Experiment No.3
To install and configure MongoDB to execute NoSQL commands
Date of Performance:
Date of Submission:

AIM: To install and configure MongoDB/ Cassandra/ HBase/ Hypertable and to execute NoSQL commands.

THEORY:

MongoDB can be downloaded from https://www.mongodb.com/try/download/community2

Now open command prompt and run the following command

```
C:\>move mongodb-win64-* mongodb
        1 dir(s) moved.
```

MongoDB requires a data folder to store its files. The default location for the MongoDB data directory is c:\data\db. So create the folder using the Command Prompt. Execute the following command sequence.

```
C:\>md data
C:\md data\db
```

In case mongodb is stored in some other location, navigate to that folder.

In command prompt navigate to the bin directory present into the mongodb installation folder. Suppose the installation folder is D:\set up\mongodb

```
C:\Users\XYZ>d:
D:\>cd "set up"
D:\set up>cd mongodb
D:\set up\mongodb>cd bin
D:\set up\mongodb\bin>mongod.exe --dbpath "d:\set up\mongodb\data"
```

Now to run the mongodb, open another command prompt and issue the following command:

```
D:\set up\mongodb\bin>mongo.exe
MongoDB shell version: 2.4.6
connecting to: test
>db.test.save( { a: 1 } )
>db.test.find()
{ "_id" : ObjectId(5879b0f65a56a454), "a" : 1 }
>
```

The use Command

MongoDB use DATABASE_NAME is used to create database. The command will create a new database, if it doesn't exist otherwise it will return the existing database

Syntax:

use DATABASE_NAME

The dropDatabase () Method

MongoDB db.dropDatabase () command is used to drop an existing database.

Syntax:

db.dropDatabase()

The createCollection() Method

MongoDB db.createCollection(name, options) is used to create collection.

Syntax:

db.createCollection(name, options)

Insert Document

To insert data into MongoDB collection, you need to use MongoDB's insert() or save()method

Syntax

>db.COLLECTION_NAME.insert(document)

Example:

```
>db.post.insert([
{
 title: 'MongoDB Overview',
 description: 'MongoDB is no sql database',
 tags: ['mongodb', 'database', 'NoSQL'],
 likes: 100
},
{
 title: 'NoSQL Database',
 description: 'NoSQL database doesn't have tables',
 tags: ['mongodb', 'database', 'NoSQL'],
 likes: 20,
 comments: [
 {
 user:'user1',
 message: 'My first comment',
 dateCreated: new Date(2022,11,10,2,35),
 like: 0
 }
 ]
}
])
```

Creating sample document:

Example

Suppose a client needs a database design for his blog website. Website has the following requirements.

 Every post has the unique title, description and url.

 Every post can have one or more tags.

 Every post has the name of its publisher and total number of likes.

 Every Post have comments given by users along with their name, message, data-time and likes.

 On each post there can be zero or more comments.

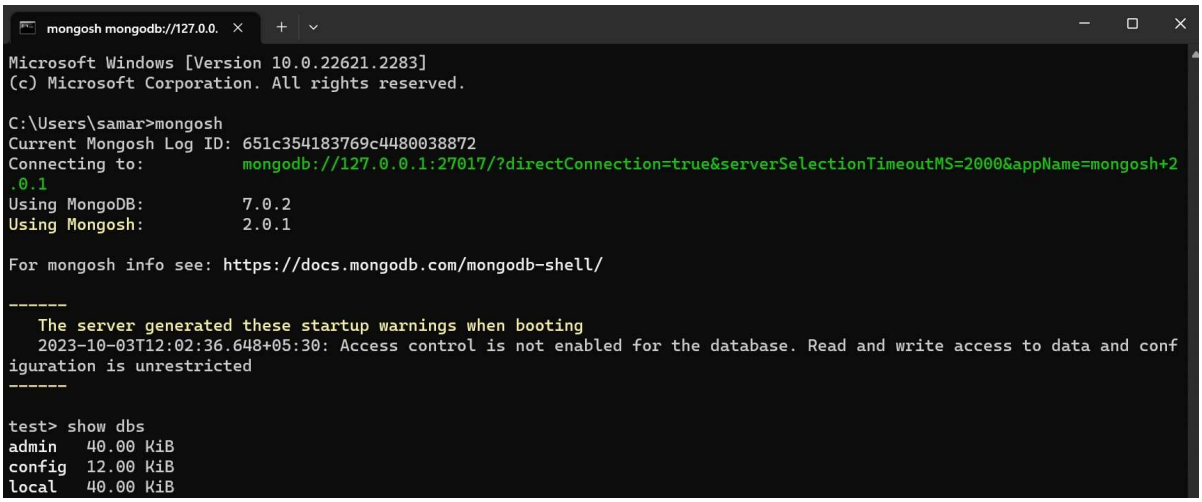
Document:

```
{
 _id: POST_ID
 title: TITLE_OF_POST,
 description: POST_DESCRIPTION,
 by: POST_BY,
 url: URL_OF_POST,
 tags: [TAG1, TAG2, TAG3],
 likes: TOTAL_LIKES,
 comments: [
 {
 user:'COMMENT_BY',
 message: TEXT,
 dateCreated: DATE_TIME,
 like: LIKES
 },
 {
 user:'COMMENT_BY',
 message: TEXT,
 dateCreated: DATE_TIME,
 like: LIKES
```

