Subject Code: 18CS5DS07L **Total Hours: 15**
Credits: 01 **L-T-P: 0-0-2**

NOSQL Database Lab Manual
Department of Computer Science
5th Semester – DS

Lab Experiments List

	• Add another piranha, and a naked mole rat called Henry. • Use find to list all the pets. Find the ID of Mikey the Gerbil. • Use find to find Mikey by id. • Use find to find all the gerbils. • Find all the creatures named Mikey. • Find all the creatures named Mikey who are gerbils. • Find all the creatures with the string "dog" in their species.	

6.	AirPhone Corp is a famous telecom company. They have customers in all locations. Customers use AirPhone Corp's network to make calls. Government has brought in a regulation that all telecom companies should store call details of their customers. This is very important from a security point of view and all telecom companies have to retain this data for 15 years. AirPhone Corp already stores all customer details data, for their analytics team. But due to a surge in mobile users in recent years, their current database cannot handle huge amounts of data. Current database stores only six months of data. AirPhone Corp now wants to scale their database and wants to store 15 years of data.
	Data contains following columns:
	Source : Phone number of caller
	Destination : Phone number of call receiver
	Source_location : Caller's city
	Destination_location : Call receiver's city
	Call_duration : phone call duration
	Roaming : Flag to check if caller is in roaming
	Call_charge : Money charged for call
	Sample Data:
	{
	source: "+919612345670",
	destination: "+919612345671",
	source_location: "Delhi",
	destination_location: "Mumbai",
	call_duration: 2.03,
	roaming: false,
	call_charge: 2.03
	}
	After discussing the requirements with database and architecture team, it has been decided that they should use MongoDb. You have been given the task to Setup a distributed system (database) such that data from different locations go to different nodes (to distribute the load)
	• Import data to sharded collection
	• Check data on each shard for distribution
		32-35

7.	Execute below sets of problem by taking reference of Experiment Number 06 and find out: • Add additional node to existing system (to test if we can add nodes easily when data increases) • Check the behavior of cluster (data movement) on adding a share. Check the behavior of query for finding a document with source location Mumbai	**36**

8.	Anand Corp is a leading corporate training provider. A lot of prestigious organizations send their employees to Anand Corp for training on different skills. As a distinct training provider, Anand Corp has decided to share analysis report with their clients. This report will help their clients know the employees who have completed training and evaluation exam, what are their strengths, and what are the areas where employees need improvement. This is going to be a unique selling feature for the Anand Corp. As Anand Corp is already doing great business and they give training to a large number of people every month, they have huge amount of data to deal with. They have hired you as an expert and want your help to solve this problem.
	Attributes of data:
	Id : id of the person who was trained
	Name : name of the person who was trained
	Evaluation : evaluation term
	Score : score achieved by the person for the specific term
	A person can undergo multiple evaluations. Each evaluation will have a unique result score.
	You can see the sample data below.
	Sample Data
	{
	"_id":0,
	"name":"Andy",
	"results": [
	{"evaluation":"term1","score":1.463179736705023},
	{"evaluation":"term2","score":11.78273309957772},
	{"evaluation":"term3","score":6.676176060654615}
	]
	}
	PQR Corp has assigned the following tasks to you to analyze the results:
	Find count and percentage of employees who failed in term 1, the passing score being 37.
		37-39

9.	Execute below sets of problem by taking reference of Experiment Number 08 and find out: • Find employees who failed in aggregate (term1 + term2 + term3). Find the Average score of trainees for term1.	**40**
10.	Execute below sets of problem by taking reference of Experiment Number 08 and find out: • Find the Average score of trainees for aggregate (term1 + term2 + term3). • Find number of employees who failed in all the three (term1 + term2 + term3). • Find the number of employees who failed in any of the three (term1 + term2 + term3	**41-44**
11	Practice Question Set -1	**45-49**
12	Practice Question Set -2	**50-52**
13	Practice Question Set -3	**53-55**

