MONGO DB

EXPERIMENT-01

1.a. Illustration of Where Clause, AND,OR operations in MongoDB.

b. Execute the Commands of MongoDB and operations in MongoDB : Insert, Query, Update, Delete and Projection.

Where Clause, AND, OR Operations.

The Where Clause: A Chameleon for Complex Queries

The `$where` operator acts like a chameleon, allowing you to craft queries using JavaScript expressions. It offers immense flexibility, letting you tailor intricate conditions for finding your treasures. However, this power comes at a cost. JavaScript evaluation for each document can be slow, and it hinders the use of indexes, which are crucial for speedy searches. Consider `$where` for unique situations where built-in operators fall short, but use it judiciously.

The AND Operator: A Matchmaker for Conjunctions

The `$and` operator acts like a meticulous matchmaker, ensuring your documents satisfy all the specified conditions before making a match. Imagine you're searching for a diamond ring (think document). You might have conditions like `color: 'red'` (for a ruby) and `cut: 'brilliant'`. The `$and` operator guarantees the document possesses both qualities for it to be considered a match.

Here's the **syntax for `$and`:**

```
{
  $and: [
    { condition1 },
    { condition2 },
```

```
    ...
  ]
}
```

Each condition within the array is like a separate interview question the document must pass to be a match.

The OR Operator: A Matchmaker for Disjunctions

The `$or` operator acts like a more lenient matchmaker, where a document only needs to fulfill one of the specified conditions to be considered a match. Think of searching for a pirate's treasure (documents). You might be interested in documents with `gold: true` or `gems: { $gt: 10 }` (more than 10 gems). The `$or` operator ensures documents with either of these qualities are presented as potential treasures.

Here's the **syntax for `$or`:**

```
{
  $or: [
    { condition1 },
    { condition 2 },
    ...
  ]
}
```

With `$or`, any condition met is like a green light for the document to be a potential match.

MongoDB's Magic (Combining AND and OR):

Using the $and operator within an OR clause

Example Query:

{ $or:

[{ $and:

[{ condition }, { condition }] },

{ ……. }, { …… }

] }

Where operation

1.Find the total number of students present in collection student1.

syntax: .count();

```
db> db.student1.find().count();
500
```

2.Find the students information **database name.collection name.find();**

```
db> db.student1.find();
[
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e0'),
    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('66645d86df4bc5f5cdf985e1'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e2'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
```

3.Find the students who have age greater than 18 using **$gt**

```
db> db.student1.find({age:{$gt:18}});
[
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e0'),
    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('66645d86df4bc5f5cdf985e1'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e2'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
```

4.Find the students count who all have age greater than 18

```
db> db.student1.find({age:{$gt:18}}).count();
442
```

5.Find the students who have age less than 18, less than or equal to, greater than or equal to, equal to, not equal to using **$lt ,$lte, $gte , $eq , $ne.**

```
db> db.student1.find({age:{$lt:18}}).count();
0
db> db.student1.find({age:{$lte:18}}).count();
58
db>
```

```
db> db.student1.find({age:{$gte:18}}).count();
500
db> db.student1.find({age:{$eq:18}}).count();
58
db> db.student1.find({age:{$ne:18}}).count();
442
```

6.Find the students who all have home city :city1

```
db> db.student1.find({home_city:"City 1"}).count();
34
db> db.student1.find({home_city:"City 1"});
[
  {
    _id: ObjectId('66645d86df4bc5f5cdf985ea'),
    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('66645d86df4bc5f5cdf985f3'),
    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('66645d86df4bc5f5cdf985fe'),
    name: 'Student 367',
    age: 25,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 2.61,
    home_city: 'City 1',
    blood_group: 'AB+',
    is_hotel_resident: true
  },
```

7. Find the students who all have hotel resident false.

```
db> db.student1.find({is_hotel_resident:false}).count();
254
db> db.student1.find({is_hotel_resident:false});
[
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e6'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e7'),
    name: 'Student 563',
    age: 18,
    courses: "['Mathematics', 'English']",
    gpa: 2.25,
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e9'),
    name: 'Student 536',
    age: 20,
    courses: "['History', 'Physics', 'English', 'Mathematics']",
    gpa: 2.87,
    home_city: 'City 3',
    blood_group: 'O-',
    is_hotel_resident: false
  },
```

