UNIT-III

1. Introduction to MongoDB

Ans: MongoDB is a type of NoSQL database that is designed specifically for web applications. It offers a high throughput, which means that it can process a large amount of data quickly. MongoDB uses a unique data model called BSON, which is similar to JSON but offers some additional features such as support for data types like dates and binary data. One of the key advantages of MongoDB is its easily scalable architecture, which allows developers to add more servers to handle an increasing amount of data. Unlike traditional relational databases, MongoDB does not use tables, rows, or columns to store data. Instead, it stores data in collections of documents, where each document is a self-contained piece of data that can be easily accessed and manipulated. This approach makes it easier to store complex, unstructured data such as social media posts, sensor data, or logs.

2. Key features of MongoDB.

Ans: Some of the key features of MongoDB include:

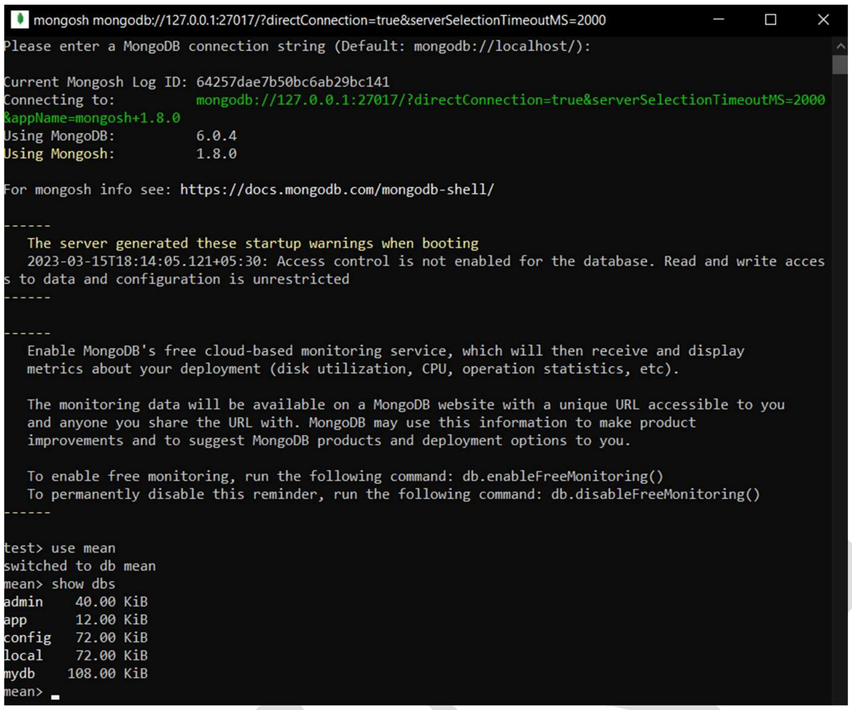
- **Document-Oriented:** MongoDB is a document-oriented database, which means that data is stored in JSON-like documents that can contain any number of fields and nested data structures. This makes it easy to work with complex data types, as you can store related data together in a single document.
- **High Performance:** MongoDB is designed to be highly performant, with built-in features like Sharding and replication to improve scalability and availability. Sharding is the process of distributing data across multiple servers in a cluster. By dividing the data into smaller chunks and distributing them across multiple shards, MongoDB can handle larger amounts of data and provide better performance. Replication is the process of synchronizing data across multiple servers to provide redundancy and high availability.
- **Flexible Data Model:** MongoDB's document model allows for flexible and dynamic schemas, which can be modified without downtime or disruption. This makes it easy to adapt to changing data needs. MongoDB uses a master-slave replication model
- **Indexing and Aggregation:** MongoDB supports a variety of indexing and aggregation features that can be used to optimize query performance.
- **Integration with Other Tools and Technologies:** MongoDB integrates easily with a wide range of tools and technologies, including programming languages like Python, Node.js, and Java, as well as data analytics and visualization tools like Tableau and Hadoop.
- **Open Source:** MongoDB is open source software, which means that the source code is freely available and can be modified and distributed by anyone. This makes it easy to customize and extend MongoDB to suit your specific needs.

3. MongoDB shell

Ans: The MongoDB shell is a command-line interface that allows you to interact with MongoDB servers. With the MongoDB shell, you can perform various administrative and data manipulation tasks such as querying data, inserting, updating, and deleting data, creating and managing indexes, and running MongoDB commands. The MongoDB shell is available for both Windows and Unix-based systems, and it uses a JavaScript-based syntax for interacting with the database. To start the MongoDB shell, you need to have MongoDB installed on your computer and the MongoDB server running. Once the server is running, open a terminal or command prompt and type **mongo** to start the shell. This will open a new interactive prompt where you can start executing commands.

