Msc(Data Sci and AI)

Practical-3: CRUD USING Mangodb

Name:Aditya Kamble **Roll No. – L011**

Write-up: -

- NOSQL Vs SQL
- Types of NOSQL
- Document document-oriented NoSQL
- Mangodb
- Mangodb shell
- Basic commands of mangodb

1. Install — mongosh

   https://www.mongodb.com/try/download/compass

2. set environment variable 3. Install mangodb shell https://www.mongodb.com/try/download/shell

Crud Operations in Mongo DB

Overview

CRUD is an acronym for Create, Read, Update, and Delete. CRUD operations in MongoDB are used to manipulate data in databases.

CRUD In MongoDB

CREATE, READ, UPDATE, and DELETE actions are referred to as CRUD operations in MongoDB. These are the basic operations used to alter data in databases. In MongoDB, the term "CRUD operations" refers to the standard set of database operations used to work with collections of data.

Create: To add new documents to a collection, use the Create operation.

Read: Data from a collection is retrieved using the Read operation.

Update: The Update operation is used to edit existing documents in a collection.

Delete: A collection of documents can be deleted using the Delete procedure.

These four basic procedures can be used to accomplish a wide range of database operations in MongoDB. These activities, for instance, can be used to add new records, get data, edit records, and remove records.

Performing CRUD Operations in MongoDB

Before we start exploring the CRUD methods. Let's first set up the db and collection which we are going to use.

Creating a New Database:

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

Example:

Let us create a database name userdb.

```
test> use userdb
```

Output:

```
switched to db userdb
```

Creating a New Collection:

To create a new collection, you can use the "createCollection" method.

Syntax:

```
db.createCollection(name, options)
```

Example:

```
userdb> db.createCollection("users")
```

Output:

```
{ ok: 1 }
```

Create Operation

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

1. insertOne():

Adding a sin argument it accepts, and it returns a result object with details about the insertion.

Syntax:

```
db.collectio gle document to a collection is done using this method. A document to be added is the only anName.insertOne()
```

Example:

```
userdb> db.createCollection("users")

{ ok: 1 }

userdb> db.users.insertOne({

... name:"krishna",

... age:21,

... });
```

Output:

```
{

  acknowledged: true,

  insertedId: ObjectId('6780f4e6d7eb94f0fcdc2419')

}


}
```

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:

```
userdb> db.users.insertMany([

... {

... name:"KP",

... age:23,

... },

... {

... name:"aditya",

... age:22,

... },

... {

... name:"adi",

... age:26,

... }

... ]);
```

Output:

```
{

  acknowledged: true,

  insertedIds: {

    '0': ObjectId('6780f676d7eb94f0fcdc241a'),
```

```
    '1': ObjectId('6780f676d7eb94f0fcdc241b'),
    '2': ObjectId('6780f676d7eb94f0fcdc241c')
  }
}


}
```

Both the insertOne() and insertMany() methods return a result object that contains information about the insertion operation. The result object includes the number of documents inserted, the unique _id field value of each inserted document, and any write errors that occurred during the operation.

Read Operations

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

1. find()

The find() method is used to retrieve data from a collection. It returns a cursor that can be iterated to access all the documents that match the specified query criteria.

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.

Note: Both query and projection are optional.

Example:

```
userdb> db.users.find()
```

Output:

```
[
  {
    _id: ObjectId('6780f4e6d7eb94f0fcdc2419'),
    name: 'krishna',
    age: 21
  },
  { _id: ObjectId('6780f676d7eb94f0fcdc241a'), name: 'KP', age: 23 },
  {
    _id: ObjectId('6780f676d7eb94f0fcdc241b'),
    name: 'aditya',
    age: 22
  },
  { _id: ObjectId('6780f676d7eb94f0fcdc241c'), name: 'adi', age: 26 }
]
```

The above example will retrieve all documents from the collection without applying any filters or projecting any specific fields.

Example:

```
userdb> db.users.find({age:{$gt:22}},{name:1,age:1})
```

Output:

```
[ull
  { _id: ObjectId('6780f676d7eb94f0fcdc241a'), name: 'KP', age: 23 },
  { _id: ObjectId('6780f676d7eb94f0fcdc241c'), name: 'adi', age: 26 }
]
```

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:

```
userdb> db.users.findOne({ name: "adi" })
```

Output:

```
{ _id: ObjectId('6780f676d7eb94f0fcdc241c'), name: 'adi', age: 26 }
```

This returns the first document in the user's collection where the name field is "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:

```
userdb> db.users.updateOne({ name: "adi" }, { $set: { email: "angela@gmail.com" } })
```

Output

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

In the example, we have used the $set operation.

Following are a few of the many available operations:

This command will update the email of the document in the "users" collection where the name is "Angela" to 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:

```
userdb> db.users.updateMany({ age: { $lt: 30 } }, { $set: { status: "active" } })
```

Output:

```
{
  acknowledged: true,
  insertedId: null,
```

```
  matchedCount: 4,

  modifiedCount: 4,

  upsertedCount: 0

}
```

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.

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

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.

Options:

- **WriteConcern (Optional):** A document expressing the write concern. Omit to use the default write concern.
- **Collation (Optional):** Specifies the collation to use for the operation. Collation allows users to specify language-specific rules for string comparison, such as rules for
letter case and accent marks.
- **Hint (Optional):** A document or string that specifies the index to use to support the query predicate.

Example:

```
userdb> db.users.deleteOne({ name: "adi" })
```

Output:

```
{ acknowledged: true, deletedCount: 1 }
```

This command will remove the first document in the "users" collection where the name is "Angela".

2. deleteMany()

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

Syntax:

```
db.collectionName.deleteMany(filter, options)
```

Example:

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

Output:

```
{ acknowledged: true, deletedCount: 3 }
```

This command will remove all documents in the "users" collection where the age is less than
30.

3. drop()

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

Syntax:

```
db.collectionName.drop()
```

Example:

```
userdb> db.users.drop()
```

Output:

```
true
```

This command will remove the users collection.