Introduction to MongoDB

MongoDB is a NoSQL database which stores the data in form of key-value pairs. It is an **Open Source**, **Document Database** which provides high performance and scalability along with data modelling and data management of huge sets of data in an enterprise application.

MongoDB is a cross platform database and can be installed across different platforms like Windows, Linux etc.

What is Document based storage?

A Document is nothing but a data structure with name-value pairs.

For example : Student object has attributes name, rollno and subjects, where subjects is a List.

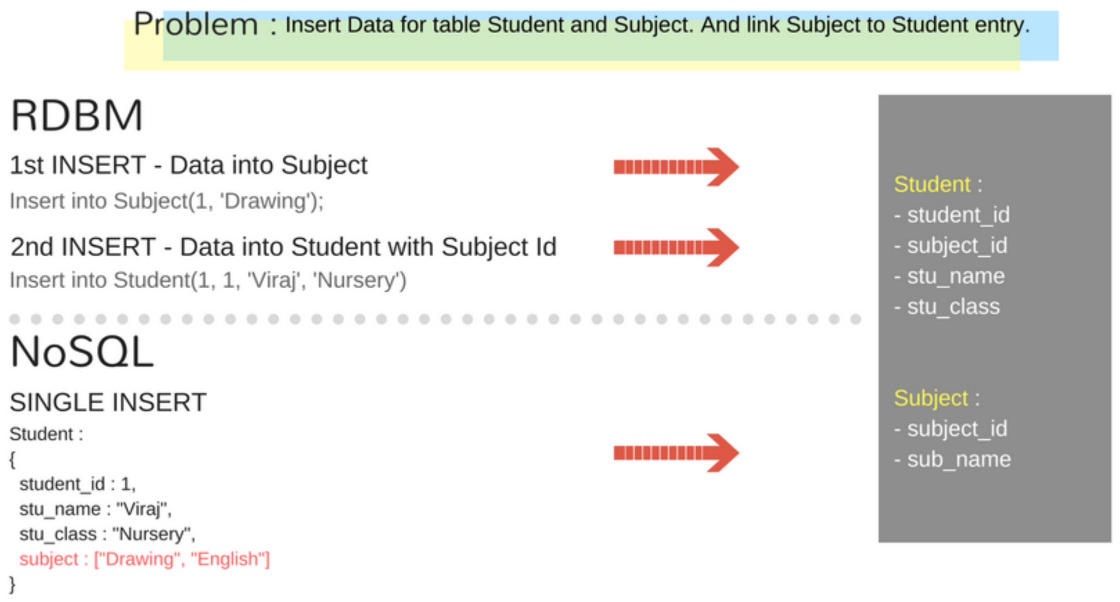
Document for Student in MongoDB will be like:

```
{
        name : "Manoj",
        rollno : 1,
        subjects : ["C Language", "C++", "Core Java"]
}
```

Key Features of MongoDB

Apart from most of the NoSQL default features, MongoDB does bring in some more, very important and useful features :

1. MongoDB provides high performance. Input/Output operations are lesser than relational databases due to support of embedded documents(data models) and Select queries are also faster as Indexes in MongoDB supports faster queries.


Problem : Insert Data for table Student and Subject. And link Subject to Student entry.
RDBM
1st INSERT - Data into Subject
2nd INSERT - Data into Student with Subject Id
NoSQL
SINGLE INSERT
Student :
- student_id
- subject_id
- stu_name
- stu_class
Subject :
- subject_id
- sub_name

```
Insert into Subject(1, 'Drawing');
```

```
Insert into Student(1, 1, 'Viraj', 'Nursery')
```

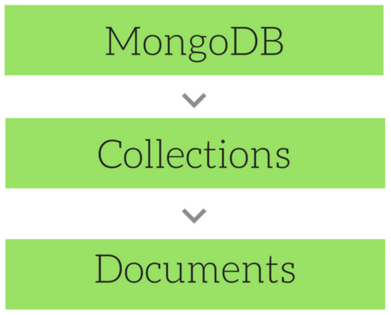
```
Student :
{
  student_id : 1,
  stu_name : "Viraj",
  stu_class : "Nursery",
  subject : ["Drawing", "English"]
}
```

2. MongoDB has a rich Query Language, supporting all the major CRUD operations. The Query Language also provides good Text Search and Aggregation features.
3. **Auto Replication** feature of MongoDB leads to High Availability. It provides an automatic failover mechanism, as data is restored through backup(replica) copy if server fails.

4. Sharding is a major feature of MongoDB. Horizontal Scalability is possible due to sharding.
5. MongoDB supports multiple Storage Engines. When we save data in form of documents(NoSQL) or tables(RDBMS) who saves the data? It's the Storage Engine. Storage Engines manages how data is saved in memory and on disk.

Overview of MongoDB

MongoDB consists of a set of databases. Each database again consists of Collections. Data in MongoDB is stored in collections. The below figure depicts the typical database structure in MongoDB.



Database in MongoDB

Database in MongoDB is nothing but a container for collections.

Collections in MongoDB

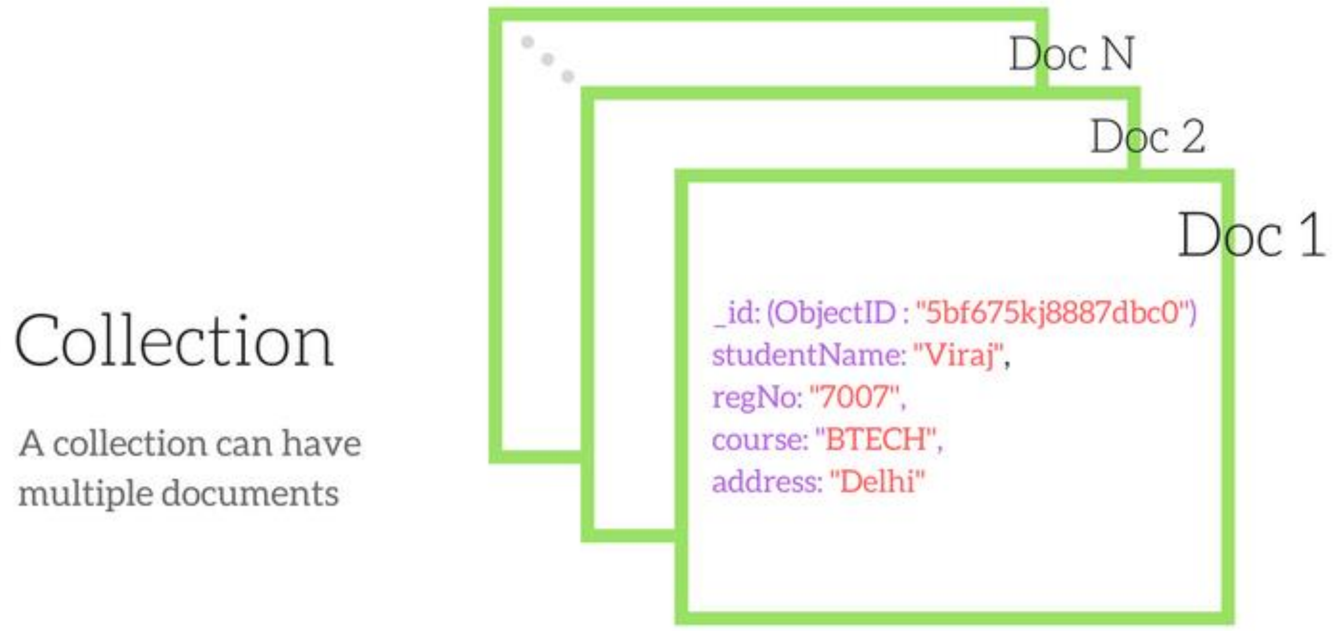
- Collection is nothing but a set of MongoDB documents. These documents are equivalent to the row of data in tables in RDBMS. But, collections in MongoDB do not relate to any set schema as compared to RDBMS. Collections are a way of storing related data. Being schema less, any type of Document can be saved in a collection, although similarity is recommended for index efficiency. Documents can have a maximum size of 4MB.
- We can use **namespace** to logically group and nest collections. **For example :** There can be one collection named db.computer.users to save user informations, then there can be others like db. computer.forum.questions and db. computer.forum.answers to store forum questions and answers respectively.
- If we create an **Index** on a namespaced collection, it will only apply on that namespace only. We will learn later in this tutorial, how to create indexes.
- A collection is physically created as soon as the first document is created in it.

