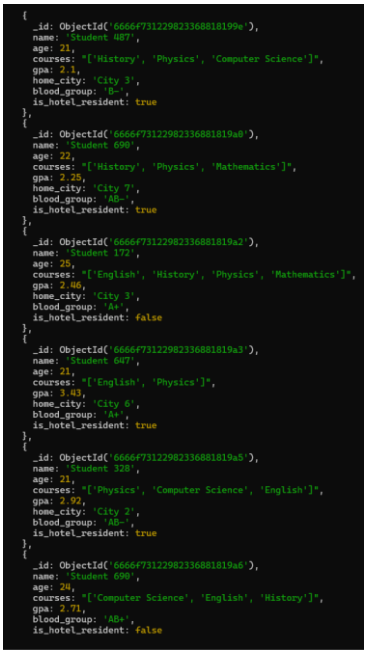
- **db.students** : This refers to the `students` collection within the MongoDB database.
- **.find({ age: { $gt: 20 } })** :
- `.find`: The `.find` method is used to find documents in a collection.
- `{ age: { $gt: 20 } }`: This is the filter document that specifies which documents to retrieve.
- `age`: The field to filter on in the documents.
- `{ $gt: 20 }`: The filter operator. In this case, `$gt` stands for "greater than" and 20 is the value to compare against. So this filter retrieves documents where the `age` field is greater than 20.

## Output :

This code snippet finds all documents in the `students` collection where the `age` field is greater than 20.








# AND operator

```
// Find students from "City 2" with blood group "B+"
db.students.find({
  $and: [
    { home_city: "City 2" },
    { blood_group: "B+" }
  ]
});
```

- **db.students.find({** : This line uses the `.find` method of the `students` collection to retrieve documents from the database that match the specified filter criteria.

  - **db.students** : This refers to the `students` collection within the MongoDB database.
  - **.find({** : The `.find` method is used to find documents in a collection. The opening curly brace `{` marks the beginning of the filter document.

- **$and: [** : This line uses the MongoDB aggregation operator `$and` to combine multiple filter conditions. The `$and` operator requires that all the specified conditions must be true for a document to be included in the results.

- **{ home_city: "City 2" },** : This is the first filter condition within the `$and` operator. It filters documents where the `home_city` field is equal to "City 2".

  - `{ home_city: "City 2" }`: This is a filter document with a single key-value pair.
  - `home_city`: The field to filter on.
  - `"City 2"`: The value to compare against.

- **{ blood_group: "B+" }** : This is the second filter condition within the `$and` operator. It filters documents where the `blood_group` field is equal to "B+".

  - `{ blood_group: "B+" }`: This is a filter document with a single key-value pair.
    - `blood_group`: The field to filter on.
    - `"B+"`: The value to compare against.

- **]}):** :

  - `]`: This closing square bracket `]` marks the end of the array of filter conditions within the `$and` operator.
  - `})`: This closing curly brace `}` marks the end of the filter document passed to the `.find` method.

## Output :

This code snippet queries the `students` collection to find documents where both the `home_city` field is "City 2" and the `blood_group` field is "B+". The `$and` operator ensures that documents must satisfy both conditions to be included in the results.

# OR operator

```
// Find students who are hotel residents OR have a GPA less than 3.0
db.students.find({
  $or: [
    { is_hotel_resident: true },
    { gpa: { $lt: 3.0 } }
  ]
});
```

- **db.students.find({** : This line uses the `.find` method of the `students` collection to retrieve documents from the database that match the specified filter criteria.

  - **db.students** : This refers to the `students` collection within the MongoDB database.
  - **.find({** : The `.find` method is used to find documents in a collection. The opening curly brace `{` marks the beginning of the filter document.

- **$or: [** : This line uses the MongoDB aggregation operator `$or` to combine multiple filter conditions. The `$or` operator allows any of the specified conditions to be true for a document to be included in the results.

- **{ is_hotel_resident: true },** : This is the first filter condition within the `$or` operator. It filters documents where the `is_hotel_resident` field is equal to `true`.

  - `{ is_hotel_resident: true }`: This is a filter document with a single key-value pair.
    - `is_hotel_resident`: The field to filter on.
    - `true`: The value to compare against.

- **{ gpa: { $lt: 3.0 } }** : This is the second filter condition within the `$or` operator. It filters documents where the `gpa` field is less than 3.0.

  - `{ gpa: { $lt: 3.0 } }`: This is a filter document with a nested filter document.
    - `gpa`: The field to filter on.
    - `{ $lt: 3.0 }`: The filter operator. In this case, `$lt` stands for "less than" and 3.0 is the value to compare against.

- **]}):**

  - `]`: This closing square bracket `]` marks the end of the array of filter conditions within the `$or` operator.
  - `})`: This closing curly brace `}` marks the end of the filter document passed to the `.find` method.