AND Operation

1.Find all the students who have age 18 and blood group of B-.

```
db> db.student1.find({
... $and:[
... {age:18},
... {blood_group:"B-"}
... ]
... });
[
  {
    _id: ObjectId('66645d86df4bc5f5cdf985ee'),
    name: 'Student 213',
    age: 18,
    courses: "['English', 'History']",
    gpa: 2.39,
    blood_group: 'B-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66645d86df4bc5f5cdf985f3'),
    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
  },
```

2. Find the students count who have age 18 and blood group of B-.

```
db> db.student1.find({ $and: [ { age: 18 }, { blood_group: "B-" } ] }).count();
9
db>
```

OR Operation

1.Find all the students who have age 18 or blood group of B- with count.

```
db> db.student1.find({ $or: [ { age: 18 }, { blood_group: "B-" } ] }).count();
109
db> db.student1.find({ $or: [ { age: 18 }, { blood_group: "B-" } ] });
[
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e7'),
    name: 'Student 563',
    age: 18,
    courses: "['Mathematics', 'English']",
    gpa: 2.25,
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66645d86df4bc5f5cdf985ed'),
    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('66645d86df4bc5f5cdf985ee'),
    name: 'Student 213',
    age: 18,
    courses: "['English', 'History']",
    gpa: 2.39,
    blood_group: 'B-',
    is_hotel_resident: true
  },
```

AND and OR(Combining AND and OR)

1.Find all the students who have home city 2 and gpa less than 3.5.

```
db> db.student1.find({ $or: [{$and:[{home_city:'City 2'},{gpa:{$lt:3.5}}]},{home_cityhome_city:'City 2'},{gpa:1}]});
[
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e0'),
    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('66645d86df4bc5f5cdf985f4'),
    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('66645d86df4bc5f5cdf985f8'),
    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
  },
```

2.Find all the students count who have home city 2 and gpa less than 3.5.

```
db> db.student1.find({ $or: [{$and:[{home_city:'City 2'},{gpa:{$lt:3.5}}]},{home_city:'City 2'},{gpa:1}]}).count();
33
```

CRUD Operations

C- Create

R-Remove

U-Update

D-Delete

1.Insert:

Adds one or more documents to a collection.

Syntax:

db.collection_name.insertOne({ document_data })

db.collection_name.insertMany([document1, document2, ...])

- **db:** Refers to the current database you're connected to.
- **collection_name:** The name of the collection where you want to insert documents.
- **insertOne():** Inserts a single document.
- **document_data:** The JavaScript object representing the document to be inserted.
- **insertMany():** Inserts multiple documents at once. Pass an array containing the documents you want to insert.

Example 1: Adding a new data to a collection student1

```
db> const studentData={
... "name":"Varshini kakade",
... "age":20,
... "courses":["mathematics","English"],
... "gpa":4.2,
... "home_city":"Banglore",
... "blood_group":"O+",
... "is_hotel_resident":false
... };db.student1.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('666714129ef56e86d4cdcdf6')
}
```

2.After adding the data total collection is 502

```
db> db.student1.find().count();
502
db>
```

2. Query (Find):

Retrieves documents from a collection based on specific criteria.

Syntax:

db.collection_name.find({ query_criteria }, { projection_document })

Example 1: Find documents matching a specific condition

db.collection_name.find({ field_name: value })

```
db> db.student1.find({home_city:"Banglore"});
[
  {
    _id: ObjectId('666714129ef56e86d4cdcdf6'),
    name: 'Varshini kakade',
    age: 20,
    courses: [ 'mathematics', 'English' ],
    gpa: 5,
    home_city: 'Banglore',
    blood_group: 'O+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666714219ef56e86d4cdcdf7'),
    name: 'Varshini kakade',
    age: 20,
    courses: [ 'mathematics', 'English' ],
    gpa: 4.2,
    home_city: 'Banglore',
    blood_group: 'O+',
    is_hotel_resident: false
  }
]
db> db.student1.find({name:"Varshini kakade"},{name:1});
[
  {
    _id: ObjectId('666714129ef56e86d4cdcdf6'),
    name: 'Varshini kakade'
  },
  {
    _id: ObjectId('666714219ef56e86d4cdcdf7'),
    name: 'Varshini kakade'
  }
]
```

- **find():** The method used to query the collection.
- **query_criteria:** An optional JavaScript object that specifies the conditions documents must meet to be returned. If omitted, all documents are returned.
- **projection_document:** An optional JavaScript object that determines which fields to include or exclude from the returned documents (covered in Projection).

