Experiment 03

MongoDB Query Selector

Introduction to query selectors

Whenever we query a MongoDB database, it fetches some data stored in the Mongolia documents from various MongoDB collections. There are different requirement data throughout the application. Sometimes, we need to use some small pre of data for a specific action in the application, sometimes, we need to update the data based on some criteria, and sometimes, we need to delete some old documents based on some condition. In these situations, we need something which will help s to work on only those documents which are needed for CRUD operations.

So, the query selectors are helpful for the following:

- Fetch the MongoDB documents from the collections based on some conditions
- Modify the multiple documents based on some conditions
- Delete the multiple documents based on some conditions

There are various types of query selectors we can use based on our requirements. In MongoDB, these query selectors are divided into the following types:

1. Comparison Selectors
2. Logical Selectors
3. Element Selectors
4. Array Selectors
5. Evaluation Selectors
6. Geospatial Selectors
7. Bitwise Selectors
8. Comment Selector.

a. Execute query selector (Comparison Selectors, Logical Selectors) and list out the result on any collection

Comparison Selector:-

These types of selectors are helpful to perform the CRUD operations based on the comparison. The following table has the list of comparison selectors.

Selector	Selection Description	Selector Use and Syntax
$eq	This selector matches the documents that have values equal to the specified value	{ <document field> : { $eq: <value to match> } }
$gt	This selector matches the documents that have values greater than the specified value	{<document field> : {$gt: <value to match>} }
$gte	This selector matches the documents that have values greater than or equal to the specified value	{<document field> : {$gte : <value to match>} }
$in	This selector matches the documents that have any of the values specified in an array	<document field> : { $in : [<array value1>, <array value2>, ... <array valueN>] }}
$lt	This selector matches the documents that have values less than the specified value	{<document field> : {$lt : <value to match>} }
$lte	This selector matches the documents that have values less than or equal to the specified	{<document field> : {$lte : <value to match>} }
$ne	This selector matches the documents that have values not equal to the specified value	{ <document field> : { $ne : <value to match> } }
$nin	This selector matches the documents that have none of the values specified in an array	{<document field> : { $nin : [<array value1>, <array value2>, ... <array valueN>] }}

Ex01- $gt Comparison Selector

//Find all students with age greater than 20

```
db> db.std.find({age:{$gt:20}}).count()
310
db> db.std.find({age:{$gt:20}})
[
  {
    _id: ObjectId('666529e3819e1a387778718d'),
    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('666529e3819e1a387778718e'),
    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('666529e3819e1a387778718f'),
    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('666529e3819e1a3877787190'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666529e3819e1a3877787192'),
    name: 'Student 440',
    age: 21,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 2.06,
    home_city: 'City 10',
    blood_group: 'O-',
    is_hotel_resident: true
  },
```

Ex02 - $lte Comparison Selector

// Find all students with age lesser than or equal 18

```
db> db.std.find({age:{$lte:18}}).count()
58
db> db.std.find({age:{$lte:18}})
[
  {
    _id: ObjectId('666529e3819e1a3877787191'),
    name: 'Student 563',
    age: 18,
    courses: "['Mathematics', 'English']",
    gpa: 2.25,
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666529e3819e1a3877787198'),
    name: 'Student 213',
    age: 18,
    courses: "['English', 'History']",
    gpa: 2.39,
    blood_group: 'B-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('666529e3819e1a387778719d'),
    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('666529e3819e1a38777871a3'),
    name: 'Student 652',
    age: 18,
    courses: "['History', 'Computer Science']",
    gpa: 3.93,
    home_city: 'City 8',
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
```

Logical Selector :-

These types of selectors are helpful to perform the CRUD operations based on the logical condition. The following table has the list of logical selectors.