## Output :

This code snippet queries the `students` collection to find documents where either the `is_hotel_resident` field is set to `true` or the `gpa` field is less than 3.0. The `$or` operator allows documents to satisfy either condition to be included in the results.

```
db> db.student.find({$or:[{is_hostel_resident:true},{gpa:{$lt:3.0}}]});
[
  {
    _id: ObjectId('6666f731229823368818992'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6666f731229823368818993'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6666f731229823368818998'),
    name: 'Student 563',
    age: 18,
    courses: "['Mathematics', 'English']",
    gpa: 2.28,
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6666f731229823368818999'),
    name: 'Student 440',
    age: 21,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 2.06,
    home_city: 'City 10',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6666f731229823368818999a'),
    name: 'Student 536',
    age: 20,
    courses: "['History', 'Physics', 'English', 'Mathematics']",
    gpa: 2.87,
    home_city: 'City 3',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6666f731229823368818999b'),
    name: 'Student 256',
    age: 19,
    courses: "['Computer Science', 'Mathematics', 'History', 'English']",
    gpa: 2.94,
    home_city: 'City 1',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6666f731229823368818999c'),
    name: 'Student 177',
    age: 23,
    courses: "['Mathematics', 'Computer Science', 'Physics']",
    gpa: 2.52,
    home_city: 'City 10',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6666f731229823368818999e'),
    name: 'Student 487',
    age: 21,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 2.1,
    home_city: 'City 3',
    blood_group: 'B-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6666f731229823368818999f'),
    name: 'Student 213',
    age: 18,
    courses: "['English', 'History']",
    gpa: 2.39,
    blood_group: 'B-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6666f731229823368818a00'),
    name: 'Student 690',
    age: 22,
    courses: "['History', 'Physics', 'Mathematics']",
    gpa: 2.25,
    home_city: 'City 7',
    blood_group: 'AB-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6666f731229823368818a02'),
    name: 'Student 172',
    age: 25,
  },
  {
    _id: ObjectId('6666f731229823368818a02'),
    name: 'Student 172',
    age: 25,
    courses: "['English', 'History', 'Physics', 'Mathematics']",
    gpa: 2.46,
    home_city: 'City 3',
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6666f731229823368818a04'),
    name: 'Student 232',
    age: 18,
    courses: "['Computer Science', 'Physics', 'History', 'Mathematics']",
    gpa: 2.54,
    home_city: 'City 1',
    blood_group: 'B-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6666f731229823368818a05'),
    name: 'Student 328',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English']",
    gpa: 2.92,
    home_city: 'City 2',
    blood_group: 'AB-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6666f731229823368818a06'),
    name: 'Student 690',
    age: 24,
    courses: "['Computer Science', 'English', 'History']",
    gpa: 2.71,
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6666f731229823368818a07'),
    name: 'Student 499',
    age: 25,
    courses: "['Mathematics', 'English', 'Computer Science', 'Physics']",
    gpa: 2.04,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6666f731229823368818a09'),
    name: 'Student 504',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
    gpa: 2.42,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6666f731229823368818a0b'),
    name: 'Student 915',
    age: 22,
    courses: "['Computer Science', 'History', 'Physics', 'English']",
    gpa: 2.87,
    blood_group: 'AB+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6666f731229823368818a0c'),
    name: 'Student 127',
    age: 19,
    courses: "['History', 'English', 'Computer Science', 'Mathematics']",
    gpa: 2.56,
    home_city: 'City 10',
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6666f731229823368818a0e'),
    name: 'Student 770',
    age: 24,
    courses: "['History', 'Computer Science', 'Mathematics', 'English']",
    gpa: 2.98,
    blood_group: 'B-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6666f731229823368818a0f'),
    name: 'Student 367',
    age: 25,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 2.61,
    home_city: 'City 1',
    blood_group: 'AB+',
    is_hotel_resident: true
  }
]
Type "it" for more
db>
```

# Let's Take new Data set

## 1.New Students Permission dataset link

**2.Explanation: Collection name: students_permission**

**3.name:** Student's name (string)
**4.age:** Student's age (number)
**5.permissions:** Bitmask representing user permissions (number)

## Bitwise Value

1.In our example its a 32 bit each bit representing different things
2.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

```
// Define bit positions for permissions
const LOBBY_PERMISSION = 1;
const CAMPUS_PERMISSION = 2;

// Query to find students with both lobby and campus permissions using
db.students_permission.find({
  permissions: { $bitsAllSet: [LOBBY_PERMISSION, CAMPUS_PERMISSION] }
});
```

- **const LOBBY_PERMISSION = 1;** : This line defines a constant named `LOBBY_PERMISSION` and assigns it the value `1`. In MongoDB, you can store permissions as bit flags, where each bit represents a specific permission. Assigning the value `1` to `LOBBY_PERMISSION` likely means that the first bit in the `permissions` field represents lobby permission.

- **const CAMPUS_PERMISSION = 2;** : This line defines another constant named `CAMPUS_PERMISSION` and assigns it the value `2`. Here, the second bit in the `permissions` field likely represents campus permission.

