MongoDB

WHERE, AND ,OR and CRUD Operations:

WHERE:

Given a collection you want to filter a subset based on a condition. That is the place “where” is used .

Example : // Find all students with GPA greater than 3.5

db. Students.find({gpa:{$gt : 3.5 }});

// Find all students from “city 3”

db.students.find({home _city :”city 3”});

AND :

Absolutely, the $and operator in MongoDB allows you to perform logical AND operations on multiple conditions within your queries. Here's how it works with an example:

Functionality:

The $and operator takes an array of expressions as input. A document is selected only if **all** the expressions in the array evaluate to `true`. MongoDB utilizes short-circuit evaluation, meaning it stops evaluating subsequent expressions once the first one evaluates to `false`. This can improve query performance.

Example:

Imagine you have a collection named "products" with documents containing information about various items. You want to find all products that are:

- In stock (quantity greater than 0)
- Priced below $50

CODE:

db.products.find({ $and: [{ quantity: { $gt: 0 } }, // Quantity greater than 0 { price: { $lt: 50 } } // Price less than 50] });

OR:

In MongoDB, the $or operator is used to perform logical OR operations on multiple conditions within your queries. Here's a breakdown of its functionality and examples:

Functionality:

- The $or operator takes an array of one or more expressions as input.
- A document is selected if **at least one** of the expressions in the array evaluates to `true`.
- MongoDB evaluates the expressions one by one. If it finds a true expression, it stops evaluating further expressions and returns the document.

Benefits:

- $or is useful for retrieving documents that meet one or more criteria.
- It allows for flexibility in your queries.

Example:

Consider a collection named "customers" with documents containing customer information. You want to find all customers who are:

- Located in "New York" or "California" (for city field)
- Have a loyalty program membership (with a "loyaltyMember" field set to true)

Here's the query using $or:

CODE:

db.customers.find({ $or: [{ city: "New York" }, // City is New York { city: "California" }, // City is California { loyaltyMember: true } // Loyalty member] });

MongoDB CRUD Operations:

CRUD stands for Create, Read, Update, and Delete, which represent the fundamental actions you can perform on documents in a MongoDB collection.

1. Create:

- This operation refers to inserting new documents into a collection. MongoDB offers two primary methods for creating documents:
    - **insertOne()**: This method inserts a single document into a specified collection.
    - EX: db.collection("products").insertOne({ name: "T-Shirt", price: 25.99, category: "Clothing" });

insertMany(): This method allows you to insert an array of documents into a collection at once.
EX: db.collection("customers").insertMany([{ name: "Alice", email: "alice@email.com" }, { name: "Bob", email: "bob@email.com" }]);

2. Read:

- This operation involves retrieving existing documents from a collection. You can use various find methods in MongoDB to retrieve data. Here's a basic example:

EX: db.collection("orders").find({ status: "shipped" });

- This query finds all documents in the "orders" collection where the "status" field is equal to "shipped".

3. Update:

- This operation allows you to modify existing documents within a collection. MongoDB provides methods to target specific documents based on criteria and update their fields.
    - **updateOne()**: Updates the first document matching a specified filter.

EX:
db.collection("users").updateOne({ username: "john" }, { $set: { age: 30 } });

This updates the "age" field to 30 for the first user with the username "john".

- **updateMany()**: Updates all documents matching a specified filter.

- EX:
  db.collection("products").updateMany({ stock: 0 }, { $inc: { stock: 10 } });

- This increases the "stock" value by 10 for all products where "stock" is currently 0.

4. Delete:

- This operation removes documents from a collection. Similar to updates, there are methods for deleting documents:
  - **deleteOne()**: Deletes the first document matching a specified filter.
  - EX: db.collection("posts").deleteOne({ author: "anonymous" });

  This deletes the first post where the "author" field is "anonymous".
- **deleteMany()**: Deletes all documents matching a specified filter.
- EX:
  db.collection("cart").deleteMany({});
- This removes all items from the "cart" collection (empty filter targets all documents).

These are fundamental CRUD operations, but they offer more functionalities. You can use operators for complex filtering, sorting, and updates within your queries

Other Examples:

CREATE:

InsertOne():

Db.collection("products").insertOne({ name: "T-Shirt", price: 25.99, category: "Clothing" });

InsertMany():

db.collection("customers").insertMany([{ name: "Alice", email: "alice@email.com" }, { name: "Bob", email: "bob@email.com" }]);

READ:

```
db.collection("orders").find({ status: "shipped" });
```

UPDATE:

UpdateOne():

```
db.collection("users").updateOne({ username: "john" },
{ $set: { age: 30 } });
```

UpdateMany():

```
db.collection("products").updateMany({ stock: 0 }, { $inc:
{ stock: 10 } });
```

DELETE:

DeleteOne():

```
db.collection("posts").deleteOne({ author: "anonymous" });
```

DeleteMany():

```
db.collection("cart").deleteMany({});
```