Document in MongoDB

Document in MongoDB is nothing but the set of key-value pairs. These documents will have dynamic schema which means that the documents in the same collection do not need to possess the same set of fields.

Since MongoDB is considered as a schema-less database, each collection can hold different type of objects. Every object in a collection is known as Document, which is represented in a JSON like (**JavaScript Object Notation**) structure(nothing but a list of key-value pair). Data is stored and queried in **BSON**, its **binary representation of JSON**-like data.

Sample Data in MongoDB



Note: In the above figure, the field `_id` represents the primary key identifier of the given document. MongoDB also stores the values in the form of arrays of value as well. Infact any type of data can be stored as values, and nothing requires to be pre-defined. In other words, you do not have to predefine the type of data to be stored, you can store anything you want. Remember, MongoDB is schema-less.

Documents are not identified by a simple ID, but by an object identifier type. The default Id is a combination of machine identifier, timestamp and process id to keep it unique, but user can changes it to anything.

Example of Array as value

```
{ _id : 112233,
  name : "Viraj",
  education : [
             {
                year : 2029,
                course : "BTECH",
                college : "IIT, Delhi"
              },
             {
                year : 2031,
                course : "MS",
                college : "Harvard College"
             }
  ]
}
```

Example of Different Datatypes in one Document

The value of fields in a document can be anything, including other documents, arrays, and arrays of documents, date object, a String etc.

```
var mydoc = {
    _id : ObjectId("5099803df3f4948bd2f98391"),
    name : { first: "Alan", last: "Turing" },
    birth : new Date('Jun 23, 1912'),
    death : new Date('Jun 07, 1954'),
    contribs : [ "Turing machine", "Turing test", "Turingery" ],
    view : NumberLong(1250000)
}
```

MongoDB vs SQL Databases

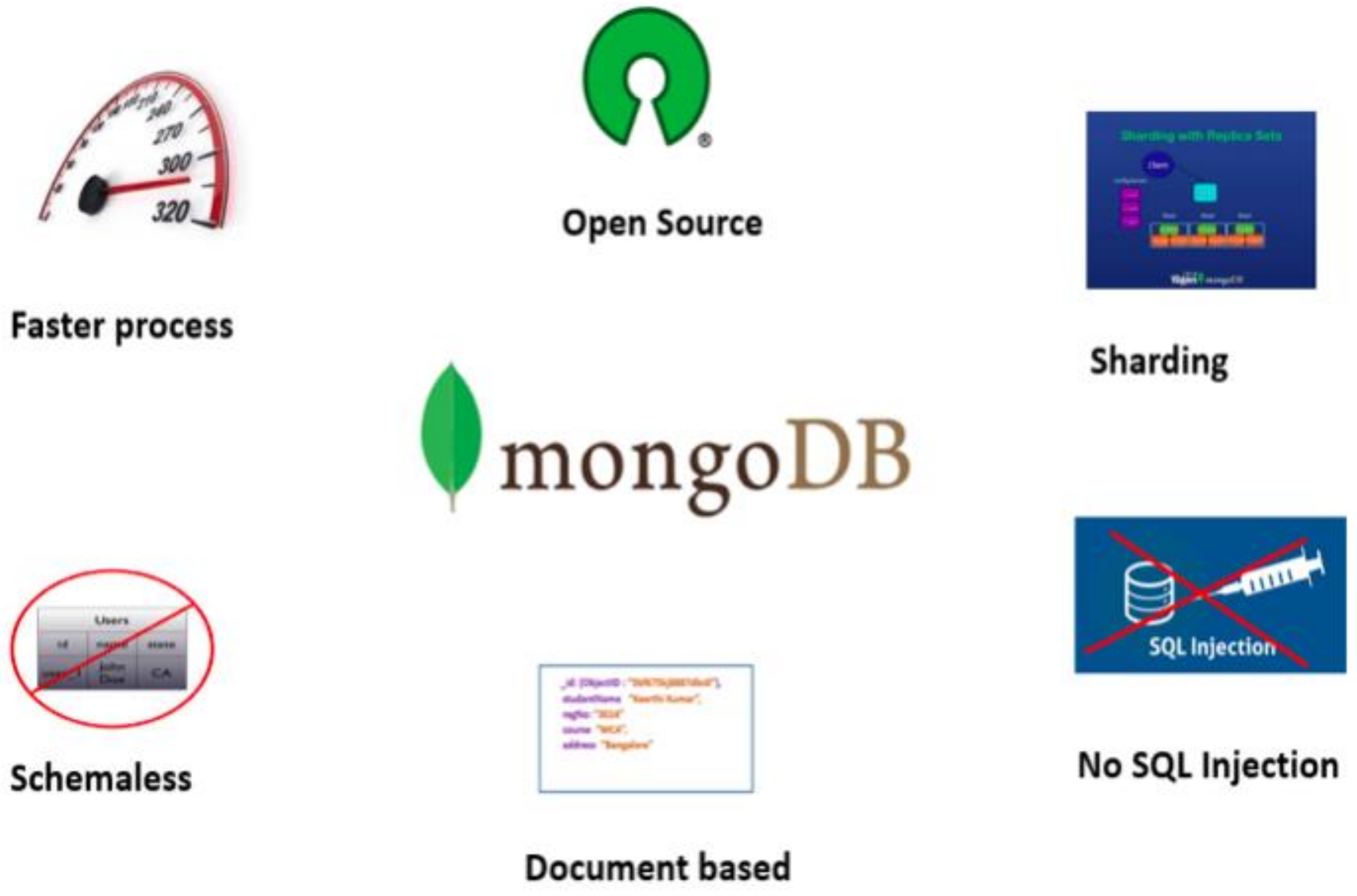
The major difference between MongoDB and SQL Databases is the way they handle data. In SQL databases, data is stored in form of traditional 2 dimensional row-column structure while in MongoDB rich data document model is followed, which allows storage of any type of data.

