Mongo db

Class-2:Add,update,delete

Commands:

In Mongo DB, commands are operations or instructions sent to the database to perform specific actions. These actions could be anything from querying data to managing databases, collections, or performing administrative tasks.

Commands are used in Mongo DB for various purposes such as:

1. **Data Manipulation**: Commands like `insert`, `update`, and `delete` are used to manipulate data in collections.
2. **Data Querying**: Commands like `find` are used to query data from collections.
3. **Index Management**: Commands like `create Index` and `drop Index` are used to manage indexes in collections for optimizing query performance.
4. **Database Administration**: Commands like `create User`, `drop User`, `create Database`, and `drop Database` are used for managing users, databases, and other administrative tasks.
5. **Replication and Sharding**: Mongo DB uses commands for managing replication and sharding configurations.
6. **Aggregation**: Commands like `aggregate` are used to perform aggregation operations on data.

Overall, commands are essential for interacting with Mongo DB databases, allowing developers and administrators to perform various tasks efficiently.

Which are the commands used:

Command	Expected Output	Notes
show dbs	admin 40.00 KiB config 72.00 KiB db 128.00 KiB local 40.00 KiB	All Databases are shown
use db	switched to db db	Connect and use db
show collections	Students	Show all tables
db.foo.insert({"bar" : "baz"})		Insert a record to collection. Create Collection if not exists

Show dbs command description:

The show dbs shell command corresponds to the list Databases admin command, which shows the size on disk for each database.

The size for sharded collections is the sum of one member for each shard, and does not include redundant storage usage across all members of the shard replica set.

Use db command description:

The MongoDB command interface provides access to all non CRUD database operations. Fetching server statistics, initializing a replica set, and running an aggregation pipeline or map-reduce job are all accomplished with commands.

Show collections command description:

For users with the required access, show collections lists the nonsystem collections for the database. For users without the required access, show collections lists only the collections for which the users has privileges.

- ## Documents,collections,database

Document:

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

we have a simple database called "bookstore" with a collection named "books". Each document in the "books" collection represents a book. Here are a couple of example documents:

```
json
Copy code
// Example document 1
{
  "_id": ObjectId("60c3c0d8b0e05e001f53d672"),
  "title": "To Kill a Mockingbird",
  "author": "Harper Lee",
  "genre": "Fiction",
  "year": 1960,
  "isbn": "9780061120084",
  "copies": 10,
  "publisher": {
    "name": "HarperCollins Publishers",
    "location": "New York"
  },
  "reviews": [
    {
      "user": "user123",
      "rating": 5,
      "comment": "A timeless classic!"
    },
    {
      "user": "user456",
      "rating": 4,
      "comment": "An important read."
    }
  ]
}
```

Collections:

MongoDB stores data records as documents (specifically BSON documents) which are gathered together in collections. A database stores one or more collections of documents. You can manage MongoDB databases and collections in the UI for deployments hosted in MongoDB Atlas.

Let's consider a simple scenario where we have an e-commerce database named "webstore" with multiple collections to store different types of data related to products, users, and orders. Here are a few example collections:

1. **Products Collection**: This collection stores information about the products available in the webstore.

   ```
   json
   Copy code
   // Example document 1
   {
     "_id":
   ObjectId("60c3c0d8b0e05e001f53d672"),
     "name": "Laptop",
     "brand": "Dell",
     "price": 800,
     "stock_quantity": 20,
     "category": "Electronics",
     "description": "High-performance
   laptop with the latest features."
   }
   ```

Users Collection: This collection stores information about the users registered on the webstore.

```
json
Copy code
// Example document 1
{
  "_id": ObjectId("60c3c0d8b0e05e001f53d674"),
  "username": "john_doe",
  "email": "john@example.com",
  "password": "hashed_password",
  "shipping_address": "123 Main St, Anytown,
USA",
  "phone": "555-123-4567"
}
```

