MongoDB

TUTORIAL

Small Codes

Programming Simplified

A SmlCodes.Com Small presentation

In Association with Idleposts.com

For more tutorials & Articles visit SmlCodes.com

MongoDB Tutorial

If you discover any errors on our website or in this tutorial, please notify us at support@smlcodes.com or smlcodes@gmail.com

First published on FEB 2016, Published by **SmlCodes.com**

Author Credits

Name : **Satya Kaveti**

Email : satyakaveti@gmail.com

Website : smlcodes.com, satyajohnny.blogspot.com

Digital Partners

1. Introduction

MongoDB is an open-source **NoSQL, Document Database** Written in **C++** that provides high performance, high availability, and automatic scaling.

1.1 NoSQL (Not Only SQL)

It provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases. NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications.

NoSQL Database Types

- **Document databases:** Documents can contain many different key-value pairs, or key-array pairs.
- **Graph stores:** are used to store networks of data, such as social connections.
- **Key-value stores:** simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value.
- **Wide-column** stores such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

1.2 Document Database

A record in MongoDB is a document, which is a data structure composed of **field and value pairs.** MongoDB documents **are similar to JSON objects**. The values of fields may include other documents, arrays, and arrays of documents.

```
{
   name: "sue",                          <-- field: value
   age: 26,                              <-- field: value
   status: "A",                          <-- field: value
   groups: [ "news", "sports" ]          <-- field: value
}
```

1.3 MongoDB Features & Advantages

High Performance : Indexes support faster queries

Rich Query Language : supports CRUD Operation, Data Aggregation, Text Search &Geospatial Queries.

High Availability : MongoDB's replication facility, called replica set provide atomatic failover and Data redundancy.

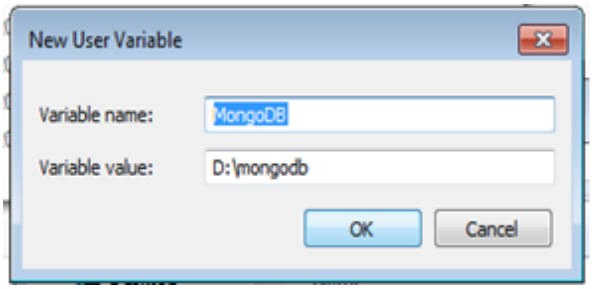
Horizontal Scalability : Sharding (partitioning) distributes data across a cluster of machines.

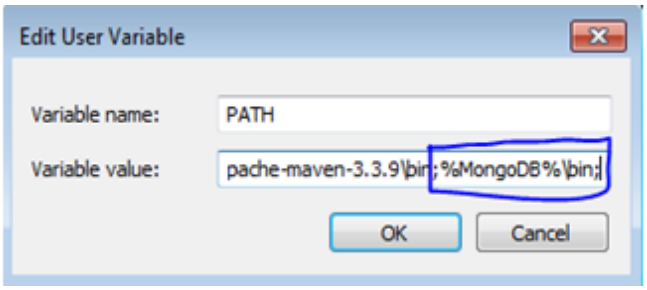
1.4 MongDB Installation & Configuration

The MongoDB does not require installation, just download and extracts the zip file, configure the data directory and start it with command "`mongod`".

- **Download** MongoDB and extract into some folder, ex: **d:/mongodb**

- Create following directories inside **d:/mongodb**
  - **D:\mongodb\data**
  - **D:\mongodb\log**
- Configure Environmnet variables `MongoDB = D:\mongodb  PATH=%MongoDB%\bin`





- Create a mongodb config file under : `d:\mongodb\mongo.config`

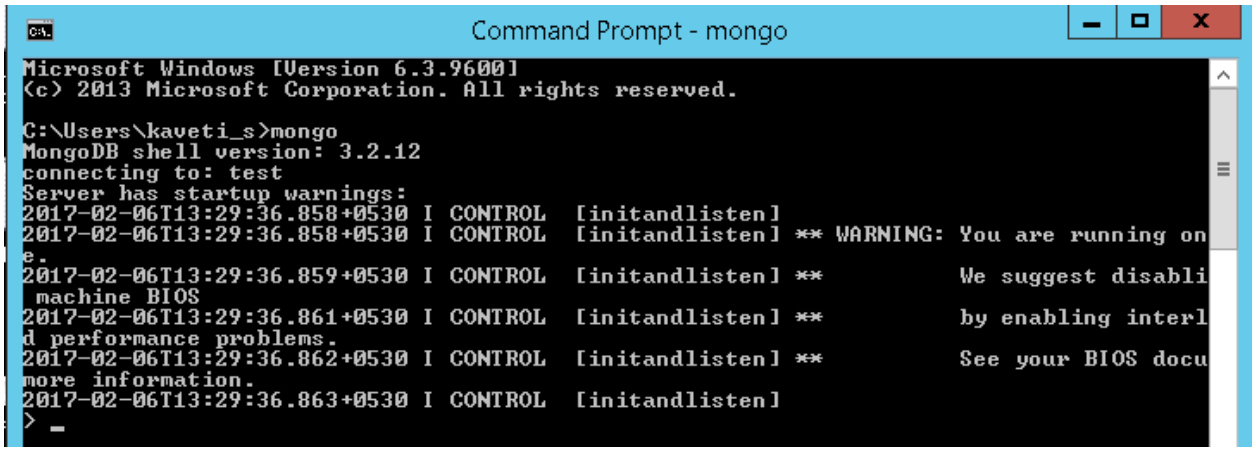
```
##store data here
dbpath=D:\mongodb\data

##all output go here
logpath=D:\mongodb\log\mongo.log

##log read and write operations
diaglog=3
```