Let us see some of the key differences between MongoDB and other SQL databases :

SQL Database	**NoSQL Database (MongoDB)**
Relational database	Non-relational database
Supports SQL query language	Supports JSON query language
Table based	Collection based and key-value pair
Row based	Document based
Column based	Field based
Support foreign key	No support for foreign key
Support for triggers	No Support for triggers
Contains schema which is predefined	Contains dynamic schema
Not fit for hierarchical data storage	Best fit for hierarchical data storage
Vertically scalable - increasing RAM	Horizontally scalable - add more servers
Emphasizes on ACID properties (Atomicity, Consistency, Isolation and Durability)	Emphasizes on CAP theorem (Consistency, Availability and Partition tolerance)

Advantages of MongoDB

Let us see some of the vital advantages of MongoDB :



1. First and foremost, it is very easy to install and setup the MongoDB.
2. The very basic feature of MongoDB is that it is a schema-less database. No schema migrations anymore. Since MongoDB is schema-free, your code defines your schema.
3. The ability to derive a document-based data model is one of the most attractive advantages of MongoDB. Because, the way it stores the data in the form of BSON (Binary JSON), ruby hashes etc, helps to store the data in a very rich way while being capable of holding arrays and other documents.
4. The document query language supported by MongoDB plays a vital role in supporting dynamic queries.
5. Very easy to scale.
6. Due to the structuring (BSON format - key value pair) way of the data in MongoDB, no complex joins are needed.
7. Performance tuning is absolutely easy compared to any relational databases.
8. No need of mapping the application objects to the data objects.
9. Enables faster access of the data due to its nature of using the internal memory for the storage.
10. Since, it is a NOSQL database, then it is obviously secure because no sql injection can be made.
11. MongoDB can also be used as a file system, which helps in easier way of load balancing.
12. MongoDB supports, the search by regex and fields as well.
13. MongoDB can be run as windows service as well.
14. Good amount of documentation is available.
15. MongoDB does not require a VM to be run.
16. MongoDB follows regular release cycle of its newer versions.
17. The support for Sharding is one of its key feature. Sharding is the process of storing the data in different machines and MongoDB's ability to process the data, as and when the size of the data grows. This results in the horizontal scaling. With sharding, more amount of data can be written and read back as and when there is an increase in the data growth.

Datatypes in MongoDB

MongoDB supports the below list of datatypes. Each datatype in MongoDB possess a unique number.

Datatype	Number	Description
Double	1	Used to stored floating point values
String	2	Commonly used datatype and it is UTF-8 valid
Object	3	Used for storing embedded objects
Array	4	Used for storing embedded objects
Binary Data	5	Used to store binary data
Undefined	6	Used to store undefined value
Object Id	7	Used to store document's ID
Boolean	9	Used to store Boolean value
Date	10	Used to store current date time in UNIX format.
Null	11	Used to store null value
Regular Expression	12	Used to store regex
Javascript	13	Used to store JavaScript data without scope
Symbol	14	Basically used to store string, but reserved for languages that use specific symbol
Javascript with scope	15	Used to store JavaScript data with scope
Integer	16 & 18	Used to store numerical value

Timestamp	10	Used to track when a document is modified.
Min/Max Key	255/127	Used to compare value against lowest and highest BSON elements

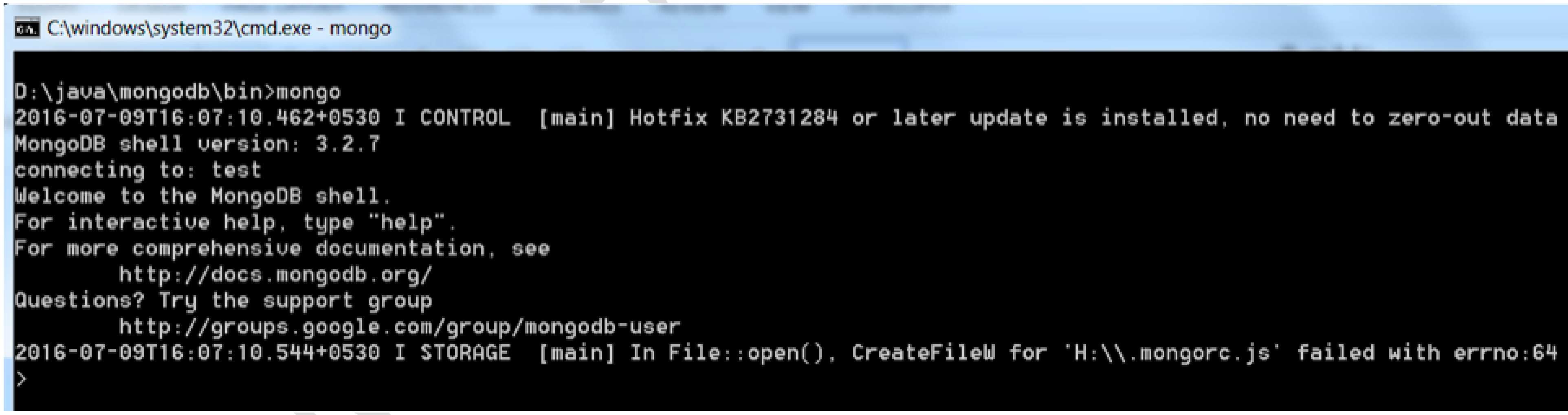
Create and Drop Database in MongoDB

In this tutorial we will learn how to create and drop a Database in MongoDB.

MongoDB: Creating a Database

Open the command prompt and navigate to the **/bin** folder of the MongoDB using the `cd` command and execute the command `mongod` there. This will initiate the MongoDB server. We have to keep this command prompt window alive, as this is running MongoDB. To stop the MongoDB server, simply enter `exit` and press Enter.

Now, Open another command prompt and navigate to the **/bin** folder of the MongoDB again and execute the command `mongo`. This will open up the client to run the MongoDB commands.

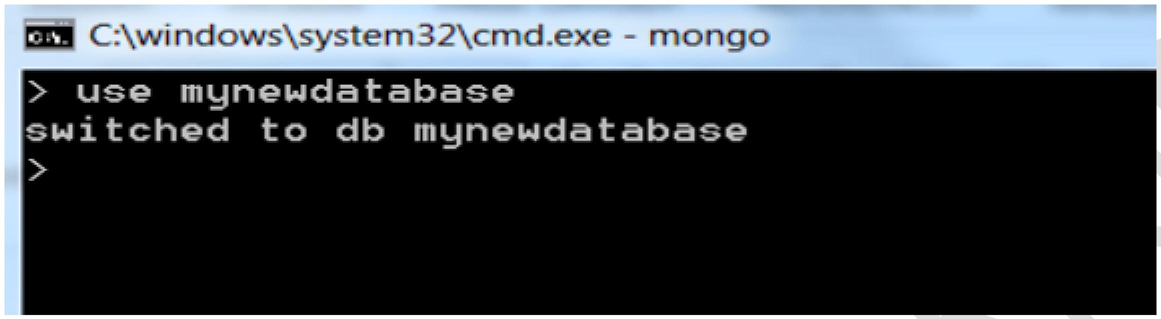


```
D:\java\mongodb\bin>mongo
2016-07-09T16:07:10.462+0530 I CONTROL  [main] Hotfix KB2731284 or later update is installed, no need to zero-out data
MongoDB shell version: 3.2.7
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
2016-07-09T16:07:10.544+0530 I STORAGE  [main] In File::open(), CreateFileW for 'H:\\.mongorc.js' failed with errno:64
>
```

In the command prompt window in which we ran the `mongo` command, after successfully connecting to the mongodb, just type the command the following :

```
use database_name
```

This will create a new database with the name **database_name** if there is no database already present with the same name. If a database already exists with the mentioned name, then it just connects to that database.

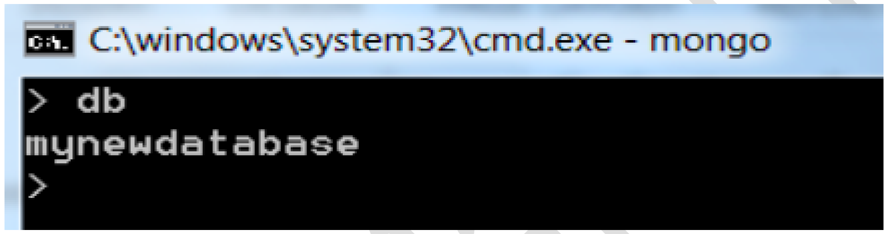


C:\windows\system32\cmd.exe - mongo

```
> use mynewdatabase
switched to db mynewdatabase
>
```

In the above picture, it creates a new database called **mynewdatabase** and will also connect to the same.