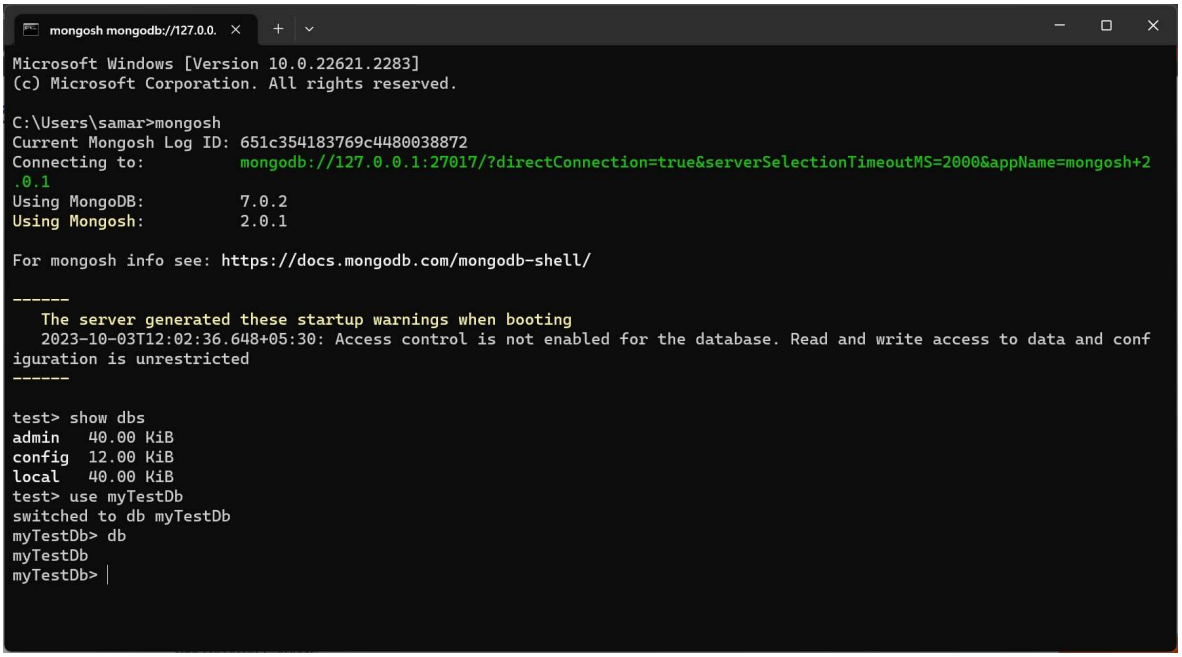
```
}
]
}
```

OUTPUT:

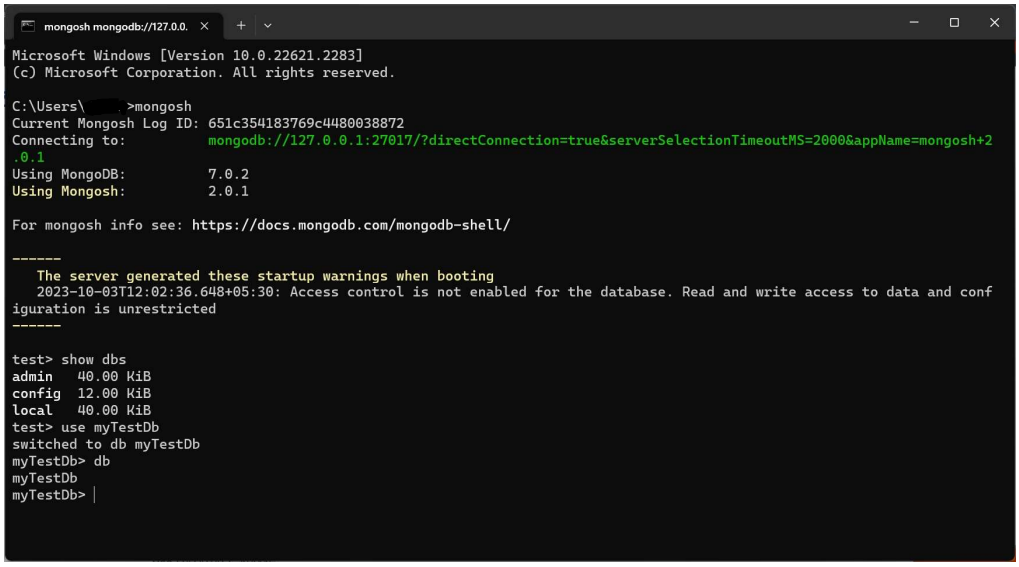
Show All Databases



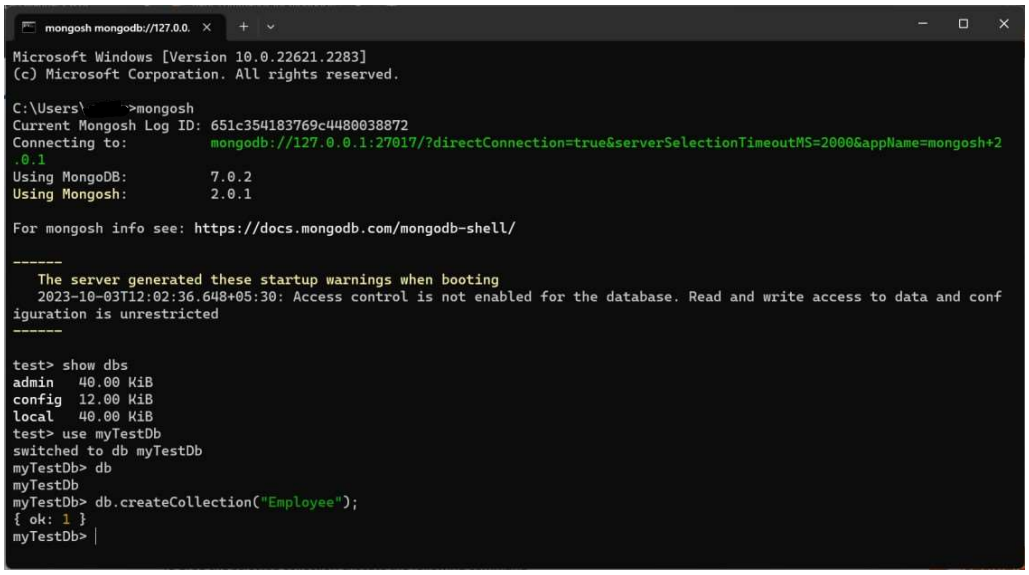
Create new database



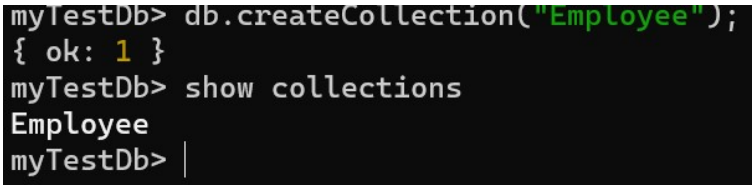
Know your current selected database



Create collection



To check collections list



Insert document in collection

```
myTestDb> db.Employee.insert({id:1 , name:'Samarth', address:'Pune'})
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrit
e.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("651c386283769c4480038873") }
}
myTestDb> db.Employee.insert({id:2 , name:'Shubham', address:'Ratnagiri'})
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("651c387883769c4480038874") }
}
myTestDb>
```

To insert multiple documents in selected collection

```
myTestDb> db.Employee.insert({id:3 , name:'Dharmesh', address:'Malvan'},{id:4, name:'Hrushike
sh',address:'kochi'})
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("651c394183769c4480038875") }
}
myTestDb>
```

Get collection document

```
myTestDb> db.Employee.find().pretty()
[
  {
    _id: ObjectId("651c386283769c4480038873"),
    id: 1,
    name: 'Samarth',
    address: 'Pune'
  },
  {
    _id: ObjectId("651c387883769c4480038874"),
    id: 2,
    name: 'Shubham',
    address: 'Ratnagiri'
  },
  {
    _id: ObjectId("651c394183769c4480038875"),
    id: 3,
    name: 'Dharmesh',
    address: 'Malvan'
  }
]
myTestDb>
```

Update document

```
myTestDb> db.Employee.update({name:'Dharmesh'},{$set:{name:'Hrushikesh'}})
DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
myTestDb> db.Employee.find().pretty()
[
  {
    _id: ObjectId("651c386283769c4480038873"),
    id: 1,
    name: 'Samarth',
    address: 'Pune'
  },
  {
    _id: ObjectId("651c387883769c4480038874"),
    id: 2,
    name: 'Shubham',
    address: 'Ratnagiri'
  },
  {
    _id: ObjectId("651c394183769c4480038875"),
    id: 3,
    name: 'Hrushikesh',
    address: 'Malvan'
  }
]
```

Drop collection

```
myTestDb> db.Employee.drop()
true
myTestDb>
```

Drop database

```
myTestDb> db.dropDatabase()
{ ok: 1, dropped: 'myTestDb' }
```

CONCLUSION:

Setting up and configuring MongoDB for NoSQL operations is a critical step in harnessing the power of this flexible database system.

By installing and configuring MongoDB, users can leverage its features to store, retrieve, and manipulate data in non-tabular, schema-less formats. This lays the foundation for efficient and scalable NoSQL database operations, offering businesses and developers a robust platform for data management and analysis.