4. MongoDB databases

Ans: In MongoDB, a server instance can store multiple databases. By default, the MongoDB shell connects to a database called **test**. You can switch to another database by using the **use** command followed by the database name. If the database does not exist, it will be created when you insert data into it. Alternatively, you can specify the database name as an argument when starting the MongoDB shell using the **mongo** command. To see a list of all databases on the MongoDB server, use the **show dbs** command. This will display a list of all available databases that have at least one document stored in them.



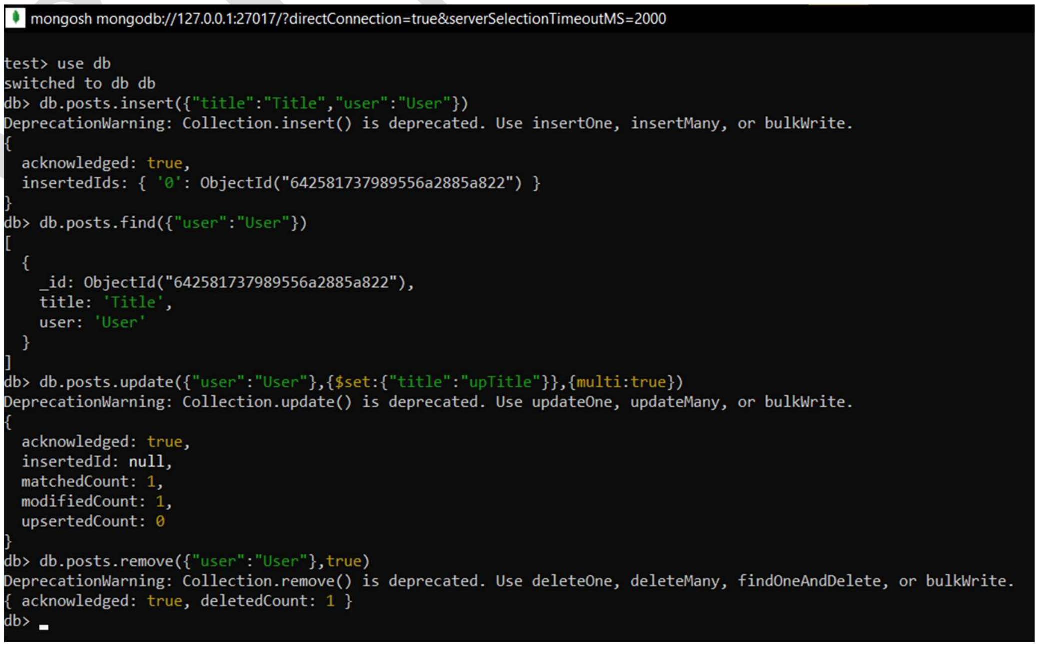
```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Please enter a MongoDB connection string (Default: mongodb://localhost/):

Current Mongosh Log ID: 64257dae7b50bc6ab29bc141
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
&appName=mongosh+1.8.0
Using MongoDB:          6.0.4
Using Mongosh:          1.8.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2023-03-15T18:14:05.121+05:30: Access control is not enabled for the database. Read and write acces
s to data and configuration is unrestricted
------

------
   Enable MongoDB's free cloud-based monitoring service, which will then receive and display
   metrics about your deployment (disk utilization, CPU, operation statistics, etc).

   The monitoring data will be available on a MongoDB website with a unique URL accessible to you
   and anyone you share the URL with. MongoDB may use this information to make product
   improvements and to suggest MongoDB products and deployment options to you.

   To enable free monitoring, run the following command: db.enableFreeMonitoring()
   To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
------

test> use mean
switched to db mean
mean> show dbs
admin    40.00 KiB
app      12.00 KiB
config   72.00 KiB
local    72.00 KiB
mydb    108.00 KiB
mean>
```

5. CRUD Operations

Ans: CRUD operations are the basic operations that are performed on a database or data storage system. They stand for Create, Read, Update, and Delete.

- **Create:** Creates a new record or document in the database.
- **Read:** Reads or retrieves one or more records or documents from the database.
- **Update**: Updates an existing record or document in the database.
- **Delete:** Deletes an existing record or document from the database.

Program:



