mongoDB

Leading No Sql Database

RamaSagar Pulidindi

C#Corner

Introduction to Mongo DB

RamaSagar
.NET developer

Sam Hobbs
Editor, C# Corner

Abstract

Mongo DB (from "humongous") is a cross-platform document-oriented database system. Classified as a NoSQL database, Mongo DB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (Mongo DB calls the format BSON), making the integration of data in certain types of applications easier and faster.

It has been written in C++ as a document-oriented database, so it manages collections of JSON-like documents.

Mongo DB supports cross-platform support (Windows, Linux, Solaris). It also has a rich set of data types (supports dates, regular expressions, code, and binary data).

Mongo DB uses memory mapped files. For Windows the data size is limited to 2GB on 32-bit machines (64-bit systems have a much larger data size).

Note:

- Most resources discussed in this paper are provided with the Mongo DB package. For a complete list of documents and references discussed, see “Resources and References” at the end of this document.
- for up-to-date documentation, Mongo DB news, and online community, see http://www.mongodb.org/

Contents

Introduction to the Mongo Database

Databases are important of every internet and enterprise applications

For scale speed and fast application development has brought on a new breed of databases broadly turned to no sql databases

This tutorial introduces the basic features of the Mongo DB with a series of exercises that show

- What no sql means versus relational databases.
- We will get the mongo server up and running
- We will know how to connect to database manipulate some data save & update and
- We will see how to use indexing to speed up the query time and review some indexing strategies...and some miscellaneous points along the series...

The main pros of Mongo DB can be divided in the following three:

- Flexibility
- Performance
- Scalability

The last section of this tutorial provides a brief exercise to introduce you to Mongo DB with Asp.net. It shows how to perform CRUD operations

Prerequisites

To take advantage of this tutorial, you should be familiar with the following:

- Microsoft® Visual Studio® 2010
- C# programming language
- NET Framework
- Basic practices for building, debugging, and testing software

Computer Configuration

These tutorials require that the following software components are installed:

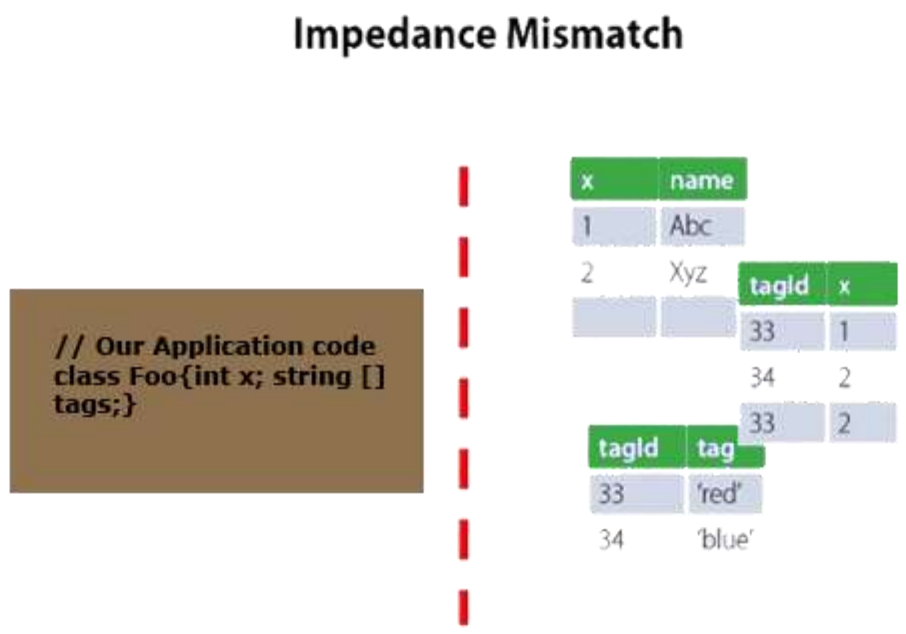
- Windows® 7, Windows Vista®, or Windows Server® 2008 R2 or later operating system
- Visual Studio Professional 2010 or later

Getting Help

- For questions, see "Resources and References" at the end of this document.
- If you have a question, post it on the C# corner forum.

Mongo DB vs RDBMS

As developers we want database that is easy to use. Relational databases save data in tables and rows. Our application hardly ever does, this means alignment of application layer objects to tables and rows is called impedance mismatch as shown below...

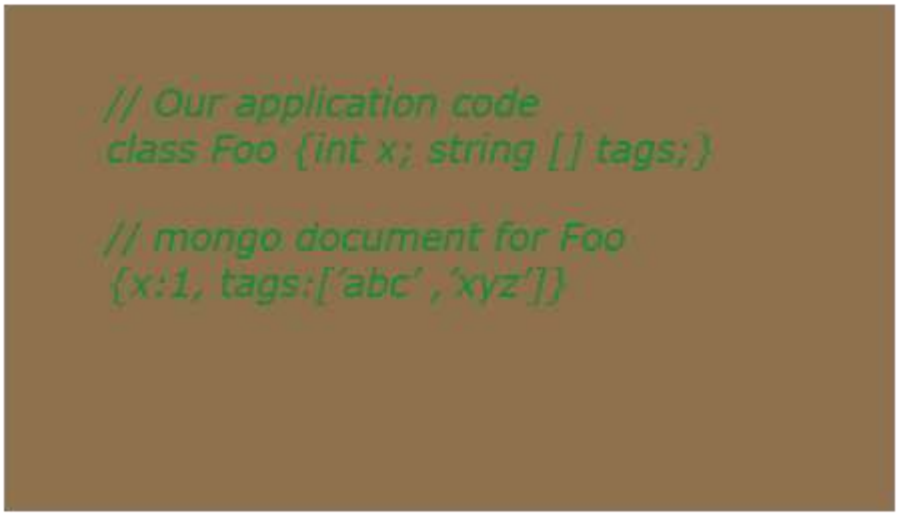


In the above figure we can notice that it has an object containing a field x and a field tags which contains bunch of strings...if we want to save in relational database we will create three tables one for the main object and other for the tags and other to map the tags to the main object. This forces us to write a mapping layer or use an ORM to translate between our object and memory and save into the database...

Many of us develop applications using object oriented concepts. Our objects are not simply tables and rows. And we may use polymorphism or inheritance our objects are not uniform...mapping those into table and rows are quite a bit of pain...

In mongo dB there is no schema to define...there are no tables and no relationships between collections of objects, every document we save in mongo can be flat and simple and as complex as the application requires. This makes life as a developer much easier and the application code much cleaner and simpler...

No Impedance Mismatch



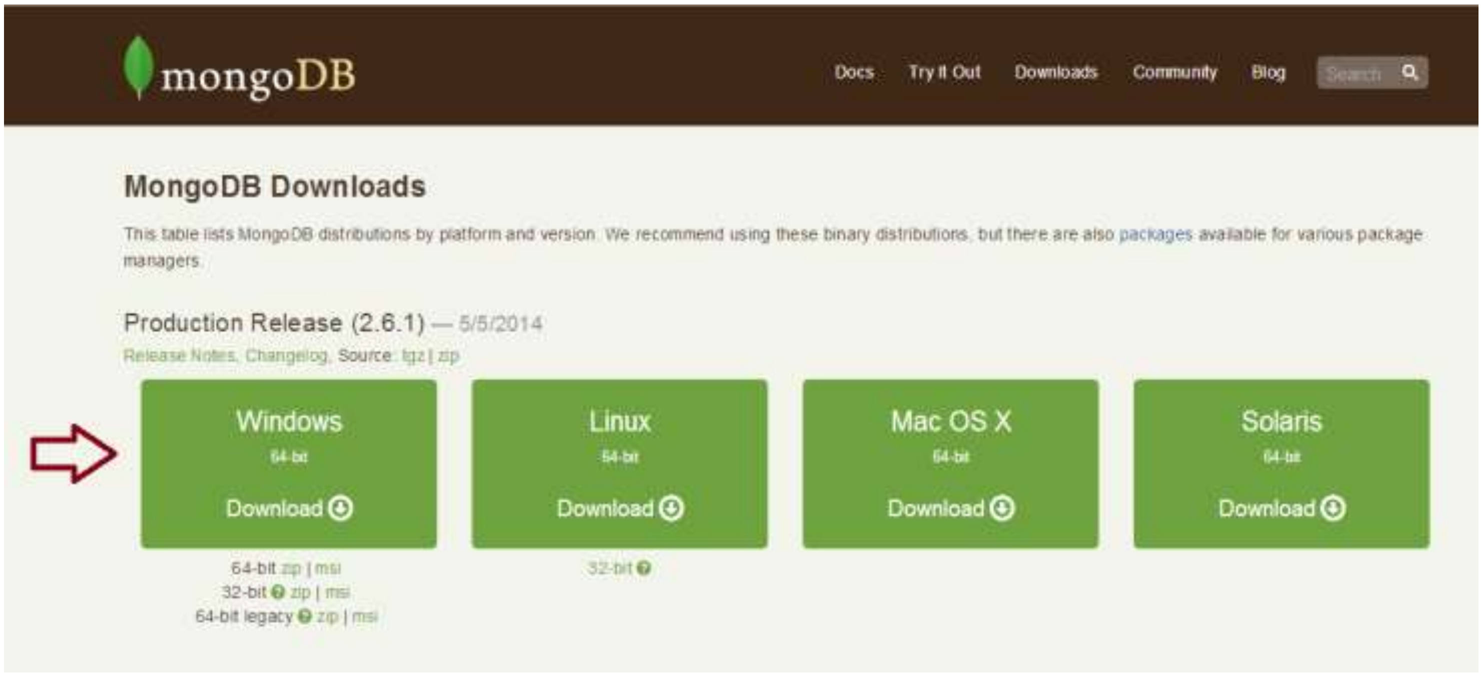
```
// Our application code
class Foo {int x; string [] tags;}

// mongo document for Foo
{x:1, tags:['abc' ,'xyz']}
```

- Mongo DB has no schema no tables no rows no columns and certainly no relationships between tables.
- In mongo DB we have single document write scope, a document lives in a collection but updating documents occur one at a time.so if any locking needed to occur it would be much simpler there is no need to extend locks across collections there are no relationships to enforce.
- Mongo also offers a special collection called capped collection which have a fixed size and automatically overwrites old documents.

Exercise 1: Setting up the server

Mongo DB is open source software which can be found on web through mongodb.org…From there we can download a distribution suitable to our platform. Mongo DB is written in C++…It's available in both 32 and 64 bit editions.

Open Mongodb.org click on downloads and select the suitable version



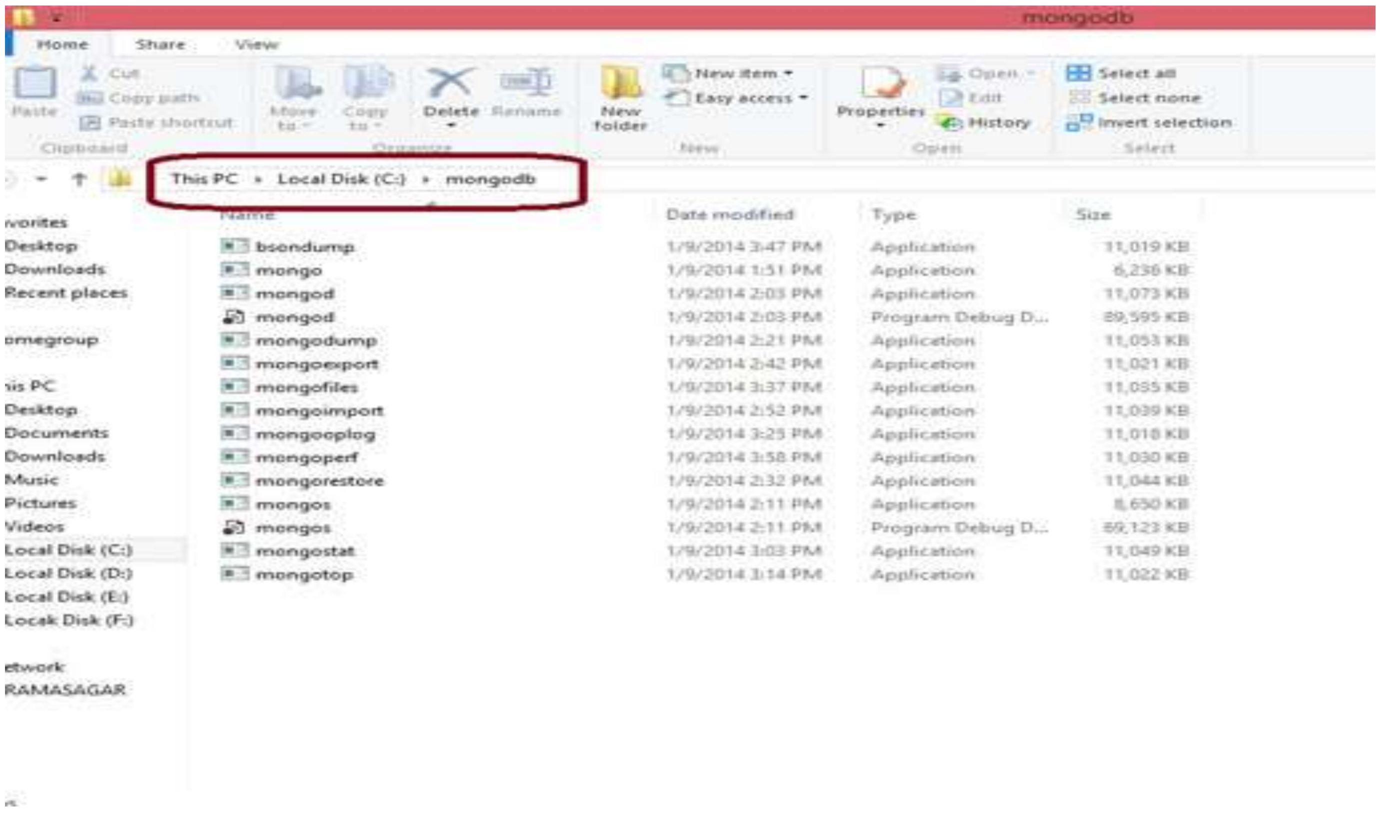
MongoDB Downloads

This table lists MongoDB distributions by platform and version. We recommend using these binary distributions, but there are also packages available for various package managers.

Production Release (2.6.1) — 5/5/2014

Release Notes, Changelog, Source: tgz | zip

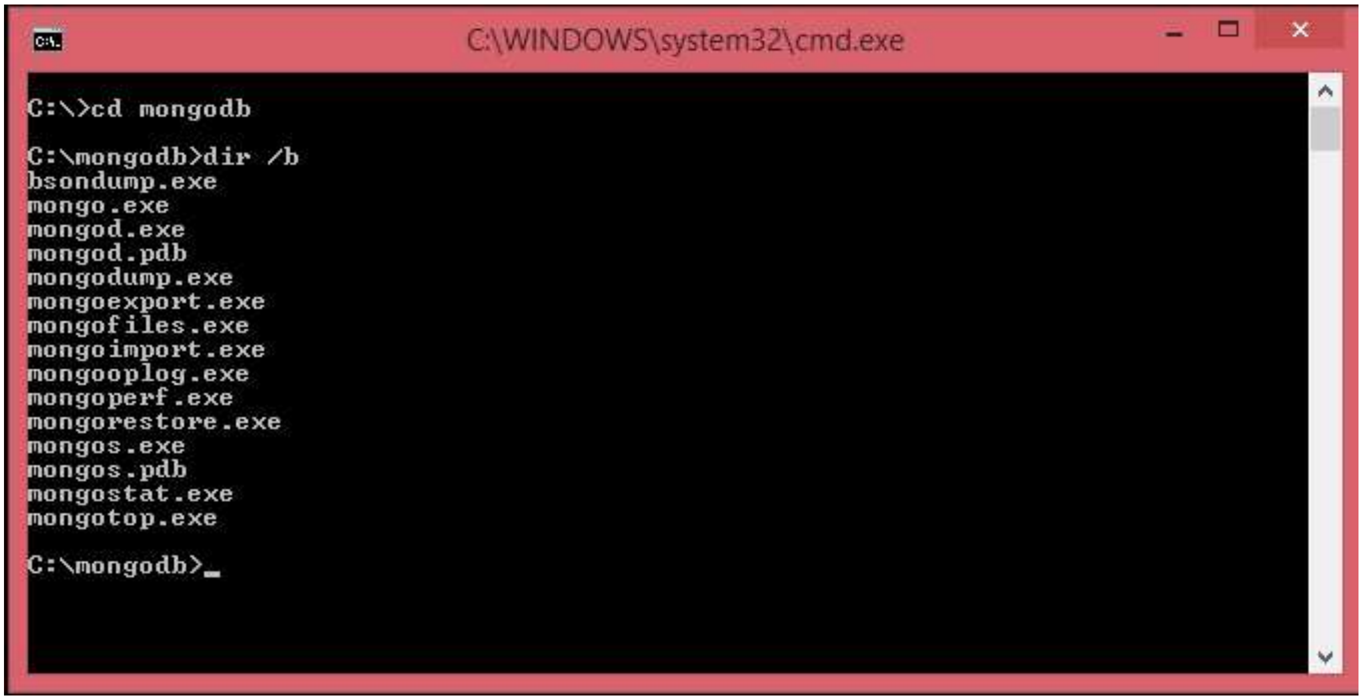
Unzip into a directory and we can find a bunch of executable files. Here a folder is created in C: drive and unzipped the files.



This PC ▸ Local Disk (C:) ▸ mongodb

Name	Date modified	Type	Size
bsondump	1/9/2014 3:47 PM	Application	11,019 KB
mongo	1/9/2014 1:51 PM	Application	6,236 KB
mongod	1/9/2014 2:03 PM	Application	11,073 KB
mongod	1/9/2014 2:03 PM	Program Debug D...	89,595 KB
mongodump	1/9/2014 2:21 PM	Application	11,053 KB
mongoexport	1/9/2014 2:42 PM	Application	11,021 KB
mongofiles	1/9/2014 3:37 PM	Application	11,035 KB
mongoimport	1/9/2014 2:52 PM	Application	11,039 KB
mongooplog	1/9/2014 3:25 PM	Application	11,018 KB
mongoperf	1/9/2014 3:58 PM	Application	11,030 KB
mongorestore	1/9/2014 2:32 PM	Application	11,044 KB
mongos	1/9/2014 2:11 PM	Application	8,650 KB
mongos	1/9/2014 2:11 PM	Program Debug D...	69,123 KB
mongostat	1/9/2014 3:03 PM	Application	11,049 KB
mongotop	1/9/2014 3:14 PM	Application	11,022 KB

Now open the cmd prompt and fetch into that directory



```
C:\>cd mongodb

C:\mongodb>dir /b
bsondump.exe
mongo.exe
mongod.exe
mongod.pdb
mongodump.exe
mongoexport.exe
mongofiles.exe
mongoimport.exe
mongooplog.exe
mongoperf.exe
mongorestore.exe
mongos.exe
mongos.pdb
mongostat.exe
mongotop.exe

C:\mongodb>_
```

Before we would run mongo server for first time we need to create a directory where the data files will reside .The default name for the directory is data...

Type the command

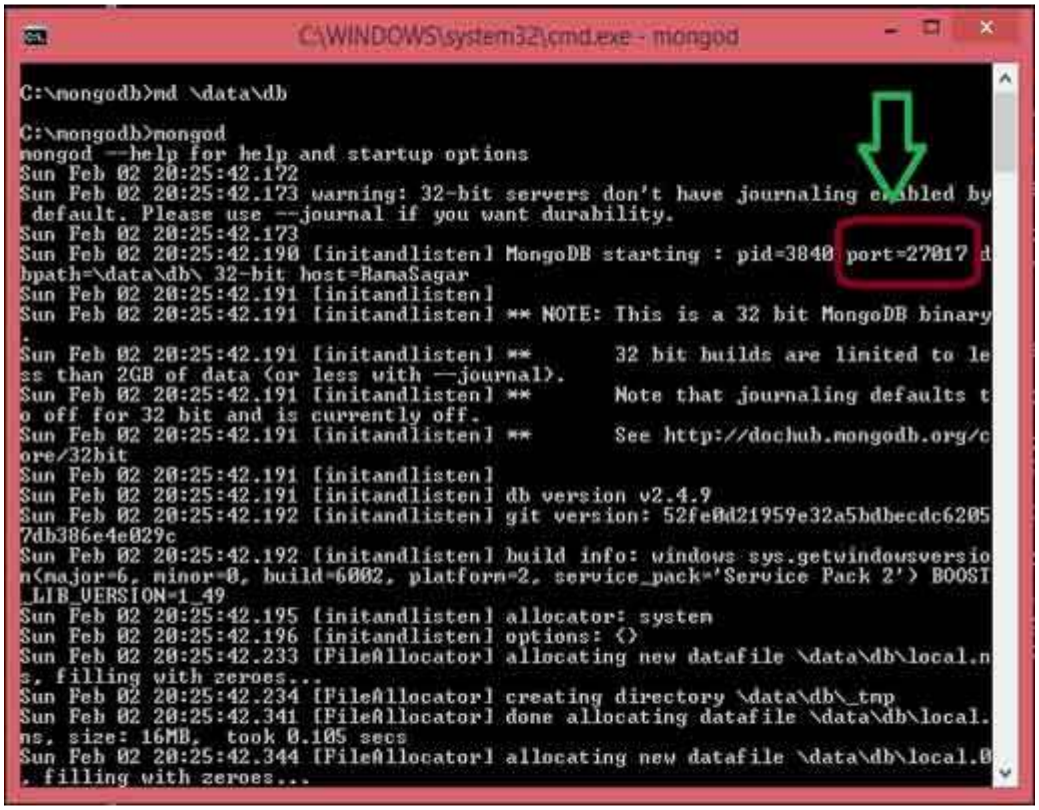
```
md \data\db
```

Once the directory exists we can simply run the mongo server without any more command line arguments...