3. Update:

Modifies existing documents in a collection.

Syntax:

db.collection_name.updateOne({ update_criteria }, { update_document })

db.collection_name.updateMany({ update_criteria }, { update_document })

db.collection_name.replaceOne({ replace_criteria }, { replacement_document })

- **updateOne():** Updates a single document that matches the update_criteria.
- **update_criteria:** An object specifying the conditions for selecting documents to be updated.
- **update_document:** An object containing the modifications to be applied to the matched documents.
- **updateMany():** Updates multiple documents that match the update_criteria. Similar to updateOne().
- replaceOne(): Replaces an entire document that matches the `replace_criteria` with the provided `replacement_document`.

Example 1:

Update a single document by matching a specific criterion

```
db> db.student1.updateOne({name:"Varshini kakade"},{$set:{gpa:5.0}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Example 02: Update multiple documents matching a criteria

db.collectionname.updateMany({ condition: value },{ field_name: new_value }})

```
db> db.student1.updateMany({gpa:{$lt:3.0}},{$inc:{gpa:0.5}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 261,
  modifiedCount: 261,
  upsertedCount: 0
}
db>
```

4. Delete:

Removes documents from a collection.

Syntax:

db.collection_name.deleteOne({ delete_criteria })

db.collection_name.deleteMany({ delete_criteria })

- **deleteOne():** Deletes the first document that matches the `delete_criteria`.
- delete_criteria: An object specifying the conditions for selecting documents to be deleted.
- **deleteMany():** Deletes all documents that match the `delete_criteria`.

Example 01:

Delete a single document by matching a specific criterion.

db.collection_name.deleteOne({ name: document_id })

```
db> db.student1.deleteOne({name:"Varshini kakade"});
{ acknowledged: true, deletedCount: 1 }
db> db.student1.find().count();
501
db>
```

Example 02:

Delete multiple documents matching a criteria.

db.collection_name.deleteMany({ condition: value })

```
db> db.student1.deleteMany({is_hotel_resident:false});
{ acknowledged: true, deletedCount: 255 }
```

5. Projection

Specifies which fields to include or exclude when retrieving documents using the find() method.

Syntax:

db.collection_name.find({ ... }, { field1: 1, field2: 0, ... })

- Use a projection document as the second argument to find().
- Set a field to `1` to include it in the results. Set it to `0` to exclude it.
- You can also use `_id: 0` to exclude the `_id` field by default.

Example 01:

Return only the name and gpa by set to 1 without any condition.

db.collection_name.find({})

db.collection_name.find({}, { projection: { field1: 1}})

```
db> db.student1.find({},{name:1,gpa:1});
[
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e0'),
    name: 'Student 948',
    gpa: 3.44
  },
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e1'),
    name: 'Student 157',
    gpa: 2.77
  },
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e2'),
    name: 'Student 316',
    gpa: 2.82
  },
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e3'),
    name: 'Student 346',
    gpa: 3.31
  },
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e4'),
    name: 'Student 930',
    gpa: 3.63
  },
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e5'),
    name: 'Student 305',
    gpa: 3.4
  },
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e8'),
    name: 'Student 440',
    gpa: 2.56
  },
```

Example 02:

Find the students who have gpa less than 3.0 and return only gpa so it is set to 1.

db.collection name.find({condition},{projection})