Orders Collection: This collection stores information about the orders placed by users.

```
json
Copy code
// Example document 1
{
  "_id": ObjectId("60c3c0d8b0e05e001f53d676"),
  "user_id":
ObjectId("60c3c0d8b0e05e001f53d674"),
  "products": [
    { "product_id":
ObjectId("60c3c0d8b0e05e001f53d672"),
"quantity": 2 },
    { "product_id":
ObjectId("60c3c0d8b0e05e001f53d673"),
"quantity": 1 }
  ],
  "total_amount": 1625,
  "status": "Pending",
  "shipping_address": "123 Main St, Anytown,
USA",
  "order_date": ISODate("2024-06-11T12:00:00Z")
}
```

Database:

here's an example of how you might create a simple database in MongoDB and perform some basic operations using the MongoDB shell:

1. **Connect to MongoDB:** First, you need to connect to your MongoDB instance. If MongoDB is running locally on the default port (27017), you can simply open a terminal and type:

   ```
   Copy code
   ```

```
mongo
```

2. **Create a Database:** Let's create a new database called "mydatabase":

   ```
   perl
   Copy code
   use mydatabase
   ```

   If the database doesn't already exist, MongoDB will create it for you.

3. **Create a Collection and Insert Data:** Now, let's create a collection called "users" and insert some data into it:

   ```
   php
   Copy code
   db.users.insertOne({ name: "John", age: 30, city:
   "New York" })
   ```

   This command will insert a document into the "users" collection with the specified fields.

4. **Query Data:** You can query the data you just inserted using the `find` command:

   ```
   lua
   Copy code
   db.users.find()
   ```

   This will return all documents in the "users" collection.

5. **Update Data:** Let's update John's age to 31:

   ```
   css
   ```

```
Copy code
db.users.updateOne({ name: "John" }, { $set: {
age: 31 } })
```

This command updates the first document in the "users" collection where the name is "John".

6. **Delete Data:** Now, let's delete John's document:

```
css
Copy code
db.users.deleteOne({ name: "John" })
```

This command deletes the first document in the "users" collection where the name is "John".

That's a basic example of how you can work with databases in MongoDB using the MongoDB shell. Of course, in a real-world scenario, you would likely be interacting with MongoDB using a programming language like JavaScript (with the MongoDB Node.js driver), Python, or any other language that has a MongoDB driver available.

Datatype:

In MongoDB, data is stored in a format called BSON (Binary JSON), which is a binary-encoded serialization of JSON-like documents. BSON supports various data types similar to JSON but includes additional types to handle data more efficiently and to support additional use cases. Here are some common data types used in MongoDB:

1. **String**: Represents a sequence of UTF-8 characters. Strings are commonly used for storing textual data.

    ```
    json
    Copy code
    { "name": "John" }
    ```

2. **Integer**: Represents a 32-bit signed integer value.

    ```
    json
    Copy code
    { "age": 30 }
    ```

3. **Double**: Represents a 64-bit floating-point number.

    ```
    css
    Copy code
    { "height": 6.2 }
    ```

4. **Boolean**: Represents a boolean value, either true or false.

    ```
    json
    Copy code
    { "isStudent": true }
    ```

5. **Date**: Represents a date and time. Dates are stored as milliseconds since the Unix epoch.

    ```
    css
    Copy code
    { "createdAt": ISODate("2024-06-11T12:00:00Z") }
    ```

6. **Array**: Represents an ordered list of values. Arrays can contain values of different data types, including other arrays or documents.

```
json
Copy code
{ "hobbies": ["reading", "painting", "gardening"]
}
```

7. **Object (Embedded Document)**: Represents a nested document within a document. Objects can contain fields with different data types, including other objects or arrays.

```
css
Copy code
{
    "address": {
        "street": "123 Main St",
        "city": "New York",
        "zipcode": "10001"
    }
}
```

8. **Null**: Represents a null value.

```
json
Copy code
{ "middleName": null }
```

9. **ObjectId**: Represents a unique identifier for documents. ObjectId values are typically generated by MongoDB drivers.

```
css
Copy code
{ "_id": ObjectId("60c3c0d8b0e05e001f53d672") }
```

These are some of the common data types used in MongoDB. BSON also supports other data types such as Regular Expression, Binary Data, JavaScript Code, and Symbol, among others, but the ones listed above are the most frequently used.