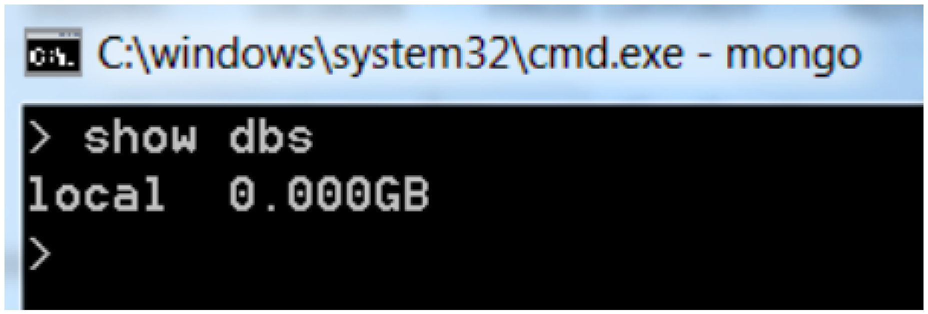
To check the current connected database, type in `db` in the command prompt window, and you will get the name of the current database as result.



C:\windows\system32\cmd.exe - mongo

```
> db
mynewdatabase
>
```

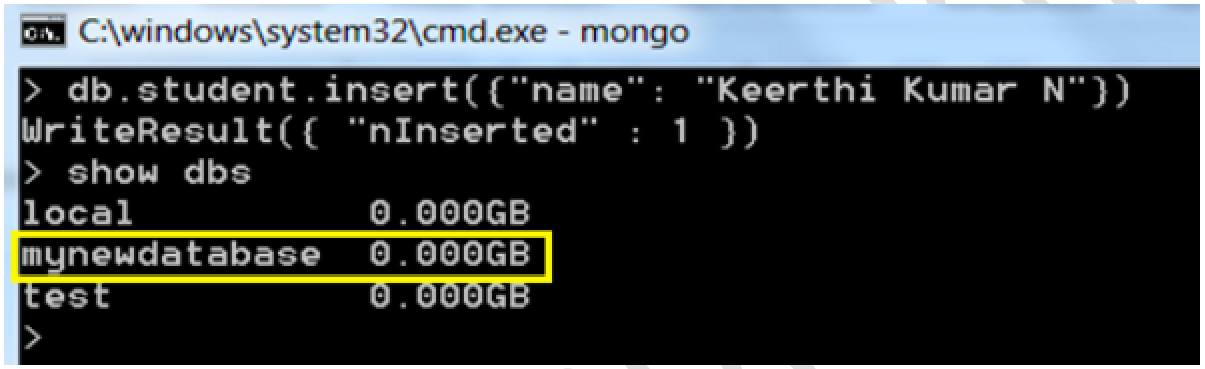
To see the list of all the databases in MongoDB, use command `show dbs`



C:\windows\system32\cmd.exe - mongo

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

Please note that the newly created database **mynewdatabase** has not been listed after running the above command. This is because, no records have been inserted into that database yet. Just insert one record and then run the command again as shown below:



C:\windows\system32\cmd.exe - mongo

```
> db.student.insert({"name": "Keerthi Kumar N"})
WriteResult({ "nInserted" : 1 })
> show dbs
local          0.000GB
mynewdatabase  0.000GB
test           0.000GB
>
```

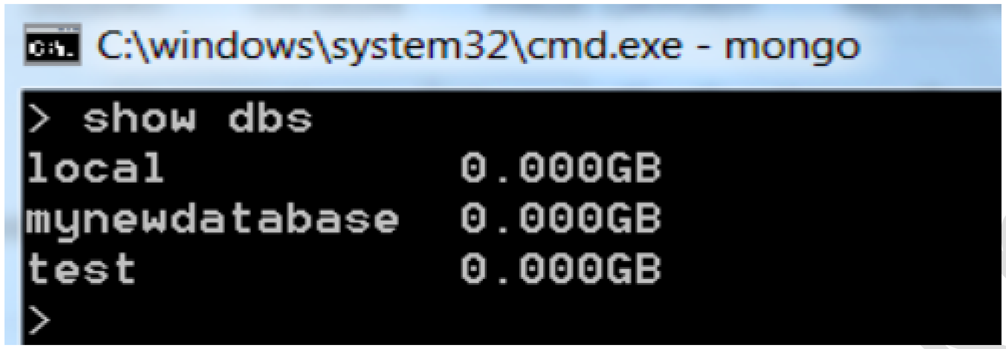
To Insert data, run the following command. Dont worry about it, we will learn this in detail in next lessons.

```
db.student.insert({name : "Viraj" })
```

NOTE: In MongoDB, `test` will be the default database. If no database is created, then all the data will be stored in the `test` database.

MongoDB: Drop a Database

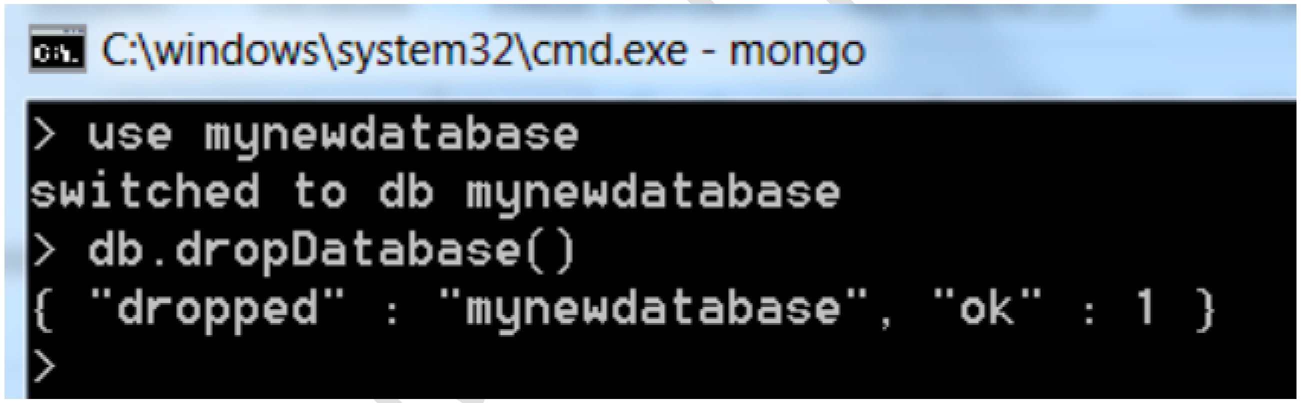
First check the list of databases available as shown below, using the `show dbs` command.



```
C:\windows\system32\cmd.exe - mongo
> show dbs
local            0.000GB
mynewdatabase    0.000GB
test             0.000GB
>
```

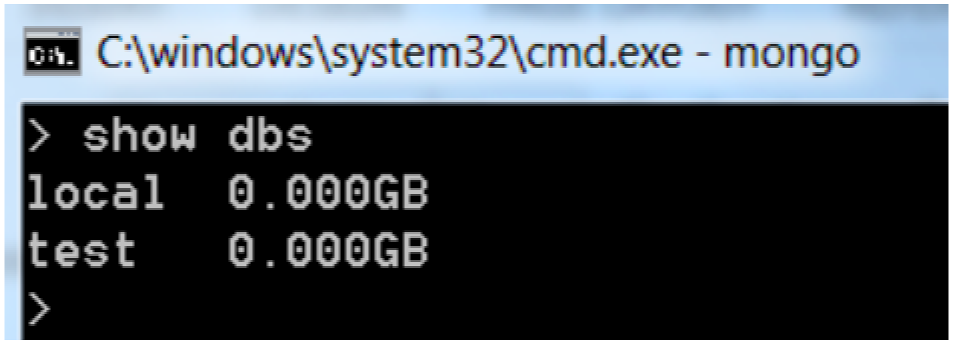
If the newly created database **mynewdatabase** has to be dropped(deleted). Run the below command to delete the database. Before deleting the database, connect to the required database which is to be deleted.

```
db.dropDatabase()
```



```
C:\windows\system32\cmd.exe - mongo
> use mynewdatabase
switched to db mynewdatabase
> db.dropDatabase()
{ "dropped" : "mynewdatabase", "ok" : 1 }
>
```

Now again check the list of databases, to verify whether the database is deleted or not.