- **// Query to find students with both lobby and campus permissions using** : This line is another comment that explains the purpose of the following line which is the actual query.

- **db.students_permission.find({** : This line uses the `.find` method of the `students_permission` collection to retrieve documents from the database that match the specified filter criteria.

    - **db.students_permission** : This refers to the `students_permission` collection within the MongoDB database. It likely stores documents that link students to their permission levels.
    - **.find({** : The `.find` method is used to find documents in a collection. The opening curly brace `{` marks the beginning of the filter document.

- **permissions: { $bitsAllSet: [LOBBY_PERMISSION, CAMPUS_PERMISSION] }** : This line defines the filter criteria for the query. It uses the MongoDB aggregation operator `$bitsAllSet` to check if all the specified permission bits are set in the `permissions` field of the documents.

    - `permissions`: The field to filter on in the documents.
    - `{ $bitsAllSet: [LOBBY_PERMISSION, CAMPUS_PERMISSION] }`: This is a filter document with a sub-document that uses the `$bitsAllSet` operator.
    - `$bitsAllSet`: The MongoDB aggregation operator that checks if multiple bit flags are all set in a bitfield.
    - `[LOBBY_PERMISSION, CAMPUS_PERMISSION]`: An array of the bit positions to check. In this case, the operator will check if both the first bit (which we assigned to `LOBBY_PERMISSION`) and the second bit (which we assigned to `CAMPUS_PERMISSION`) are set in the `permissions` field of the documents.

- **})** : This closing curly brace `}` marks the end of the filter document passed to the `.find` method.

## Output :

This code snippet first defines constants to represent bit positions for lobby and campus permissions. Then, it finds all documents in the `students_permission` collection where both the lobby permission bit and the campus permission bit are set in the `permissions` field. This indicates that the students have both lobby and campus permissions.

```
db> const LOBBY_PERMISSION=1;

db> const CAMPUS_PERMISSION=2;

db> db.student_permission.find({permissions:{$bitsAllSet:[LOBBY_PERMISSION,CAMPUS_PERMISSION]}});
[
  {
    _id: ObjectId('6650c5cbfbe86b0d030ac88b'),
    name: 'George',
    age: 21,
    permissions: 6
  },
  {
    _id: ObjectId('6650c5cbfbe86b0d030ac88c'),
    name: 'Henry',
    age: 27,
    permissions: 7
  },
  {
    _id: ObjectId('6650c5cbfbe86b0d030ac88d'),
    name: 'Isla',
    age: 18,
    permissions: 6
  }
]
db>
```

# Geospatial

1.Official Documentation link
2.Create collection called “locations”
3.Upload the dataset using json link

---

```
_id: 1
name : "Coffee Shop A"
▾ location : Object
    type : "Point"
  ▸ coordinates : Array (2)
```

# Query:

```
db.locations.find({
  location: {
    $geoWithin: {
      $centerSphere: [[-74.005, 40.712], 0.00621376] // 1 kilometer in
    }
  }
});
```

- **db.locations.find({** : This line uses the `.find` method of the `locations` collection to retrieve documents from the database that match the specified filter criteria.

- **db.locations**: This refers to the `locations` collection within the MongoDB database. It likely stores geospatial data about various locations.
- **.find({** : The `.find` method is used to find documents in a collection. The opening curly brace `{` marks the beginning of the filter document that specifies which documents to retrieve.

- **location: {** : This line specifies that the filter criteria pertain to the `location` field within the documents.

- **$geoWithin: {** : This line uses the MongoDB geospatial aggregation operator `$geoWithin` to filter for documents where the `location` field falls within a specific area.

- **$centerSphere: [[-74.005, 40.712], 0.00621376] // 1 kilometer in radians** : This sub-document defines the criteria for the `$geoWithin` operator. It uses another geospatial operator `$centerSphere` to specify a spherical area around a center point.

  - `$centerSphere`: This operator filters documents that have a geospatial location within a specified spherical area.
  - `[[-74.005, 40.712], 0.00621376]`: This array defines the center point and the radius of the sphere.
    - `[-74.005, 40.712]`: This is an array containing two elements representing the longitude and latitude coordinates of the center point. In this case, it's `[-74.005, 40.712]`.
    - `0.00621376`: This is the radius of the sphere specified in radians. The comment indicates this is equivalent to 1 kilometer.

- **}** : This closing curly brace `}` marks the end of the sub-document that defines the criteria for the `$geoWithin` operator.

- **}** : This closing curly brace `}` marks the end of the filter document passed to the `.find` method.

## Output :

this code snippet queries the `locations` collection to find all documents where the `location` field is within a one-kilometer radius of the center point specified by the longitude and latitude coordinates [-74.005, 40.712]. The conversion from kilometers to radians is likely done outside of the code snippet you provided.

# Data types and Operations

## DataType

1.Point
2.Line String
3.Polygon

# 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`. |