**Session 15: MongoDB Queries**

**o Create Operations**

**Create a database in MongoDB**

Create a new database run any command against a non-existing database, and MongoDB will automatically create it for you.

Use: **insertOne()** or **insertMany()** methods. If database is not available then first it will create the database.

**insertOne():** The insertOne() method is used to insert a new document into a collection.

*[Note:* ***cls command will be used for clear the screen.]***

**To show whether data base is created or not use**

**Show dbs** command.

**To switch between the database, use the command:**

**Use database\_name;**

**Creating a New Collection:**

**Syntax:** db.createCollection(name)

**Example:** db.createCollection("users")

**Topic2.Creating Records: Create new documents to a collection in MongoDB:**

**1. insertOne():**Adding a single document to a collection is done using this method. A document to be added is the only argument it accepts, and it returns a result object with details about the insertion.

**Syntax:** **db.collectionName.insertOne()**

**Example:** db.users.insertOne({ name: "Angela",age: 27,});

**2. insertMany()**

This method is used to insert multiple documents into a collection at once. It takes an array of documents as its argument and returns a result object that contains information about the insertion.

**Syntax:** db.collectionName.insertMany([ ]);

**Example:** db.users.insertMany([

{

name: "Angela",

age: 27,

},

{

name: "Dwight",

age: 30,

},

{

name: "Jim",

age: 29,

}

]);

**B.[Read Operations]**

1. **Read**: The process of retrieving or viewing data from your database. In MongoDB, this is done using the **find()** and **findOne()** methods.

**Syntax:** db.collectionName.find(query, projection)

**Query** - It specifies the selection criteria for the documents to be retrieved.

**Projection** - It specifies which fields to include or exclude in the result set.

Note: Both query and projection are optional.

* **to include (1) or exclude (0) the field in the result set.**

1. **Find() : db.students.find({"regNo":"3014"})**

**Example:** db.users.find()

**Example:** db.users.find({ age: { $gt: 29 } }, { name: 1, age: 1 })

1. **findOne():**The findOne() method is used to retrieve a single document from a collection.

**Syntax:** db.collectionName.findOne()

let student = db.collection('students').findOne({ name: 'Logged In Student' });

console.log(student);

**Example:** db.users.findOne({ name: "Jim" })

**o Update Operations:**

Update" operation is used to modify existing documents in a collection. We have three **updateOne(), updateMany(), or replaceOne()** methods.

1. **updateOne()**

The updateOne() method is used to update a single document that matches a specified filter.

**Syntax:** db.collectionName.updateOne(filter, update, options)

a.**Update**: an optional Boolean that specifies whether to insert a new document if no document matches the filter.

* + - * If Update is set to true and no document matches the filter, a new document will be inserted. The default value of Update is false.

**Example:** db.users.updateOne({ name: "Angela" },

{ $set: { email: "angela@gmail.com" } })

**Available operations:**

* **$set:** Sets the value of a field in a document. If the field does not exist, the set will create it.
* **$unset:** Removes a field from a document.
* **$inc:** Increments the value of a field in a document by a specified amount.

db.products.insertOne(

{

\_id: 1,

Product: "Car",

quantity: 10,

metrics: { orders: 3, ratings: 4.5 }

}

)

* **increase the "metrics.orders" field by 1**
* **increase the quantity field by -2 (which decreases quantity)**

db.products.updateOne(

{ **Product**: "Car" },

{ $inc: { quantity: -3, "metrics.orders": 1 } }

)

**db.products.find() // output**

{

\_id: 1,

Product: 'Car',

quantity: 7,

metrics: {

orders: 4,

ratings: 4.5

}

}

* **$push:** Adds an element to the end of an array field in a document. If the field does not exist, push will create it as an array with the specified element.

db.sunil2.updateOne({ name: "Angela" },

{ $push : { "Gender":"Male" } })

* **$pull:**Removes all occurrences of a specified value from an array field in a document.

db.fruit.insertMany( [

{

\_id: 1,

fruits: [ "apples", "pears", "oranges", "grapes", "bananas" ],

vegetables: [ "carrots", "celery", "squash", "carrots" ]

},

{

\_id: 2,

fruits: [ "plums", "kiwis", "oranges", "bananas", "apples" ],

vegetables: [ "broccoli", "zucchini", "carrots", "onions" ]

}

] )

**db.fruit.find()**

**B. updateMany:** The updateMany () method is used to update multiple documents that match a specified filter.

**Syntax:** db.collectionName.updateMany(filter, update, options)

**Example:** db.users.updateMany({ age: { $lt: 30 } }, { $set: { status: "active" } })

**Deleting Data**

**Delete**: Used to remove documents from a collection. We can delete documents using the **deleteOne( )** or **deleteMany( )** methods.

A. **deleteOne() :** The deleteOne() method is used to remove a single document that matches a specified filter.

**Syntax: db.collectionName.deleteOne(filter, options)**

* + - **filter:** Specifies deletion criteria using query operators. Specify an empty document { } to delete the first document returned in the collection

**Example:** db.users.deleteOne({ name: "Angela" })

**B. deleteMany()**:used to remove multiple documents that match a specified filter.

**Syntax: db.collectionName.deleteMany(filter, options)**

**Example:** db.users.deleteMany({ age: { $lt: 30 } })

db.collection('students').**deleteOne**({ name: 'Aniket' });

db.collection('students').**deleteMany**({ major: 'Undeclared' });

**3. drop() :**The drop() method is used to remove an entire collection.

**Syntax:** db.collectionName.drop()

**Example:** db.users.drop() // Output: true

**Note:** This operation is irreversible, and all data in the collection will be permanently deleted.

**Querying NoSQL Stores: There are few key queries in NOSQL-**

1. **greater than [ $gt ]**

**test** {

"Brand":"Benz"

"Max\_Speed":250

"Color":"Green"

}

db.test.find({Max\_speed: {$gt:100}}).pretty() // test is the database name

* **pretty()** method is used to configure the cursor to display results in an easy-to-read format.

1. **$eq – This operator is used to check 2 values and returns the data which is equal to the specified value.**

db.test.find({Max\_speed: {$eq:250}}}.pretty()

1. **$gte (greater than or equal to)**
2. **$lte (lesser than or equal to)**
3. **$lt(less than ),**
4. **$ne (Not equal)**
5. **($and) operator.**

db.test.find({$and:[{Max\_speed:{$lt:500}},{Brand: {$eq:"Benz"}}}.pretty()}]})