C:\windows\system32\cmd.exe - mongo

```
> show dbs
local   0.000GB
test    0.000GB
>
```

Note that the database **mynewdatabase** has been deleted and hence, it will not be listed in the list of the databases.

MongoDB: Creating a Collection

In MongoDB a collection is automatically created when it is referenced in any command. For example, if you run an insert command :

```
db.student.insert({
        name: "Viraj"
})
```

Above command will create a collection named **student** if it doesn't exist already in the database. But we can explicitly create a collection using the `createCollection()` command.

MongoDB: Drop a Collection

Any collection in a database in MongoDB can be dropped easily using the following command:

```
db.collection_name.drop()
```

`drop()` method will return true is the collection is dropped successfully, else it will return false.

CRUD Operations in MongoDB

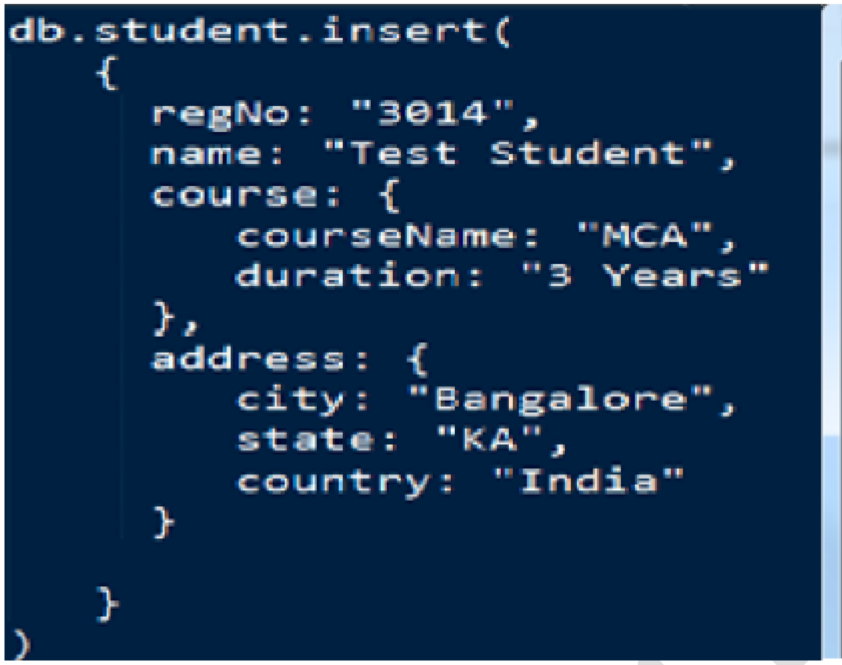
CRUD operations refer to the basic Insert, Read, Update and Delete operations. In the previous chapter, we learnt about how to create a database and drop the database in MongoDB. Now, let us learn how to perform CRUD (Create/Read/Update/Delete) operations in MongoDB.

MongoDB: Inserting a document into a collection(Create)

The command `db.collection.insert()` will perform an insert operation into a collection of a document.

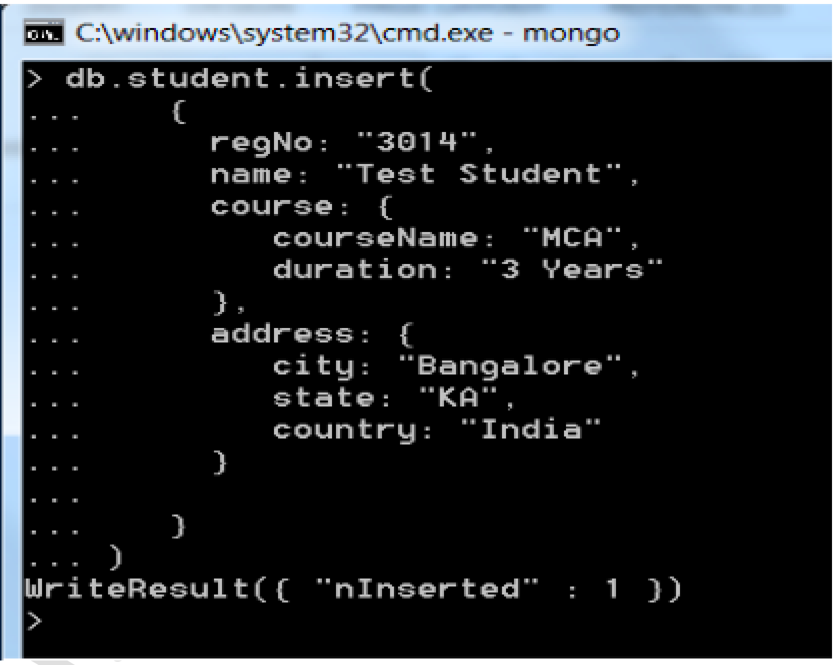
Let us insert a document to a `student` collection. You must be connected to a database for doing any insert. It is done as follows:

```
db.student.insert({
        regNo: "3014",
        name: "Test Student",
        course: {
                courseName: "MCA",
                duration: "3 Years"
        },
        address: {
                city: "Bangalore",
                state: "KA",
                country: "India"
        }
})
```



```
db.student.insert(
    {
      regNo: "3014",
      name: "Test Student",
      course: {
          courseName: "MCA",
          duration: "3 Years"
      },
      address: {
          city: "Bangalore",
          state: "KA",
          country: "India"
      }

    }
)
```