Type: mongod

Where d is the daemon

Mongo has started initialize data in that directory and it is up and running open on port 27017. We can start it using now.



```
C:\mongodb>md \data\db

C:\mongodb>mongod
mongod --help for help and startup options
Sun Feb 02 20:25:42.172
Sun Feb 02 20:25:42.173 warning: 32-bit servers don't have journaling enabled by
 default. Please use --journal if you want durability.
Sun Feb 02 20:25:42.173
Sun Feb 02 20:25:42.190 [initandlisten] MongoDB starting : pid=3840 port=27017 d
bpath=\data\db\ 32-bit host=RamaSagar
Sun Feb 02 20:25:42.191 [initandlisten]
Sun Feb 02 20:25:42.191 [initandlisten] ** NOTE: This is a 32 bit MongoDB binary
.
Sun Feb 02 20:25:42.191 [initandlisten] **       32 bit builds are limited to le
ss than 2GB of data (or less with --journal).
Sun Feb 02 20:25:42.191 [initandlisten] **       Note that journaling defaults t
o off for 32 bit and is currently off.
Sun Feb 02 20:25:42.191 [initandlisten] **       See http://dochub.mongodb.org/c
ore/32bit
Sun Feb 02 20:25:42.191 [initandlisten]
Sun Feb 02 20:25:42.191 [initandlisten] db version v2.4.9
Sun Feb 02 20:25:42.192 [initandlisten] git version: 52fe0d21959e32a5bdbecdc6205
7db386e4e029c
Sun Feb 02 20:25:42.192 [initandlisten] build info: windows sys.getwindowsversio
n(major=6, minor=0, build=6002, platform=2, service_pack='Service Pack 2') BOOST
_LIB_VERSION=1_49
Sun Feb 02 20:25:42.195 [initandlisten] allocator: system
Sun Feb 02 20:25:42.196 [initandlisten] options: {}
Sun Feb 02 20:25:42.233 [FileAllocator] allocating new datafile \data\db\local.n
s, filling with zeroes...
Sun Feb 02 20:25:42.234 [FileAllocator] creating directory \data\db\_tmp
Sun Feb 02 20:25:42.341 [FileAllocator] done allocating datafile \data\db\local.
ns, size: 16MB,  took 0.105 secs
Sun Feb 02 20:25:42.344 [FileAllocator] allocating new datafile \data\db\local.0
, filling with zeroes...
```

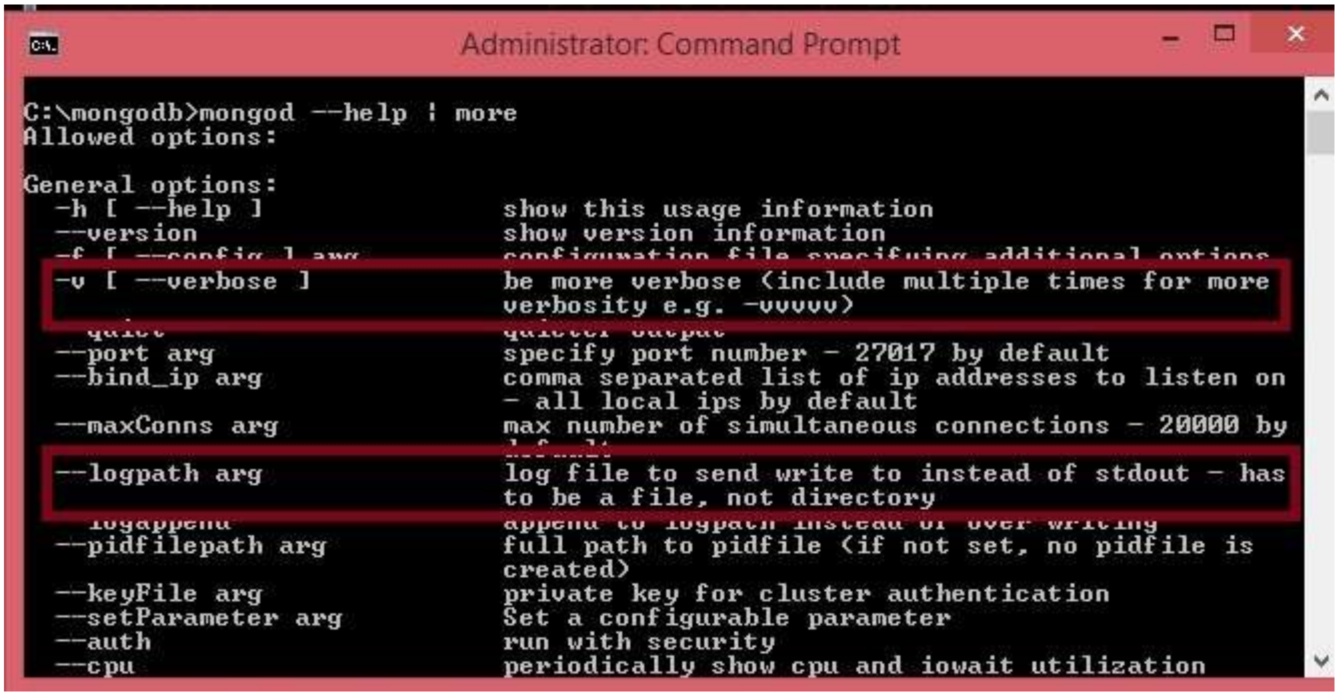
This was simple isn't all we have to do is to create a directory where mongo stores the data files and then start the daemon, with many other databases and packages we have to do prerequisite installations that's not the case with mongo all we need to do is to run the mongo daemon there are no other operating system installations and no other frameworks or dependencies to be installed.it all compiled in c++ and available in single executable file.

Exercise 2: Command Line Options

Mongo has many command line options we can use let's look at few

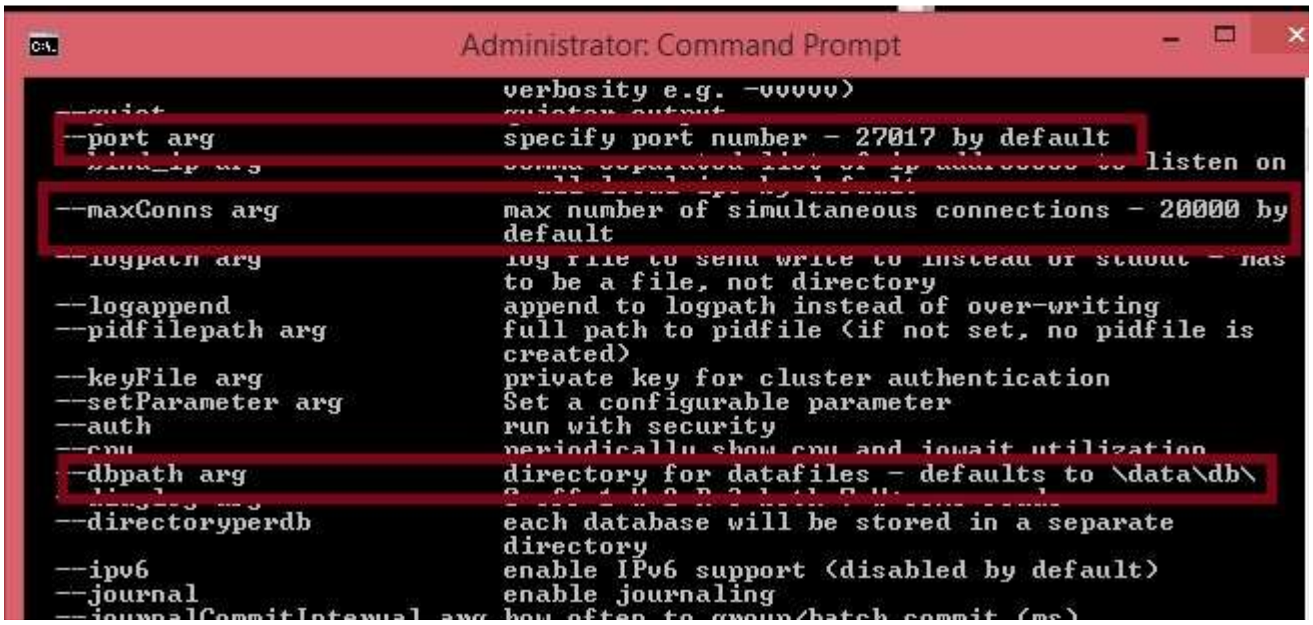
Type the following command

```
Mongd --help | more
```



```
C:\mongodb>mongod --help | more
Allowed options:

General options:
  -h [ --help ]               show this usage information
  --version                   show version information
  -f [ --config ] arg         configuration file specifying additional options
  -v [ --verbose ]            be more verbose (include multiple times for more
                              verbosity e.g. -vvvvv)
  --quiet                     quieter output
  --port arg                  specify port number - 27017 by default
  --bind_ip arg               comma separated list of ip addresses to listen on
                              - all local ips by default
  --maxConns arg              max number of simultaneous connections - 20000 by
                              default
  --logpath arg               log file to send write to instead of stdout - has
                              to be a file, not directory
  --logappend                 append to logpath instead of over-writing
  --pidfilepath arg           full path to pidfile (if not set, no pidfile is
                              created)
  --keyFile arg               private key for cluster authentication
  --setParameter arg          Set a configurable parameter
  --auth                      run with security
  --cpu                       periodically show cpu and iowait utilization
```

It has the ability to log more verbosely or less into a specific log file



```
                                         verbosity e.g. -vvvvv)
  --quiet                                quieter output
  --port arg                             specify port number - 27017 by default
  --bind_ip arg                          comma separated list of ip addresses to listen on
                                           all local ips by default
  --maxConns arg                         max number of simultaneous connections - 20000 by
                                         default
  --logpath arg                          log file to send write to instead of stdout - has
                                         to be a file, not directory
  --logappend                            append to logpath instead of over-writing
  --pidfilepath arg                      full path to pidfile (if not set, no pidfile is
                                         created)
  --keyFile arg                          private key for cluster authentication
  --setParameter arg                     Set a configurable parameter
  --auth                                 run with security
  --cpu                                  periodically show cpu and iowait utilization
  --dbpath arg                           directory for datafiles - defaults to \data\db\
  --diaglog arg                          0=off 1=W 2=R 3=both 7=W+some reads
  --directoryperdb                       each database will be stored in a separate
                                         directory
  --ipv6                                 enable IPv6 support (disabled by default)
  --journal                              enable journaling
  --journalCommitInterval arg            how often to group/batch commit (ms)
```

It has the ability to run on a different TCP port.

We can have more connections.

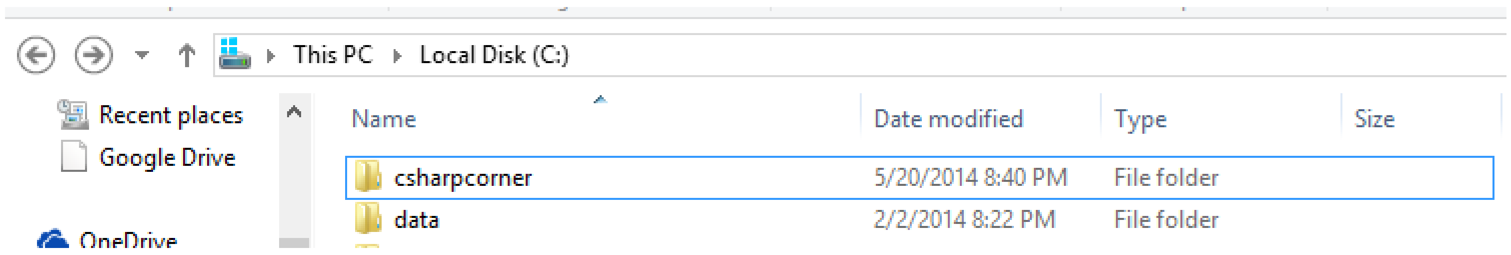
We can specify different directory where data files should reside.

Let us see how to specify different directory and to provide high verbosity level which logs everything.

We will have to specify a different data directory so let's create that directory

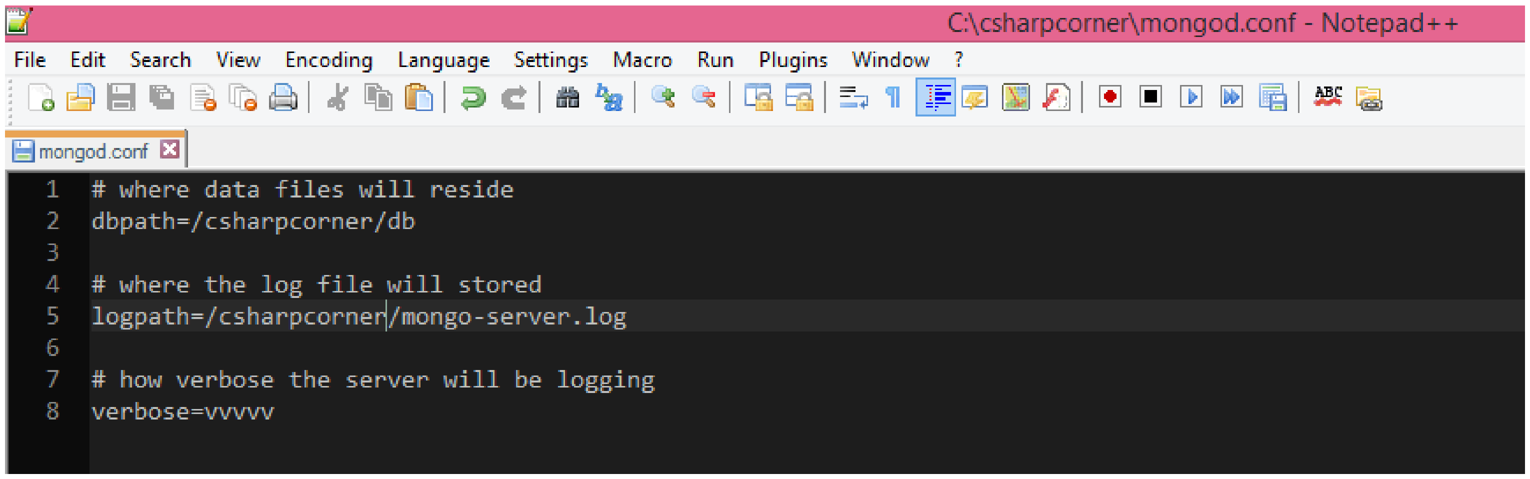
Type

md \csharpcorner\db


This PC ▸ Local Disk (C:)

Name	Date modified	Type	Size
csharpcorner	5/20/2014 8:40 PM	File folder	
data	2/2/2014 8:22 PM	File folder	

Now let's create a Text document inside csharpcorner folder


C:\csharpcorner\mongod.conf - Notepad++

```
# where data files will reside
dbpath=/csharpcorner/db

# where the log file will stored
logpath=/csharpcorner/mongo-server.log

# how verbose the server will be logging
verbose=vvvvv
```

The above is the configuration file being used for the mongo server. Data files will be residing in the csharpcorner db directory and store in log in csharpcorner/mongo-server.log and to log very verbosely am specifying 5 v's here where 1 being least verbose and 5 being most verbose.

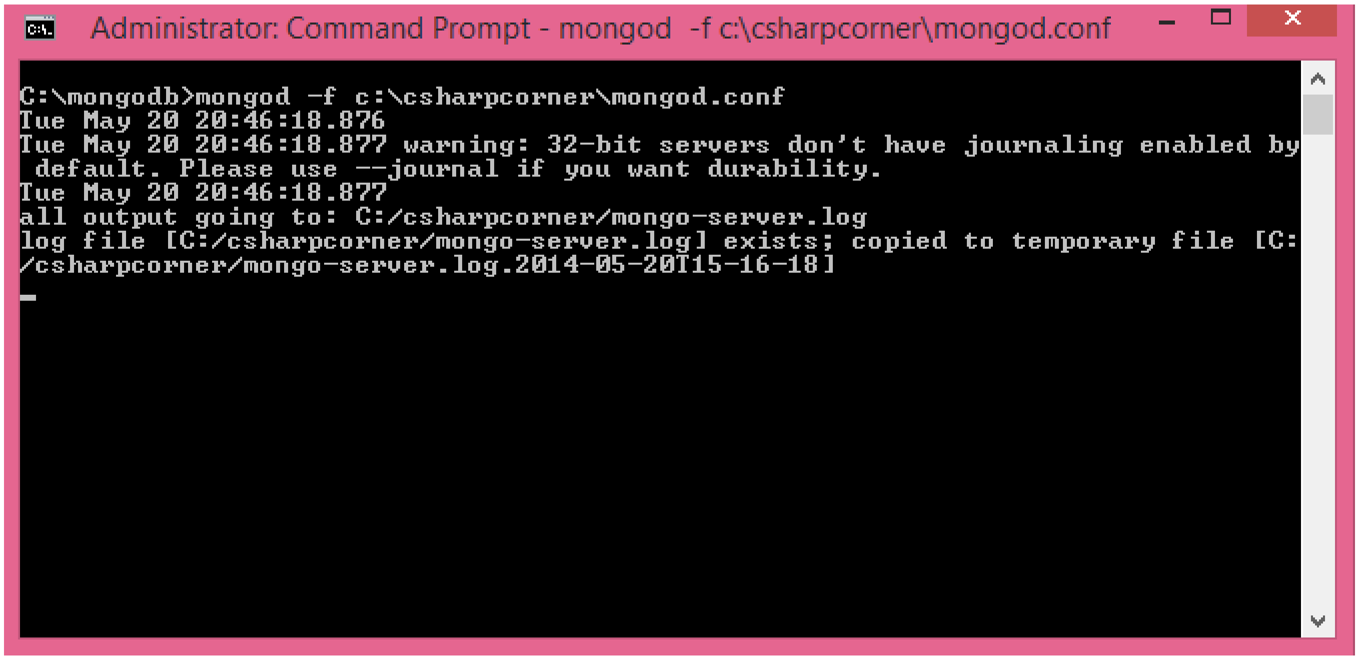
Now we are ready to start the mongo server using the configuration file

Type

```
mongod -f c:\csharpcorner\mongod.conf
```

-f for the configuration file

c: is the path where the configuration file exists



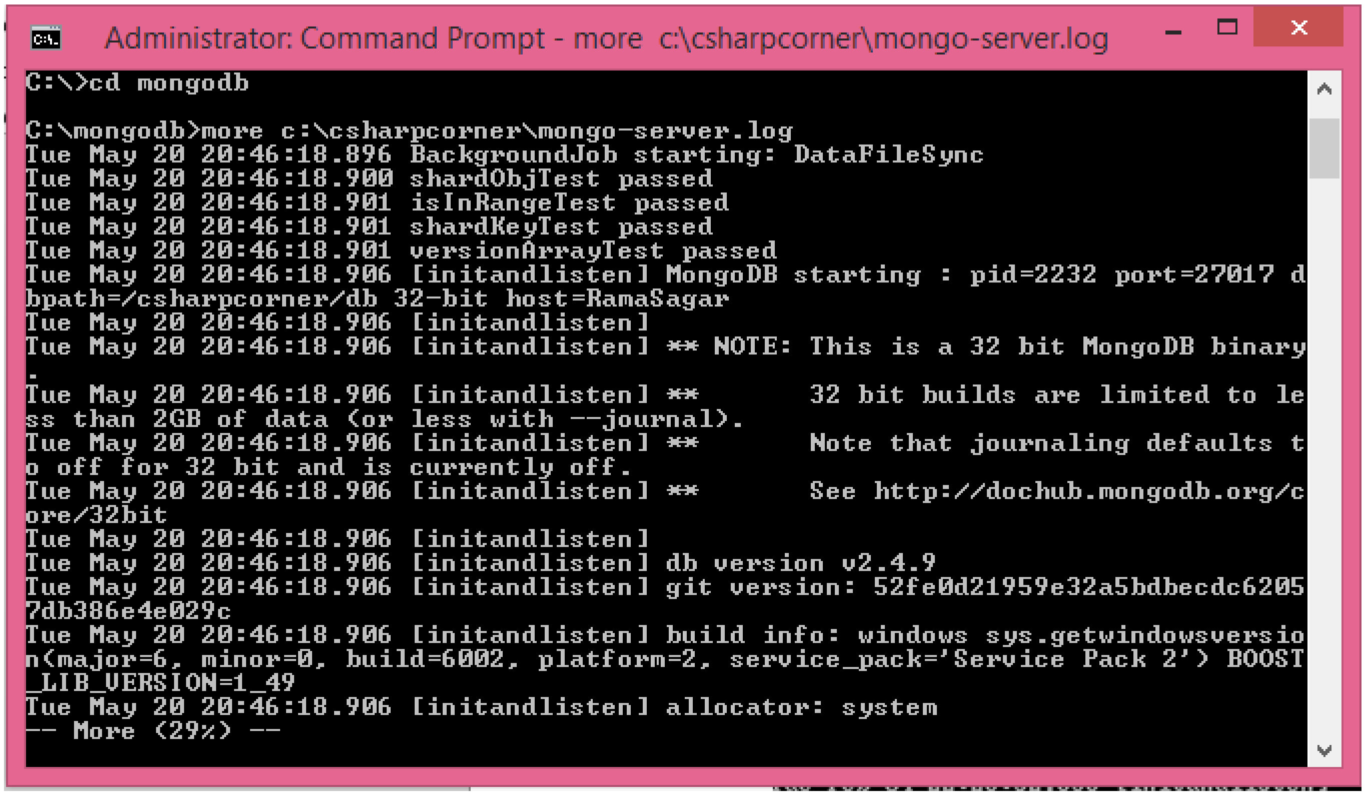
Administrator: Command Prompt - mongod -f c:\csharpcorner\mongod.conf

```
C:\mongodb>mongod -f c:\csharpcorner\mongod.conf
Tue May 20 20:46:18.876
Tue May 20 20:46:18.877 warning: 32-bit servers don't have journaling enabled by
 default. Please use --journal if you want durability.
Tue May 20 20:46:18.877
all output going to: C:/csharpcorner/mongo-server.log
log file [C:/csharpcorner/mongo-server.log] exists; copied to temporary file [C:
/csharpcorner/mongo-server.log.2014-05-20T15-16-18]
_
```

The server is started and it is indicating that all its output is going to log file which we have provided

We can also inspect what is in that log file by the command

```
more c:\csharpcorner\mongo-server.log
```



Administrator: Command Prompt - more c:\csharpcorner\mongo-server.log

```
C:\>cd mongodb

C:\mongodb>more c:\csharpcorner\mongo-server.log
Tue May 20 20:46:18.896 BackgroundJob starting: DataFileSync
Tue May 20 20:46:18.900 shardObjTest passed
Tue May 20 20:46:18.901 isInRangeTest passed
Tue May 20 20:46:18.901 shardKeyTest passed
Tue May 20 20:46:18.901 versionArrayTest passed
Tue May 20 20:46:18.906 [initandlisten] MongoDB starting : pid=2232 port=27017 d
bpath=/csharpcorner/db 32-bit host=RamaSagar
Tue May 20 20:46:18.906 [initandlisten]
Tue May 20 20:46:18.906 [initandlisten] ** NOTE: This is a 32 bit MongoDB binary
.
Tue May 20 20:46:18.906 [initandlisten] **       32 bit builds are limited to le
ss than 2GB of data (or less with --journal).
Tue May 20 20:46:18.906 [initandlisten] **       Note that journaling defaults t
o off for 32 bit and is currently off.
Tue May 20 20:46:18.906 [initandlisten] **       See http://dochub.mongodb.org/c
ore/32bit
Tue May 20 20:46:18.906 [initandlisten]
Tue May 20 20:46:18.906 [initandlisten] db version v2.4.9
Tue May 20 20:46:18.906 [initandlisten] git version: 52fe0d21959e32a5bdbecdc6205
7db386e4e029c
Tue May 20 20:46:18.906 [initandlisten] build info: windows sys.getwindowsversio
n(major=6, minor=0, build=6002, platform=2, service_pack='Service Pack 2') BOOST
_LIB_VERSION=1_49
Tue May 20 20:46:18.906 [initandlisten] allocator: system
-- More (29%) --
```

It was showing all the verbose logging that the server produced

Exercise 3: Install as a Service

Till now we have been using the commands to start and stop the mongo server. If we want the mongo server to be running all the time may be even when the system starts up. On a windows machine we can install mongo as a service.