- Start MongoDB using any of below commands
  - ➔ **D:\mongodb\bin>mongod –dbpath=D:/mongodb**
  - ➔ **d:\mongodb\bin>mongod.exe --config="D:\mongodb\mongo.config"**

- Connect to MongoDB using **mongo** command



To start MongoDB Service

```
net start MongoDB
```

To stop MongoDB Service

```
net stop MongoDB
```

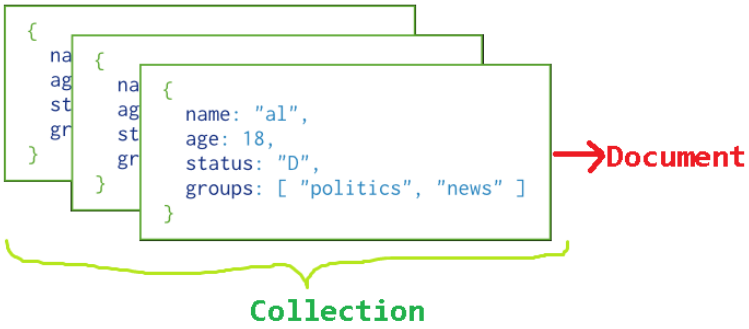
To remove MongoDB Service

```
c:\mongodb\bin>mongod --remove
```

2. MongoDB Operations

2.1 Database – Collection – Documents

- **Document**: is a single entry / record. i.e., row in a database
- **Collection**: Group of Documents are known as Collection. i.e., Table
- **Database**: Group of Collections are known as Database



2.2 Database Operations

1. Create Database

Syntax: use <database_name>

```
> use smlcodes
switched to db smlcodes
```

Here, your created database "smlcodes" is not present in the list, insert at least one document into it to display database

2. Check the currently selected database

Syntax: >db

```
> db
Smlcodes
```

3. Show all Databases

Syntax: >show dbs

```
> show dbs
local     0.000GB
smlcodes  0.000GB
```

4. Drop Database

Syntax: >db.dropDatabase()

```
> db.dropDatabase()
{ "dropped" : "smlcodes", "ok" : 1 }
```

2.3 Collection Operations

Usually we don't need to create collection. MongoDB creates collection automatically when you insert some documents.

Example: Insert a document named "admin" into a collection named "users". The operation will create the collection if the collection does not currently exist

```
> db.users.insert({"username":"admin", "password":"Admin@123"})
WriteResult({ "nInserted" : 1 })
> show collections
users
```

We can also create collection by using **db.createCollection(name, options)**

1. Create Collection

Syntax: **db.createCollection(name, options)**

- **Name**: is a string type, specifies the name of the collection to be created.
- **Options**: is a document type, specifies the memory size and indexing of the collection. (optional)

```
> db.createCollection("books")
{ "ok" : 1 }
```

2.check the collections in the database

Syntax: **show collections()**

```
> show collections
books
users
```

3. Drop Collection

Syntax: **db.<COLLECTION_NAME>.drop()**

```
> db.books.drop();
true
> show collections
users
```

The drop command returns true if it successfully drops a collection. It returns false when there is no existing collection to drop.

2.4 Document Operations

Data Types	Description
String	String is the most commonly used datatype. It is used to store data
Integer	Integer is used to store the numeric value. It can be 32 bit or 64 bit depends on server
Boolean	This datatype is used to store boolean values. It just shows YES/NO values.
Double	Double datatype stores floating point values.
Min/Max Keys	This datatype compare a value against the lowest and highest bson elements.
Arrays	This datatype is used to store a list or multiple values into a single key.
Object	Object datatype is used for embedded documents.
Null	It is used to store null values.
Symbol	It is generally used for languages that use a specific type.
Date	This datatype stores the current date or time in unix time format

2.4.1 Insert Documents

MongoDB provides the following methods for inserting documents into a collection:

1. **db.collection.insert()**
2. **db.collection.insertOne()**
3. **db.collection.insertMany()**

If the collection does not currently exist, insert operations will create the collection.

_id Field: In MongoDB, each document stored in a collection requires a **unique _id field that acts as a primary key**. If an inserted document omits the _id field, the MongoDB driver automatically generates an ObjectId for the _id field.

1.db.collection.insert():

Inserts a single document or multiple documents into a collection. To insert a single document, pass a document to the method; to insert multiple documents, pass an array of documents to the method

```
db.users.insert(
   {
      username: "Satya",
      password: "Satya@134",
      age:27,
      status:"active"
   }
)
 WriteResult({ "nInserted" : 1 })
```

2.db.collection.insertOne()

Inserts a **single document** into a collection

```
> db.users.insertOne(
   {
      username: "Smlcodes",
      password: "Smlcodes@134",
      age:27,
      status:"active"
   }
)
{
        "acknowledged" : true,
        "insertedId" : ObjectId("58986791fb8a774546289da0")
}
```

3.db.collection.insertMany()

Inserts multiple documents into a collection.

```
db.users.insertMany(
   [
     { username:"Surya",   password:"Password@1345", age: 42, status: "inactive", },
     { username:"Ravi",    password:"Password@1345", age: 22, status: "inactive", },
     { username:"Rakesh", password:"Password@1345", age: 34, status: "active", }
   ]
)
----
... )
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("589868c4fb8a774546289da1"),
                ObjectId("589868c4fb8a774546289da2"),
                ObjectId("589868c4fb8a774546289da3")
        ]
}
```

2.4.2 Query Documents (find Operations)

MongoDB provides the **db.collection.find()** method to read documents from a collection. The **db.collection.find()** method returns **a cursor** to the matching documents.

