Experiment No.3
To install and configure MongoDB to execute NoSQL commands
Date of Performance:09/08/2023
Date of Submission:16/08/2023

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,

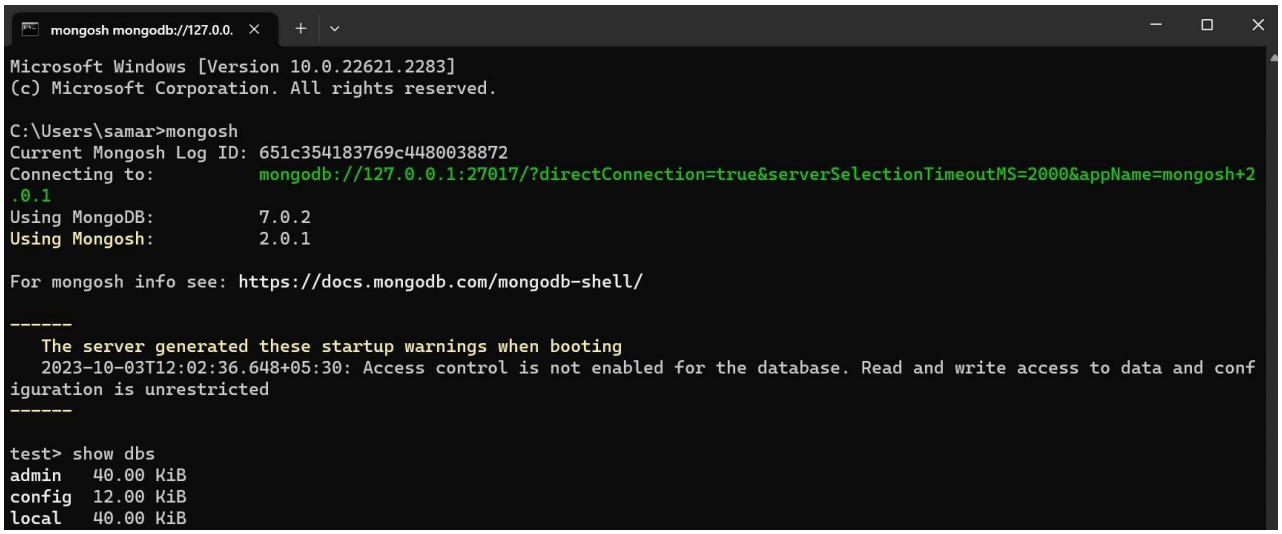
dateCreated: DATE_TIME,

like: LIKES

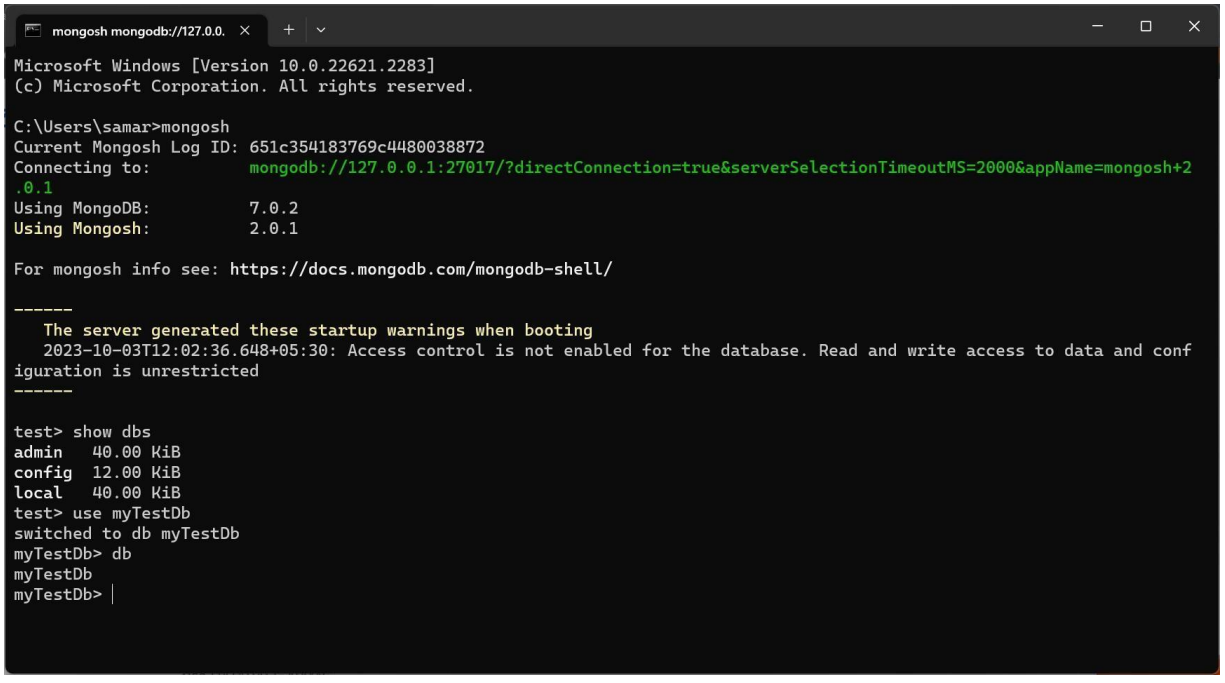
```
}
]
}
```

OUTPUT:

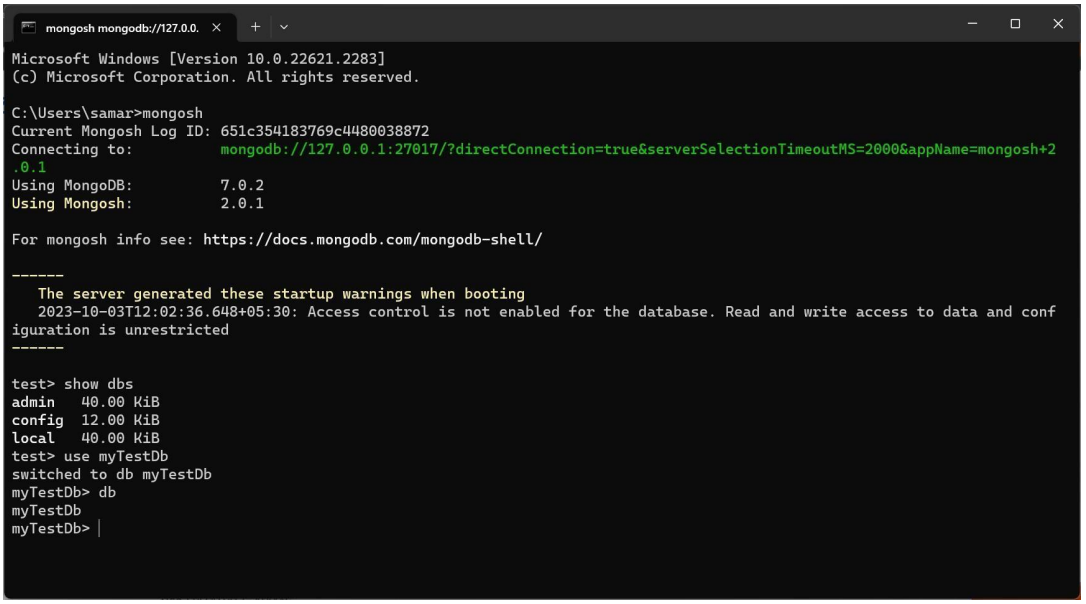
Show All Databases



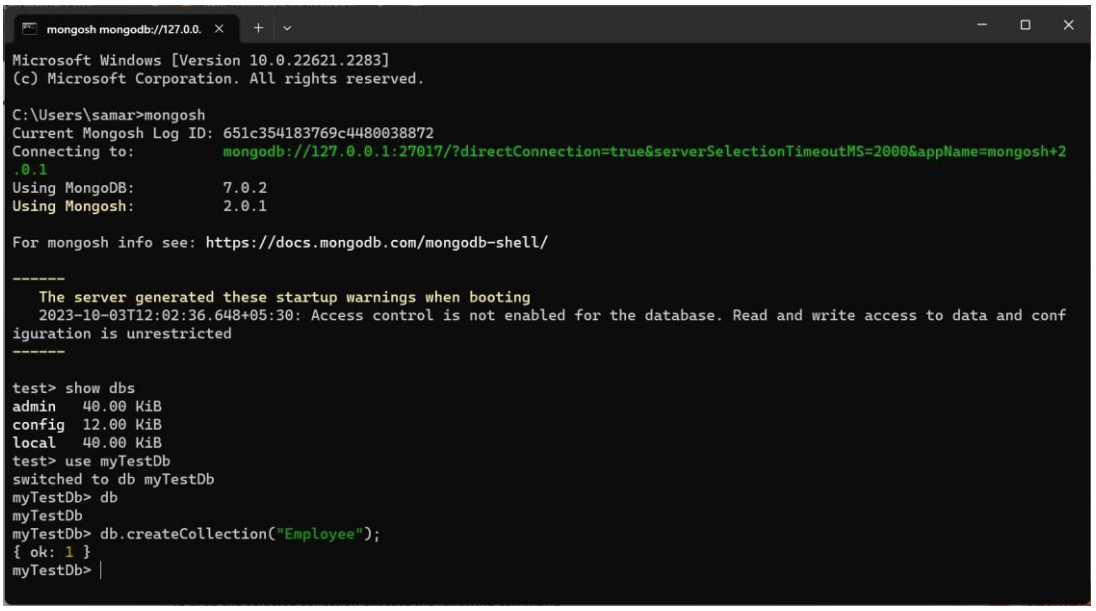
Create new database



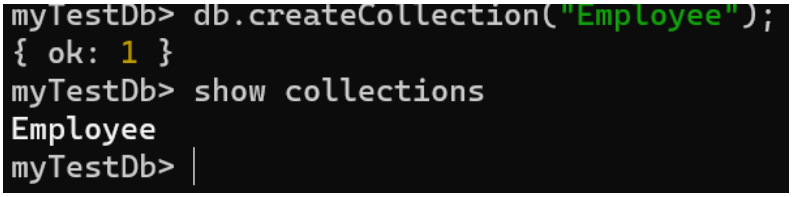
Know your current selected database



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
}
```

Drop database

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

CONCLUSION:

The experiment's objective was to set up MongoDB and tailor it to particular needs, incorporating security features and system settings. We gained proficiency in employing NoSQL commands for diverse database tasks, such as data input, searching, and indexing. MongoDB demonstrated its ability to handle unstructured data effectively, proving its suitability for NoSQL applications. The availability of comprehensive documentation and community assistance played a pivotal role in our successful experiment, equipping us with valuable expertise in proficiently managing NoSQL data through MongoDB