Assignment
No. 12

AIM: Create a database with suitable example using MongoDB and implement
- Inserting and saving document (batch insert, insertvalidation)
- Removingdocument
- Updating document (document replacement, using modifiers, upserts,updating multiple documents, returning updateddocuments)

THEORY:
MongoDB

MongoDB is an open-source document database and leading NoSQL database.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)

Advantages of MongoDB over RDBMS

- **Schema less** − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
- Structure of a single object isclear.
- No complexjoins.
- Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful asSQL.
- Tuning.
- **Ease of scale-out** − MongoDB is easy toscale.
- Conversion/mapping of application objects to database objects notneeded.
- Uses internal memory for storing the (windowed) working set, enabling fasteraccess ofdata.

Why Use MongoDB?

- **Document Oriented Storage** − Data is stored in the form of JSON styledocuments.
- Index on anyattribute
- Replication and highavailability
- Auto-sharding
- Richqueries
- Fast in-placeupdates
- Professional support by

MongoDBWheretoUseMongoDB?

- BigData
- Content Management and Delivery
- Mobile and SocialInfrastructure
- User DataManagement
- DataHub

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
```

Example

```
>use mydb
switched todbmydb
```

To check your currently selected database, use the command **db**

```
>db
mydb
```

If you want to check your databases list, use the command **show dbs**.

```
>show dbs
local     0.78125GB
test      0.23012GB
```

Your created database (mydb) is not present in list. To display database, you need
to insert at least one document into it.

The dropDatabase() Method

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

Syntax

```
db.dropDatabase()
```

This will delete the selected database. If you have not selected any database,
then it will delete default 'test' database.

Example

First, check the list of available databases by using the command, **show dbs**.

```
>show dbs
local0.78125GB
mydb0.23012GB
test0.23012GB
>
```

If you want to delete new database **<mydb>**, then **dropDatabase**()command
would be as follows −

```
>use mydb
switched todbmydb
>db.dropDatabase()
>{"dropped":"mydb","ok":1}
```

```
>show dbs
local0.78125GB
test0.23012GB
>
```

The createCollection() Method

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

Syntax

```
db.createCollection(name, options)
```

Examples

```
>use test
switched to db test
>db.createCollection("mycollection")
{"ok":1}
>
```

You can check the created collection by using the command **show collections**.

```
>show collections
mycollection
system.indexes
```

The drop() Method

MongoDB's **db.collection.drop()** is used to drop a collection from the

database. Syntax

Basic syntax of **drop()** command is as follows −

```
db.COLLECTION_NAME.drop()
```

The insert() Method

To   insert   data   into   MongoDB   collection,   you   need   to   use MongoDB's **ins**

Syntax

The basic syntax of **insert()** command is as

```
>db.COLLECTION_NAME.insert(document)
```

follows − Example

```
>db.mycol.insert({
  _id:ObjectId(7df78ad8902c),
title:'MongoDB Overview',
```

```
description:'MongoDB is no sql database',
by:'tutorials point',
url:'http://www.tutorialspoint.com',
tags:['mongodb','database','NoSQL'],
likes:100
})
```

Here **mycol** is our collection name, as created in the previous chapter. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.

In the inserted document, if we don't specify the _id parameter, then MongoDB assigns a unique ObjectId for this document.

_id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows −

```
_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id,
   3 bytes incrementer)
```

To insert multiple documents in a single query, you can pass an array of documents in insert() command.

Example

```
>db.post.insert([
{
title:'MongoDB Overview',
description:'MongoDB is no sql database',
by:'tutorials point',
url:'http://www.tutorialspoint.com',
tags:['mongodb','database','NoSQL'],
likes:100
},

{
title:'NoSQL Database',
description:"NoSQL database doesn't have tables",
by:'tutorials point',
url:'http://www.tutorialspoint.com',
tags:['mongodb','database','NoSQL'],
likes:20,
comments:[
{
user:'user1',
message:'My first comment',
dateCreated:newDate(2013,11,10,2,35),
like:0
}
```

```
]
}
])
```

The find() Method

To query data from MongoDB collection, you need to use MongoDB's
find()method. Syntax
The basic syntax of **find()** method is as follows −

```
>db.COLLECTION_NAME.find()
```

find() method will display all the documents in a non-
structured way. The pretty() Method
To display the results in a formatted way, you can use **pretty()**
method. Syntax

```
>db.mycol.find().pretty()
```

MongoDBUpdate() Method

The update() method updates the values in the existing document.

Syntax

The basic syntax of **update()** method is as follows −

```
>db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)
```

Example

Consider the mycol collection has the following data.