Selector	Selection Description	Selector Use and Syntax
$and	This selector performs the logical AND operation on different expressions and joins both the queries to deliver the combined result by returning all the documents based on the Join	{ $and : [{ <expression1> }, { <expression2> }, ..., { <expressionN> }] }
4not	This selector performs the logical NOT operation and returns the documents that do not match the expression	{ <document field> : { $not : { <expression> } } }
$or	This selector performs the logical OR operation on different expressions and	{ $or : [{ <expression1> }, {

	joins both the queries to deliver the combined result by returning all the documents based on the Join	<expression2> }, ..., { <expressionN> }] }
$nor	This selector performs the logical NOR operation on different expressions and selects all the documents that fail all the query expressions	{ $nor: [{ <expression1> }, { <expression2> }, ... { <expressionN> }] }

Ex01 - $and logical selector

//Find students from ”City 2” with blood group “B+”

```
db> db.std.find({ $and:[{ home_city:"City 2"},{blood_group:"B+"}]}).count()
7
db> db.std.find({ $and:[{ home_city:"City 2"},{blood_group:"B+"}]})
[
  {
    _id: ObjectId('666529e3819e1a38777871a2'),
    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('666529e3819e1a38777871d9'),
    name: 'Student 367',
    age: 19,
    courses: "['English', 'Physics', 'History', 'Mathematics']",
    gpa: 2.81,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666529e3819e1a3877787265'),
    name: 'Student 255',
    age: 21,
    courses: "['English', 'Physics']",
    gpa: 2.85,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666529e3819e1a3877787271'),
    name: 'Student 281',
    age: 18,
    courses: "['History', 'Mathematics', 'Physics', 'Computer Science']",
    gpa: 2.2,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666529e3819e1a38777872bf'),
    name: 'Student 289',
    age: 18,
    courses: "['History', 'Physics']",
    gpa: 2.89,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666529e3819e1a38777872ea'),
    name: 'Student 303',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 3.08,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666529e3819e1a3877787350'),
    name: 'Student 872',
    age: 24,
    courses: "['English', 'Mathematics', 'History']",
    gpa: 3.36,
    home_city: 'City 2',
    blood_group: 'B+',
```

Ex02 - $or logical selector

Find students who are hostel resident OR have a GPA less than 3.0

```
db> db.std.find({ $or:[{is_hotel_resident:true},{gpa:{$lt:3.0}}]}).count()
3774
db> db.std.find({ $or:[{is_hotel_resident:true},{gpa:{$lt:3.0}}]})
[
  {
    _id: ObjectId('666529e3819e1a3877778718a'),
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('666529e3819e1a3877778718b'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('666529e3819e1a3877778718c'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('666529e3819e1a3877778718d'),
    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('666529e3819e1a3877778718e'),
    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('666529e3819e1a3877778718f'),
    name: 'Student 385',
    age: 24,
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']",
    gpa: 3.4,
    home_city: 'City 6',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('666529e3819e1a3877787191'),
    name: 'Student 563',
    age: 18,
    courses: "['Mathematics', 'English']",
    gpa: 2.25,
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666529e3819e1a3877787192'),
    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('666529e3819e1a3877787193'),
    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('666529e3819e1a3877787194'),
    name: 'Student 256',
```

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

Geospatial Selectors:-

These types of selectors are used to perform operations based on the data which is geographical in nature while we select the documents from the database, data like coordinates, address, city, or ZIP code. GIS data are some examples of the data formats which are geospatial in nature.

MongoDB provides the following geospatial query operators:

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`.

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] }
});
```

Geospatial

- Official Documentation link
- Create collection called “locations”
- Upload the dataset using json link

Example: Geospatial Query

```
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> db.locations.find({location:{$geoWithin:{$centerSphere:[[-74.005,40.712],0.00621376]}}}).count()
3
```

Bitwise Selector:-

These types of selectors are used to perform bitwise operations while we select the documents from the database. The bitwise operations are done at a bit-level, which is the smallest unit of data in the computer system. While we compare the bitwise data, the document field against which we are comparing type or BinData Type (binary data type). the data

Let's Take new Data set

- New Students Permission dataset link

Explanation: Collection name: students_permission

- **name:** Student's name (string)
- **age:** Student's age (number)
- **permissions:** Bitmask representing user permissions (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 café

  Bit 2 campus

  Bit 1 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`.