Let's look installing it as a service

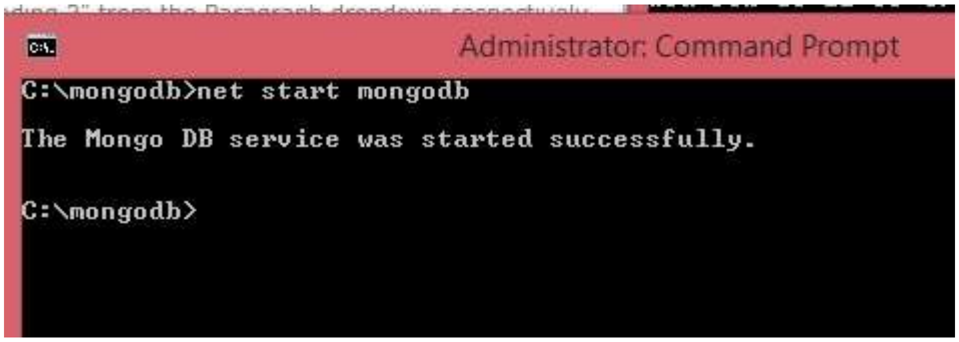
We have already created a configuration file

Open command prompt in administrator mode fetch to the mongo directory and type the following command

```
mongod -f c:\ csharpcorner \mongod.conf --install
```

It’s installed as a service

We can start the service by issuing net start mongodb

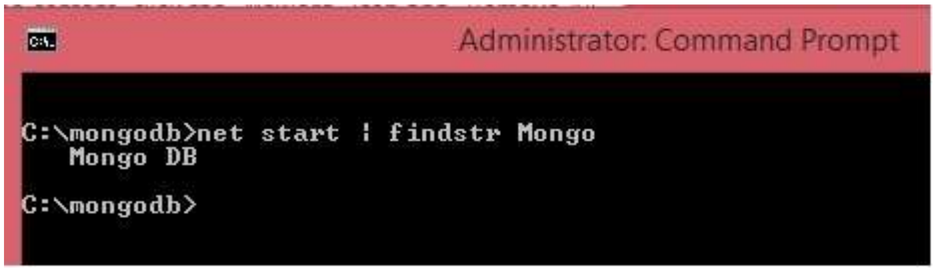


```
C:\mongodb>net start mongodb

The Mongo DB service was started successfully.

C:\mongodb>
```

If we want to check the mongo server is running or not we can use the following command

```
net start | findstr Mongo
```

We can find the Mongo service listed as shown below

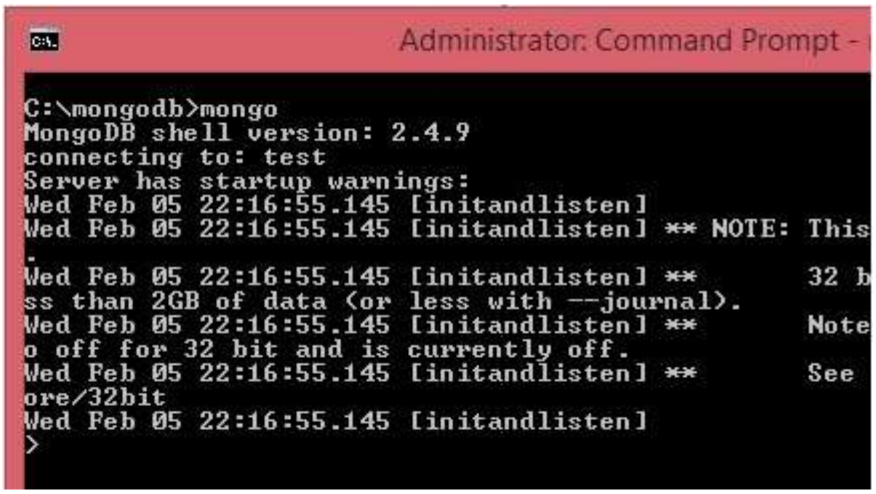


```
C:\mongodb>net start | findstr Mongo
   Mongo DB

C:\mongodb>
```

We can also stop the service by the following command

```
net stop mongodb
```

That's it now we have the Mongo server up and running we can connect to it using the mongo shell. Since we are running the server on local machine and on default port we no need to specify any command line arguments to the shell, we can simply connect by typing mongo.



```
C:\mongodb>mongo
MongoDB shell version: 2.4.9
connecting to: test
Server has startup warnings:
Wed Feb 05 22:16:55.145 [initandlisten]
Wed Feb 05 22:16:55.145 [initandlisten] ** NOTE: This
.
Wed Feb 05 22:16:55.145 [initandlisten] **       32 b
ss than 2GB of data (or less with --journal).
Wed Feb 05 22:16:55.145 [initandlisten] **       Note
o off for 32 bit and is currently off.
Wed Feb 05 22:16:55.145 [initandlisten] **       See
ore/32bit
Wed Feb 05 22:16:55.145 [initandlisten]
>
```

Let check the database inside mongo lets connect to server and we can check the databases by the command

```
show dbs
```

It will show that we have exactly one local database...

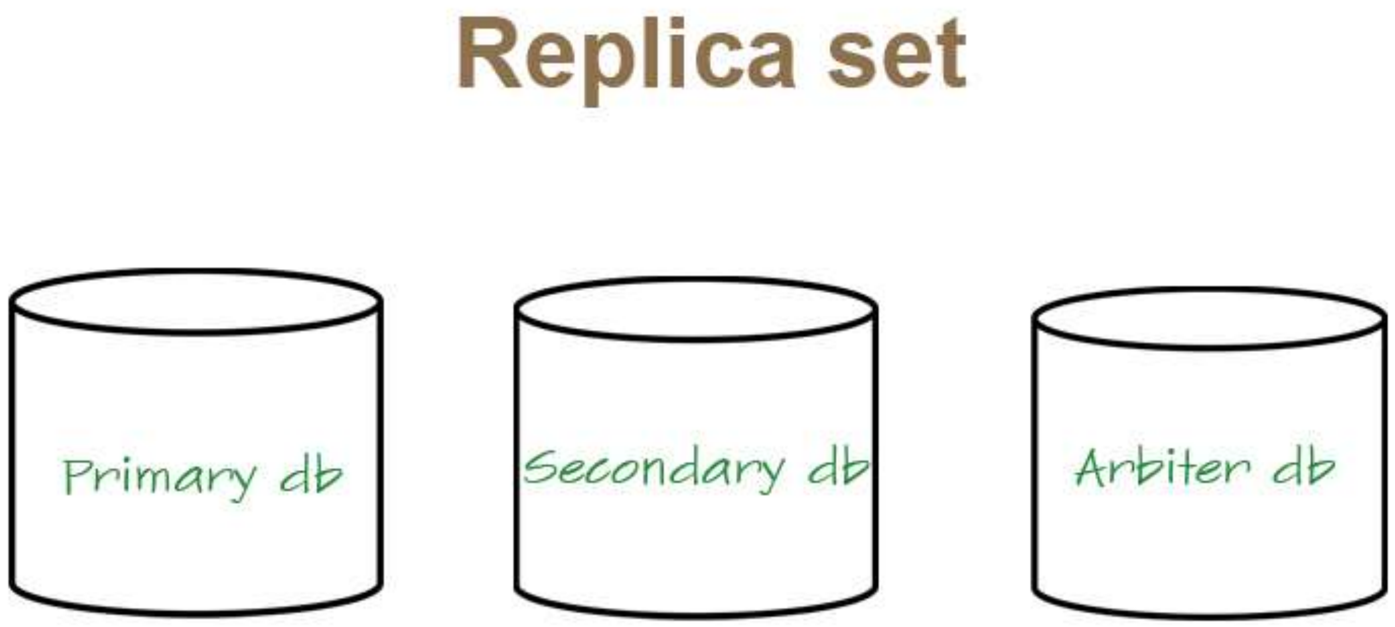
Exercise 4: Replica Set

A *replica set* in MongoDB is a group of `mongod` processes that maintain the same data set. Replica sets provide redundancy and high availability, and are the basis for all production deployments.

Sometimes things don't always go our way...sometimes our server may crash and when that happens...what happens to our application??

If we have a backup we will spend some time and will restore the data which is a traditional way of doing.......

But we can do much better than that where the replica set of mongo dB comes into role.........



Mongo dB supports an arrangement called a replica set, the members of replica set are primary ,secondary or number of secondary's and potentially an arbitrary. Let's look at the rows of each one of those in a replica set.

The primary database is the one and only writable instance in a replica set that means any of the clients that want to write data to the database have to be connected to the primary and have to issue write commands against the primary, An attempt to write to secondary will fail...

The secondary databases are read only instances…we can have many number of secondary databases this means that we also have scalability because we can perform many more reads against the replicas rather than attacking a single server with all requests for crud operations..

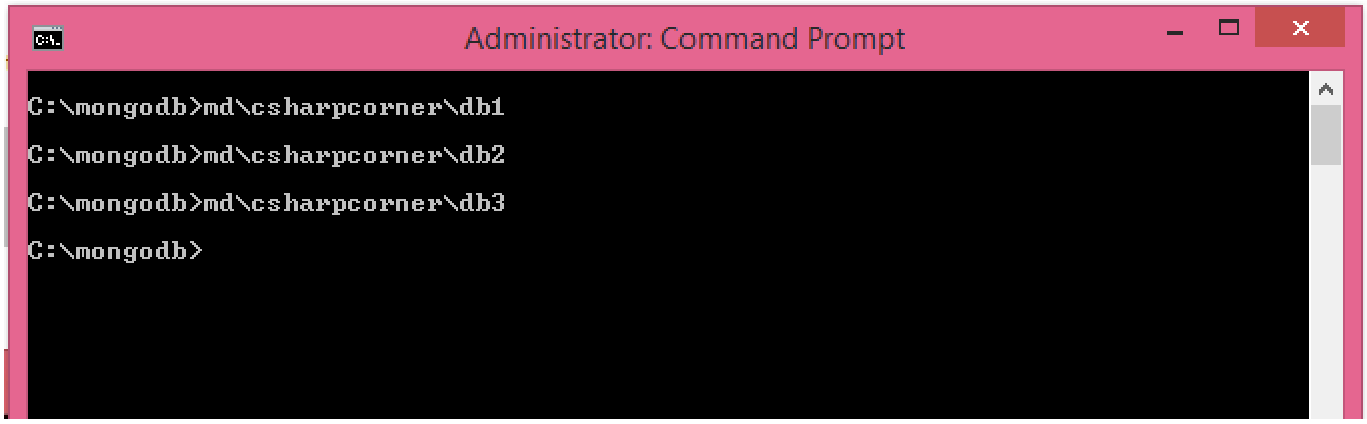
In the secondary databases the data is going to be replicated from the primary eventually which we call eventual consistency. At some point, if primary database is failed one of the secondary will take over and will become the primary where we can get automatic recovery from the crash of the primary…..

The third type of member in replica set is the arbiter. Arbiters are mongod instances that are part of replica set but do not hold data. Arbiters participate in elections in order to break ties. If a replica set has an even number of members, add an arbiter.

Arbiters have minimal resource requirements and do not require dedicated hardware. We can deploy an arbiter on an application server, monitoring host.

Now let’s create the databases and see them in action...

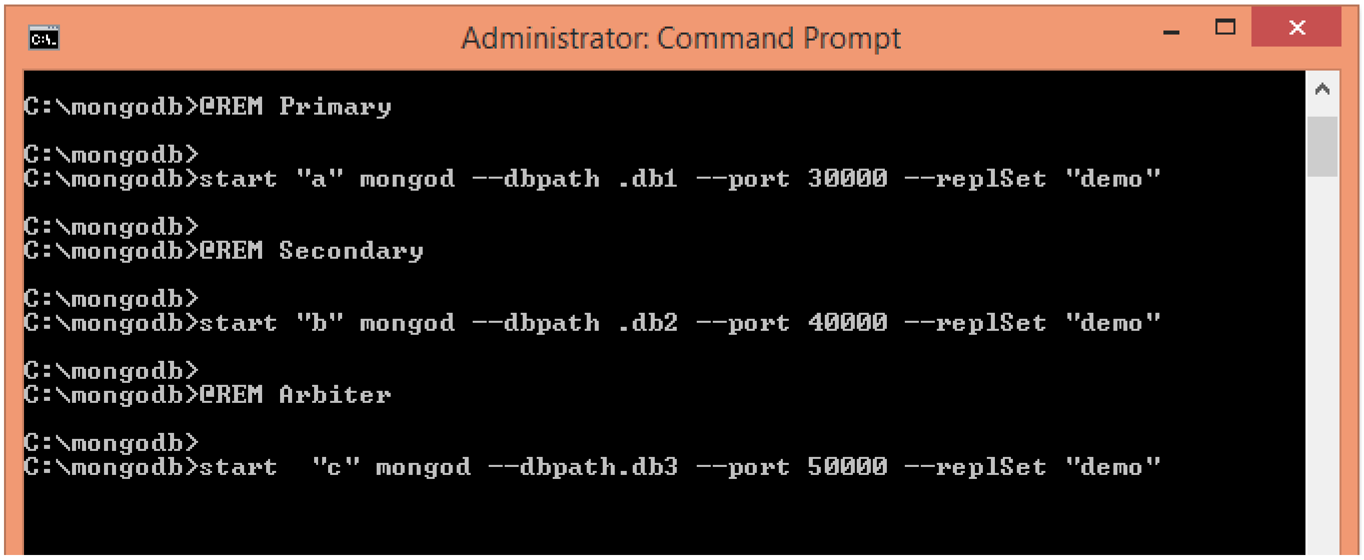
Here we going to create three directories in our csharpcorner directory named db1, db2, db3



```
C:\mongodb>md\csharpcorner\db1

C:\mongodb>md\csharpcorner\db2

C:\mongodb>md\csharpcorner\db3

C:\mongodb>
```

Now let’s start the three instances give the command

```
@REM Primary

start "a" mongod --dbpath .db1 --port 30000 --replSet "demo"

@REM Secondary

start "b" mongod --dbpath .db2 --port 40000 --replSet "demo"

@REM Arbiter

start  "c" mongod --dbpath.db3 --port 50000 --replSet "demo"
```



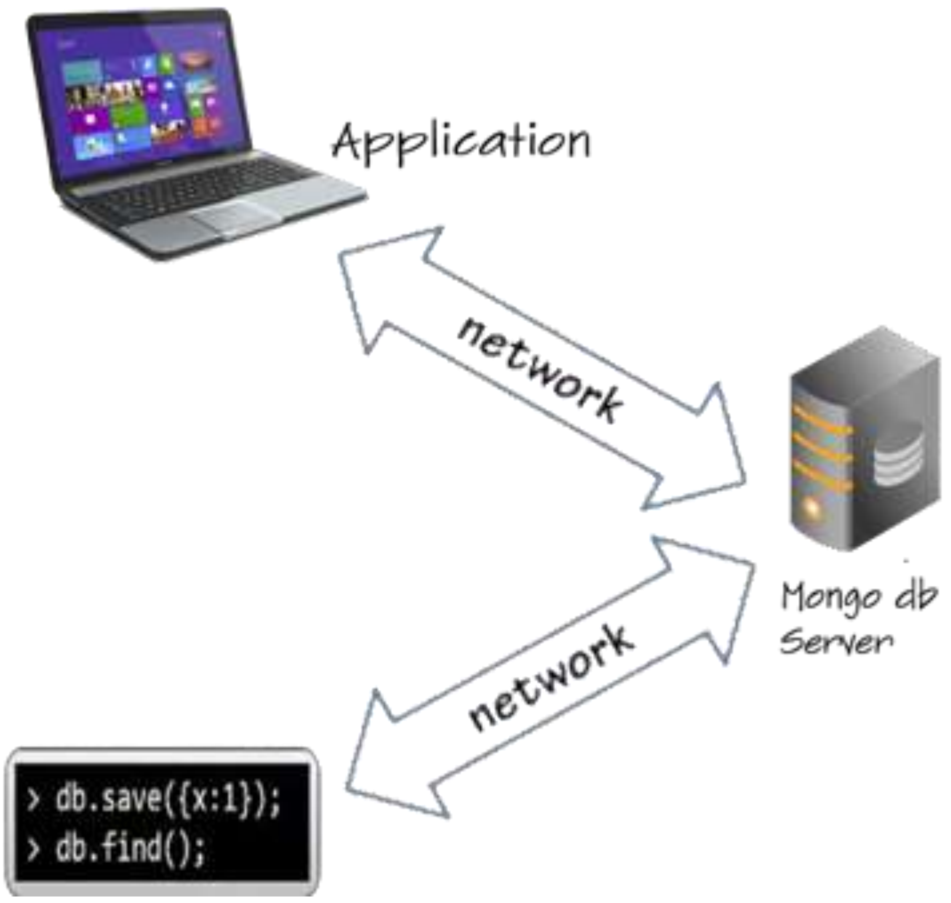
```
C:\mongodb>@REM Primary

C:\mongodb>
C:\mongodb>start "a" mongod --dbpath .db1 --port 30000 --replSet "demo"

C:\mongodb>
C:\mongodb>@REM Secondary

C:\mongodb>
C:\mongodb>start "b" mongod --dbpath .db2 --port 40000 --replSet "demo"

C:\mongodb>
C:\mongodb>@REM Arbiter

C:\mongodb>
C:\mongodb>start  "c" mongod --dbpath.db3 --port 50000 --replSet "demo"
```

Exercise 5: Mongo Shell

The `mongo` shell is an interactive JavaScript shell for MongoDB, and is part of all MongoDB distributions.

The Shell is simply an application that allows us to interactively get insight into what the mongo server is doing?

The shell uses the same wire protocol as our application would have. It's simply another application that talks to mongod and is able to understand the wire protocol but it provides many capabilities...

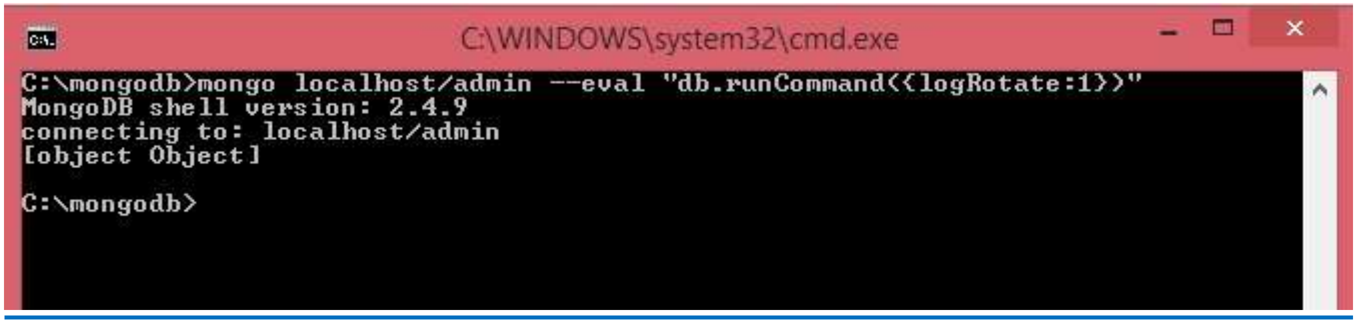


```
> db.save({x:1});
> db.find();
```

Let’s see some examples:

If we wanna rotate the log file we can provide the command

```
mongo localhost/admin --eval "db.runCommand({logRotate:1})"
```



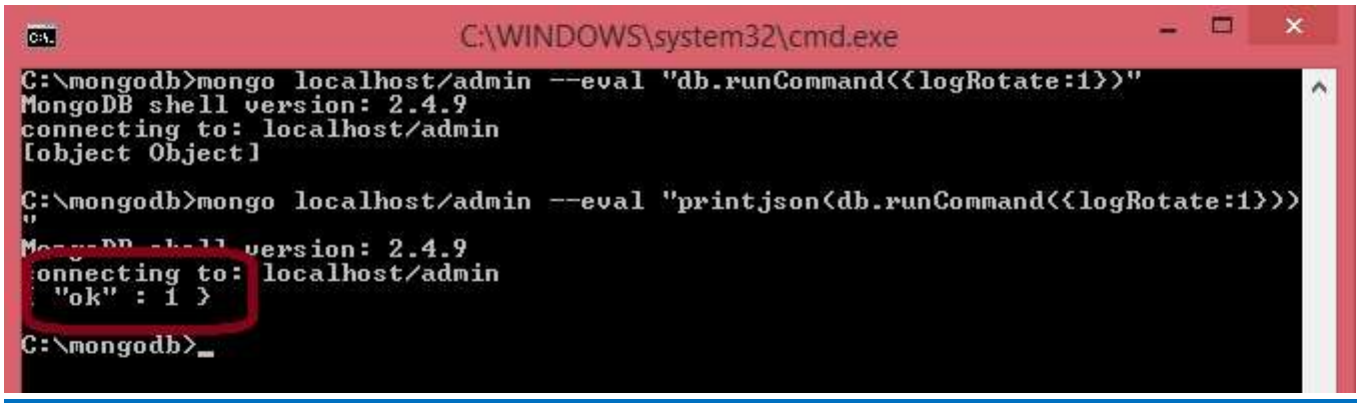
```
C:\mongodb>mongo localhost/admin --eval "db.runCommand({logRotate:1})"
MongoDB shell version: 2.4.9
connecting to: localhost/admin
[object Object]

C:\mongodb>
```

We can notice a return of object object on screen if we wanna know what that thing we can provide the command

printjson (which prints the result of the command)

We can wrap it around our orginal command

mongo localhost/admin --eval "printjson(db.runCommand({logRotate:1}))".