Syntax **> db.collection.find(<query filter>, <projection>)**

- **<query filter>:** a query filter to specify which documents to return.
- **< projection>:** which fields from the matching documents to return

1.Select All Documents in a Collection

An empty query filter document ({}) selects all documents in the collection

Syntax **> db.users.find({}) or db.users.find()**

```
> db.users.find({})
{ "_id" : ObjectId("58986156fb8a774546289d9e"), "username" : "admin", "password" :
"Admin@123" }
{ "_id" : ObjectId("58986716fb8a774546289d9f"), "username" : "Satya", "password" :
"Satya@134", "age" : 27, "status" : "active" }
{ "_id" : ObjectId("58986791fb8a774546289da0"), "username" : "Smlcodes", "password" :
"Smlcodes@134", "age" : 27, "status" : "active" }
{ "_id" : ObjectId("589868c4fb8a774546289da1"), "username" : "Surya", "password" :
"Password@1345", "age" : 42, "status" : "inactive" }
```

2.Select All Documents with Condition

```
> db.users.find( { status: "active" } )
{ "_id" : ObjectId("58986716fb8a774546289d9f"), "username" : "Satya", "password" :
"Satya@134", "age" : 27, "status" : "active" }
{ "_id" : ObjectId("58986791fb8a774546289da0"), "username" : "Smlcodes", "password" :
"Smlcodes@134", "age" : 27, "status" : "active" }
{ "_id" : ObjectId("589868c4fb8a774546289da3"), "username" : "Rakesh", "password" :
"Password@1345", "age" : 34, "status" : "active" }
```

3.Select All Documents with AND Condition

We can specify the the no. of conditions by using **comma (,)** operator

Retrieves all documents where **status equals "active" and age is less than ($lt) 30:**

```
> db.users.find( { status: "active", age: { $lt: 30 } } )
{ "_id" : ObjectId("58986716fb8a774546289d9f"), "username" : "Satya", "password" :
"Satya@134", "age" : 27, "status" : "active" }
{ "_id" : ObjectId("58986791fb8a774546289da0"), "username" : "Smlcodes", "password" :
"Smlcodes@134", "age" : 27, "status" : "active" }
```

4.Select All Documents with OR Condition

Using the **$or** operator, you can specify a compound query that joins each clause with a logical OR conjunction so that the query selects the documents in the collection that match at least one condition.

Retrieves all documents where the **status equals "inactive" or age is less than ($lt) 30:**

```
db.users.find(
   {
     $or: [ { status: "inactive" }, { age: { $lt: 30 } } ]
   }
)
-------
{ "_id" : ObjectId("58986716fb8a774546289d9f"), "username" : "Satya", "password" :
"Satya@134", "age" : 27, "status" : "active" }
{ "_id" : ObjectId("589868c4fb8a774546289da1"), "username" : "Surya", "password" :
"Password@1345", "age" : 42, "status" : "inactive" }
```

For more related Query Documents visit [MongoDB Offcial website]

2.4.3 Update Documents

MongoDB provides the following methods for updating documents in a collection

1. **db.collection.update()**
2. **db.collection.updateOne()**
3. **db.collection.updateMany()**
4. **db.collection.replaceOne()**

Once set, you cannot update the value of the _id field nor can you replace an existing document with a replacement document that has a different _id field value.

1.db.collection.update()

Either updates or replaces a **single document** that match a specified filter **or updates all documents** that match a specified filter.

```
db.users.update(
   { "status": "inactive" },
   {
     $set: { "Level": 2}
   }
)
------
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

2.db.collection.updateOne()

Updates at most a single document that match a specified filter even though multiple documents may match the specified filter.

```
db.users.updateOne(
   { "username": "Surya" },
   {
     $set: { "password": "901290190", age: 20 }
   }
)
------
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

3.db.collection.updateMany()

Update all documents that match a specified filter.

```
db.users.updateMany(
   { "status": "active" },
   {
     $set: { "Level": 1}
   }
)
----------
{ "acknowledged" : true, "matchedCount" : 6, "modifiedCount" : 6 }
```

4.db.collection.replaceOne()

Replaces at most a single document that match a specified filter even though multiple documents may match the specified filter.

2.4.4 Delete Documents

MongoDB provides the following methods to delete documents of a collection:

1. **db.collection.remove()**
2. **db.collection.deleteOne()**
3. **db.collection.deleteMany()**

Delete operations do not drop indexes, even if deleting all documents from a collection

1.db.collection.remove()

To remove all documents from the collection based on condition

```
> db.users.remove( { age: 27 } )
WriteResult({ "nRemoved" : 2 })
```

2.db.collection.deleteOne()

Delete at most a single document that match a specified filter,even though multiple documents may match

```
> db.users.deleteOne( { status: "active" } )
"acknowledged" : true, "deletedCount" : 1 }
```

3.db.collection.deleteMany()

To remove all documents from the collection based on condition

```
> db.users.deleteMany({ status : "inactive" })
 "acknowledged" : true, "deletedCount" : 5 }
```

The following methods can also delete documents from a collection:

- **db.collection.findOneAndDelete().**
- **findOneAndDelete()** provides a sort option. The option allows for the deletion of the first document sorted by the specified order.
- **db.collection.findOneAndModify().**
- **db.collection.findOneAndModify()** provides a sort option. The option allows for the deletion of the first document sorted by the specified order.
- **db.collection.bulkWrite().**