```
{"_id":ObjectId(5983548781331adf45ec5),"title":"MongoDB Overview"}

{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}

{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
```

Following example will set the new title 'New MongoDB Tutorial' of the
documents whose title is 'MongoDB Overview'.

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'NewMongoDB Tutorial'}})

>db.mycol.find()

{"_id":ObjectId(5983548781331adf45ec5),"title":"NewMongoDB Tutorial"}

{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}

{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}

>
```

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

```
>db.mycol.update({'title':'MongoDB Overview'},

{$set:{'title':'NewMongoDB Tutorial'}},{multi:true})
```

MongoDBSave() Method

The **save()** method replaces the existing document with the new document passed in the save() method.

Syntax

The basic syntax of MongoDB **save()** method is shown below −

```
>db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

The remove() Method

MongoDB's **remove()** method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

- **deletion criteria** − (Optional) deletion criteria according to documents will be removed.

- **justOne** − (Optional) if set to true or 1, then remove only onedocument.

Syntax

Basic syntax of **remove()** method is as follows −

```
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)
```

Example

Consider the mycol collection has the following data.

```
{"_id":ObjectId(5983548781331adf45ec5),"title":"MongoDB Overview"}

{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}

{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
```

Following example will remove all the documents whose title is 'MongoDB Overview'.

```
>db.mycol.remove({'title':'MongoDB Overview'})

>db.mycol.find()

{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}

{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
```

```
>
```

Remove Only One

If there are multiple records and you want to delete only the first record, then set **justOne** parameter in **remove**()method.

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Remove All Documents

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**

```
>db.mycol.remove()
>db.mycol.find()
>
```

Conclusion:

Created a Database in MongoDB and performed basic operations.

Output:

```
admin
(empty)
local
0.078GB
test0.078
GB
> use user
switched to db  user
> show
dbs;
admin
(empty)
local
0.078G
B
test0.0
78GB
> db.createCollection(" user")
{ "ok" : 1 }
> db. user.insert("_id":1,"rollno":1,"name":kirti)
2019-10-17T23:40:48.115+0530 SyntaxError: Unexpected token :
> db. user.insert("_id":1,"rollno":1,"name":"kirti")
2019-10-17T23:40:58.849+0530 SyntaxError: Unexpected token :
> db. user.insert({"_id":1,"rollno":1,"name":kirti})
2019-10-17T23:41:36.538+0530 ReferenceError: kirti is not defined
> db.
user.insert({"_id":1,"rollno":1,"name":"kirti"}
) WriteResult({ "nInserted" : 1})
> db. user.insert("_id":2,"rollno":2,"name":"aditi")
2019-10-17T23:43:08.786+0530 SyntaxError: Unexpected token :
> db.
user.insert({"_id":2,"rollno":2,"name":"aditi"}
) WriteResult({ "nInserted" : 1})
```

```
> db. user.find().pretty()
{ "_id" : 1, "roll no" : 1, "name" : "kirti" }
{ "_id" : 2, "roll no" : 2, "name" : "aditi" }
> show
dbs;
admin
(empty)
local
0.078GB
user
0.078GB
test
   0.078GB
> db. user.update({"roll no":1},{$set:{"name":"suraj"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0,
"nModified" : 1})
> db. user.find().pretty()
{ "_id" : 1, "roll no" : 1, "name" : "suraj" }
{ "_id" : 2, "roll no" : 2, "name" : "aditi" }
> db.
user.insert({"_id":3,"rollno":3,"name":"aditya"})
WriteResult({ "nInserted" : 1})
> db. user.find().pretty()
{ "_id" : 1, "roll no" : 1, "name" : "suraj" }
{ "_id" : 2, "roll no" : 2, "name" : "aditi" }
{ "_id" : 3, "roll no" : 3, "name" : "aditya" }
> db.
user.update({"_id":1},{$set:{"class":"TE","address":"pune"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" :
1})
> db. user.find().pretty()
{
    "_id" : 1,
    "roll no" :
    1, "name" :
    "suraj",
    "class" : "TE",
    "address" : "pune"
}
{ "_id" : 2, "roll no" : 2, "name" : "aditi" }
{ "_id" : 3, "roll no" : 3, "name" : "aditya" }
> db.
user.update({"_id":1},{$set:{"class":"SE","address":"pune"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" :
1})
> db. user.find().pretty()
{
    "_id" : 1,
    "roll no" :
    1, "name" :
    "suraj",
```