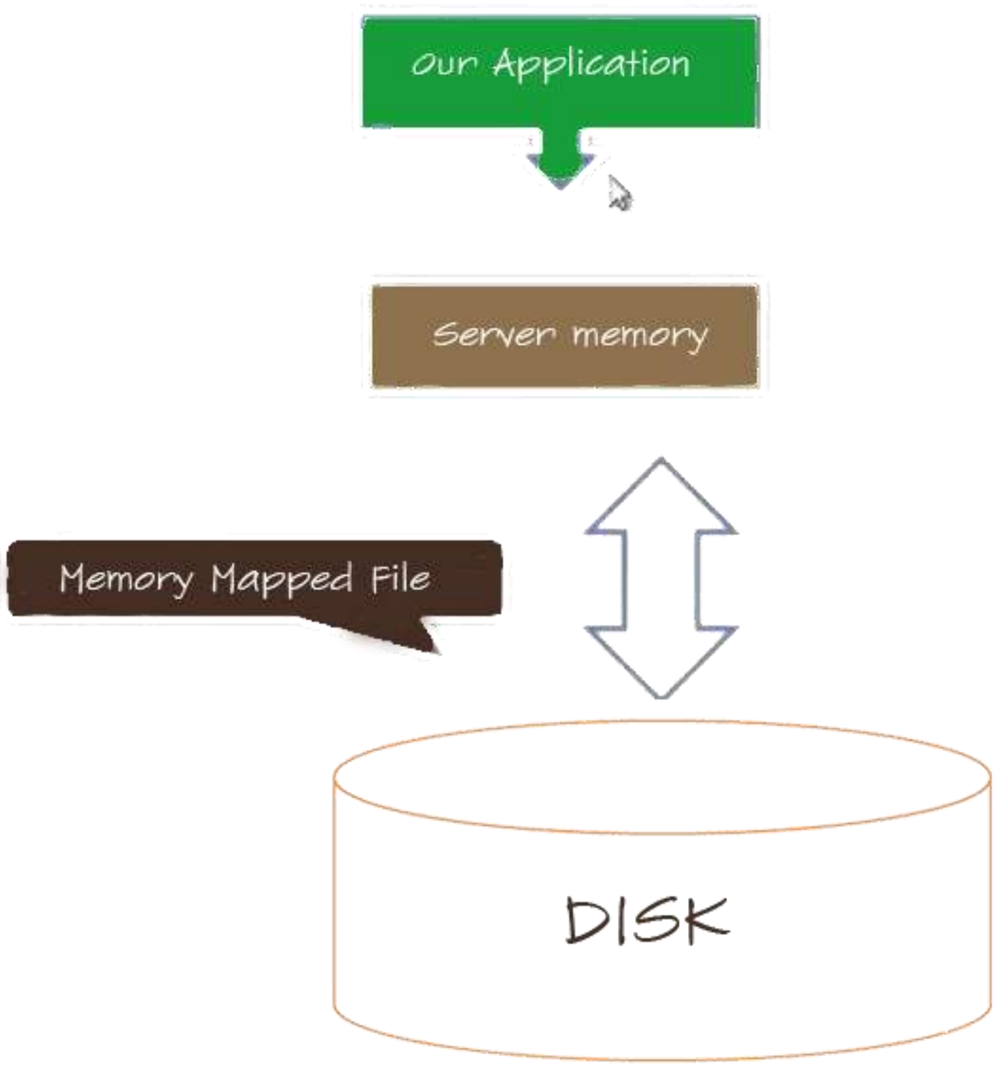


```
C:\mongodb>mongo localhost/admin --eval "db.runCommand({logRotate:1})"
MongoDB shell version: 2.4.9
connecting to: localhost/admin
[object Object]

C:\mongodb>mongo localhost/admin --eval "printjson(db.runCommand({logRotate:1}))
"
MongoDB shell version: 2.4.9
connecting to: localhost/admin
{ "ok" : 1 }

C:\mongodb>_
```

It was trying to tell us that everything is ok...

Exercise 6: Storing Data

First let us talk about how data is stored by the engine…Now if our application wants to interact with some information that information is a memory in our application…It can talk to the server and the server has preeminence storage named in the disk….



Mongo uses Memory mapped Files the server cannot store all its information in memory. But it would like to think of information has just existing in being available to it at any given moment. So what it does is it creates a Mongo server and maps it using memory mapped files whenever it calls it into play a portion of that array the operating system takes care of loading it or saving it to the disk. When we want to store a bit of information we handle it over the server, and the server scribbles it over a memory and that memory gets managed and serialized to disk... The same process in reverse happens when we want to read data the server will attempt to access a portion of the large byte array which will be loaded as needed from the operating system...

so now byte arrays can be stored on disk...now the question arises here..How does our document which doesn't have schema that get saved and what format does it get saved??

The answer is BSON...The BSON specifications can be found on http://bsonspec.org/

The BSON data format has several advantages some key advantages are that there is very little marshaling necessary from BSON elementary data types into c data types that makes reading and writing very fast in any of the programming languages. We can learn more at http://bsonspec.org/.

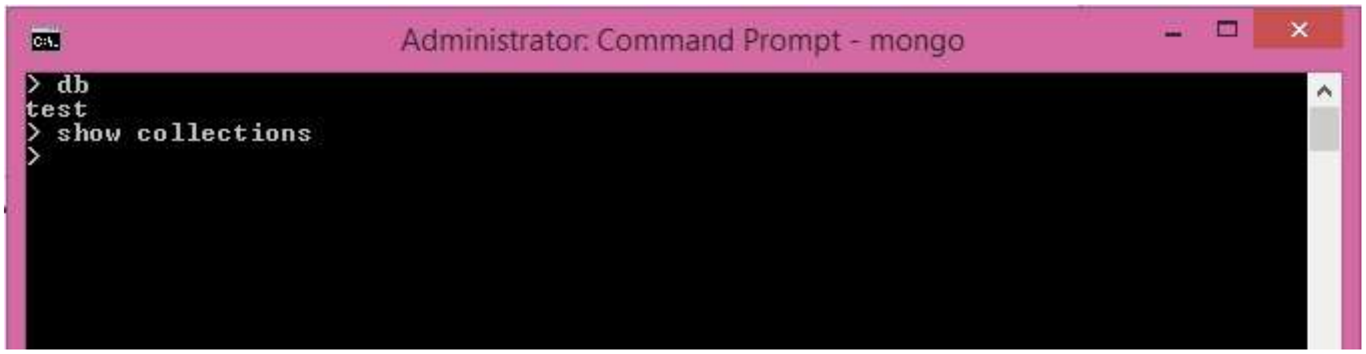
Rules for saving data

Rule 1: A document must have an ID field._id every document in mongo must have an Id if we save one without an Id mongo will assign an id field, but every document we save must have an ID...

Rule 2: The size of the document in Mongo is currently limited to 16 Mega Bytes. If we need to store more than that
We will have to store across multiple documents. This is something that may evolve in future releases. But this is the current limitation...

Saving Data

We are in some default database where dB is our current database…Lets us first check if there is any data in there we can do it by issuing a command as shown below…



```
> db
test
> show collections
>
```

The image indicates there are no collections. A collection in mongo defines the scope of interaction with the documents.so we can issue commands against a specific collection to store and retrieve data. Because it's not a relational database we cannot issue commands across several collections…

Ok now let's save our first record

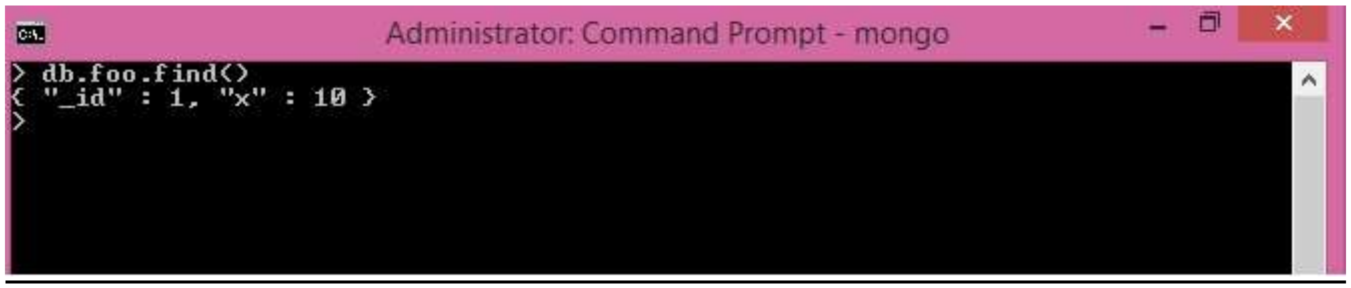
```
db.foo.save ({_id:1, x: 10})
```

Here dB means the database name where we are operating.

Foo is the name of the collection we are going to save this document into

Save indicates the saving of record.

We can find the document we just saved by issuing the command db.foo.find ()



```
> db.foo.find()
{ "_id" : 1, "x" : 10 }
>
```

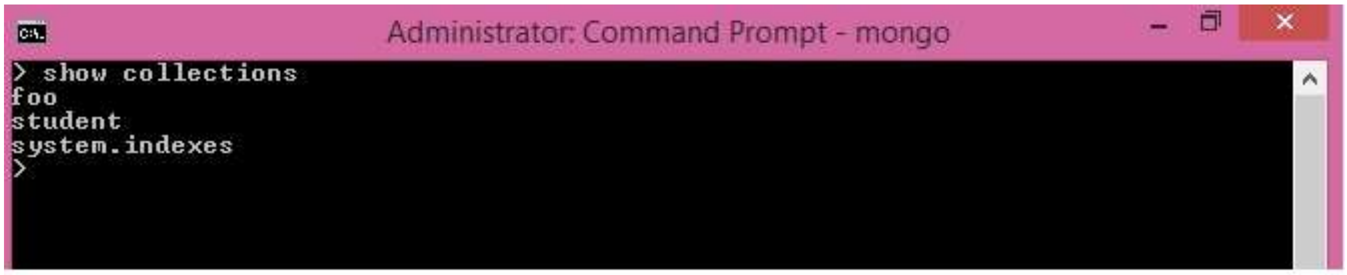
Now if we check the collections we will have two collections one is foo and other is system. Indexes.

Every document must have an id that's because in order to have fast access we want an index in an Id field.

Let's insert same record into student collection

db.foo.save ({_id:1, x: 10})

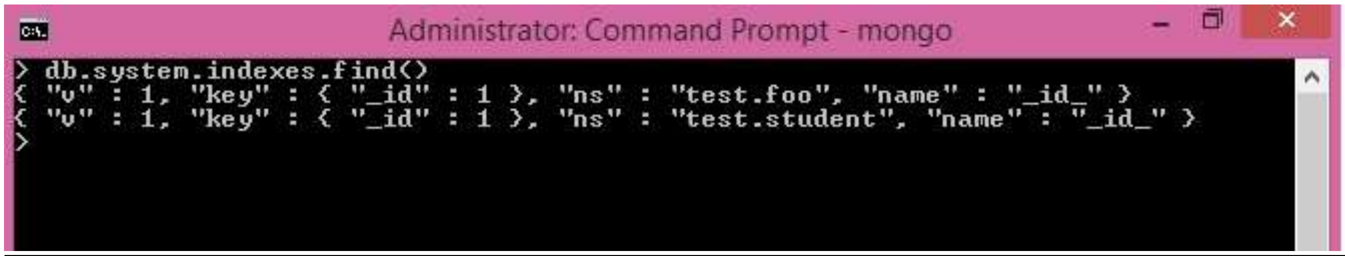
Now if we see the collections we can have the following result



```
> show collections
foo
student
system.indexes
>
```

Let’s check the indexes by issuing command

db.system.indexes.find ()



```
> db.system.indexes.find()
{ "v" : 1, "key" : { "_id" : 1 }, "ns" : "test.foo", "name" : "_id_" }
{ "v" : 1, "key" : { "_id" : 1 }, "ns" : "test.student", "name" : "_id_" }
>
```

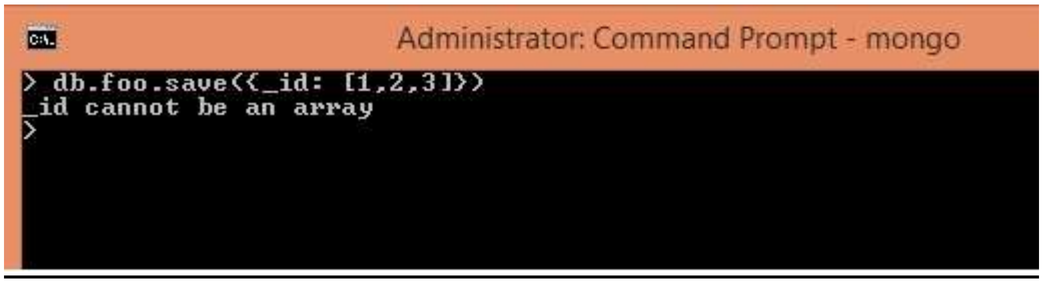
We can see mongo has created an index on the Id field the key is id on the test foo and test student collections inside the database…

Let us see what data field the id supports...

Mongo can have an id that is numeric, an integer, a floating point or a UTC date time structure.as shown below…

```
db.foo.save({_id: 1})
db.foo.save({_id 3.14})
db.foo.save({_id "hello"})
db.foo.save({_id ISODate()})
```

The only data type which is excluded is an array if we try to use an array it throws an exception as shown below.



```
> db.foo.save({_id: [1,2,3]})
_id cannot be an array
>
```

Exercise 7: Updating Document

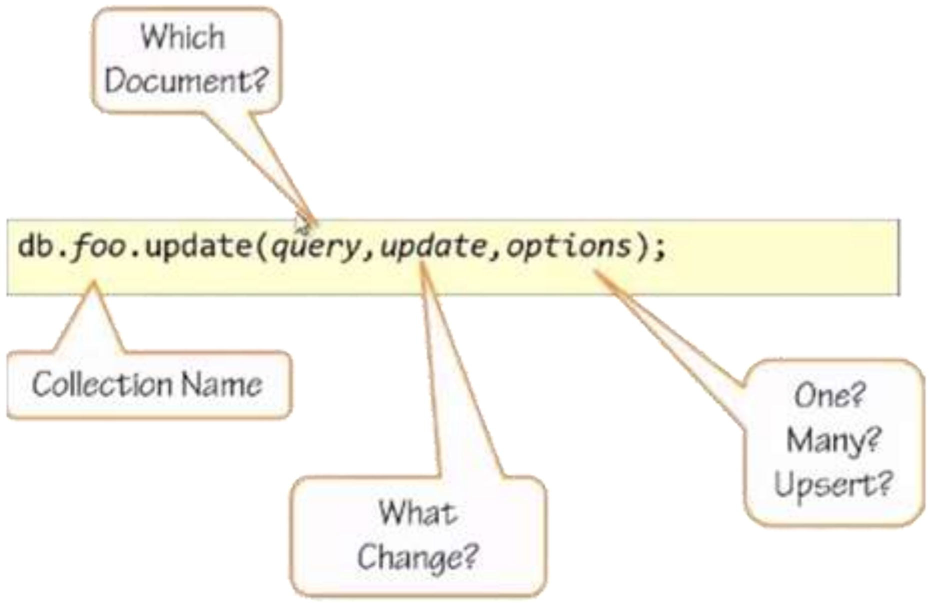
The mongo update command is atomic within a document. No two clients may update the same document at the same time, two update commands issued concurrently will be executed one after another. Here is the syntax of the update command...

First we have to specify which collection we going to update

Second we will need to specify which document we are targeting

Third we will need to specify what change we want to see in active i.e. update parameter

Lastly we may specify other options such as do we want to change only one the first document found matching the query or do we want to upsert(do we want to save a new record in case the query doesn't match any document will generate the document on the fly)



Let’s see them in a practical example

First let’s save a record with the id 1 and a value x of 10...

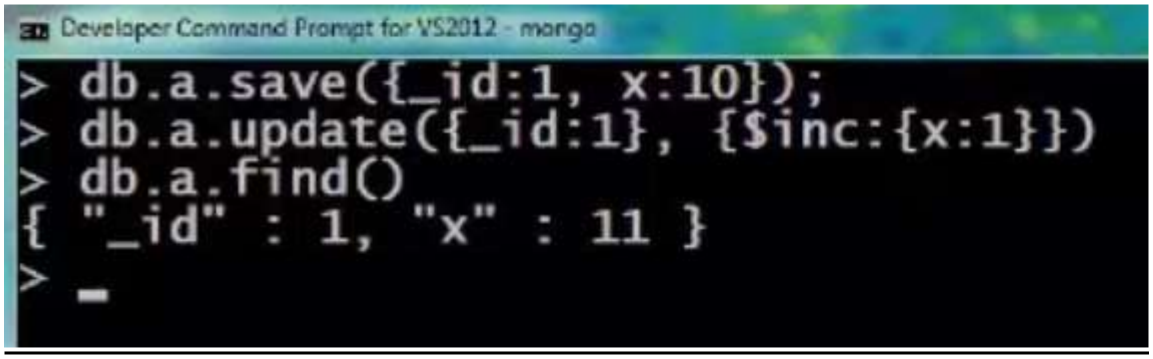
```
db.a.save({_id:1, X:10});
```

Here we use the increment operator which takes the field name and the amount of which we want to implement.

```
db.a.update({_id:1, {$inc:{X:1}});
```

Now let's check the value

db.a.find()



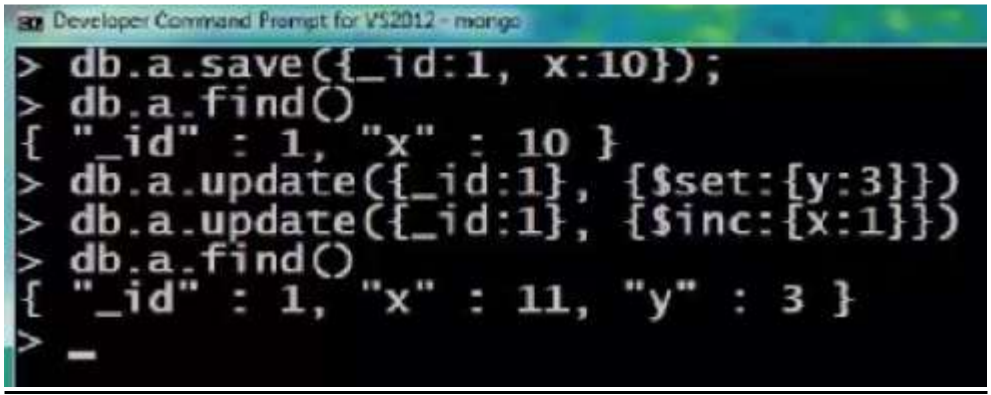
```
Developer Command Prompt for VS2012 - mongo
> db.a.save({_id:1, x:10});
> db.a.update({_id:1}, {$inc:{x:1}})
> db.a.find()
{ "_id" : 1, "x" : 11 }
> _
```

In the second scenario one client wants to add the field to the document while another client is trying to implement. So we already have our self a record in there with only one field x=10.