2.4.5 Advanced Operations

limit() To limit the records in MongoDB, you need to use **limit()** method

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

Skip() used to skip the number of documents.

```
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

sort() specify sorting order. 1 is used for ascending order while -1 is used for descending order.

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

aggregate() aggregation in MongoDB, you should use aggregate() method.

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
AGGREGATE_OPERATION = $sum, $avg, $min, $max, $push, $addToSet, $first, $last
```

db.collection.save()

Updates an existing document or inserts a new document, depending on its document parameter.

```
db.products.save( { item: "book", qty: 40 } )
```

help() uses to guide you how to do things in MongoDB.

```
 db.help()                    help on db methods
 db.mycoll.help()             help on collection methods
 sh.help()                    sharding helpers
 rs.help()                    replica set helpers
 help admin                   administrative help
 help connect                 connecting to a db help
 help keys                    key shortcuts
 help misc                    misc things to know
 help mr                      mapreduce

 show dbs                     show database names
 show collections             show collections in current database
 show users                   show users in current database
 show profile                 show most recent system.profile entries time >= 1ms
 show logs                    show the accessible logger names
 show log [name]              prints out the last segment of log in memory,
 use <db_name>                set current database
 db.foo.find()                list objects in collection foo
 db.foo.find( { a : 1 } )     list objects in foo where a == 1
 it                           result of the last line evaluated; use to further
iterate
 DBQuery.shellBatchSize = x   set default number of items to display on shell
 exit                         quit the mongo shell
```

3. MongoDB with Java

To use MongoDB in our Java programs, we need MongoDB JDBC driver. Follow the below steps to do so.

Steps to Connect with MongoDB Using Java

1.Create Java Project using Elipse Convet that into Maven Project

2. Download mongo-java driver from github. Or declare mongo-java driver in pom.xml

```
	<dependencies>
		<dependency>
			<groupId>org.mongodb</groupId>
			<artifactId>mongo-java-driver</artifactId>
			<version>2.10.1</version>
		</dependency>
	</dependencies>
```

3. Write a Java class to connect with MongoDB & perform operations

```
package core;
import java.net.UnknownHostException;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.MongoClient;
import com.mongodb.MongoException;

public class MongoDBConnect {
	public static void main(String[] args) {
		try {
			/**** Connect to MongoDB ****/
			// Since 2.10.0, uses MongoClient
			MongoClient mongo = new MongoClient("localhost", 27017);

			/**** Get database ****/
			// if database doesn't exists, MongoDB will create it for you
			DB db = mongo.getDB("smlcodes");

			/**** Get collection / table from 'testdb' ****/
			// if collection doesn't exists, MongoDB will create it for you
			DBCollection table = db.getCollection("user");

			if (mongo != null) {
		System.out.println("============\n MongoDB Connected!!! \n===========");
				System.out.println("Database Name : " + db.getName());
				System.out.println("Collection : " + table.getName());
			}

		} catch (UnknownHostException e) {
			e.printStackTrace();
		} catch (MongoException e) {
			e.printStackTrace();
		}
	}
}
```

```
Output
 MongoDB Connected!!!
===========
Database Name : smlcodes
Collection : user
```

3.1 MongoDB with Java

1.Mongo Connection

Connect to MongoDB server. For MongoDB version >= 2.10.0, uses MongoClient.

```
// Old version, uses Mongo
Mongo mongo = new Mongo("localhost", 27017);

// Since 2.10.0, uses MongoClient
MongoClient mongo = new MongoClient( "localhost" , 27017 );
```

2.Mongo Database

Get database. If the database doesn't exist, MongoDB will create it for you.

```
DB db = mongo.getDB("database name");
```

If MongoDB in secure mode, authentication is required

```
boolean auth = db.authenticate("username", "password".toCharArray());
```

Display all databases.

```
List<String> dbs = mongo.getDatabaseNames();
for(String db : dbs){
        System.out.println(db);
}
```

3.Mongo Collection

Get collection / table.

```
DB db = mongo.getDB("testdb");
DBCollection table = db.getCollection("user");
```

Display all collections from selected database.

```
DB db = mongo.getDB("testdb");
Set<String> tables = db.getCollectionNames();

for(String coll : tables){
        System.out.println(coll);
}
```

Steps to develop any MongoDB Application

1.Connect with MongoDB Server

```
MongoClient mongoClient = new MongoClient("localhost", 27017);
```

2.Connect with Database

```
DB db = mongoClient.getDB("smlcodes");
```

3.Get the Collection , on which collection you want to work

```
DBCollection collection = db.getCollection("users");
```

4.Get Document Object to perform CURD operaions on Document

```
BasicDBObject document = new BasicDBObject();
```

1. MongoDB Authenetication Example

Add user to smlcodes Collection for testing purpose

```
db.createUser(
                {
                  user: "admin",
                  pwd: "admin",
                  roles: [
                     {role: "readWrite", db: "smlcodes"}
                           ]
                  }
           )
```

Example

```
import com.mongodb.DB;
import com.mongodb.MongoClient;

public class MongoDB_Authentication {
        public static void main(String args[]) {
                try {
                        // To connect to mongodb server
                        MongoClient mongoClient = new MongoClient("localhost", 27017);

                        // Now connect to your databases
                        DB db = mongoClient.getDB("smlcodes");
                        System.out.println("Connect to database successfully");
                        boolean auth = db.authenticate("admin", "admin".toCharArray());
                        System.out.println("Authentication: " + auth);

                } catch (Exception e) {
                        System.err.println(e.getClass().getName() + ": " + e.getMessage());
                }
        }
}
```

