MONGODB

CLASS 2: ADD , UPDATE , AND DELETE

Few Commands to test after connections

1.show dbs

Absolutely, here's how you can list databases in MongoDB using the `show dbs` command along with an example:

Connecting to MongoDB Shell:

1. Make sure you have MongoDB installed and running.
2. Open your terminal or command prompt.
3. Type `mongo` and press enter to connect to the MongoDB shell. This will connect to the local MongoDB instance by default (usually on port 27017).

Listing Databases:

1. Once connected, type `show dbs` and press enter.

Example:

```
mongo
show dbs
```

This will display a list of all databases available on the MongoDB server. The output might look something like this:

```
local   0.0GB
admin   0.0GB
config  0.0GB
<your_database_name>   0.0GB
... (other databases)
```

Explanation:

- `local`: This is a special internal database used by MongoDB itself.
- `admin`: This database stores administrative data for MongoDB.
- `config`: This database stores configuration data for a deployment with multiple MongoDB instances (replica set or sharded cluster).
- `<your_database_name>`: This represents the names of your custom databases where you store your application data.
- The size (e.g., 0.0GB) indicates the approximate disk space used by the database.

2.use db

The `use db` command in MongoDB is used to switch between databases. Here's how it works with an example:

Scenario:

Imagine you have two databases: `users` and `products`. You want to work with the `products` database.

Using `use db`:

1. Connect to the MongoDB shell using `mongo`.
2. By default, you'll be in the `admin` database. To switch to the `products` database, type:

```
use products
```

Example:

```
mongo
```

```
use products
```

Explanation:

- The `use products` command tells MongoDB to start using the `products` database for subsequent operations (inserting/finding documents etc.).

Verifying the Active Database:

- To confirm the currently selected database, you can use the `db` command without any arguments:

```
db
```

This will print the name of the active database. In our example, it should display `products`.

3.show collections

In MongoDB, you can list the collections within a specific database using the `show collections` command. Here's how it works with an example:

Steps:

1. **Switch to the target database:** Use the `use <database_name>` command to select the database that contains the collections you want to list. Replace `<database_name>` with the actual name of your database.
2. **List collections:** Once you're in the desired database, type `show collections` and press enter.

Example:

Let's say you have a database named `my_store` with collections for `customers` and `orders`. Here's how to list them:

```
mongo
use my_store
show collections
```

This might output something like:

```
customers  0.0GB
orders      0.0GB
```

Explanation:

- The command lists the collection names (`customers` and `orders`) along with their approximate sizes (0.0GB in this case)

4. db.foo.insert({"bar" : "baz"})

The code `db.foo.insert({"bar" : "baz"})` in MongoDB inserts a new document into a collection named "foo".

Example:

Assuming you're connected to MongoDB and have a database named "my_app" selected (using `use my_app`), this code would insert a new document into the "foo" collection within that database:

```
db.foo.insert({"bar" : "baz"})
```

This would create a document like this in the "foo" collection:

```
{
  "_id" : ObjectId("..."),  // MongoDB-
generated unique identifier for the
document
  "bar" : "baz"
}
```

5.db.foo.batchInsert([{"_id" : 0}, {"_id" : 1}, {"_id" : 2}])

The code `db.foo.batchInsert([{"_id" : 0}, {"_id" : 1}, {"_id" : 2}])` in MongoDB attempts to insert multiple documents into a collection using batch insertion, but there's a caveat.

There are two common approaches to achieve batch insertion in MongoDB:

1. Using `insert` with an Array:

The correct way to insert multiple documents at once is to use the `insert` method with an array of document objects:

```
db.foo.insert([
  {"_id" : 0}, {"_id" : 1}, {"_id" : 2}
])
```

This will insert three documents into the "foo" collection, each with the specified `_id` field.

2. Using `insertMany` (Preferred):

MongoDB provides a dedicated method called `insertMany` for bulk insertion. It's generally preferred over using `insert` with an array because `insertMany` offers more options and better performance for larger datasets:

```
db.foo.insertMany([
```

```
  {"_id" : 0}, {"_id" : 1}, {"_id" : 2}
])
```

This achieves the same result as the previous approach but leverages the optimized `insertMany` method.

6. db.foo.find()

The command `db.foo.find()` in MongoDB is used to search for documents in a collection named "foo" within the currently active database.

Example:

Assuming you have a collection named "foo" with various documents, this command would find all of them:

```
db.foo.find()
```

7. db.foo.remove()

The command `db.foo.remove()` in MongoDB removes documents from the collection named "foo" within the currently active database.

Using `remove` (Deprecated):

- This removes all documents from the "foo" collection by default.
- You can optionally pass a query document as the first argument to filter which documents are removed:

```
db.foo.remove({ "name": "Alice" })  // Removes documents where "name" is "Alice"
```

Using `deleteOne` (Recommended):

- This removes at most one document that matches the specified query document.
- It's generally preferred for targeted removal of a single matching document.

```
db.foo.deleteOne({ "name": "Alice" })  // Removes one document where "name" is "alice"
```

Using `deleteMany` (Recommended):

- This removes all documents that match the specified query document.
- It's ideal when you want to remove multiple documents based on certain criteria.

db.foo.deleteMany({ "age": { $gt: 30 } }) // Removes documents where "age" is greater than 30

Documents, Collections, Database

DOCUMENTS

At the heart of MongoDB is the document:

an ordered set of keys with associated values.

The representation of a document varies by programming language, but most languages have a data structure that is a natural fit, such as a map, hash, or dictionary.

{"greeting" : "Hello, world!"}

Common Operations with Documents:

1. *Insert*:

- Adding a new document to a collection.

- Example:

```
db.users.insertOne({
  "name": "Eve",
  "age": 28,
  "city": "Los Angeles"
})
```

2. *Find*:

- Querying documents in a collection.

Example:

```
db.users.find({ "city": "New York" })
```

3. *Update*:

- Modifying existing documents.

- Example:

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

4. *Delete*:

- Removing documents from a collection.

- Example:

```
db.users.deleteOne({ "name": "Bob" })
```

COLLECTIONS

Collections A collection is a group of documents.

If a document is the MongoDB analog of a row in a relational database, then a collection can be thought of as the analog to a table.

DATABASE

MongoDB groups collections into databases.

A single instance of MongoDB can host several databases, each grouping together zero or more collections.

A database has its own permissions, and each database is stored in separate files on disk.

A good rule of thumb is to store all data for a single application in the same database.

DATATYPE

Basically each document will be in JSON format which will be as follows. Where each attributes inside can be of multiple data types.

```
{
    "name" : "John Doe",
    "address" : {
        "street" : "123 Park Street",
        "city" : "Anytown",
        "state" : "NY"
    }
}
```