// db.a.save({_id:1, x:10});

Now comes along the client that wants to add the field…we can issue a update command...and again the same client want to increment the value of x we can specify only the x to be incremented as shown below

```
db.a.update({_id:1},{$set:{y:3}})

db.a.save({_id:1, {$inc:{x:1}})
```


Developer Command Prompt for VS2012 - mongo

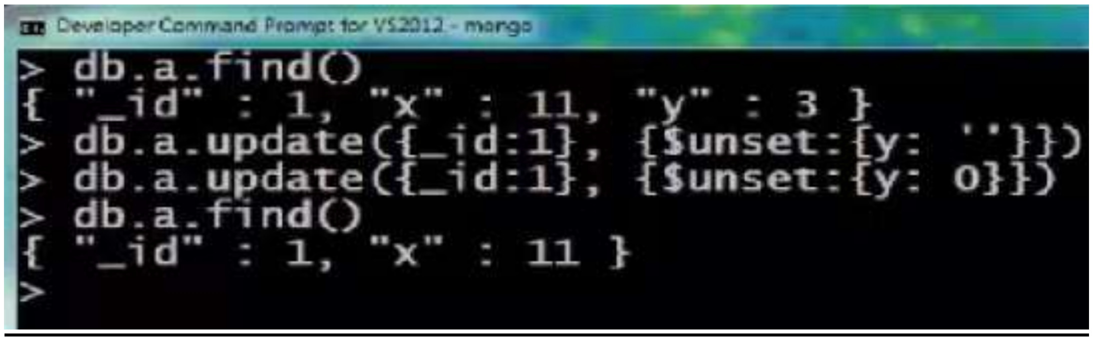
```
> db.a.save({_id:1, x:10});
> db.a.find()
{ "_id" : 1, "x" : 10 }
> db.a.update({_id:1}, {$set:{y:3}})
> db.a.update({_id:1}, {$inc:{x:1}})
> db.a.find()
{ "_id" : 1, "x" : 11, "y" : 3 }
> _
```

ok right now we have the fields x and y now if we wanna remove the field y we can issue the unset command the unset command takes the field name and a arbitary value.

db.a.update({_id:1},{$unset:{y:''}})

or

db.a.update({_id:1},{$unset:{y: 0}})


Developer Command Prompt for VS2012 - mongo

```
> db.a.find()
{ "_id" : 1, "x" : 11, "y" : 3 }
> db.a.update({_id:1}, {$unset:{y: ''}})
> db.a.update({_id:1}, {$unset:{y: 0}})
> db.a.find()
{ "_id" : 1, "x" : 11 }
>
```

Now let's consider some array operations

We can see below that I have a document with only an id field

```
db.a.save ({_id:1});
```

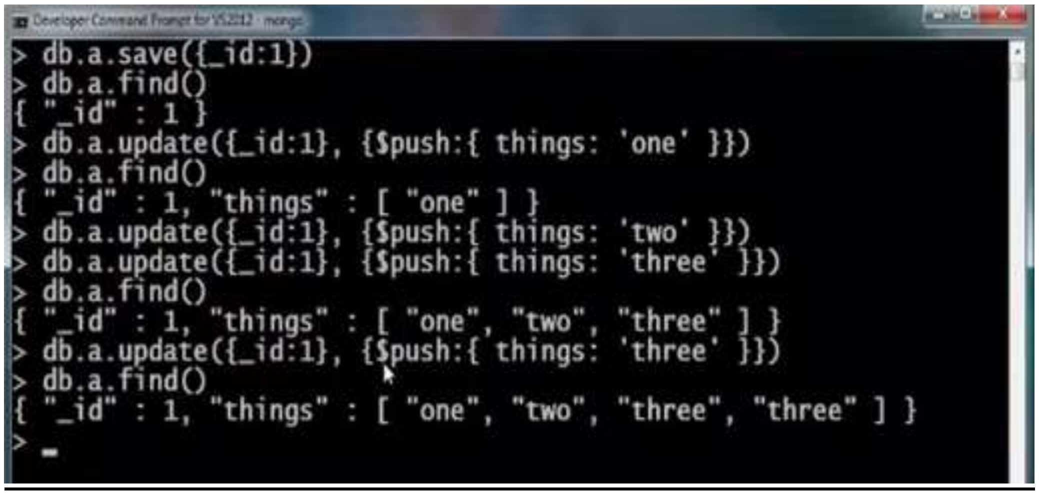
Now if we want here an array containing some values in this document

We can issue the update with the push command to add an item to an array

```
db.a.update ({_id:1}, {$push :{ things: 'one'}});
```

Let's put some more items

```
db.a.update ({_id:1}, {$push :{ things: 'two'}});
db.a.update ({_id:1}, {$push :{ things: 'three'}});
db.a.update ({_id:1}, {$push :{ things: 'three'}});
```

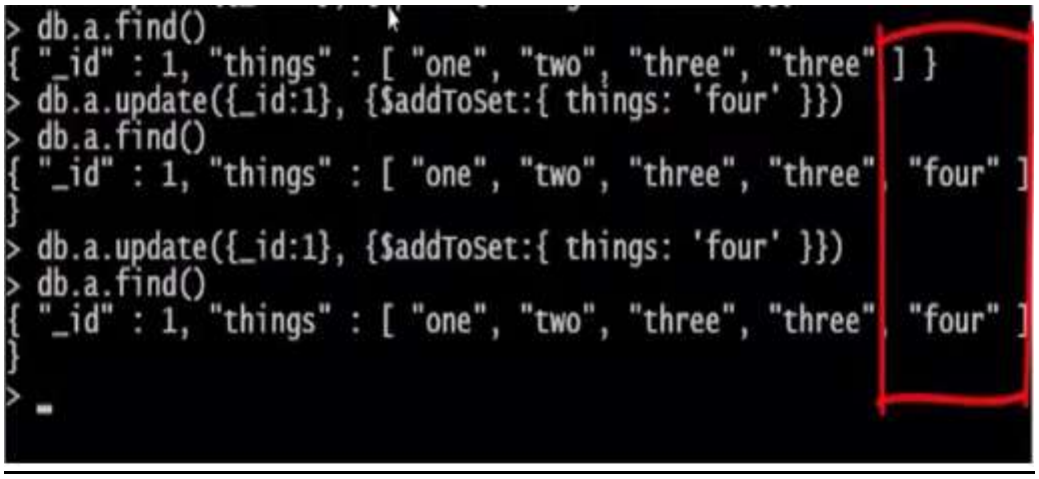


```
> db.a.save({_id:1})
> db.a.find()
{ "_id" : 1 }
> db.a.update({_id:1}, {$push:{ things: 'one' }})
> db.a.find()
{ "_id" : 1, "things" : [ "one" ] }
> db.a.update({_id:1}, {$push:{ things: 'two' }})
> db.a.update({_id:1}, {$push:{ things: 'three' }})
> db.a.find()
{ "_id" : 1, "things" : [ "one", "two", "three" ] }
> db.a.update({_id:1}, {$push:{ things: 'three' }})
> db.a.find()
{ "_id" : 1, "things" : [ "one", "two", "three", "three" ] }
> _
```

We can find that it has the element three twice that might be something we don’t want.. We can restrict it by using addToSet operator

db.a.update ({_id:1}, {$addToSet :{ things: ‘four’}});

We can notice that the element four didn't added again

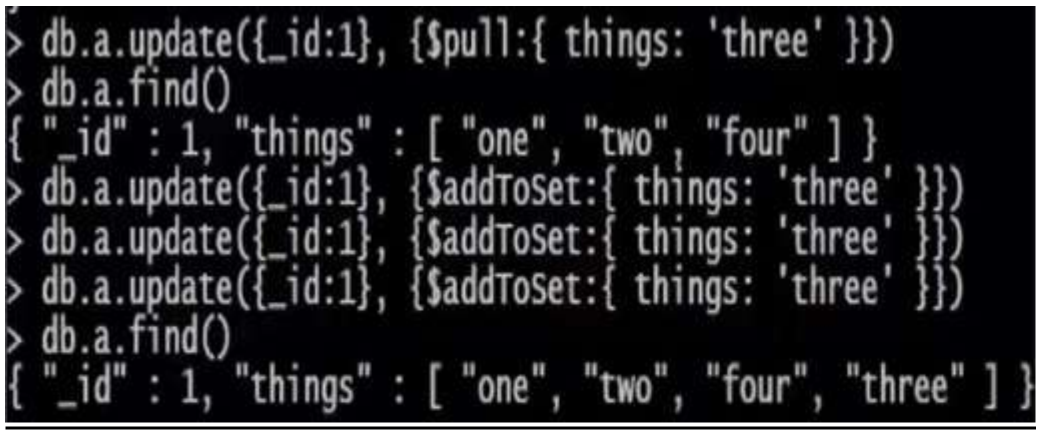


```
> db.a.find()
{ "_id" : 1, "things" : [ "one", "two", "three", "three" ] }
> db.a.update({_id:1}, {$addToSet:{ things: 'four' }})
> db.a.find()
{ "_id" : 1, "things" : [ "one", "two", "three", "three", "four" ]
}
> db.a.update({_id:1}, {$addToSet:{ things: 'four' }})
> db.a.find()
{ "_id" : 1, "things" : [ "one", "two", "three", "three", "four" ]
}
> _
```

We can still see three is hanging in which we doesn’t need we can remove it out of the array by using pull operator and can add it again as shown..

db.a.update ({_id:1}, {$pull :{ things: ‘three}});

It will delete all the instances of three in the array as shown below



```
> db.a.update({_id:1}, {$pull:{ things: 'three' }})
> db.a.find()
{ "_id" : 1, "things" : [ "one", "two", "four" ] }
> db.a.update({_id:1}, {$addToSet:{ things: 'three' }})
> db.a.update({_id:1}, {$addToSet:{ things: 'three' }})
> db.a.update({_id:1}, {$addToSet:{ things: 'three' }})
> db.a.find()
{ "_id" : 1, "things" : [ "one", "two", "four", "three" ] }
```

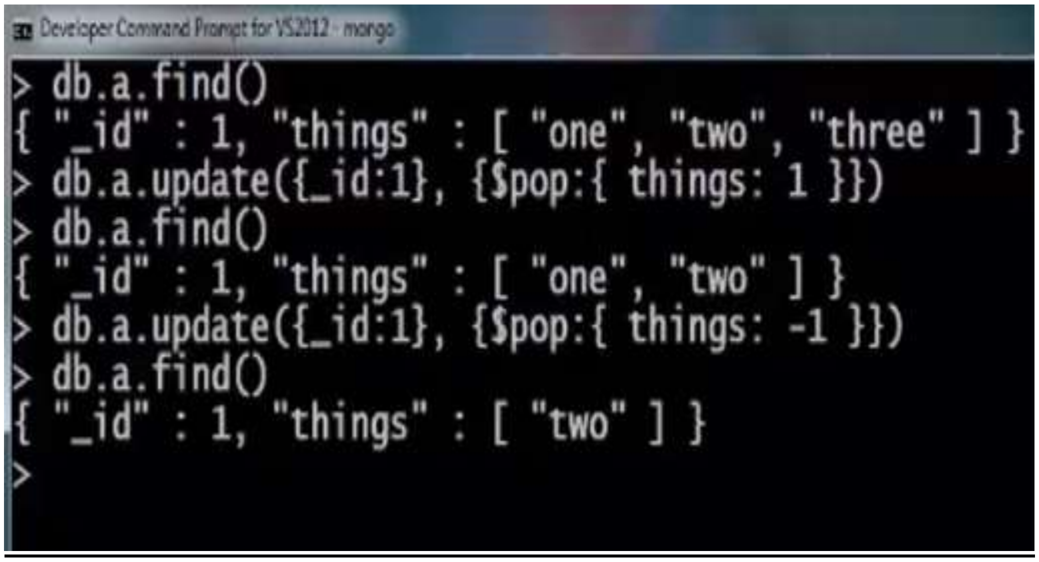
Now if we want to remove the last element and first element in an array we can use pop operator as shown below

For last element

```
db.a.update ({_id:1}, {$pop :{ things: 1}});
```

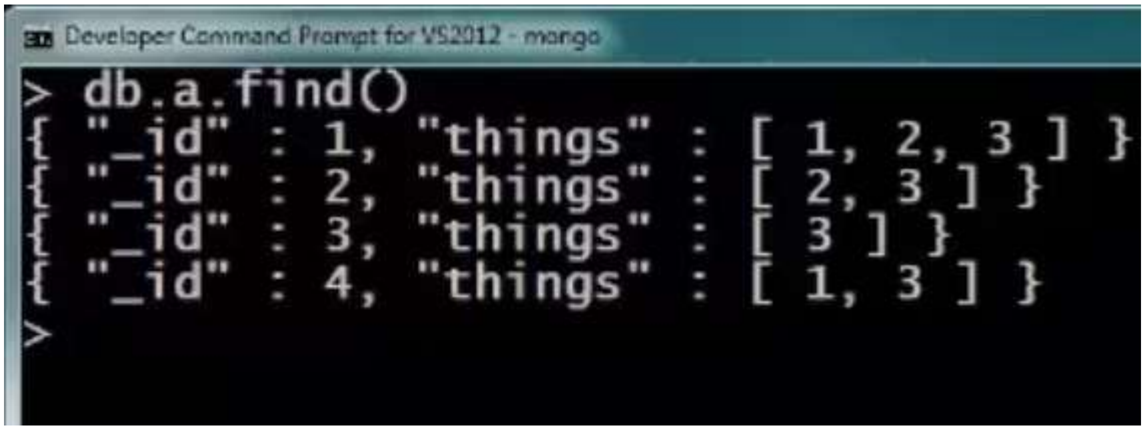
For First element we can use negative value

```
db.a.update ({_id:1}, {$pop :{ things: -1}});
```



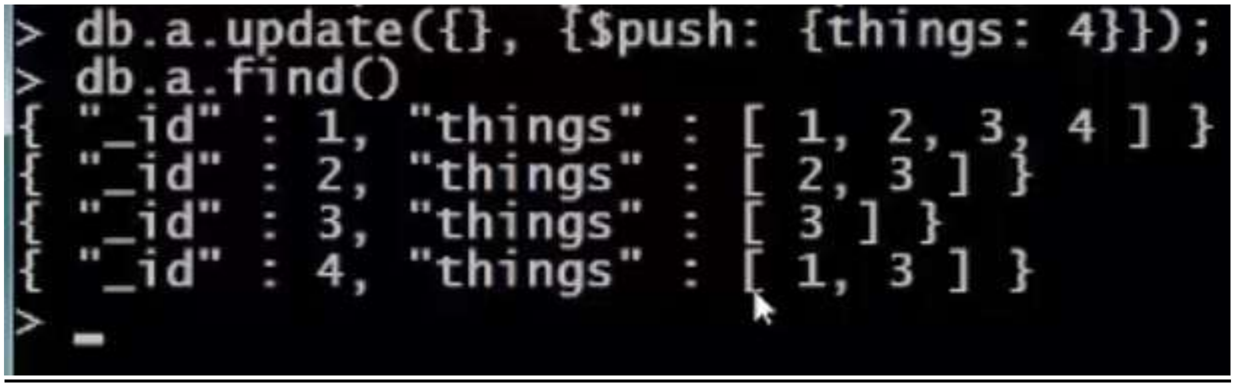
```
> db.a.find()
{ "_id" : 1, "things" : [ "one", "two", "three" ] }
> db.a.update({_id:1}, {$pop:{ things: 1 }})
> db.a.find()
{ "_id" : 1, "things" : [ "one", "two" ] }
> db.a.update({_id:1}, {$pop:{ things: -1 }})
> db.a.find()
{ "_id" : 1, "things" : [ "two" ] }
>
```

Now let's consider a scenario where we had multiple records in the database...if we wanna apply an update to several of them.



```
> db.a.find()
{ "_id" : 1, "things" : [ 1, 2, 3 ] }
{ "_id" : 2, "things" : [ 2, 3 ] }
{ "_id" : 3, "things" : [ 3 ] }
{ "_id" : 4, "things" : [ 1, 3 ] }
>
```

Let's see if we want to push another element in them using the update command with an empty query we can do it db.a.update ({}, {$push :{ things: 4}});

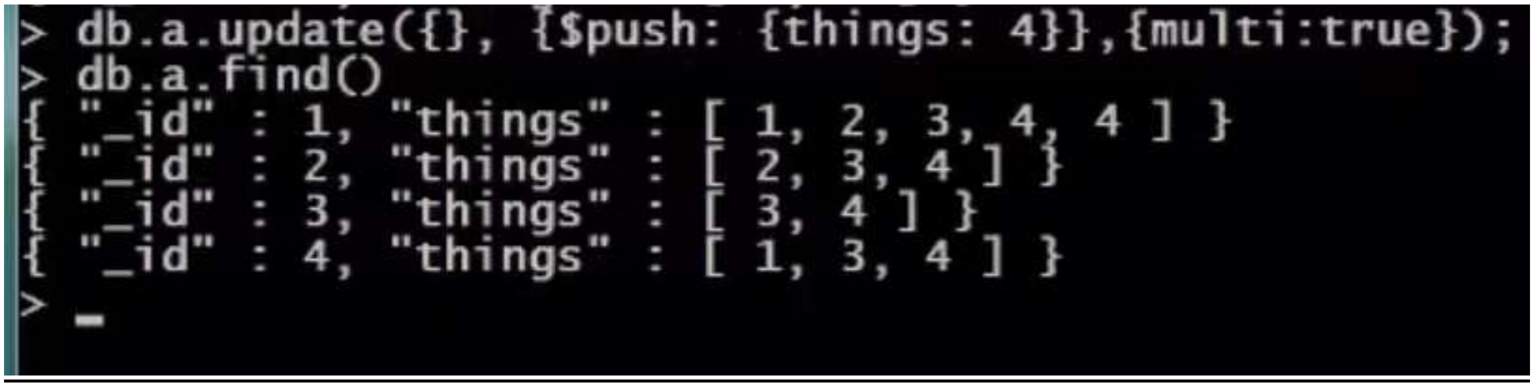


```
> db.a.update({}, {$push: {things: 4}});
> db.a.find()
{ "_id" : 1, "things" : [ 1, 2, 3, 4 ] }
{ "_id" : 2, "things" : [ 2, 3 ] }
{ "_id" : 3, "things" : [ 3 ] }
{ "_id" : 4, "things" : [ 1, 3 ] }
> _
```

But we can see that only one record was effected this is because the default options of an update is fixed to one record.

We can fix it by using the multiple options true

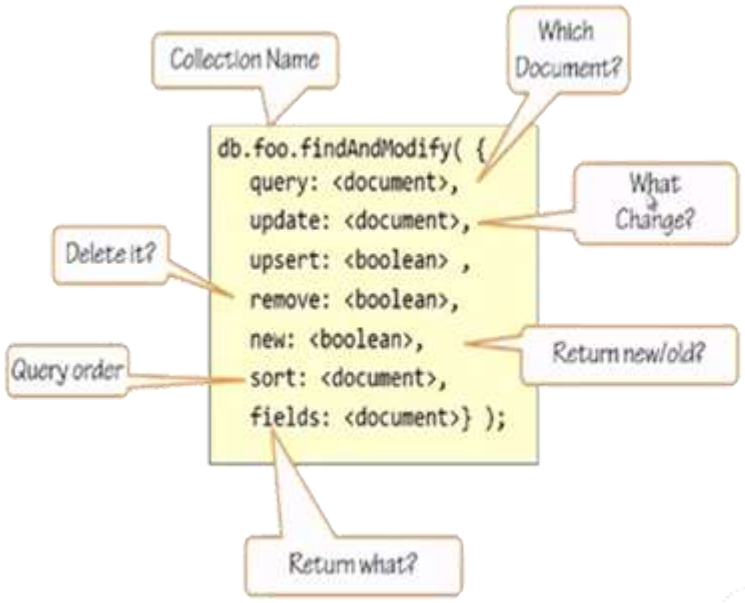
db.a.update ({}, {$push :{ things: 4}},{multi:true));



```
> db.a.update({}, {$push: {things: 4}},{multi:true});
> db.a.find()
{ "_id" : 1, "things" : [ 1, 2, 3, 4, 4 ] }
{ "_id" : 2, "things" : [ 2, 3, 4 ] }
{ "_id" : 3, "things" : [ 3, 4 ] }
{ "_id" : 4, "things" : [ 1, 3, 4 ] }
> _
```

If we want to effect exactly one record there is more concise command called find and modify.

Here is the signature of find and modify command



```
db.foo.findAndModify( {
   query: <document>,
   update: <document>,
   upsert: <boolean> ,
   remove: <boolean>,
   new: <boolean>,
   sort: <document>,
   fields: <document>} );
```

Collection name: - we need to know which collection we are looking into

Document:-we need to look which was the exact document we want to modify

Query order:-sort order of the document if we want to specify multiple documents

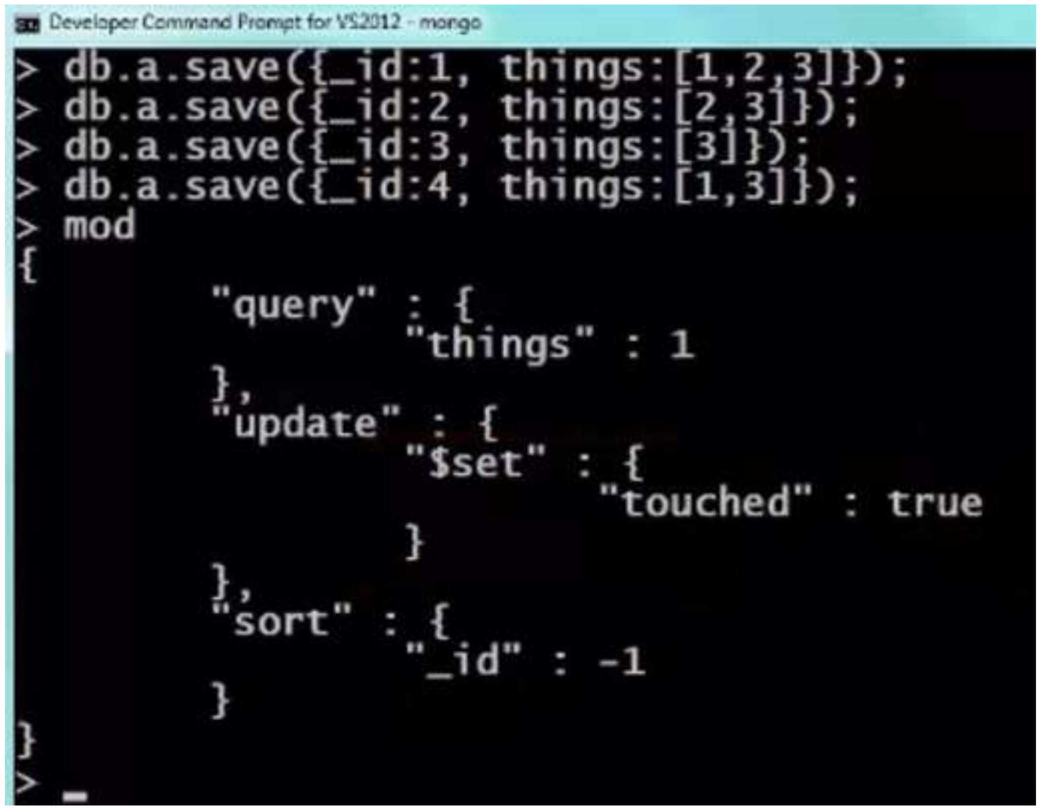
What change:-we need to specify what the change we want to make to the document we may want to upsert (set the upsert true) meaning create a new record if one doesn't exist

Delete it:-we can remove the document which deleted the record.

Return new:-Find and modify also returns the record that we are going to update from the database. By default find and modify returns the version of the before the change is made to it...if we set new to true it will return the document after the change was made to it.