```
db> db.student1.find({gpa:{$lt:3.0}},{gpa:1});
[
  { _id: ObjectId('66645d86df4bc5f5cdf985e1'), gpa: 2.77 },
  { _id: ObjectId('66645d86df4bc5f5cdf985e2'), gpa: 2.82 },
  { _id: ObjectId('66645d86df4bc5f5cdf985e7'), gpa: 2.75 },
  { _id: ObjectId('66645d86df4bc5f5cdf985e8'), gpa: 2.56 },
  { _id: ObjectId('66645d86df4bc5f5cdf985ed'), gpa: 2.6 },
  { _id: ObjectId('66645d86df4bc5f5cdf985ee'), gpa: 2.89 },
  { _id: ObjectId('66645d86df4bc5f5cdf985ef'), gpa: 2.75 },
  { _id: ObjectId('66645d86df4bc5f5cdf985f1'), gpa: 2.96 },
  { _id: ObjectId('66645d86df4bc5f5cdf985f6'), gpa: 2.54 },
  { _id: ObjectId('66645d86df4bc5f5cdf985f8'), gpa: 2.92 },
  { _id: ObjectId('66645d86df4bc5f5cdf98605'), gpa: 2.9 },
  { _id: ObjectId('66645d86df4bc5f5cdf98606'), gpa: 2.82 },
  { _id: ObjectId('66645d86df4bc5f5cdf98609'), gpa: 2.69 },
  { _id: ObjectId('66645d86df4bc5f5cdf9860c'), gpa: 2.67 },
  { _id: ObjectId('66645d86df4bc5f5cdf98616'), gpa: 2.53 },
  { _id: ObjectId('66645d86df4bc5f5cdf98619'), gpa: 2.87 },
  { _id: ObjectId('66645d86df4bc5f5cdf98621'), gpa: 2.56 },
  { _id: ObjectId('66645d86df4bc5f5cdf98622'), gpa: 2.9 },
  { _id: ObjectId('66645d86df4bc5f5cdf98625'), gpa: 2.86 },
  { _id: ObjectId('66645d86df4bc5f5cdf98626'), gpa: 2.93 }
]
```

NOTES:

- ✓ MongoDB uses JSON-like syntax for specifying criteria and documents.
- ✓ You can use operators like `$eq` (equal to), `$gt` (greater than), `$in` (within a list), etc., to build complex queries and updates.

Example 03:

Return name and gpa so it is set to 1 and exclude id is set to 0.

db.collection name.find({},{projection :{ field1: 1, field2: 1, field3: 0 } })

```
db> db.student1.find({},{name:1,gpa:1,_id:0});
[
  { name: 'Student 948', gpa: 3.44 },
  { name: 'Student 157', gpa: 2.77 },
  { name: 'Student 316', gpa: 2.82 },
  { name: 'Student 346', gpa: 3.31 },
  { name: 'Student 930', gpa: 3.63 },
  { name: 'Student 305', gpa: 3.4 },
  { name: 'Student 440', gpa: 2.56 },
  { name: 'Student 256', gpa: 3.44 },
  { name: 'Student 177', gpa: 3.02 },
  { name: 'Student 487', gpa: 2.6 },
  { name: 'Student 213', gpa: 2.89 },
  { name: 'Student 690', gpa: 2.75 },
  { name: 'Student 647', gpa: 3.43 },
  { name: 'Student 232', gpa: 3.04 },
  { name: 'Student 328', gpa: 3.42 },
  { name: 'Student 468', gpa: 3.97 },
  { name: 'Student 504', gpa: 2.92 },
  { name: 'Student 915', gpa: 3.37 },
  { name: 'Student 367', gpa: 3.11 },
  { name: 'Student 969', gpa: 3.71 }
]
```

Example 04:

Find the students who have home city - city 2.Include home city and gpa ,exclude _id

```
db> db.student1.find({home_city:"City 2"},{home_city:1,gpa:1,_id:0});
[
  { gpa: 3.44, home_city: 'City 2' },
  { gpa: 3.42, home_city: 'City 2' },
  { gpa: 2.92, home_city: 'City 2' },
  { gpa: 2.69, home_city: 'City 2' },
  { gpa: 3.59, home_city: 'City 2' },
  { gpa: 3.4, home_city: 'City 2' },
  { gpa: 3.18, home_city: 'City 2' },
  { gpa: 2.77, home_city: 'City 2' },
  { gpa: 3.68, home_city: 'City 2' },
  { gpa: 3.77, home_city: 'City 2' },
  { gpa: 3.37, home_city: 'City 2' },
  { gpa: 3.36, home_city: 'City 2' },
  { gpa: 3.04, home_city: 'City 2' },
  { gpa: 3.53, home_city: 'City 2' },
  { gpa: 3.42, home_city: 'City 2' },
  { gpa: 2.66, home_city: 'City 2' },
  { gpa: 3.77, home_city: 'City 2' },
  { gpa: 3.21, home_city: 'City 2' },
  { gpa: 3.31, home_city: 'City 2' },
  { gpa: 3.36, home_city: 'City 2' }
]
```

Example 05:

Find all students who have hotel resident true without any projection.

db.collection name.find({condition:value})

```
db> db.student1.find({is_hotel_resident:true});
[
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e0'),
    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('66645d86df4bc5f5cdf985e1'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.77,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66645d86df4bc5f5cdf985e2'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.82,
    blood_group: 'B+',
    is_hotel_resident: true
  },
```