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

test> use db
switched to db db
db> db.posts.insert({"title":"Title","user":"User"})
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("642581737989556a2885a822") }
}
db> db.posts.find({"user":"User"})
[
  {
    _id: ObjectId("642581737989556a2885a822"),
    title: 'Title',
    user: 'User'
  }
]
db> db.posts.update({"user":"User"},{$set:{"title":"upTitle"}},{multi:true})
DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
db> db.posts.remove({"user":"User"},true)
DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDelete, or bulkWrite.
{ acknowledged: true, deletedCount: 1 }
db>
```

6. Introduction to Mongoose

Ans: Mongoose is a tool for Node.js that allows developers to create and save objects in MongoDB. MongoDB is a type of database that doesn't require a specific structure, but Mongoose lets developers use both structured and unstructured approaches. To use Mongoose, you need to install it like any other module in Node.js.

7. Understanding Mongoose schemas

Ans: A Mongoose schema defines the structure of a document in a MongoDB collection, including the fields and their data types. In Mongoose, you define a schema using the Schema class provided by the Mongoose module i.e. you create a schema object in Mongoose and define the properties of your documents. Each property has a type and constraints that define what data can be stored in that property. Once you've defined your schema, you create a Model constructor that you can use to create, retrieve, and update documents that match the schema.

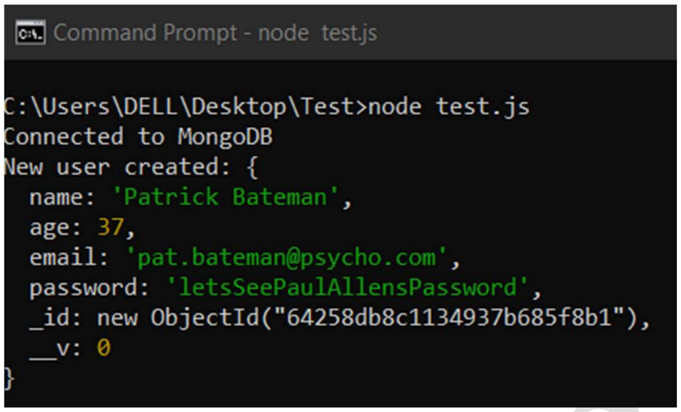
8. Creating the user schema and model

Ans:

Program:

```
// userModel.js
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/mydb', {
 useNewUrlParser: true,
 useUnifiedTopology: true
}).then(() => {
 console.log('Connected to MongoDB');
}).catch((err) => {
 console.log('Error connecting to MongoDB', err);
});
const userSchema = new mongoose.Schema({
   name: String,
   age: Number,
   email: String,
   password: String
});
module.exports = mongoose.model('User', userSchema);

//test.js
const User = require('./userModel');
const newUser = new User({
 name: 'Patrick Bateman',
 email: 'pat.bateman@psycho.com',
 password: 'letsSeePaulAllensPassword',
 age: 37
});
newUser.save()
 .then(result => {
 console.log('New user created:', result);
 })
 .catch(error => {
 console.error('Error creating user:', error);
 });
```

Output:



```
Command Prompt - node  test.js

C:\Users\DELL\Desktop\Test>node test.js
Connected to MongoDB
New user created: {
  name: 'Patrick Bateman',
  age: 37,
  email: 'pat.bateman@psycho.com',
  password: 'letsSeePaulAllensPassword',
  _id: new ObjectId("64258db8c1134937b685f8b1"),
  __v: 0
}
```

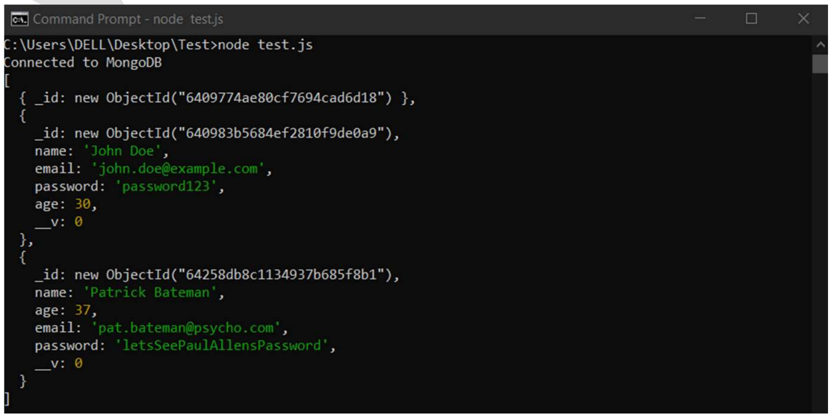
9. Registering the User model
Ans: Once you've defined the schema and created the model constructor, you need to register the model with Mongoose so that it knows how to interact with the MongoDB database. You can do this by calling the mongoose.model() method and passing in the name of the model and the schema object you created. **Refer 8.**

10. Creating new users using save()
Ans: To create a new user in MongoDB using Mongoose, you can use the save() method on a new instance of the User model. **Refer 8.**

11. Finding multiple user documents using find()
Ans: In Mongoose, you can use the find() method to search for multiple documents that match a given query criteria.
Program:
Refer userModel from 8.