3.1.2 MongoDB Java Complete Example

```
package core;
import java.net.UnknownHostException;
import java.util.HashMap;
import java.util.Map;
import com.mongodb.BasicDBObject;
import com.mongodb.BasicDBObjectBuilder;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.Mongo;
import com.mongodb.MongoException;
import com.mongodb.util.JSON;
public class MongoDB_Insert {
        public static void main(String[] args) {

                try {

                        Mongo mongo = new Mongo("localhost", 27017);
                        DB db = mongo.getDB("smlcodes");

                        DBCollection collection = db.getCollection("users");
                        collection.remove(new BasicDBObject());

                        // 1. BasicDBObject example
                        System.out.println("1.BasicDBObject example...");
                        System.out.println("==========================");
                        BasicDBObject document = new BasicDBObject();
                        document.put("username", "satyajohnny");
                        document.put("password", "password254");

                        BasicDBObject documentDetail = new BasicDBObject();
                        documentDetail.put("street", "RAMALAYAM");
                        documentDetail.put("city", "VIJAYAWADA");
                        documentDetail.put("state", "ANDHRA PRADESH");
                        document.put("address", documentDetail);
                        collection.insert(document);

                        DBCursor cursorDoc = collection.find();
                        while (cursorDoc.hasNext()) {
                                System.out.println(cursorDoc.next());
                        }
                        collection.remove(new BasicDBObject());

                        // 2. BasicDBObjectBuilder example
                        System.out.println("\n\n 2.BasicDBObjectBuilder Insert");
                        System.out.println("==========================");
                        BasicDBObjectBuilder documentBuilder = BasicDBObjectBuilder.start()
                                        .add("username", "Anil")
                                        .add("password",        "Anigirekula123");

                        BasicDBObjectBuilder documentBuilderDetail = BasicDBObjectBuilder.start()
                                .add("street", "NTR STREET")
                                        .add("city", "HYDERABAD").add("state", "TN");
                        documentBuilder.add("detail", documentBuilderDetail.get());
                        collection.insert(documentBuilder.get());

                        DBCursor cursorDocBuilder = collection.find();
                        while (cursorDocBuilder.hasNext()) {
                                System.out.println(cursorDocBuilder.next());
                        }
                        collection.remove(new BasicDBObject());

                        // 3. Map example
                        System.out.println("\n\n 3.MAP Insert");
                        System.out.println("==========================");
                        Map<String, Object> documentMap = new HashMap<String, Object>();
                        documentMap.put("username", "mapuser");
                        documentMap.put("password", "mapassword");
```

```
                        Map<String, Object> documentMapDetail = new HashMap<String, Object>();
                        documentMapDetail.put("street", "JAMES STREET");
                        documentMapDetail.put("city", "GEORGIO");
                        documentMapDetail.put("state", "U.S");
                        documentMap.put("detail", documentMapDetail);
                        collection.insert(new BasicDBObject(documentMap));

                        DBCursor cursorDocMap = collection.find();
                        while (cursorDocMap.hasNext()) {
                                System.out.println(cursorDocMap.next());
                        }
                        collection.remove(new BasicDBObject());

                        // 4. JSON parse example
                        System.out.println("\n\n 4.JSON Insert");
                        System.out.println("============================");

        String json = "{'username' : 'jsonuser','password' : 'JsonPass',"
        + "'detail' : {'street' : 'FIGHTCLUB STREET', 'city' : 'MELBORN', 'state' : 'AUS'}}}";

                        DBObject dbObject = (DBObject) JSON.parse(json);
                        collection.insert(dbObject);

                        DBCursor cursorDocJSON = collection.find();
                        while (cursorDocJSON.hasNext()) {
                                System.out.println(cursorDocJSON.next());
                        }
                        collection.remove(new BasicDBObject());

                } catch (UnknownHostException e) {
                        e.printStackTrace();
                } catch (MongoException e) {
                        e.printStackTrace();
                }

        }
}
```

```
//Output
1.BasicDBObject example...
============================
{ "_id" : { "$oid" : "589b07a32989f6de61c17c09"} , "username" : "satyajohnny" , "password" :
"password254" , "address" : { "street" : "RAMALAYAM" , "city" : "VIJAYAWADA" , "state" : "ANDHRA
PRADESH"}}


 2.BasicDBObjectBuilder Insert
============================
{ "_id" : { "$oid" : "589b07a32989f6de61c17c0a"} , "username" : "Anil" , "password" :
"Anigirekula123" , "detail" : { "street" : "NTR STREET" , "city" : "HYDERABAD" , "state" : "TN"}}


 3.MAP Insert
============================
{ "_id" : { "$oid" : "589b07a32989f6de61c17c0b"} , "password" : "mapassword" , "detail" : { "city"
: "GEORGIO" , "street" : "JAMES STREET" , "state" : "U.S"} , "username" : "mapuser"}


 4.JSON Insert
============================
{ "_id" : { "$oid" : "589b07a32989f6de61c17c0c"} , "username" : "jsonuser" , "password" :
"JsonPass" , "detail" : { "street" : "FIGHTCLUB STREET" , "city" : "MELBORN" , "state" : "AUS"}}
```

In above we are removing inserted Object for display purpose only

```
collection.remove(new BasicDBObject());
```