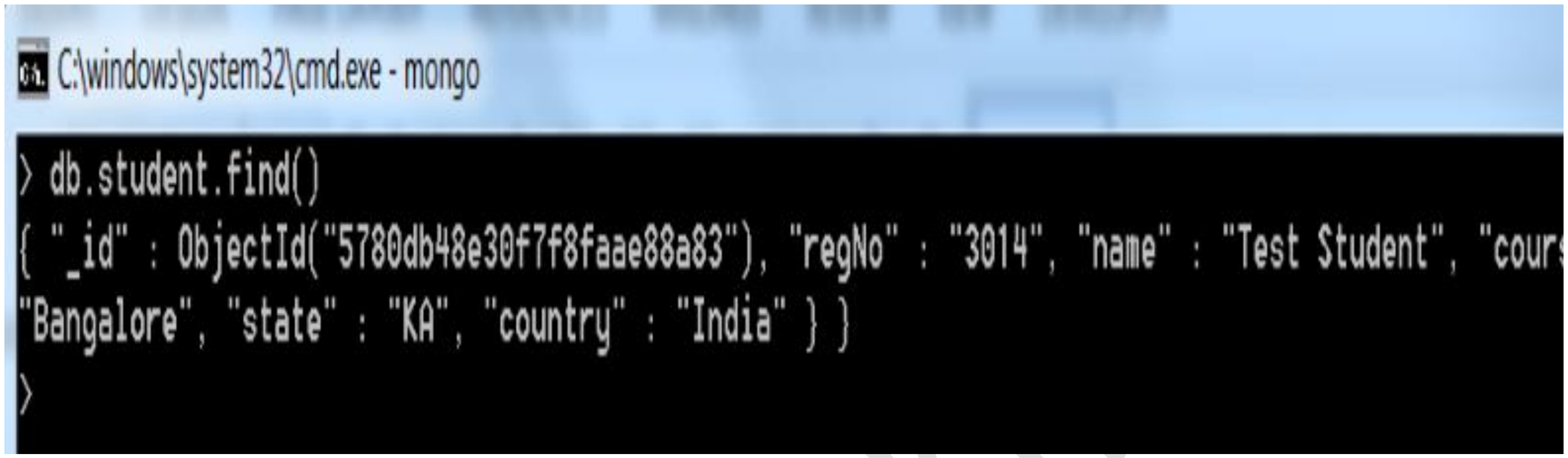
```
> db.student.insert(
...     {
...       regNo: "3014",
...       name: "Test Student",
...       course: {
...           courseName: "MCA",
...           duration: "3 Years"
...       },
...       address: {
...           city: "Bangalore",
...           state: "KA",
...           country: "India"
...       }
...
...     }
... )
WriteResult({ "nInserted" : 1 })
>
```

Note that an entry has been made into the collection called student.

MongoDB: Querying a document from a collection(Read)

To retrieve (Select) the inserted document, run the below command. The `find()` command will retrieve all the documents of the given collection.

```
db.collection_name.find()
```



```
> db.student.find()
{ "_id" : ObjectId("5780db48e30f7f8faae88a83"), "regNo" : "3014", "name" : "Test Student", "cour
"Bangalore", "state" : "KA", "country" : "India" } }
>
```

NOTE: Please observe that the record retrieved contains an attribute called `_id` with some unique identifier value called **ObjectId** which acts as a document identifier.

If a record is to be retrieved based on some criteria, the `find()` method should be called passing parameters, then the record will be retrieved based on the attributes specified.

```
db.collection_name.find({"fieldname":"value"})
```

For Example: Let us retrieve the record from the **student** collection where the attribute **regNo** is **3014** and the query for the same is as shown below:

```
db.students.find({"regNo":"3014"})
```

MongoDB: Updating a document in a collection(Update)

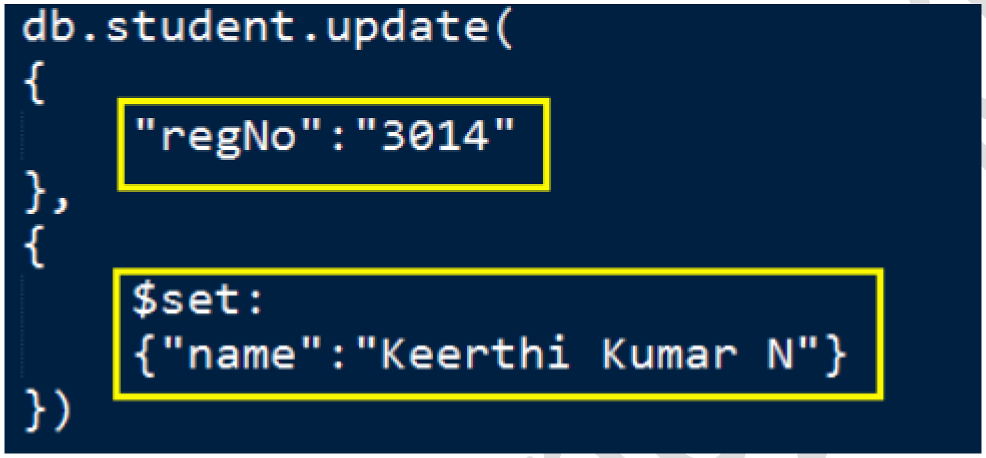
In order to update specific field values of a collection in MongoDB, run the below query.

```
db.collection_name.update()
```

`update()` method specified above will take the fieldname and the new value as argument to update a document.

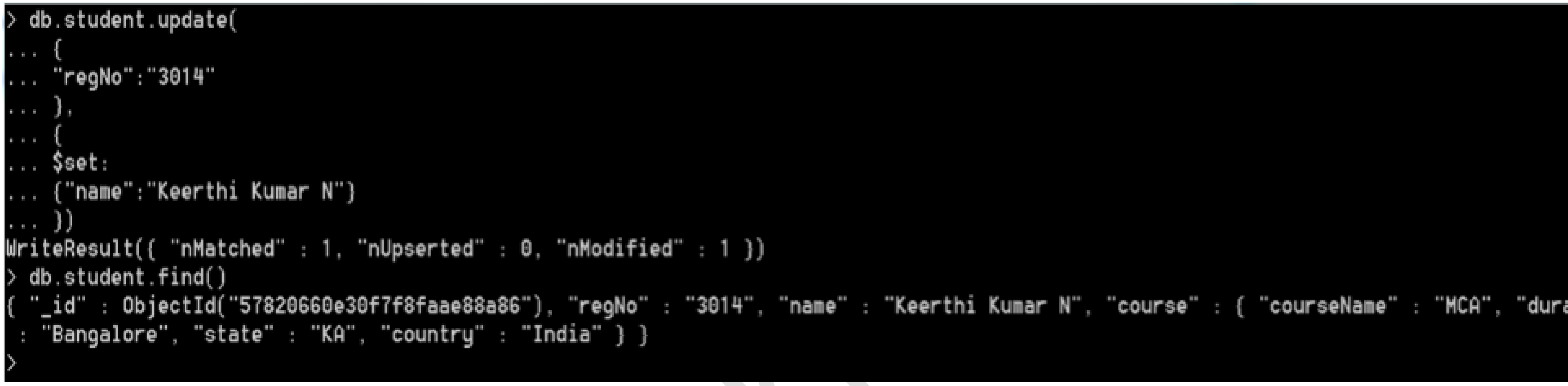
Let us update the attribute **name** of the collection **student** for the document with **regNo** 3014.

```
db.student.update({
        "regNo": "3014"
},
$set:
{
        "name":"Viraj"
})
```



```
db.student.update(
{
      "regNo":"3014"
},
{
      $set:
      {"name":"Keerthi Kumar N"}
})
```

You will see the following in the Command Prompt:

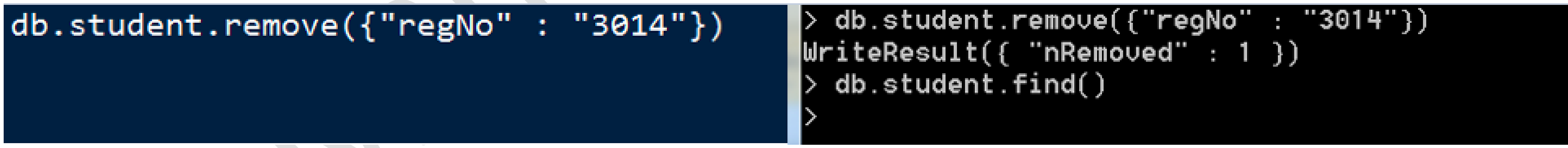

```
> db.student.update(
... {
... "regNo":"3014"
... },
... {
... $set:
... {"name":"Keerthi Kumar N"}
... })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find()
{ "_id" : ObjectId("57820660e30f7f8faae88a86"), "regNo" : "3014", "name" : "Keerthi Kumar N", "course" : { "courseName" : "MCA", "dura
 : "Bangalore", "state" : "KA", "country" : "India" } }
>
```

MongoDB: Removing an entry from the collection(Delete)

Let us now look into the deleting an entry from a collection. In order to delete an entry from a collection, run the command as shown below:

`db.collection_name.remove({"fieldname":"value"})`

For Example: `db.student.remove({"regNo":"3014"})`


```
db.student.remove({"regNo" : "3014"})
```
```
> db.student.remove({"regNo" : "3014"})
WriteResult({ "nRemoved" : 1 })
> db.student.find()
>
```

Note that after running the `remove()` method, the entry has been deleted from the student collection.