```
}
{
"
_
i
d
"
:
2
,
"
r
o
ll
n
o
"
:
2
```

```
, "name" : "aditi" }
{ "_id" : 3, "roll no" : 3, "name" : "aditya" }
> db. user.find()
{ "_id" : 1, "roll no" : 1, "name" : "suraj", "class" : "SE", "address" : "pune" }
{ "_id" : 2, "roll no" : 2, "name" : "aditi" }
{ "_id" : 3, "roll no" : 3, "name" : "aditya" }
> db.
user.remove({"_id":3})
WriteResult({
"nRemoved" : 1})
> db. user.find()
{ "_id" : 1, "roll no" : 1, "name" : "suraj", "class" : "SE", "address" : "pune" }
{ "_id" : 2, "roll no" : 2, "name" : "aditi" }


> db. user.remove({"name":"aditya"})
WriteResult({ "nRemoved" : 0})
> db. user.find()
{ "_id" : 1, "roll no" : 1, "name" : "suraj", "class" : "SE", "address" : "pune" }
{ "_id" : 2, "roll no" : 2, "name" : "aditi" }
> db. user.remove({"name":"aditi"})
WriteResult({ "nRemoved" : 1})
> db. user.find()
{ "_id" : 1, "roll no" : 1, "name" : "suraj", "class" : "SE", "address" : "pune" }
> db. user.insert({"_id":2,"rollno":3,"name":"aditya"})
WriteResult({ "nInserted" : 1})
> db. user.find()
{ "_id" : 1, "roll no" : 1, "name" : "suraj", "class" : "SE", "address" : "pune" }
{ "_id" : 2, "roll no" : 3, "name" : "aditya" }
> db. user.insert({"_id":2,"rollno":4,"name":"adi"})
WriteResult({
    "nInserted" : 0,
    "writeError" : {
            "code" : 11000,
            "errmsg" : "insertDocument :: caused by :: 11000 E11000 duplicate key error index:
user. user.$_id_ dup key: { : 2.0 }"
    }
})
> db.
user.insert({"_id":3,"rollno":3,"name":"adi"}
) WriteResult({ "nInserted" : 1})
> db. user.find()
{ "_id" : 1, "roll no" : 1, "name" : "suraj", "class" : "SE", "address" : "pune" }
{ "_id" : 2, "roll no" : 3, "name" : "aditya" }
{ "_id" : 3, "roll no" : 3, "name" : "adi" }
> db.
user.remove({"rollno":3}
) WriteResult({
"nRemoved" : 2})
> db. user.find()
{ "_id" : 1, "roll no" : 1, "name" : "suraj", "class" : "SE", "address" : "pune" }
```

```
> db.
user.insert({"_id":3,"rollno":3,"name":"adi"}
) WriteResult({ "nInserted" : 1})
> db.
user.insert({"_id":2,"rollno":4,"name":"adi"}
) WriteResult({ "nInserted" : 1})
> db.
user.remove({"name":"adi"}
) WriteResult({ "nRemoved"
: 2})
> db. user.find()
{ "_id" : 1, "roll no" : 1, "name" : "suraj", "class" : "SE", "address" : "pune" }
> db.
user.insert({"_id":1,"rollno":3,"name":"adi"}
) WriteResult({
    "nInsert
    ed" : 0,
    "writeE
    rror" : {
            "code" : 11000,
            "errmsg" : "insertDocument :: caused by :: 11000 E11000 duplicate key
error index:  user. user.$_id_ dup key: { : 1.0 }"
    }
}
```