Let’s create a query in object mod

```
var mod={
 "query" :{
 "things" :1
},
"update":{
"$set" :{
"touched":true
}
},
"sort":{
"_id" : -1
}
}
```



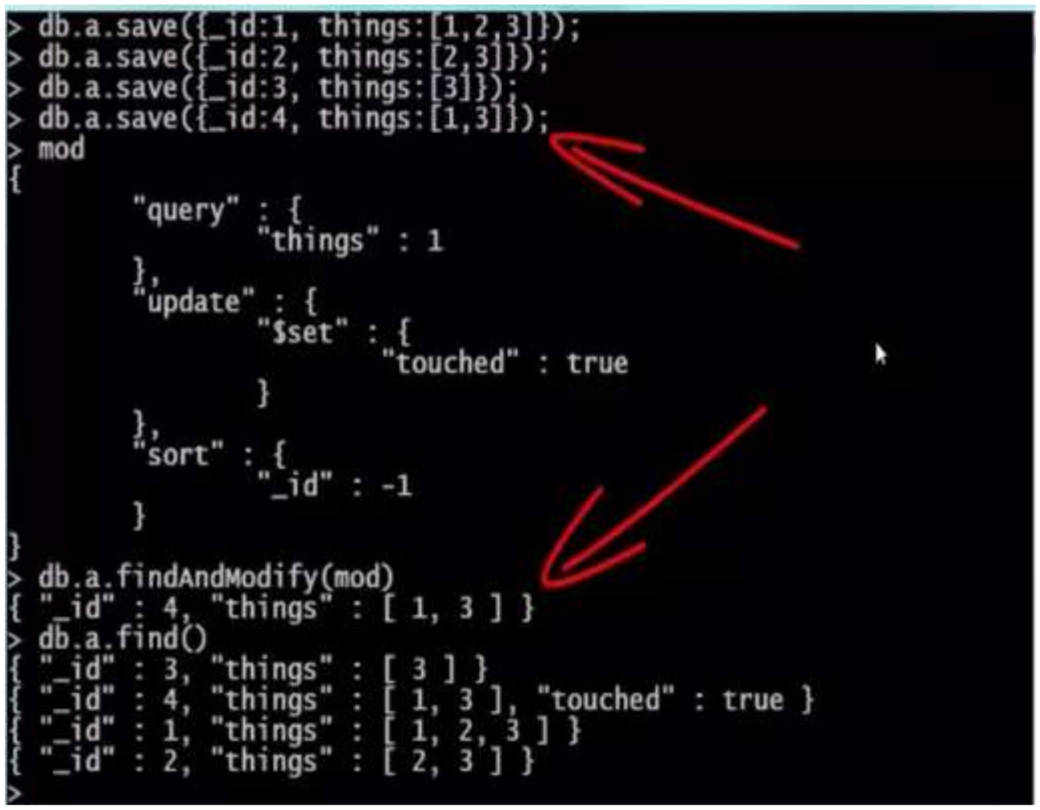
```
> db.a.save({_id:1, things:[1,2,3]});
> db.a.save({_id:2, things:[2,3]});
> db.a.save({_id:3, things:[3]});
> db.a.save({_id:4, things:[1,3]});
> mod
{
        "query" : {
                "things" : 1
        },
        "update" : {
                "$set" : {
                        "touched" : true
                }
        },
        "sort" : {
                "_id" : -1
        }
}
> _
```

The object mod has query finding something that has things with element 1.it has an update it wanna set the field touched to the value true...and it wanna sort by id in descending order as shown below..

Now if we issue the command

```
db.a.findAndModify(mod)
```

we get the document before it is modified as shown below and the current state is it has the touched field which we have just appended to it.

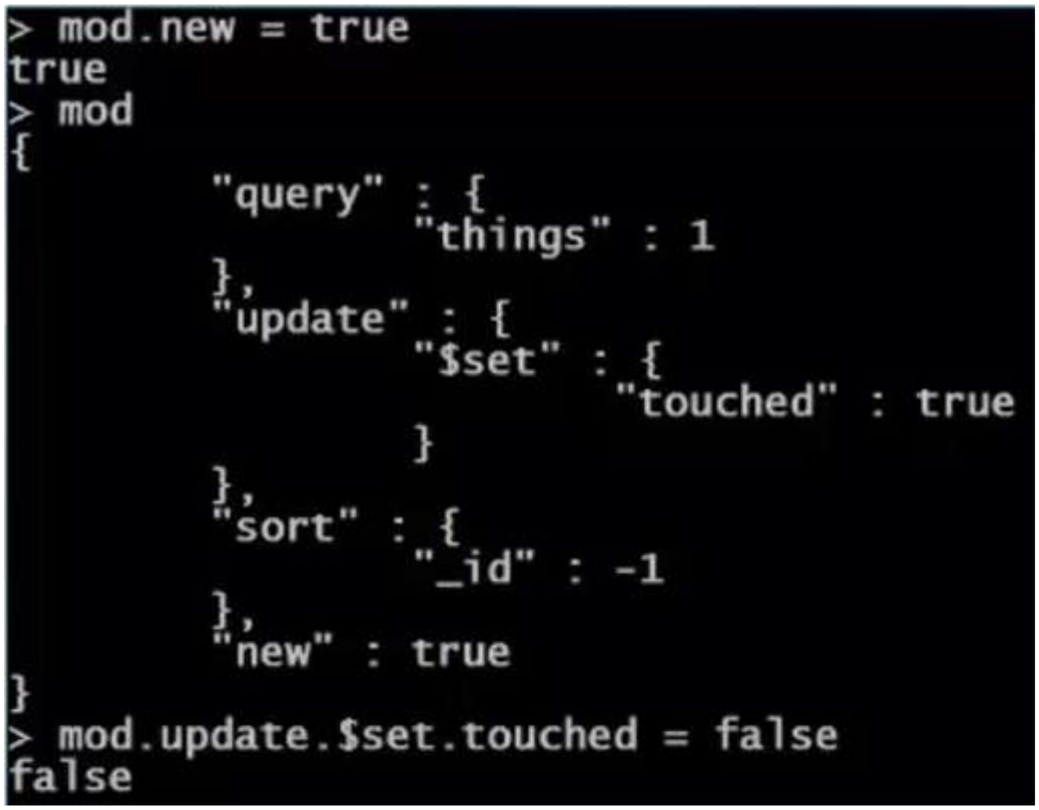

```
> db.a.save({_id:1, things:[1,2,3]});
> db.a.save({_id:2, things:[2,3]});
> db.a.save({_id:3, things:[3]});
> db.a.save({_id:4, things:[1,3]});
> mod
{
        "query" : {
                "things" : 1
        },
        "update" : {
                "$set" : {
                        "touched" : true
                }
        },
        "sort" : {
                "_id" : -1
        }
}
> db.a.findAndModify(mod)
{ "_id" : 4, "things" : [ 1, 3 ] }
> db.a.find()
{ "_id" : 3, "things" : [ 3 ] }
{ "_id" : 4, "things" : [ 1, 3 ], "touched" : true }
{ "_id" : 1, "things" : [ 1, 2, 3 ] }
{ "_id" : 2, "things" : [ 2, 3 ] }
>
```

Now let's set the touched field to false but this time lets return the record after it was modified

```
mod.new=true
```

Update touched field to be false this time

```
mod.update.$set.touched=false
```

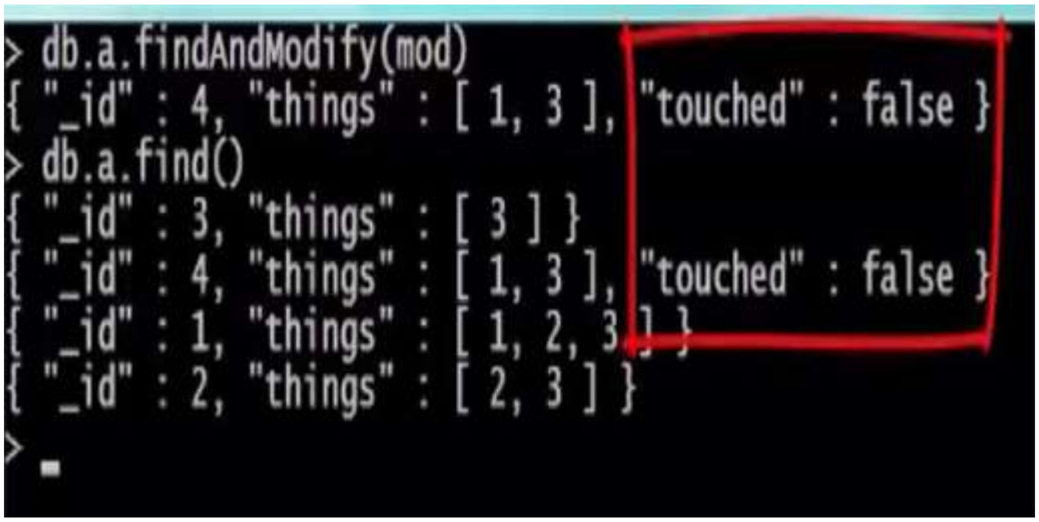


```
> mod.new = true
true
> mod
{
        "query" : {
                "things" : 1
        },
        "update" : {
                "$set" : {
                        "touched" : true
                }
        },
        "sort" : {
                "_id" : -1
        },
        "new" : true
}
> mod.update.$set.touched = false
false
```

Now let’s find the document

```
db.a.findAndModify(mod)
```

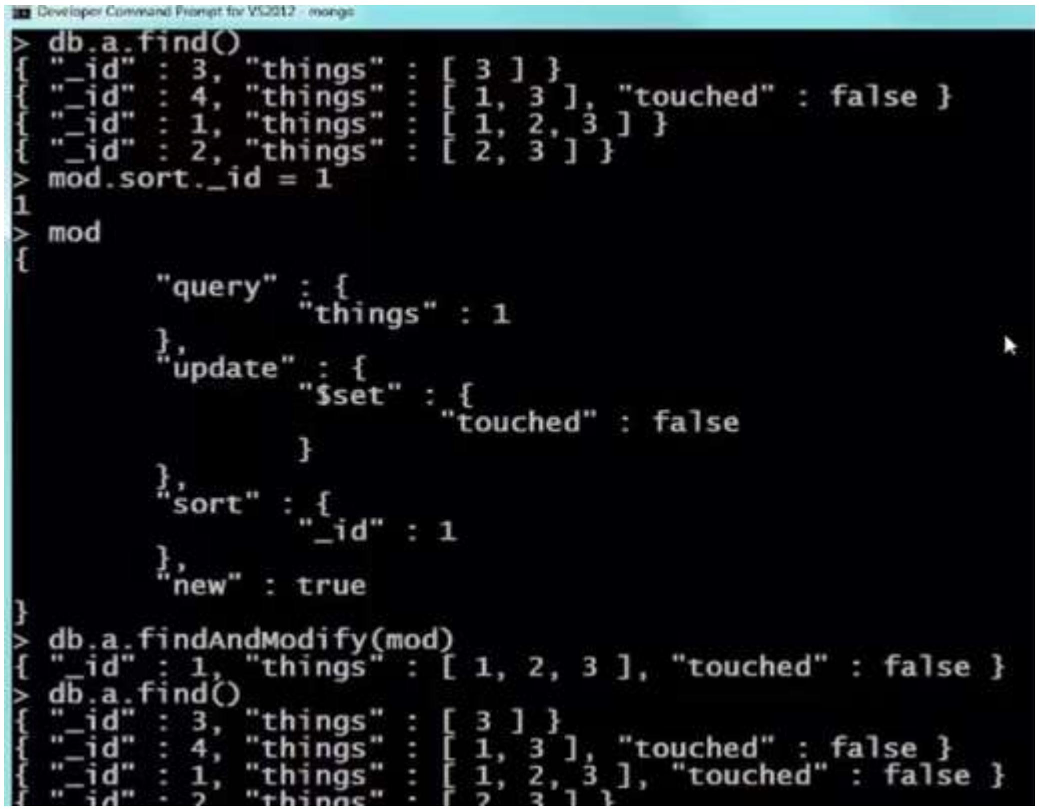
It returns the document after the modification as shown below



```
> db.a.findAndModify(mod)
{ "_id" : 4, "things" : [ 1, 3 ], "touched" : false }
> db.a.find()
{ "_id" : 3, "things" : [ 3 ] }
{ "_id" : 4, "things" : [ 1, 3 ], "touched" : false }
{ "_id" : 1, "things" : [ 1, 2, 3 ] }
{ "_id" : 2, "things" : [ 2, 3 ] }
>
```

Now let’s sort to find the first record for that we can issue the command

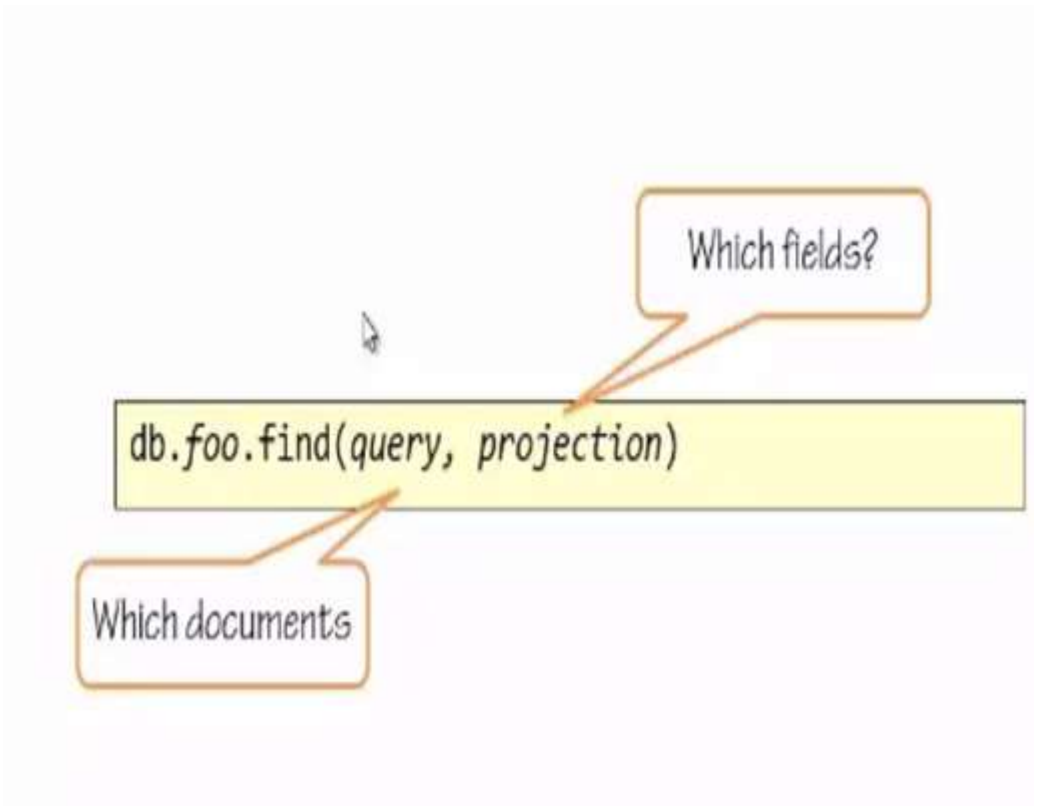
```
mod.sort._id=1
```


Developer Command Prompt for VS2012 - mongo

```
> db.a.find()
{ "_id" : 3, "things" : [ 3 ] }
{ "_id" : 4, "things" : [ 1, 3 ], "touched" : false }
{ "_id" : 1, "things" : [ 1, 2, 3 ] }
{ "_id" : 2, "things" : [ 2, 3 ] }
> mod.sort._id = 1
1
> mod
{
        "query" : {
                "things" : 1
        },
        "update" : {
                "$set" : {
                        "touched" : false
                }
        },
        "sort" : {
                "_id" : 1
        },
        "new" : true
}
> db.a.findAndModify(mod)
{ "_id" : 1, "things" : [ 1, 2, 3 ], "touched" : false }
> db.a.find()
{ "_id" : 3, "things" : [ 3 ] }
{ "_id" : 4, "things" : [ 1, 3 ], "touched" : false }
{ "_id" : 1, "things" : [ 1, 2, 3 ], "touched" : false }
{ "_id" : 2, "things" : [ 2, 3 ] }
```

We can see that the document with id 1 has been modified.

Exercise 8: Finding Documents

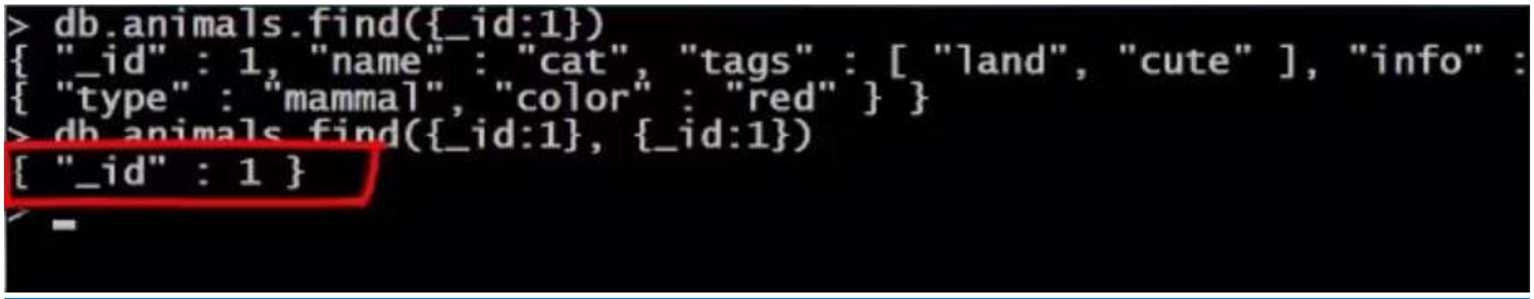


It had a query parameter and a projection parameter. The query parameter is a filter it defines the matching criteria to run against the documents

The Projection parameter defines up those documents to which part to be returned which is optional.

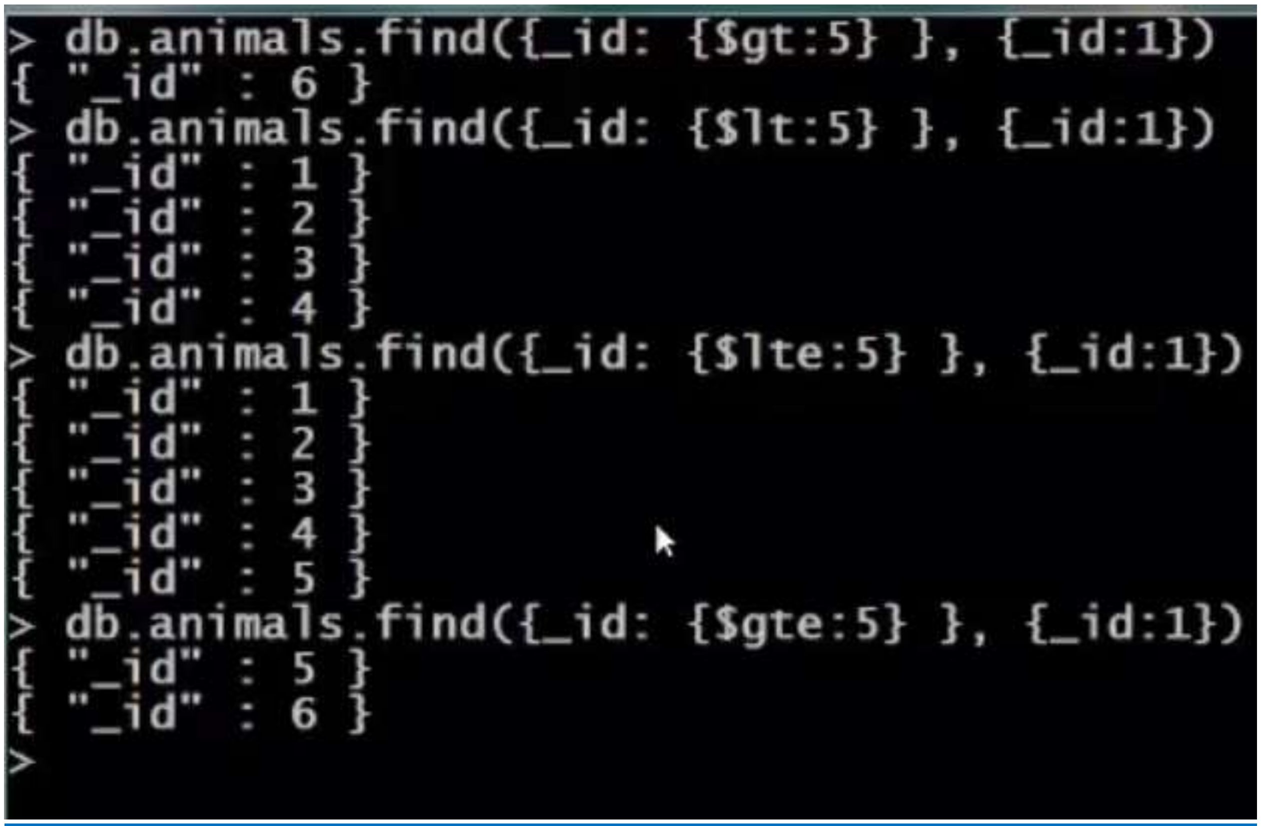
We will start using the find command which we have seen in previous articles.

