I. Create a database with suitable example using MongoDB and implement
- Inserting and saving document (batch insert, insert validation)
- Removing document
-Updating document (document replacement, using modifiers, upserts, updating multiple documents, returning updated documents)

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.

Basic syntax of **use DATABASE** statement is as follows −

```
use DATABASE_NAME
```

```
>use mydb
switched to db mydb
```

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.

```
>db.movie.insert({"name":"Yogesh Murumkar"})
>show dbs
local      0.78125GB
mydb       0.23012GB
test       0.23012GB
```

In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.

1. Inserting a document

```
db.inventory.insertOne(
   { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom:
"cm" } })

db.inventory.insertMany([
   { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21,
uom: "cm" } },
   { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" }
},
   { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85,
uom: "cm" } }
])
```

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

Syntax

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

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

```
>db.mycol.insert({
   _id: ObjectId(7df78ad8902c),
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by: 'sachin',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
})
```

Here mycol is our collection name, as created earlier. 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.

```
>db.post.insert([
   {
      title: 'MongoDB Overview',
      description: 'MongoDB is no sql database',
      by: 'You tube 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: new Date(2013,11,10,2,35),
            like: 0
         }
      ]
   }
])
```

2. Saving document

To insert the document you can use db.post.save(document) also. If you don't specify _id in the document then save() method will work same as insert() method. If you specify _id then it will replace whole data of document containing _id as specified in save() method.

Save - insert or update a document.
Insert- does only an insertion.

Save an Apple

```
db.fruit.save({"name":"apple", "color":"red","shape":"round"})
```

Save an apple with _id of previously saved apple and then the apple we saved will have color updated from red to real red

Save an apple with _id

Apple will get inserted as there is no apple with the same Object Id to do an update

```
db.fruit.save(
{"_id" : ObjectId("53fa1809132c1f084b005cd0"),"name":"apple",
"color":"real red","shape":"round"})
```

3. Batch Insert

```
>db.post1.insert([
   {
      title: 'MongoDB Overview',
      description: 'MongoDB is no sql database',
      by: 'You tube 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: new Date(2013,11,10,2,35),
            like: 0
         }
      ]
   }
])
```

4. Validation

MongoDB provides the capability to validate documents during updates and insertions. Validation rules are specified on a per-collection basis using the `validator` option, which takes a document that specifies the validation rules or expressions. Specify the expressions using any query operators.

The following example creates a `contacts` collection with a validator that specifies that inserted or updated documents should match at least one of three following conditions:

- the `phone` field is a string
- the `email` field matches the regular expression
- the `status` field is either `Unknown` or `Incomplete`.

```
db.createCollection( "contacts",
   {
      validator: { $or:
         [
            { phone: { $type: "string" } },
            { email: { $regex: /@mongodb\.com$/ } },
            { status: { $in: [ "Unknown", "Incomplete" ] } }
         ]
      },
      validationAction: "error"
   }
)

Now try to insert using following command

db.contacts.insert( { name: "Amanda", status: "Updated" } )
```

5. Removing document

```
db.inventory.insertMany([
   { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21,
uom: "cm" } },
   { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" }
},
   { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85,
uom: "cm" } }
])
```

Remove Documents that Match a Condition

```
db.inventory.remove( { item : "mousepad" } )
```

Remove All Documents

```
db.inventory.remove({})
```

6. Updating Document

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

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

Consider the mycol collection has the following data.

```
db.ypm.insertMany([{"title":"MongoDB   Overview"},{"title":"NoSQL   Overview"},
{"title":"Big Data Overview"}])
```

Following example will set the new title 'New MongoDB Tutorial with Yogesh Murumkar' for the documents whose title is 'MongoDB Overview'.

```
db.ypm.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial
with Yogesh Murumkar'}})
```

7. Documement replacement

`db.collection.replaceOne`(*filter*, *replacement*, *options*)

Replaces a single document within the collection based on the filter.

The `replaceOne()` method has the following form:

```
db.collection.replaceOne(

   <filter>,
   <replacement>,
   {
     upsert: <boolean>,
     writeConcern: <document>
   }
)
```

`replaceOne()` replaces the first matching document in the collection that matches the `filter`, using the `replacement` document.

If `upsert: true` and no documents match the `filter`, `replaceOne()` creates a new document based on the `replacement` document.

The `restaurant1` collection contains the following documents:

```
db.restaurant1.insert([{ "_id" : 1, "name" : "Central Perk Cafe", "Borough" :
"Manhattan" },

{ "_id" : 2, "name" : "Rock A Feller Bar and Grill", "Borough" : "Queens",
"violations" : 2 },
{ "_id" : 3, "name" : "Empire State Pub", "Borough" : "Brooklyn", "violations" :
5 }])
```

The following operation replaces a single document where `name: "Central Perk Cafe"`:

```
db.restaurant1.replaceOne(

      { "name" : "Central Perk Cafe" },
      { "name" : "Central Pork Cafe", "Borough" : "Manhattan" }
   );
```

8. Replace with Upsert

The `restaurant2` collection contains the following documents:

```
db.restaurant2.insert([{ "_id" : 1, "name" : "Central Perk Cafe", "Borough" :
"Manhattan",  "violations" : 3 },

{ "_id" : 2, "name" : "Rock A Feller Bar and Grill", "Borough" : "Queens",
"violations" : 2 },
{ "_id" : 3, "name" : "Empire State Pub", "Borough" : "Brooklyn", "violations" :
5 }])
```

The following operation attempts to replace the document with `name : "Pizza Rat's Pizzaria"`, with `upsert : true`:

```
db.restaurant2.replaceOne(

      { "name" : "Pizza Rat's Pizzaria" },
      { "_id": 4, "name" : "Pizza Rat's Pizzaria", "Borough" : "Manhattan",
"violations" : 8 },
      { upsert: true }
   );
```

Since `upsert : true` the document is inserted based on the `replacement` document.

9. Update Multiple Documents

The `restaurant3` collection contains the following documents:

```
db.restaurant3.insert([{ "_id" : 1, "name" : "Central Perk Cafe", "violations" :
3 }

{ "_id" : 2, "name" : "Rock A Feller Bar and Grill", "violations" : 2 }
{ "_id" : 3, "name" : "Empire State Sub", "violations" : 5 }
{ "_id" : 4, "name" : "Pizza Rat's Pizzaria", "violations" : 8 }])
```

The following operation updates all documents where `violations` are greater than `4` and `$set` a flag for review:

```
 db.restaurant.updateMany(

      { violations: { $gt: 4 } },
      { $set: { "Review" : true } });
```