NAME: MUSUNURU VARSHA

TRACK:FULL STACK

COLLEGE:Vignan’s Nirula Institute of Technology and Science for Women(20NN1A0590)

Phone: 7093688917

ASSIGNMENT-4

Creating a Database Using MongoDB and Mongosh

OBJECTIVE:

The objective of this assignment is to familiarize yourself with MongoDB and

its command-line interface, Mongosh, and to understand how to create, manage,

and query databases and collections in MongoDB.

STEPS:

1. Database Setup:

MongoDB organizes data into databases, each of which contains collections. To create a new MongoDB database, we use the 'use' command followed by the database name.

# Database Setup: Create a new MongoDB database called myDatabase.

- **Use myDatabase**

  Output: switched to db myDatabase

  This command switches to the 'myDatabase' database. If the database does not exist, MongoDB creates it automatically.

2. Collection Creation:

Collections in MongoDB are analogous to tables in relational databases. They store documents, which are JSON-like data structures. To create a new collection named 'users' within the 'myDatabase' database, we use the 'createCollection' command. For example:

# Collection Creation: Create a collection named users within the myDatabase database.

- **db.createCollection("users")**

  Output: **{ "ok" : 1 }**

  This command creates the 'users' collection within the 'myDatabase' database.

3. Document Insertion:

Documents are the basic unit of data storage in MongoDB. We can insert documents into a collection using the 'insertMany' command. Each document represents a user with fields such as name, email, and age. For example:

# Document Insertion: Insert at least three documents into the users collection, each representing a user with fields such as name, email, and age.

- **db.users.insertMany([
  { name: "John Doe", email: "john@example.com", age: 25 },
  { name: "Jane Smith", email: "jane@example.com", age: 35 },
  { name: "Alice Johnson", email: "alice@example.com", age: 40 }
  ])**

Output: **{ "acknowledged" : true,
insertedIds : { ‘-
‘0’:ObjectId("603df15e6f7d162d36f3c141"),
‘1’:ObjectId("603df15e6f7d162d36f3c142"),
‘2’:ObjectId("603df15e6f7d162d36f3c143")
}
}**

This command inserts three user documents into the 'users' collection.

4. Querying:

Querying allows us to retrieve data from MongoDB collections based on specific criteria. We can use the 'find' command to retrieve all users from the 'users' collection. For example:

```
# Querying: Write queries to retrieve:
# All users from the users collection.
```

- **db.users.find()**

Output:

```
{
_id : ObjectId("603df15e6f7d162d36f3c141"),
name : "John Doe",
email : "john@example.com",
age : 25
}
{
_id : ObjectId("603df15e6f7d162d36f3c142"),
 name : "Jane Smith",
 email : "jane@example.com",
 age: 35
}
{ _id : ObjectId("603df15e6f7d162d36f3c143"),
name : "Alice Johnson",
email : "alice@example.com",
age : 40
}
```

This command retrieves all documents from the 'users' collection.

write queries to retrieve users with an age greater than or equal to 30.

For example:

# Users with an age greater than or equal to 30.

- **db.users.find({ age: { $gte: 30 } })**

Output:

```
{
_id : ObjectId("603df15e6f7d162d36f3c142"),
 name : "Jane Smith",
 email : "jane@example.com",
 age: 35
}
{ _id : ObjectId("603df15e6f7d162d36f3c143"),
name : "Alice Johnson",
email : "alice@example.com",
age : 40
}
```

This command retrieves documents from the 'users' collection where the 'age' field is greater than or equal to 30.

5. Update Operation:

Updating allows us to modify existing documents in a collection. We can use the 'updateOne' command to update the age of a user with a specific email address. For example:

# Update Operation: Update the age of a user with a specific email address.

```
➢ db.users.updateOne(
  { email: "john@example.com" },
  { $set: { age: 28 } }
)
```

Output:

```
{
 acknowledged : true,
 matchedCount: 1,
 modifiedCount: 1
}
```

This command updates the age of the user with the email address 'john@example.com' to 28.

6. Deletion Operation:

Deletion allows us to remove documents from a collection. We can use the 'deleteOne' command to delete a user document based on a specific email address. For example:

# Deletion Operation: Delete a user document based on a specific email address.

- **db.users.deleteOne({ email: "alice@example.com" })**

Output:

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

This command deletes the user document with the email address 'alice@example.com'.

7.Index Creation:

Indexes in MongoDB improve query performance by allowing faster retrieval of data. We can create an index on the email field of the users collection using the 'createIndex' command. For example:

# Index Creation: Create an index on the email field of the users collection.

- **db.users.createIndex({ email: 1 }, {unique: true})**

Output: **email_1**

This command creates an index on the 'email' field of the 'users' collection.