```
//test.js
const User = require('./userModel');
User.find({})
  .exec()
  .then((users) => {
    console.log(users);
  })
  .catch((err) => {
    console.error(err);
  });
```

Output:

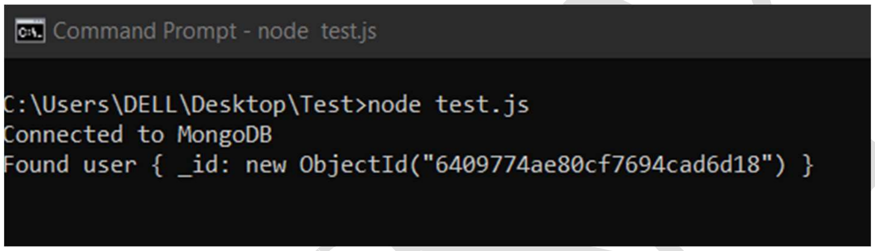


```
Command Prompt - node  test.js
C:\Users\DELL\Desktop\Test>node test.js
Connected to MongoDB
[
  { _id: new ObjectId("6409774ae80cf7694cad6d18") },
  {
    _id: new ObjectId("640983b5684ef2810f9de0a9"),
    name: 'John Doe',
    email: 'john.doe@example.com',
    password: 'password123',
    age: 30,
    __v: 0
  },
  {
    _id: new ObjectId("64258db8c1134937b685f8b1"),
    name: 'Patrick Bateman',
    age: 37,
    email: 'pat.bateman@psycho.com',
    password: 'letsSeePaulAllensPassword',
    __v: 0
  }
]
```

12. Reading a single user document using findOne()
Ans: In MongoDB, you can retrieve a single document from a collection using the findOne() method. Similarly, in Mongoose, you can use the findOne() method to retrieve a single document from a model.
Program:
Refer userModel from 8.

```
//test.js
const User = require('./userModel');
User.findOne({ _id: '6409774ae80cf7694cad6d18' })
 .then((user) => {
 console.log('Found user', user);
 })
 .catch((err) => {
 console.log('Error finding user', err);
 });
```

Output:

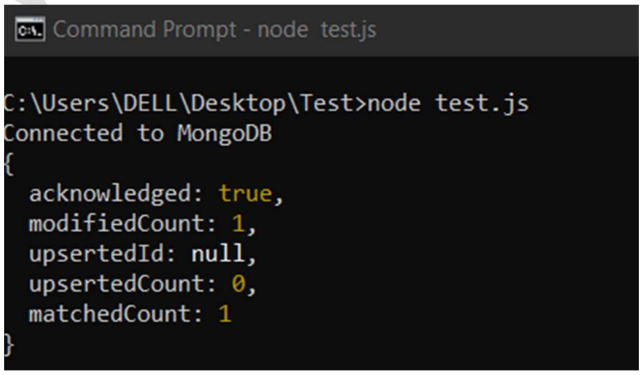


```
Command Prompt - node  test.js

C:\Users\DELL\Desktop\Test>node test.js
Connected to MongoDB
Found user { _id: new ObjectId("6409774ae80cf7694cad6d18") }
```

13. Updating an existing user document
Ans: The Mongoose model has several available methods to update an existing document. Among these are the update() , updateOne(), findOneAndUpdate() , and findByIdAndUpdate() methods.
Program:
Refer userModel from 8.

```
//test.js
 const User = require('./userModel');
 const filter = { email: 'john.doe@example.com' };
 const update = { $set: { age: 48 } };
 User.updateOne(filter, update)
   .then(result => {
     console.log(result);
   })
   .catch(err => {
     console.error(err);
   });
```

Output:

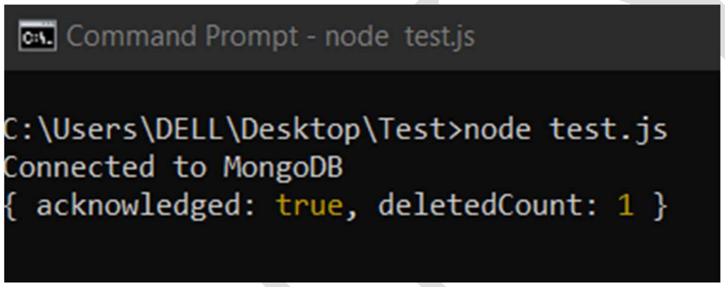


```
Command Prompt - node  test.js

C:\Users\DELL\Desktop\Test>node test.js
Connected to MongoDB
{
  acknowledged: true,
  modifiedCount: 1,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 1
}
```

14. Deleting an existing user document
Ans: The Mongoose model has several available methods for removing an existing document. Among these are the remove() , deleteOne() , findOneAndRemove() , and findByIdAndRemove() methods
Program:
Refer userModel from 8.

```
//test.js
 const User = require('./userModel');
 const filter = { email: 'john.doe@example.com' };
 User.deleteOne(filter)
   .then(result => {
     console.log(result);
   })
   .catch(err => {
     console.error(err);
   });
```

Output:



```
Command Prompt - node  test.js

C:\Users\DELL\Desktop\Test>node test.js
Connected to MongoDB
{ acknowledged: true, deletedCount: 1 }
```