Practical 4 : MongoDB CRUD Operations

Create Database: Let us create a database name userdb.

use userdb;

Creating a New Collection

db.createCollection("users")

Create Operation: There are two ways to create new documents to a collection in MongoDB:

1. insertOne():

Syntax:

db.collectionName.insertOne()

Example:

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

2. insertMany()

Syntax:

db.collectionName.insertMany();

Example:

```
db.users.insertMany([
    {
        name: "Angela",
        age: 27,
    },
    {
        name: "Dwight",
        age: 30,
    },
    {
        name: "Jim",
        age: 29,
    }
]);
```

Read Operations: In MongoDB, read operations are used to retrieve data from the database.

1. find()

Syntax:

db.collectionName.find(query, projection)

Query - It specifies the selection criteria for the documents to be retrieved. It is an object that contains one or more key-value pairs, where each key represents a field in the document and the value represents the value to match for that field.

Projection - It specifies which fields to include or exclude in the result set. It is an object that contains one or more key-value pairs, where each key represents a field in the document and the value represents whether to include (1) or exclude (0) the field in the result set.

Example:

db.users.find()

retrieve all documents from the collection without applying any filters or projecting any specific fields.

Example:

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

This command will return all the documents in the "users" collection where the age is greater than 29, and only return the "name" and "age" fields.

2. findOne()

The findOne() method returns a single document object, or null if no document is found. You can pass a query object to this method to filter the results.

Syntax:

db.collectionName.findOne()

Example:

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

Update Operations

In MongoDB, the "update" operation is used to modify existing documents in a collection.

Methods:

There are several ways to perform an update operation, including the following:

1. updateOne()

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

Syntax:

db.collectionName.updateOne(filter, update, options)

Options include the following parameters:

Upsert is an optional boolean that specifies whether to insert a new document if no document matches the filter. If upsert is set to true and no document matches the filter, a new document will be inserted. The default value of upsert is false.

WriteConcern is an optional document that specifies the level of acknowledgement requested from MongoDB for write operations. If not specified, the default write concern will be used.

Example:

db.users.updateOne({ name: "Angela" }, { $set: { email: "angela@gmail.com" } })

2. 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" } })

This command will update the status of all documents in the "users" collection where the age is less than 30 to "active".

Delete Operations

In MongoDB, the "delete" operation is used to remove documents from a collection.

There are several ways to perform a delete operation, including the following:

1. 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" })

2. deleteMany()

The deleteMany() method is used to remove multiple documents that match a specified filter.

Syntax:

db.collectionName.deleteMany(filter, options)

Example:

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

3. drop()

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

Syntax:

db.collectionName.drop()

Example:

db.users.drop()

This command will remove the users collection.

Practical 5: MongoDB aggregation operations

The operations on each stage can be one of the following:

- $project – select fields for the output documents.
- $match – select documents to be processed.
- $limit – limit the number of documents to be passed to the next stage.
- $skip – skip a specified number of documents.
- $sort – sort documents.
- $group – group documents by a specified key.

The following shows the syntax for defining an aggregation pipeline:

db.collection.aggregate([{ $match:...},{$group:...},{$sort:...}]);

In this syntax:

- First, call the aggregate() method on the collection.
- Second, pass an array of documents, where each document describes a stage in the pipeline.

MongoDB aggregation example

First, switch to the coffeeshop database that stores the coffee sales:

– use coffeeshop

Second, insert documents into the sales collection:

– db.sales.insertMany([

{ "_id" : 1, "item" : "Americanos", "price" : 5, "size": "Short", "quantity" : 22, "date" : ISODate("2022-01-15T08:00:00Z") },

{ "_id" : 2, "item" : "Cappuccino", "price" : 6, "size": "Short","quantity" : 12, "date" : ISODate("2022-01-16T09:00:00Z") },

{ "_id" : 3, "item" : "Lattes", "price" : 15, "size": "Grande","quantity" : 25, "date" : ISODate("2022-01-16T09:05:00Z") },

{ "_id" : 4, "item" : "Mochas", "price" : 25,"size": "Tall", "quantity" : 11, "date" : ISODate("2022-02-17T08:00:00Z") },

{ "_id" : 5, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 12, "date" : ISODate("2022-02-18T21:06:00Z") },

{ "_id" : 6, "item" : "Cappuccino", "price" : 7, "size": "Tall","quantity" : 20, "date" : ISODate("2022-02-20T10:07:00Z") },

{ "_id" : 7, "item" : "Lattes", "price" : 25,"size": "Tall", "quantity" : 30, "date" : ISODate("2022-02-21T10:08:00Z") },

{ "_id" : 8, "item" : "Americanos", "price" : 10, "size": "Grande","quantity" : 21, "date" : ISODate("2022-02-22T14:09:00Z") },

{ "_id" : 9, "item" : "Cappuccino", "price" : 10, "size": "Grande","quantity" : 17, "date" : ISODate("2022-02-23T14:09:00Z") },

{ "_id" : 10, "item" : "Americanos", "price" : 8, "size": "Tall","quantity" : 15, "date" : ISODate("2022-02-25T14:09:00Z")}

]);

Third, use an aggregation pipeline to filter the sales by the Americanos, calculate the sum of quantity grouped by sizes, and sort the result document by the total quantity in descending order.

```
db.sales.aggregate([
    {
        $match: { item: "Americanos" }
    },
    {
        $group: {
            _id: "$size",
            totalQty: {$sum: "$quantity"}
        }
```

```
        },
        {
                $sort: { totalQty : -1}
        }
]);
```

```
[
  { _id: 'Grande', totalQty: 33 },
  { _id: 'Short', totalQty: 22 },
  { _id: 'Tall', totalQty: 15 }
]
```