Similarly we can perform CURD operations using below methods in the same way

Update Operation

Update a document where "username"="satya" to SatyaKaveti.

```
DBCollection table = db.getCollection("user");

BasicDBObject query = new BasicDBObject();
query.put("username", "satya");

BasicDBObject newDocument = new BasicDBObject();
newDocument.put("username", "SatyaKaveti");

BasicDBObject updateObj = new BasicDBObject();
updateObj.put("$set", newDocument);

table.update(query, updateObj);
```

Find/Query/Search Operation

Find document where "username =satya", and display it with DBCursor

```
DBCollection table = db.getCollection("user");

BasicDBObject searchQuery = new BasicDBObject();
searchQuery.put("username ", " satya ");

DBCursor cursor = table.find(searchQuery);

while (cursor.hasNext()) {
       System.out.println(cursor.next());
}
```

Delete Operation

Find document where "username =satya", and delete it.

```
DBCollection table = db.getCollection("user");

BasicDBObject searchQuery = new BasicDBObject();
searchQuery.put("username ", " satya ");

table.remove(searchQuery);
```

3.2 MongoDB with Spring Data

To work with MogoDB with Spring Data, we need following dependencies. So, add these dependencies in pom.xml of your project & run maven install

```
<dependencies>
		<!-- Spring framework -->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-core</artifactId>
			<version>3.2.2.RELEASE</version>
		</dependency>

		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>3.2.2.RELEASE</version>
		</dependency>

		<!-- mongodb java driver -->
		<dependency>
			<groupId>org.mongodb</groupId>
			<artifactId>mongo-java-driver</artifactId>
			<version>2.11.0</version>
		</dependency>

		<!-- Spring data mongodb -->
		<dependency>
			<groupId>org.springframework.data</groupId>
			<artifactId>spring-data-mongodb</artifactId>
			<version>1.2.0.RELEASE</version>
		</dependency>

		<dependency>
			<groupId>cglib</groupId>
			<artifactId>cglib</artifactId>
			<version>2.2.2</version>
		</dependency>
</dependencies>
```

1. We need to create SpringMongoConfig.java to connect with MongoDB database

```
package spring;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.mongodb.core.MongoTemplate;

import com.mongodb.MongoClient;

@Configuration
public class SpringMongoConfig {

public @Bean MongoTemplate mongoTemplate() throws Exception {
	MongoTemplate mongoTemplate = new MongoTemplate(new MongoClient("127.0.0.1"), "Emp");
	return mongoTemplate;
	}
}
```

2. We need to create an **Employee.java** Bean class. Uses **@Document** to define a "collection name" when you save this object. In this case, when "Employee" object saves, it will save into "employee" collection

```
package spring;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;
```

```
@Document(collection = "employee")
public class Employee {
	@Id
	private String id;

	@Indexed // means Unique
	private String email;

	private String name;
	private int age;
	private String address;

	public String getId() {
		return id;
	}

	public void setId(String id) {
		this.id = id;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	public String getAddress() {
		return address;
	}

	public void setAddress(String address) {
		this.address = address;
	}

	public Employee(String id, String email, String name, int age, String address) {
		super();
		this.id = id;
		this.email = email;
		this.name = name;
		this.age = age;
		this.address = address;
	}

	@Override
	public String toString() {
		return "User [id=" + id + ", email=" + email + ",
			name=" + name + ", age=" + age + ", address=" + address + "]";
	}

}
```

Above **SpringMongoConfig.java,Employee.java** classes are common for all Examples

1. Spring Data + MongoDB Insert Operation

In Spring data MongoDB, you can use **save(), insert()** to save objects into mongoDB database.

```
User user = new User("...");

//save user object into "user" collection / table
//class name will be used as collection name
mongoOperation.save(user);

//save user object into "users" collection
mongoOperation.save(user,"users");

//insert user object into "user" collection
//class name will be used as collection name
mongoOperation.insert(user);

//insert user object into " users " collection
mongoOperation.insert(user, " users ");

//insert a list of user objects
mongoOperation.insert(listofUser);
```

- **Save** (**saveOrUpdate()**) it performs **insert()** if "_id" is NOT exist or **update()** if "_id" is existed".
- **Insert** – Only insert, if "_id" is existed, an error is generated.

Example

```
package spring;
import java.util.ArrayList;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.data.mongodb.core.MongoOperations;

public class SpringMongo_Insert {

	public static void main(String[] args) {
		// For Annotation
		ApplicationContext ctx = new AnnotationConfigApplicationContext(SpringMongoConfig.class);
		MongoOperations mongoOperation = (MongoOperations) ctx.getBean("mongoTemplate");

		System.out.println("1 - insert a Employee, put 'employee' as collection name");
		Employee emp1 = new Employee("101", "1.akil@one.com", "AKHIL", 30, "Hyderabad");
		mongoOperation.save(emp1, "employee");

		System.out.println("2- insert a Emmployee, put bean name as collection name");
		Employee emp2 = new Employee("102", "2.balu@two.com", "BALU", 25, "Vijayawada");
		mongoOperation.save(emp2);

		System.out.println("3 - insert a list of employees");
		Employee emp3 = new Employee("103", "3.chandu@two.com", "CHANDU", 35, "Mumbai");
		Employee emp4 = new Employee("104", "4.delip@two.com", "DELIP", 43, "Delhi");
		Employee emp5 = new Employee("105", "5.Ervin@two.com", "ERVIN", 29, "Kolkata");
		List<Employee> empList = new ArrayList<Employee>();
		empList.add(emp3);
		empList.add(emp4);
		empList.add(emp5);
		mongoOperation.insert(empList, Employee.class);

		System.out.println("List Of all Saved Employees \n==================");
		List<Employee> employees = mongoOperation.findAll(Employee.class);

		for (Employee employee : employees) {
			System.out.println(employee);
			mongoOperation.remove(employee);
		}
	}
}
```

```
//Output
1 - insert a Employee, put 'employee' as collection name
2- insert a Emmployee, put bean name as collection name
3 - insert a list of employees
List Of all Saved Employees
==================
User [id=101, email=1.akil@one.com, name=AKHIL, age=30, address=Hyderabad]
User [id=102, email=2.balu@two.com, name=BALU, age=25, address=Vijayawada]
User [id=103, email=3.chandu@two.com, name=CHANDU, age=35, address=Mumbai]
User [id=104, email=4.delip@two.com, name=DELIP, age=43, address=Delhi]
User [id=105, email=5.Ervin@two.com, name=ERVIN, age=29, address=Kolkata]
```