If our document is containing huge data and we want to retrieve only some data from it we can specify a projection to which fields we want to return as shown below...



```
> db.animals.find({_id:1})
{ "_id" : 1, "name" : "cat", "tags" : [ "land", "cute" ], "info" :
{ "type" : "mammal", "color" : "red" } }
> db.animals.find({_id:1}, {_id:1})
{ "_id" : 1 }
> _
```

Here our animal database contains all the animals from 1 t0 6 so now let's find all the animals by issuing a commands shown below…



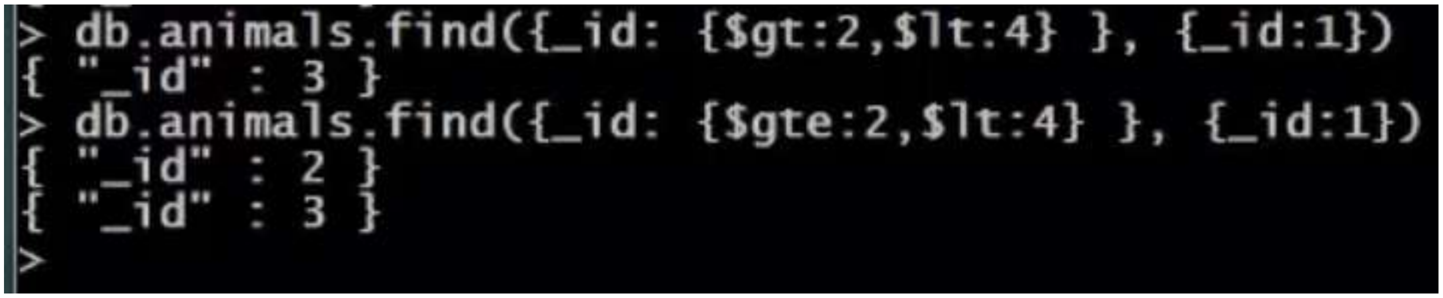
```
> db.animals.find({_id: {$gt:5} }, {_id:1})
{ "_id" : 6 }
> db.animals.find({_id: {$lt:5} }, {_id:1})
{ "_id" : 1 }
{ "_id" : 2 }
{ "_id" : 3 }
{ "_id" : 4 }
> db.animals.find({_id: {$lte:5} }, {_id:1})
{ "_id" : 1 }
{ "_id" : 2 }
{ "_id" : 3 }
{ "_id" : 4 }
{ "_id" : 5 }
> db.animals.find({_id: {$gte:5} }, {_id:1})
{ "_id" : 5 }
{ "_id" : 6 }
>
```

Here the gt operator means greater than and lt operator means less than

Also gte means greater equal and lte less than equal

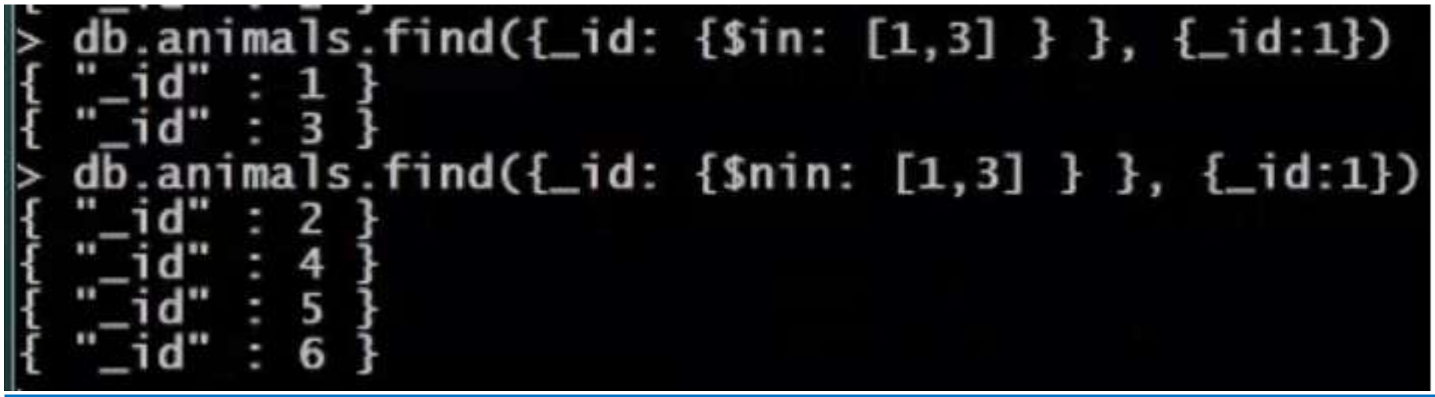
So we can use these operators to query our database as shown above

We can also specify a range



```
> db.animals.find({_id: {$gt:2,$lt:4} }, {_id:1})
{ "_id" : 3 }
> db.animals.find({_id: {$gte:2,$lt:4} }, {_id:1})
{ "_id" : 2 }
{ "_id" : 3 }
>
```

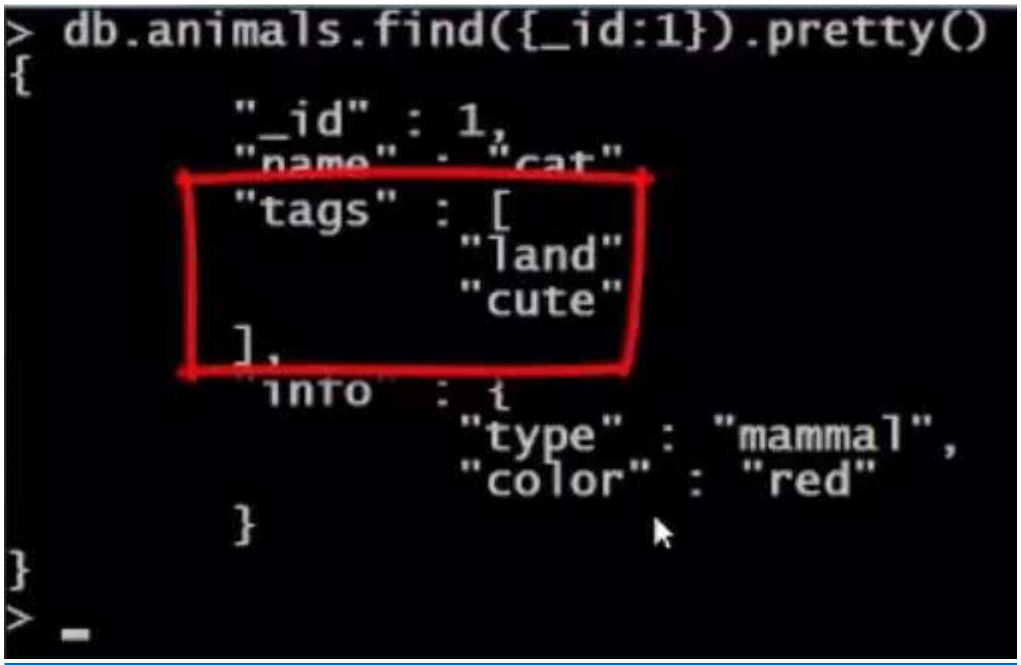
We can also provide the matching query



```
> db.animals.find({_id: {$in: [1,3] } }, {_id:1})
{ "_id" : 1 }
{ "_id" : 3 }
> db.animals.find({_id: {$nin: [1,3] } }, {_id:1})
{ "_id" : 2 }
{ "_id" : 4 }
{ "_id" : 5 }
{ "_id" : 6 }
```

Here we can see that we are finding the documents which are 1 and 3 and the same in opposite

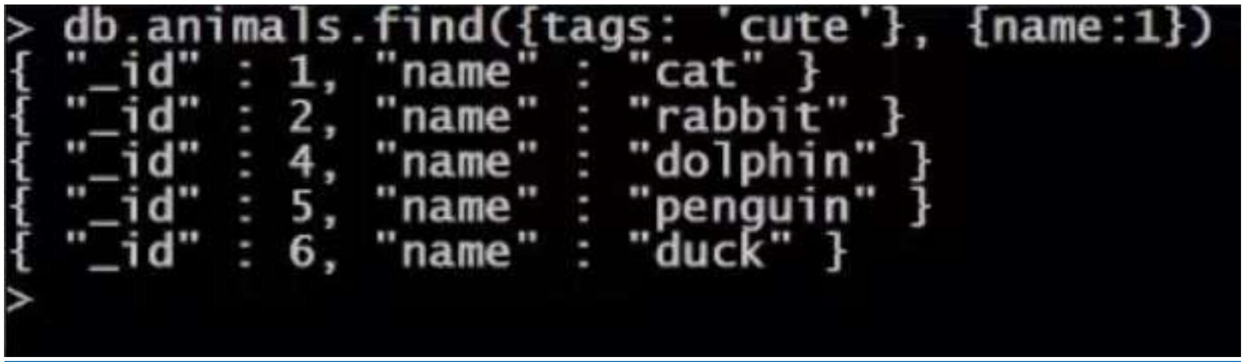
Our animal document contains a tag field and the tags field contains an array of tags



```
> db.animals.find({_id:1}).pretty()
{
        "_id" : 1,
        "name" : "cat",
        "tags" : [
                "land"
                "cute"
        ],
        "info" : {
                "type" : "mammal",
                "color" : "red"
        }
}
> _
```

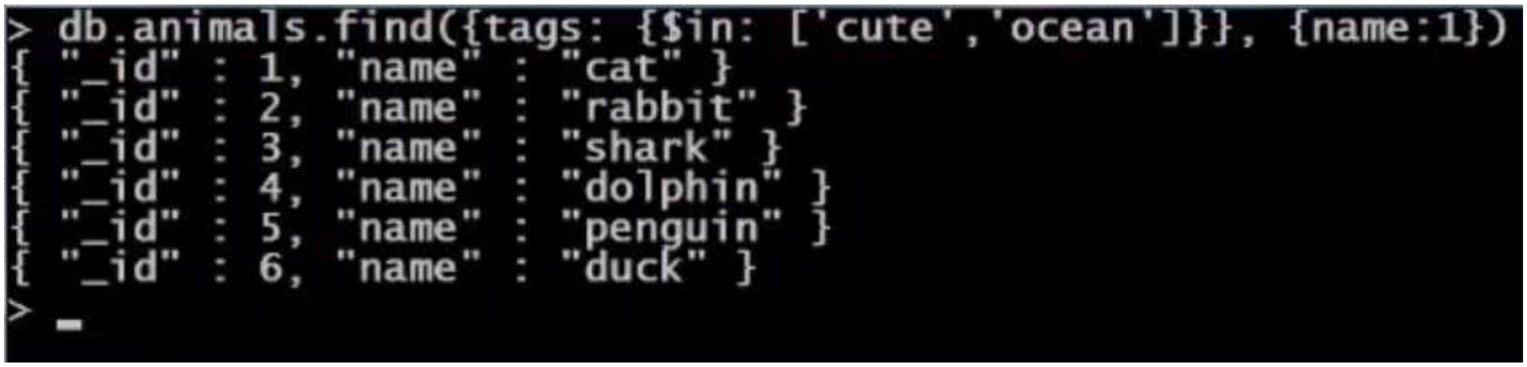
So let us see how to match among those…

We can notice that we are finding the tag field of tag which matches cute



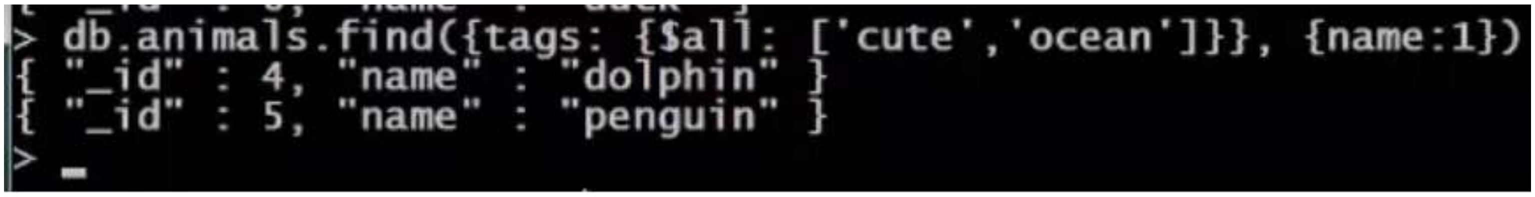
```
> db.animals.find({tags: 'cute'}, {name:1})
{ "_id" : 1, "name" : "cat" }
{ "_id" : 2, "name" : "rabbit" }
{ "_id" : 4, "name" : "dolphin" }
{ "_id" : 5, "name" : "penguin" }
{ "_id" : 6, "name" : "duck" }
>
```

Now let’s find animals who are cute or in the ocean we will get some animals which are cute and some which are in ocean but not necessarily both of them as shown below ….



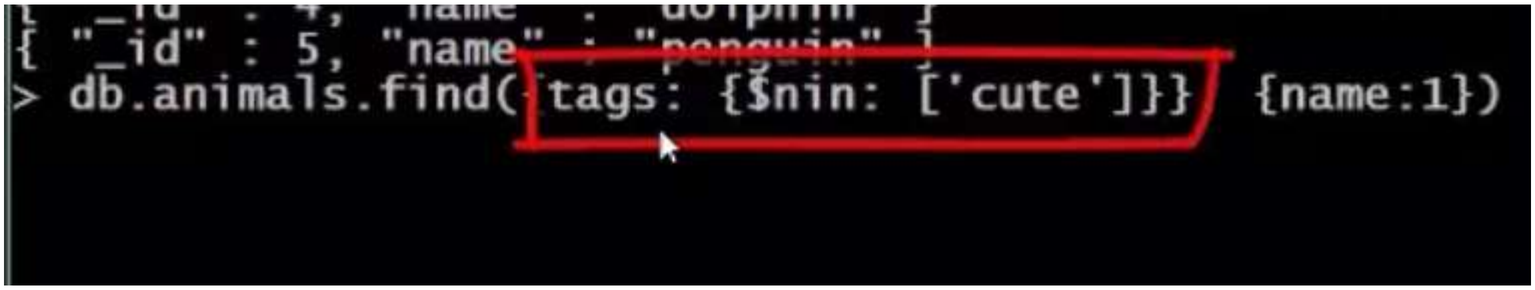
```
> db.animals.find({tags: {$in: ['cute','ocean']}}, {name:1})
{ "_id" : 1, "name" : "cat" }
{ "_id" : 2, "name" : "rabbit" }
{ "_id" : 3, "name" : "shark" }
{ "_id" : 4, "name" : "dolphin" }
{ "_id" : 5, "name" : "penguin" }
{ "_id" : 6, "name" : "duck" }
> _
```

Now if we want to insist that there would be both ocean and cute tagged we can issue the following command by using all operator as shown below…



```
> db.animals.find({tags: {$all: ['cute','ocean']}}, {name:1})
{ "_id" : 4, "name" : "dolphin" }
{ "_id" : 5, "name" : "penguin" }
> _
```

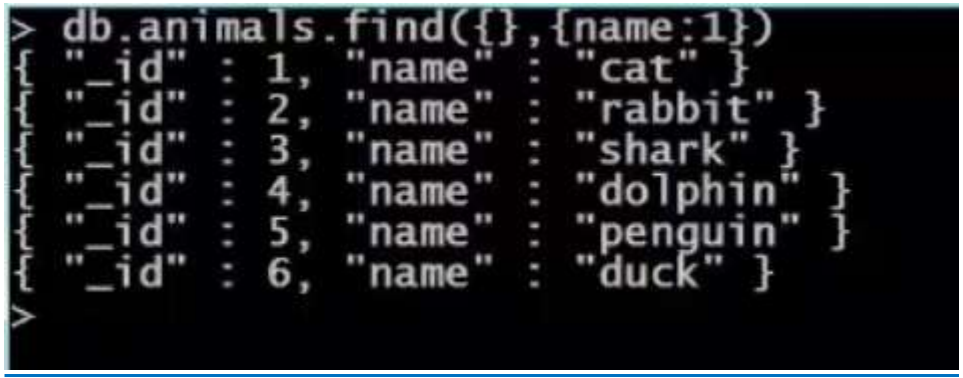
If we want to find the document which are none in the tags we can use nin operator as shown below



```
{ "_id" : 5, "name" : "penguin" }
> db.animals.find({tags: {$nin: ['cute']}}, {name:1})
```

Exercise 9: Cursor Operations

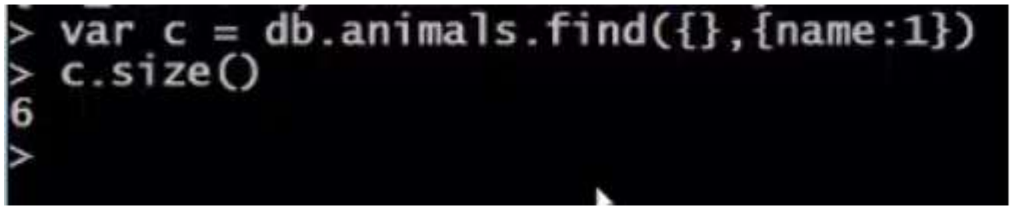
For example if we issue the query as shown



```
> db.animals.find({},{name:1})
{ "_id" : 1, "name" : "cat" }
{ "_id" : 2, "name" : "rabbit" }
{ "_id" : 3, "name" : "shark" }
{ "_id" : 4, "name" : "dolphin" }
{ "_id" : 5, "name" : "penguin" }
{ "_id" : 6, "name" : "duck" }
>
```

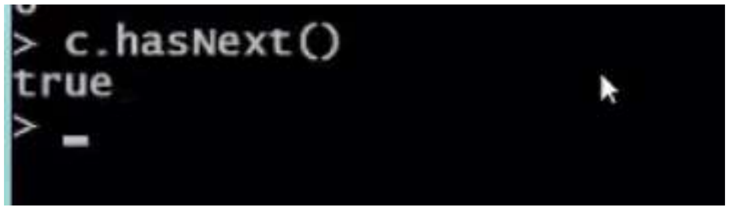
The shell has enumerated all other documents matching the criteria.

Now let's capture this cursor into a variable and we can look at the cursor size as shown



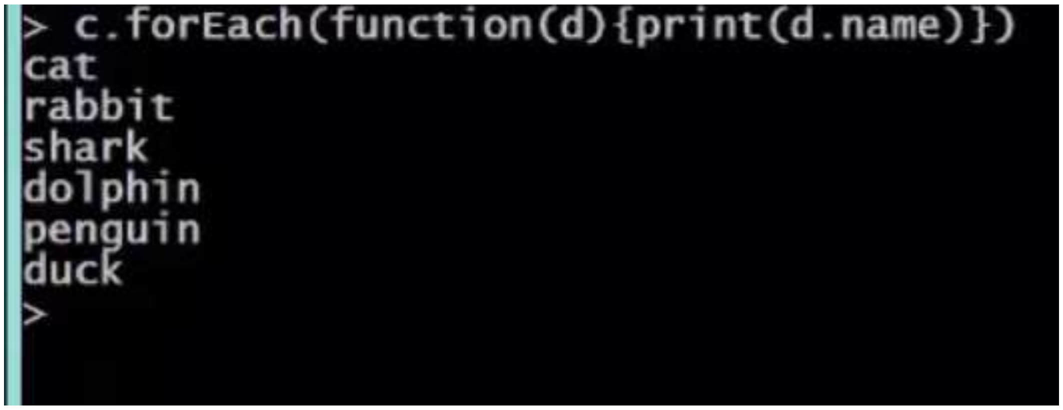
```
> var c = db.animals.find({},{name:1})
> c.size()
6
>
```

To support iterating to the cursor we can query the cursor whether it had items we can see that it had items as shown



```
> c.hasNext()
true
> _
```

If we want to iterate to the cursor in mongo db we have the foreach method as shown below



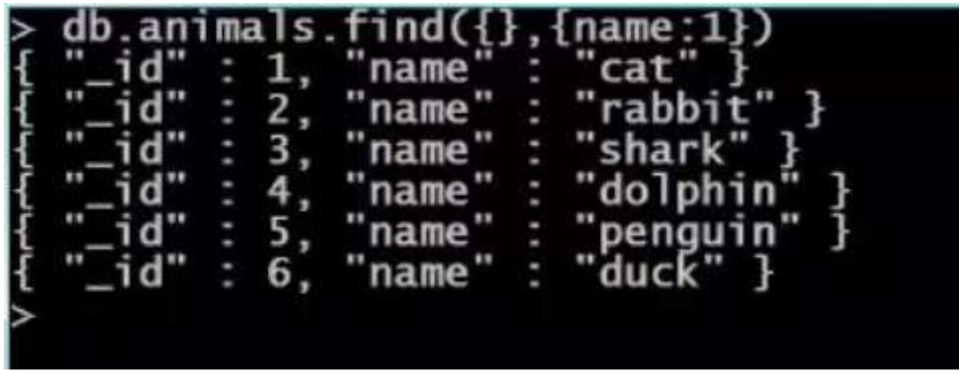
```
> c.forEach(function(d){print(d.name)})
cat
rabbit
shark
dolphin
penguin
duck
>
```

So we iterated through the whole cursor and for each one we executed a function

Sorting

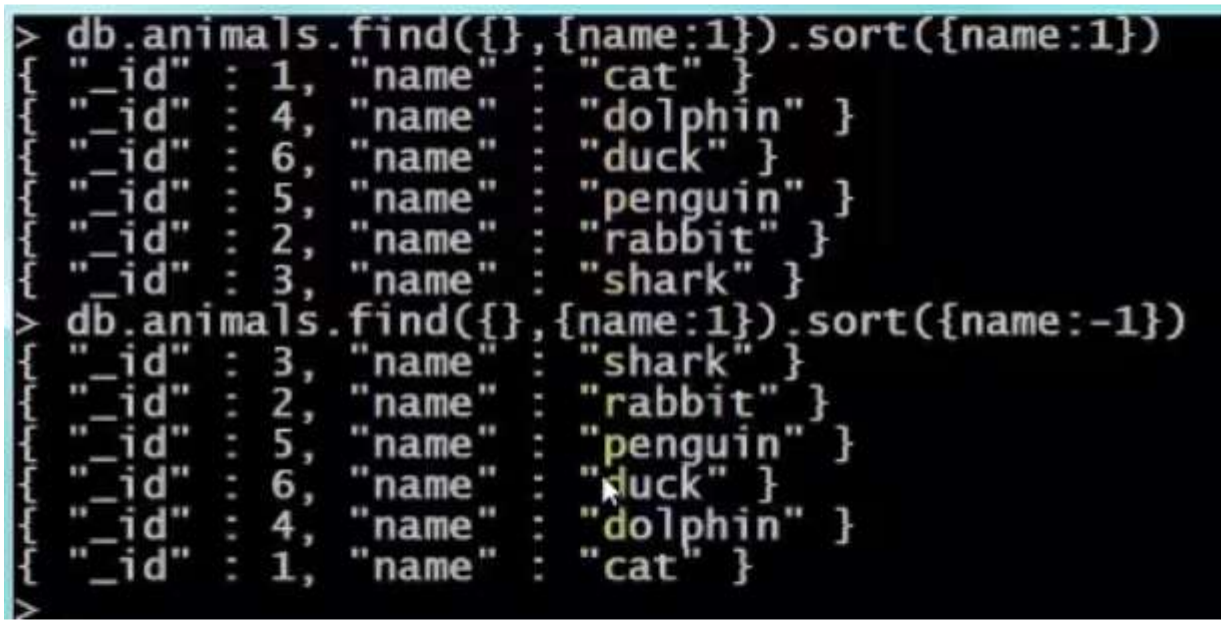
The more obvious and popular cursor option is sorting

Now let's find some animals in our animal's database and retrieve their names and sort them by the name



```
> db.animals.find({},{name:1})
{ "_id" : 1, "name" : "cat" }
{ "_id" : 2, "name" : "rabbit" }
{ "_id" : 3, "name" : "shark" }
{ "_id" : 4, "name" : "dolphin" }
{ "_id" : 5, "name" : "penguin" }
{ "_id" : 6, "name" : "duck" }
>
```

The sort direction positive is ascending and negative is descending



```
> db.animals.find({},{name:1}).sort({name:1})
{ "_id" : 1, "name" : "cat" }
{ "_id" : 4, "name" : "dolphin" }
{ "_id" : 6, "name" : "duck" }
{ "_id" : 5, "name" : "penguin" }
{ "_id" : 2, "name" : "rabbit" }
{ "_id" : 3, "name" : "shark" }
> db.animals.find({},{name:1}).sort({name:-1})
{ "_id" : 3, "name" : "shark" }
{ "_id" : 2, "name" : "rabbit" }
{ "_id" : 5, "name" : "penguin" }
{ "_id" : 6, "name" : "duck" }
{ "_id" : 4, "name" : "dolphin" }
{ "_id" : 1, "name" : "cat" }
>
```

Exercise 10: Indexing

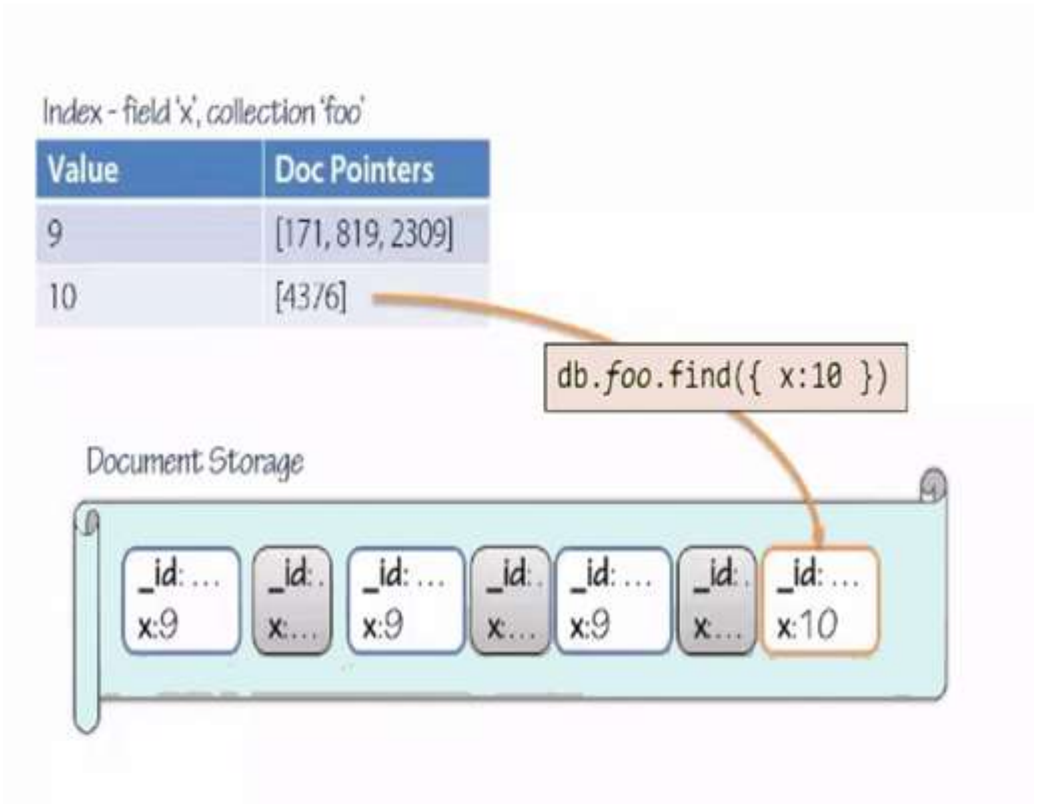
First let us assume we have a collection named foo and we want to find all the collections with 10 for the value of the field x.