Indexing in MongoDB

Indexes in SQL programming are nothing but a special data structure used to easily and quickly locate the record in a given table of the database without being required to traverse through each and every record of the table. Indexes are easily generated using one or more columns of a given table. As a note, the data structure used by an index is a **Binary Tree** (B-Tree).

In MongoDB, indexes plays a vital role in efficiently execution of the queries. Basically, if no index is defined in MongoDB, then it has to scan every document of a given collection. Hence, MongoDB uses index to reduce the number of documents to be scanned in a given collection. In fact, MongoDB's index is more or less similar to the indexes used in other relational databases.

The fact is that the MongoDB defines the indexes at the collection level and supports indexing on any fields in a MongoDB collection.

Default Index in MongoDB

Mongodb provides a default index named `_id` which acts as a primary key to access any document in a collection. This `_id` index basically avoids the insertion of 2 documents with the same value for the `_id` field.

Creating an Index using `createIndex()`

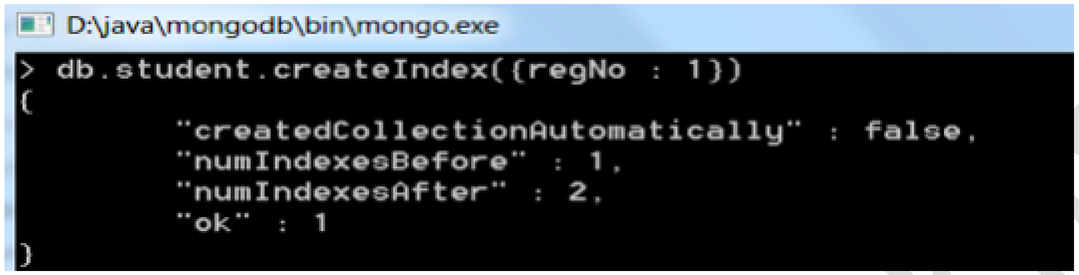
To create an index in MongoDB, run the following command :

```
db.collection_name.createIndex({field : value })
```

To create an index on the field `regNo` for a student collection, run the command `db.student.createIndex({regNo : 1})`

Following will be the output upon running the above command :

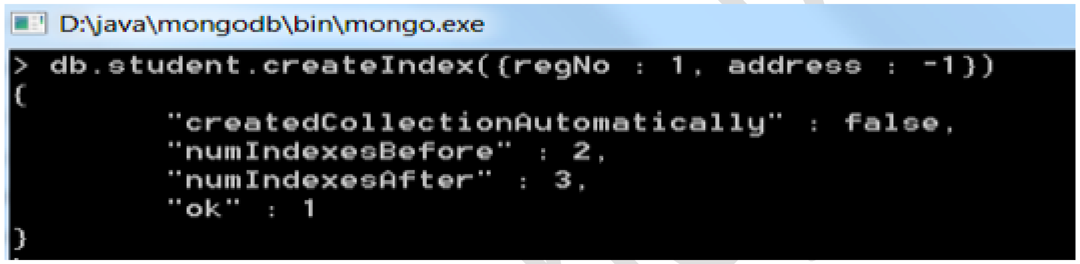
```
{
        "createdCollectionAutomatically": false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```



```
D:\java\mongodb\bin\mongo.exe
> db.student.createIndex({regNo : 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

We can also create Index on multiple fields by running a single command. The command will be :

```
db.student.createIndex({regNo : 1, address : -1})
```



```
D:\java\mongodb\bin\mongo.exe
> db.student.createIndex({regNo : 1, address : -1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
```

Sorting in MongoDB

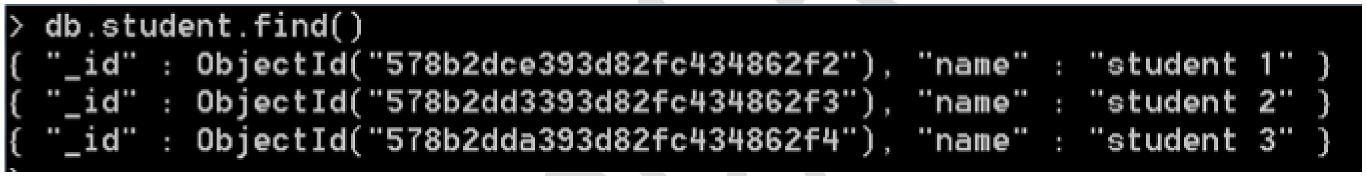
Sorting the data in any database is one of the vital operations in any database management system. MongoDB provides `sort()` function in order to sort the data in a collection. Sort function in MongoDB accepts a list of values and an integer value 1 or -1 which states whether the collection to be sorted in ascending (1) or descending (-1) order.

Syntax for sort function:

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

Consider a collection named **student** containing 3 records. Let us now see how the data can be sorted using the `sort()` function in MongoDB.

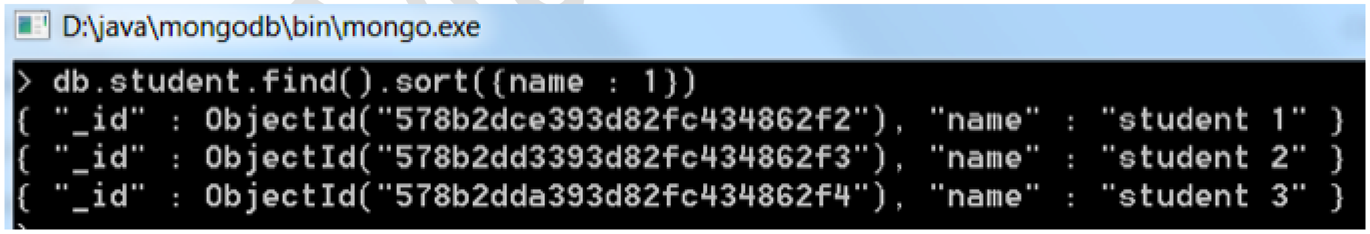
To list down all the data in a collection, use the `find()` command. To create the same sample data as used here in the example, create a collection named `student` and insert 3 documents with one field **name** and some value for it. In the next step we will run the sort command on this sample data.



```
> db.student.find()
{ "_id" : ObjectId("578b2dce393d82fc434862f2"), "name" : "student 1" }
{ "_id" : ObjectId("578b2dd3393d82fc434862f3"), "name" : "student 2" }
{ "_id" : ObjectId("578b2dda393d82fc434862f4"), "name" : "student 3" }
```

Now run the below query to sort the data by the **name** field in ascending order :

```
db.student.find().sort({name : 1})
```