2. Spring Data + MongoDB Update Operation

In spring data – MongoDB, you can use following methods to update documents.

1. **save** – Update the whole object, if "_id" is present, perform an update, else insert it.
2. **updateFirst** – Updates the first document that matches the query.
3. **updateMulti** – Updates all documents that match the query.
4. **Upserting** – If no document that matches the query, a new document is created by combining the query and update object.
5. **findAndModify** – Same with updateMulti, but it has an extra option to return either the old or newly updated document

Find the document, modify and update it with `save()` method.

```
Query query = new Query();
query.addCriteria(Criteria.where("name").is("appleA"));

User userTest1 = mongoOperation.findOne(query, User.class);

System.out.println("userTest1 - " + userTest1);

//modify and update with save()
userTest1.setAge(99);
mongoOperation.save(userTest1);

//get the updated object again
User userTest1_1 = mongoOperation.findOne(query, User.class);

System.out.println("userTest1_1 - " + userTest1_1);
```

Example

```
package spring;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.data.mongodb.core.MongoOperations;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;

public class SpringMongo_Update {

        public static void main(String[] args) {
                // For Annotation
                ApplicationContext ctx = new AnnotationConfigApplicationContext(SpringMongoConfig.class);
                MongoOperations mongoOperation = (MongoOperations) ctx.getBean("mongoTemplate");
```

```
                System.out.println("1- find and update");
                Query query1 = new Query();
                query1.addCriteria(Criteria.where("name").is("BALU"));
                Employee emp1 = mongoOperation.findOne(query1, Employee.class);
                System.out.println("Before Update : " + emp1);
                emp1.setAge(75);
                mongoOperation.save(emp1);
                Employee emp1_1 = mongoOperation.findOne(query1, Employee.class);
                System.out.println("After Update : " + emp1_1 + "\n------------------");



                System.out.println("2- select single field only");
                Query query2 = new Query();
                query2.addCriteria(Criteria.where("name").is("CHANDU"));
                query2.fields().include("name");
                query2.fields().include("age");
                Employee emp2 = mongoOperation.findOne(query2, Employee.class);
                System.out.println("Before Update : " + emp2);
                emp2.setAge(88);
                mongoOperation.save(emp2);
                Employee emp11 = mongoOperation.findOne(query2, Employee.class);
                System.out.println("After Update : " + emp11 + "\n------------------");
        }

}
```

```
//Output
1- find and update
Before Update : User [id=102, email=2.balu@two.com, name=BALU, age=75, address=Vijayawada]
After Update : User [id=102, email=2.balu@two.com, name=BALU, age=25, address=Vijayawada]
------------------
2- select single field only
Before Update : User [id=103, email=null, name=CHANDU, age=88, address=null]
After Update : User [id=103, email=null, name=CHANDU, age=38, address=null]
```

3. Spring Data + MongoDB Query Operation

Here we show you a few examples to query documents from MongoDB, by using Query, Criteria and along with some of the common operators.

1. BasicQuery example: If you are familiar with the core MongoDB console find() command, just put the "raw" query inside the `BasicQuery`.

2. findOne example: findOne will return the single document that matches the query, and you can a combine few criteria with `Criteria.and()` method. See example 4 for more details.

3. find and $inc example: Find and return a list of documents that match the query. This example also shows the use of `$inc` operator.

4. find and $gt, $lt, $and example: Find and return a list of documents that match the query. This example also shows the use of `$gt`, `$lt` and `$and` operators.