So we say:

db.foo.find({ x :10})

Now let us think, what does the server do to find the document?
The server will visit each and every document and check the value of x is equal to 10 and if so return it. That's an over-simplification but it shows us the problem. We will need to scan every location on disk to dig up every document and compare the value of the field x. That is a very, very slow operation and if we want to do it quickly then we need to find a better strategy. Indexing is done by Mongo by storing all the documents in its own location on disk. An index logically holds a mapping to those locations from field values. In the preceding example an index on field x of the collection foo has an entry of each possible value of x associated with a list of document locations and each of

those documents contain a value or key. So for example we have a bunch of documents with x equaling 9 and x equaling 10.



We will design a query to find the documents where the field x matches the value 10. Mongo will look in the index to find the entry whose value is 10 and jump directly to that document. This is much much faster than scanning the entire disk and much much faster than loading each and every document.

Regular Indexes

The regular index is an index that we can use on a single or multiple field with multiple values as well.

Compound Indexes

A compound index includes more than one field of the documents in a collection.

Multikey Indexes

A multikey index references an array and records a match if a query includes any value in the array.

Geo Indexes

The geo index is optimized for geographical queries. This supports proximity of points to the center.

Text Indexes

Text indexes allows us to do the things like search engines do, parsing text queries and comparing them against text fields.

Time to Live Index

This supports Expiring documents using a TTL index we can designate a date time field on our document to be an expiration date and Mongo will automatically remove the document from our collection when it expires. This again reduces our overhead in writing all kind of patch works in removing our self.

Getting Started With MongoDB and ASP.Net

Create a simple ASP.NET application that retrieves data. You can have a look at my other articles of MongoDB from here.

Installing Mongo db

Installing MongoDB in Windows is very easy and simple. Use the following procedure to get it running:

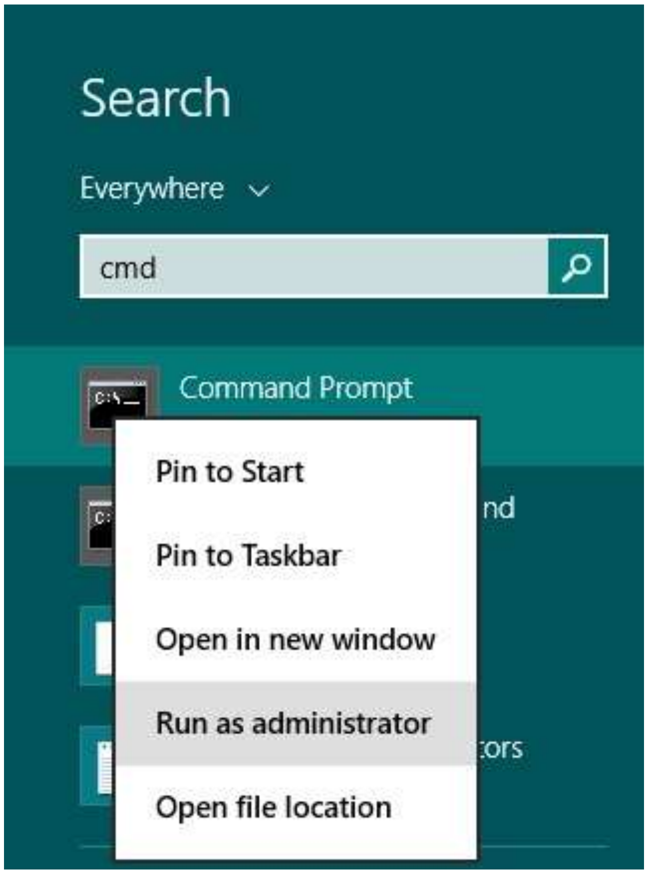
1. Download **MongoDB** for Windows from "http://www.mongodb.org/downloads".
2. After downloading, create a folder named MongoDB and extract the Zip file into that folder.

That's all. MongoDB is now installed. We can find many files in the folder but the key files are:

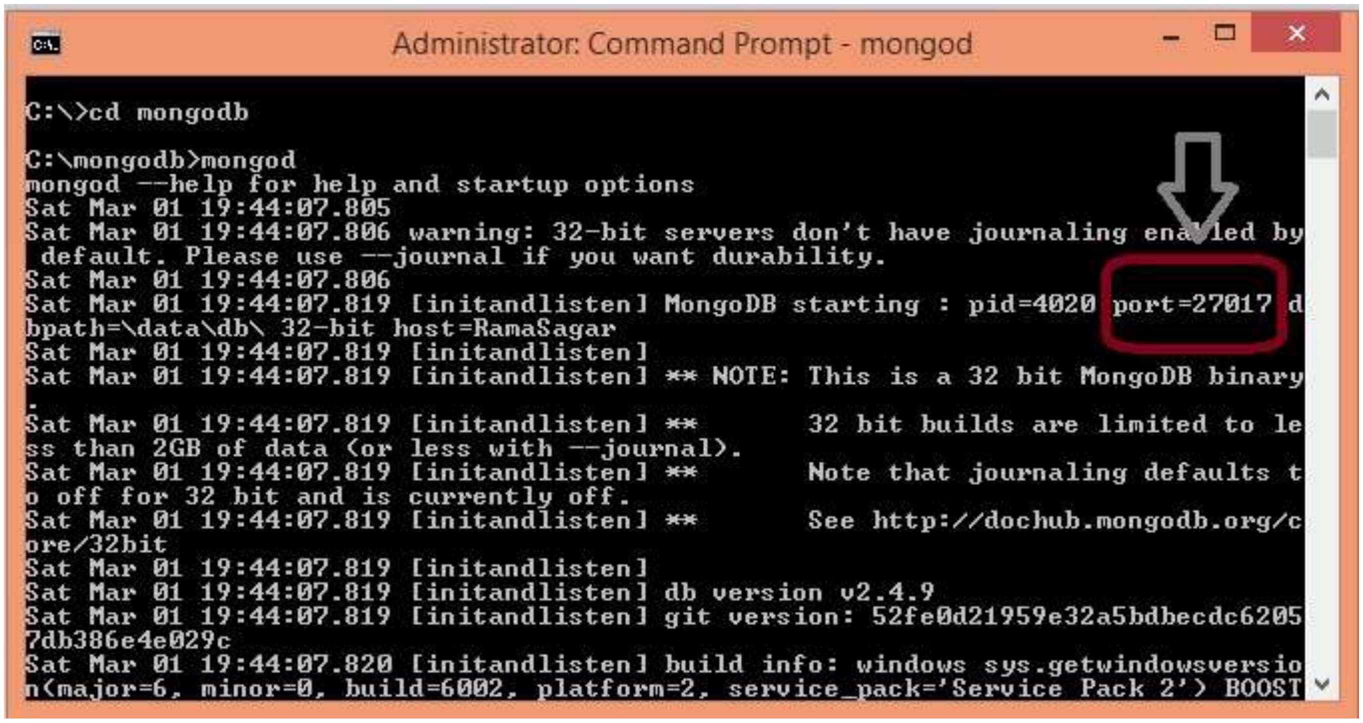
- *mongod.exe*: The Mongo database
- *mongo.exe*: The administrative shell
- *Mongos.exe*: The sharding controller (Sharding is the process of storing data records across multiple machines and is MongoDB's approach to meeting the demands of data growth.)

Now let's get started by setting up the server and creation of a database; use the following procedure to do that.

- Step 1: Open the command prompt in administrator mode as shown and go to the MongoDB's directory. (The folder where we extracted the Zip contentto.) For mine, the folder name is **mongodb.**



- Step 2: Type "mongod" and press Enter. And Mongo DB has started. It uses port **27017** by default as shown below.

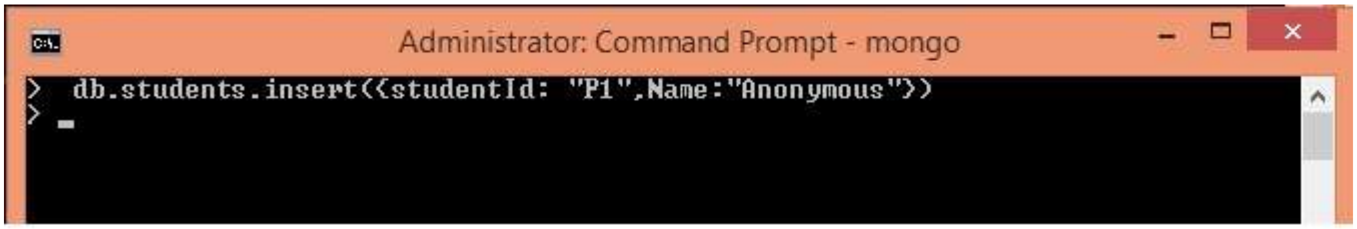

Administrator: Command Prompt - mongod

```
C:\>cd mongodb

C:\mongodb>mongod
mongod --help for help and startup options
Sat Mar 01 19:44:07.805
Sat Mar 01 19:44:07.806 warning: 32-bit servers don't have journaling enabled by
 default. Please use --journal if you want durability.
Sat Mar 01 19:44:07.806
Sat Mar 01 19:44:07.819 [initandlisten] MongoDB starting : pid=4020 port=27017 d
bpath=\data\db\ 32-bit host=RamaSagar
Sat Mar 01 19:44:07.819 [initandlisten]
Sat Mar 01 19:44:07.819 [initandlisten] ** NOTE: This is a 32 bit MongoDB binary
.
Sat Mar 01 19:44:07.819 [initandlisten] **       32 bit builds are limited to le
ss than 2GB of data (or less with --journal).
Sat Mar 01 19:44:07.819 [initandlisten] **       Note that journaling defaults t
o off for 32 bit and is currently off.
Sat Mar 01 19:44:07.819 [initandlisten] **       See http://dochub.mongodb.org/c
ore/32bit
Sat Mar 01 19:44:07.819 [initandlisten]
Sat Mar 01 19:44:07.819 [initandlisten] db version v2.4.9
Sat Mar 01 19:44:07.819 [initandlisten] git version: 52fe0d21959e32a5bdbecdc6205
7db386e4e029c
Sat Mar 01 19:44:07.820 [initandlisten] build info: windows sys.getwindowsversio
n(major=6, minor=0, build=6002, platform=2, service_pack='Service Pack 2') BOOST
```

- Step 3: Now open another command prompt and go to mongodb's directory. Type "mongo". This command will connect to the default **test** database.

MongoDB is schemaless and contains no table or relation. It maintains a collection of data. So, for now to keep things simple, Let's create a "Students" collection in the "test" database with a student "studentId = P1 and Name = Anonymous". Just type the following command:

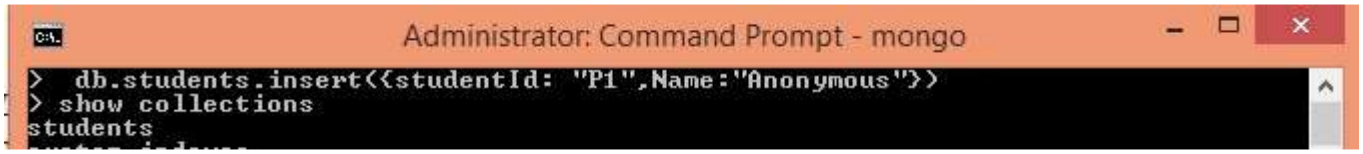
```
db.students.insert({studentId: "P1",Name:"Anonymous"})
```


Administrator: Command Prompt - mongo

```
> db.students.insert({studentId: "P1",Name:"Anonymous"})
> _
```

Let's check whether or not the students collection was created by issuing the command:

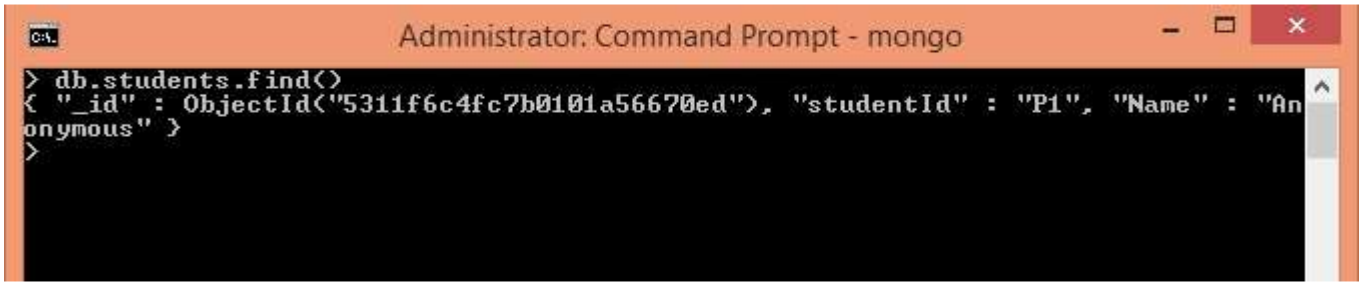
show collections



```
> db.students.insert({studentId: "P1",Name:"Anonymous"})
> show collections
students
```

Let us find the data in the collection by issuing:

db.students.find()



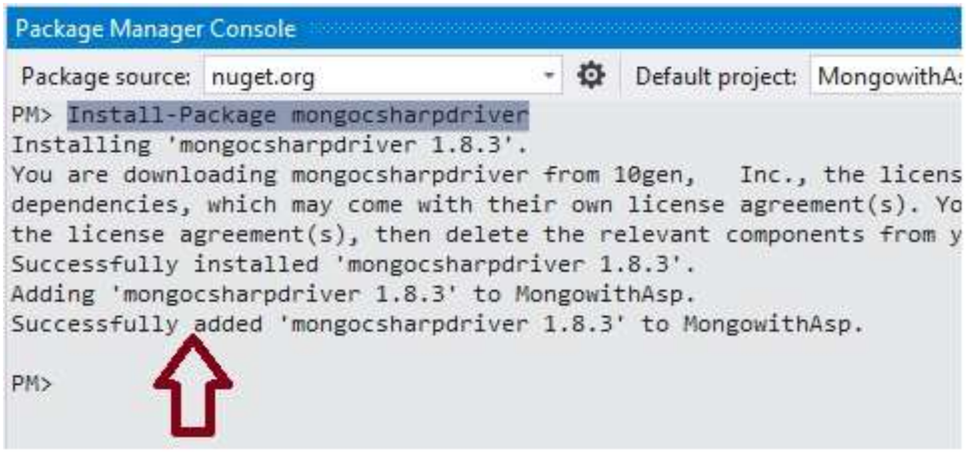
```
> db.students.find()
{ "_id" : ObjectId("5311f6c4fc7b0101a56670ed"), "studentId" : "P1", "Name" : "An
onymous" }
>
```

We can see our data in the collection.

Now let's us retrieve the data using an ASP.NET application.

Create an ASP.NET application and and add the mongocsharpdriver using the package manger console as shown below.

Install-Package mongocsharpdriver



```
PM> Install-Package mongocsharpdriver
Installing 'mongocsharpdriver 1.8.3'.
You are downloading mongocsharpdriver from 10gen,  Inc., the licens
dependencies, which may come with their own license agreement(s). Yo
the license agreement(s), then delete the relevant components from y
Successfully installed 'mongocsharpdriver 1.8.3'.
Adding 'mongocsharpdriver 1.8.3' to MongowithAsp.
Successfully added 'mongocsharpdriver 1.8.3' to MongowithAsp.

PM>
```

Now let's define the connection string for the **MongoDB** server. By default, it runs on the port 27017, so, define the connection string inside the "web.config" file as in the following:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=169433
  -->
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
  </system.web>
<appSettings>
    <add key="connectionString" value="Server=localhost:27017"/>
  </appSettings>
</configuration>
```

Our application is now ready to communicate with the **MongoDB**.

Now create a helping class named "studentinfo" that contains "_id" that one is an ObjectId type and uses MongoDB.Bson, studentId and Name; all are string type.

```
using MongoDB.Bson;

namespace MongowithAsp
{
    public class studentsinfo
    {
        public ObjectId _id { get; set; }
        public string studentId { get; set; }
        public string Name { get; set; }
    }
}
```

Now, create a simple button and a label as shown below:

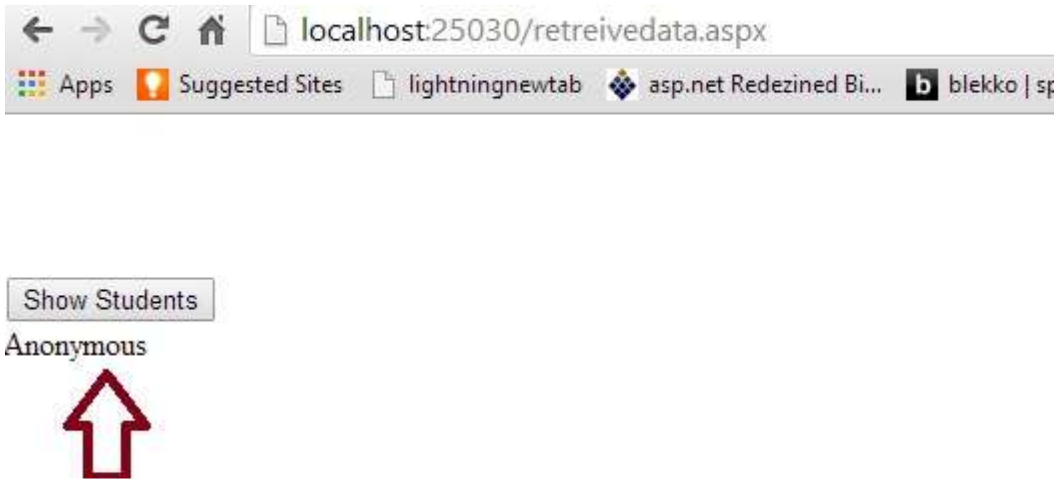
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="retreivedata.aspx.cs" Inherits="MongowithAsp.retre
ivedata" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>ASP.NET with Mongo</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1> </h1>
        <asp:Button ID="showButton" runat="server" Text="Show Students" OnClick="showButton_Click" />
    </div>
        <asp:Label ID="nameLabel" runat="server"></asp:Label>
    </form>
</body>
</html>
```

And write the following code for the button click event:

```
using MongoDB.Driver;
using System;
using System.Collections.Generic;
using System.Configuration;

namespace MongowithAsp
{
    public partial class retreivedata : System.Web.UI.Page
    {
        string name = "";
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void showButton_Click(object sender, EventArgs e)
        {
            List<studentsinfo> names = new List<studentsinfo>();
            MongoServer server = MongoServer.Create(ConfigurationManager.AppSettings["connectionString"]);
            MongoDatabase myDB = server.GetDatabase("test");
            MongoCollection<studentsinfo> Students = myDB.GetCollection<studentsinfo>("students");
            foreach (studentsinfo Astudent in Students.FindAll())

            {
                namename = name + " " + Astudent.Name;
                names.Add(Astudent);
            }
            namenameLabel.Text = name;
        }
    }
}
```

We have created an instance of MongoServer using the connection string and by iterating through the collection, we are getting the individual student and adding it in the "names" list.

Let's debug to check the output.



```
Anonymous
```

Resources and References

Mongo DB Resources, Publications, and Channel 9 Videos

Mongo DB Documentation Site

http://docs.mongodb.org/manual/

Community

http://www.mongodb.org/get-involved