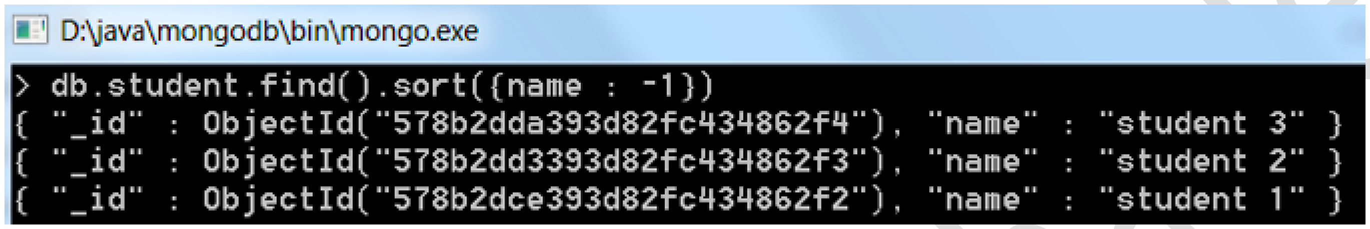


D:\java\mongodb\bin\mongo.exe

```
> db.student.find().sort({name : 1})
{ "_id" : ObjectId("578b2dce393d82fc434862f2"), "name" : "student 1" }
{ "_id" : ObjectId("578b2dd3393d82fc434862f3"), "name" : "student 2" }
{ "_id" : ObjectId("578b2dda393d82fc434862f4"), "name" : "student 3" }
```

Now run the below query to sort the data by the **name** field in descending order :

```
db.student.find().sort({name : -1})
```



D:\java\mongodb\bin\mongo.exe

```
> db.student.find().sort({name : -1})
{ "_id" : ObjectId("578b2dda393d82fc434862f4"), "name" : "student 3" }
{ "_id" : ObjectId("578b2dd3393d82fc434862f3"), "name" : "student 2" }
{ "_id" : ObjectId("578b2dce393d82fc434862f2"), "name" : "student 1" }
```

NOTE: If you do not specify the sorting preference(i.e 1 or -1), then by default documents in a collection are sorted in ascending order.

Aggregation in MongoDB

Aggregation in MongoDB is nothing but an operation used to process the data that returns the computed results. Aggregation basically groups the data from multiple documents and operates in many ways on those grouped data in order to return one combined result. In sql `count(*)` and with **group by** is an equivalent of MongoDB aggregation.

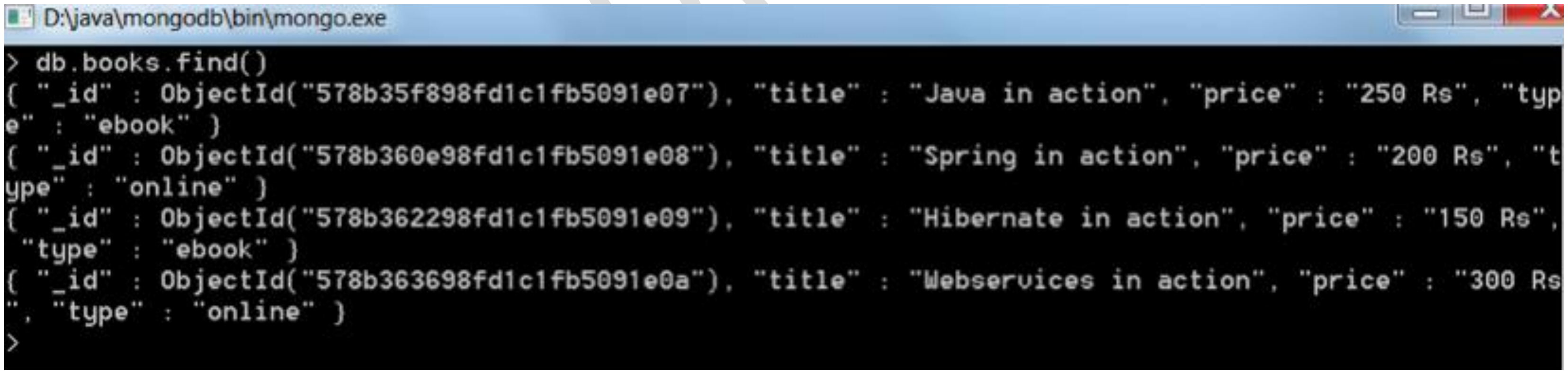
Aggregate function groups the records in a collection, and can be used to provide total number(sum), average, minimum, maximum etc out of the group selected.

In order to perform the aggregate function in MongoDB, `aggregate ()` is the function to be used. Following is the syntax for aggregation :

```
db.collection_name.aggregate(aggregate_operation)
```

Let us now see how to make use of the aggregate function in MongoDB. Consider a collection named **books** with the data as shown below :

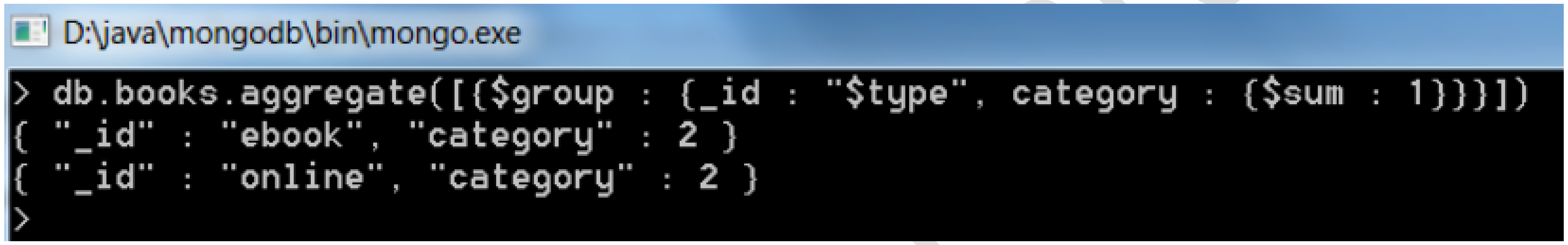
NOTE : Create a collection named **books**, with fieldnames - title, price and type. Use `db.books.find()` to list down the contents of the collection.



```
> db.books.find()
{ "_id" : ObjectId("578b35f898fd1c1fb5091e07"), "title" : "Java in action", "price" : "250 Rs", "typ
e" : "ebook" }
{ "_id" : ObjectId("578b360e98fd1c1fb5091e08"), "title" : "Spring in action", "price" : "200 Rs", "t
ype" : "online" }
{ "_id" : ObjectId("578b362298fd1c1fb5091e09"), "title" : "Hibernate in action", "price" : "150 Rs",
 "type" : "ebook" }
{ "_id" : ObjectId("578b363698fd1c1fb5091e0a"), "title" : "Webservices in action", "price" : "300 Rs
", "type" : "online" }
>
```

Now, from the above collection, let us use the aggregate function to **group** the books which are of the type **ebook** and **online**. Following command will be used :

```
db.books.aggregate([{$group : {_id: "$type", category: {$sum : 1}}}])
```



```
D:\java\mongodb\bin\mongo.exe
> db.books.aggregate([{$group : {_id : "$type", category : {$sum : 1}}}])
{ "_id" : "ebook", "category" : 2 }
{ "_id" : "online", "category" : 2 }
>
```

The above aggregate function will give result, which says there are 2 records of the type *ebook* and 2 records of the type *online*. So the above aggregation command, has grouped our collection data, based on their **type**.

Different expressions used by Aggregate function

Expression	**Description**
$sum	Summates the defined values from all the documents in a collection
$avg	Calculates the average values from all the documents in a collection
$min	Return the minimum of all values of documents in a collection
$max	Return the maximum of all values of documents in a collection
$addToSet	Inserts values to an array but no duplicates in the resulting document

$push	Inserts values to an array in the resulting document
$first	Returns the first document from the source document
$last	Returns the last document from the source document