Example

```
package spring;

import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.data.domain.Sort;
import org.springframework.data.mongodb.core.MongoOperations;
import org.springframework.data.mongodb.core.query.BasicQuery;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;

public class SpringMongo_QueryFind {

	public static void main(String[] args) {
		// For Annotation
		ApplicationContext ctx = new AnnotationConfigApplicationContext(SpringMongoConfig.class);
		MongoOperations mongoOperation = (MongoOperations) ctx.getBean("mongoTemplate");

		System.out.println("1 - findOne- with BasicQuery example");
		BasicQuery query1 = new BasicQuery("{ age : { $lt : 40 } }");
		Employee emp1 = mongoOperation.findOne(query1, Employee.class);
		System.out.println("query1 - " + query1.toString());
		System.out.println("emp1 - " + emp1);
		System.out.println("------------------------------");

		System.out.println("2 - findOne AND example");
		Query query2 = new Query();
		query2.addCriteria(Criteria.where("name").is("DELIP").and("age").is(43));
		Employee emp2 = mongoOperation.findOne(query2, Employee.class);
		System.out.println("query2 - " + query2.toString());
		System.out.println("emp2 - " + emp2);
		System.out.println("------------------------------");

		System.out.println("3 - findlist $and $lt, $gt example");
		Query query4 = new Query();

	query4.addCriteria(Criteria.where("age").lt(40).andOperator(Criteria.where("age").gt(10)));
		List<Employee> emp4 = mongoOperation.find(query4, Employee.class);
		System.out.println("query4 - " + query4.toString());
		for (Employee employee : emp4) {
			System.out.println("emp4 - " + employee);
		}
		System.out.println("------------------------------");

		System.out.println("4 - find list and sorting example");
		Query query5 = new Query();
		query5.addCriteria(Criteria.where("age").gte(20));
		query5.with(new Sort(Sort.Direction.DESC, "age"));
		List<Employee> emp5 = mongoOperation.find(query5, Employee.class);
		System.out.println("query5 - " + query5.toString());
		for (Employee employee : emp5) {
			System.out.println("emp5 - " + employee);
		}
		System.out.println("------------------------------");

		System.out.println(" 5- find by regex example");
		Query query6 = new Query();
		query6.addCriteria(Criteria.where("name").regex("A.*U", "i"));
		List<Employee> emp6 = mongoOperation.find(query6, Employee.class);
		System.out.println("query6 - " + query6.toString());
		for (Employee user : emp6) {
			System.out.println("emp6 - " + user);
		}
	}
}
```

```
//Output
1 - findOne- with BasicQuery example
query1 - Query: { "age" : { "$lt" : 40}}, Fields: null, Sort: { }
emp1 - User [id=101, email=1.akil@one.com, name=AKHIL, age=30, address=Hyderabad]
-----------------------------
2 - findOne AND example
query2 - Query: { "name" : "DELIP" , "age" : 43}, Fields: null, Sort: null
emp2 - User [id=104, email=4.delip@two.com, name=DELIP, age=43, address=Delhi]
-----------------------------
3 - findlist $and $lt, $gt example
query4 - Query: { "age" : { "$lt" : 40} , "$and" : [ { "age" : { "$gt" : 10}}]}, Fields: null, Sort: null
emp4 - User [id=101, email=1.akil@one.com, name=AKHIL, age=30, address=Hyderabad]
emp4 - User [id=102, email=2.balu@two.com, name=BALU, age=25, address=Vijayawada]
emp4 - User [id=103, email=null, name=CHANDU, age=38, address=null]
emp4 - User [id=105, email=5.Ervin@two.com, name=ERVIN, age=29, address=Kolkata]
-----------------------------
4 - find list and sorting example
query5 - Query: { "age" : { "$gte" : 20}}, Fields: null, Sort: { "age" : -1}
emp5 - User [id=104, email=4.delip@two.com, name=DELIP, age=43, address=Delhi]
emp5 - User [id=103, email=null, name=CHANDU, age=38, address=null]
emp5 - User [id=101, email=1.akil@one.com, name=AKHIL, age=30, address=Hyderabad]
emp5 - User [id=105, email=5.Ervin@two.com, name=ERVIN, age=29, address=Kolkata]
emp5 - User [id=102, email=2.balu@two.com, name=BALU, age=25, address=Vijayawada]
-----------------------------
 5- find by regex example
query6 - Query: { "name" : { "$regex" : "A.*U" , "$options" : "i"}}, Fields: null, Sort: null
emp6 - User [id=102, email=2.balu@two.com, name=BALU, age=25, address=Vijayawada]
emp6 - User [id=103, email=null, name=CHANDU, age=38, address=null]
```

4. Spring Data + MongoDB Delete Operation

In Spring data for MongoDB, you can use `remove()` and `findAndRemove()` to delete documents from MongoDB.

- **remove**() – delete single or multiple documents.
- **findAndRemove**() – delete single document, and returns the deleted document.

Example

```
package spring;

import java.util.List;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.data.mongodb.core.MongoOperations;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;

public class SpringMongo_Delete {

	public static void main(String[] args) {
		// For Annotation
		ApplicationContext ctx = new AnnotationConfigApplicationContext(SpringMongoConfig.class);
		MongoOperations mongoOperation = (MongoOperations) ctx.getBean("mongoTemplate");


		Query query1 = new Query();
		query1.addCriteria(Criteria.where("name").is("DELIP").and("age").is(43));
		Employee emp2 = mongoOperation.findOne(query1, Employee.class);
		mongoOperation.remove(query1, Employee.class);
		System.out.println("\nAll users : ");
```

```
                List<Employee> allEmployees = mongoOperation.findAll(Employee.class);
                for (Employee user : allEmployees) {
                        System.out.println(user);
                }
                //mongoOperation.dropCollection(Employee.class);

                System.out.println("ALL Employees DELETED");
                System.out.println("================\n THE END MONGODB\n ==============");

        }

}
```

```
//Output
All users :
User [id=101, email=1.akil@one.com, name=AKHIL, age=30, address=Hyderabad]
User [id=102, email=2.balu@two.com, name=BALU, age=25, address=Vijayawada]
User [id=103, email=null, name=CHANDU, age=38, address=null]
User [id=105, email=5.Ervin@two.com, name=ERVIN, age=29, address=Kolkata]
ALL Employees DELETED
================
 THE END MONGODB
 ==============
```

References

http://www.javatpoint.com/mongodb-tutorial

https://docs.mongodb.com/v3.2/tutorial/

http://www.mkyong.com/tutorials/java-mongodb-tutorials/

https://www.tutorialspoint.com/mongodb/mongodb_environment.htm