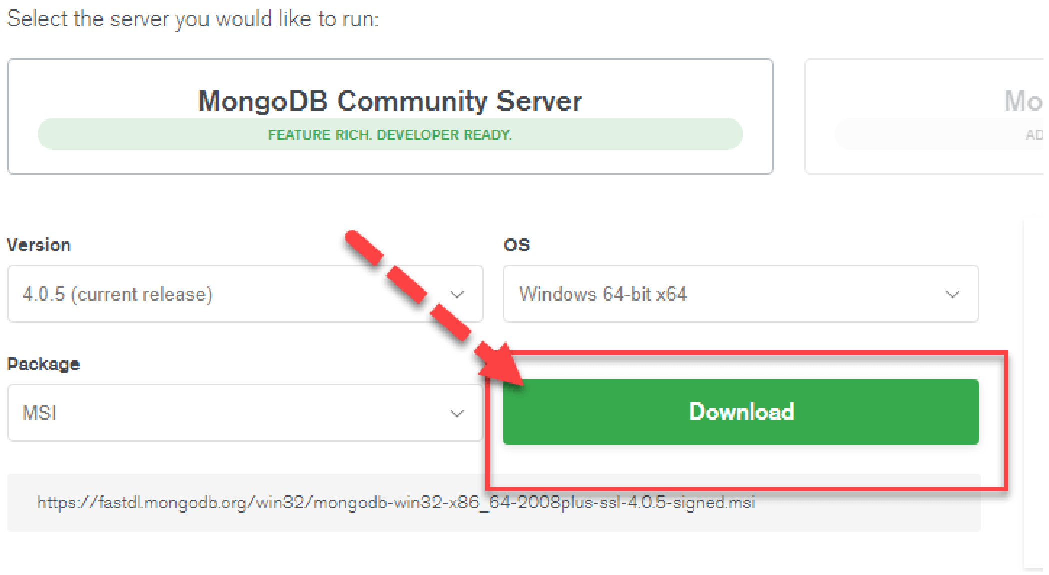
Experiment 1

Prepare and install infrastructure for setting up MongoDB lab.
•Install MongoDB Community Edition

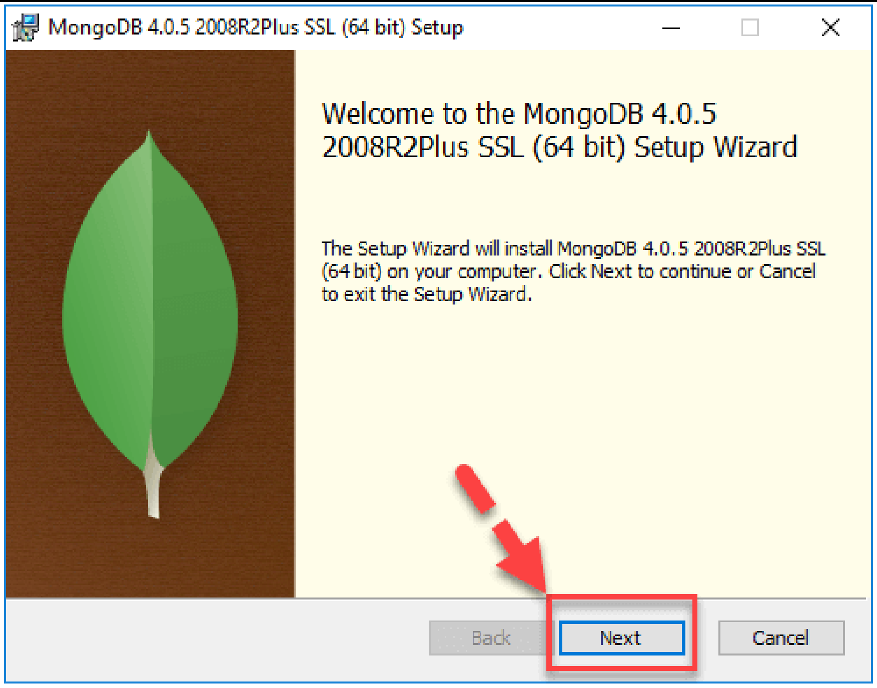
- Download MongoDB Community Edition
- Run the MongoDB installer
- Follow the MongoDB Community Edition installation wizard

•Run MongoDB Community Edition as a Windows Service
•Run MongoDB Community Edition from the Command Interpreter
It is advised to follow below URL:
https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/

How to Download & Install MongoDB on Windows
The following steps can be used to download and install MongoDB on Windows 10
Step 1) Download MongoDB Community Server
Go to link and Download MongoDB Community Server. We will install the 64-bit version for Windows.


Select the server you would like to run:
MongoDB Community Server
FEATURE RICH. DEVELOPER READY.
Version: 4.0.5 (current release)
OS: Windows 64-bit x64
Package: MSI
https://fastdl.mongodb.org/win32/mongodb-win32-x86_64-2008plus-ssl-4.0.5-signed.msi

Step 2) Click on Setup
Once download is complete open the msi file. Click Next in the start up screen

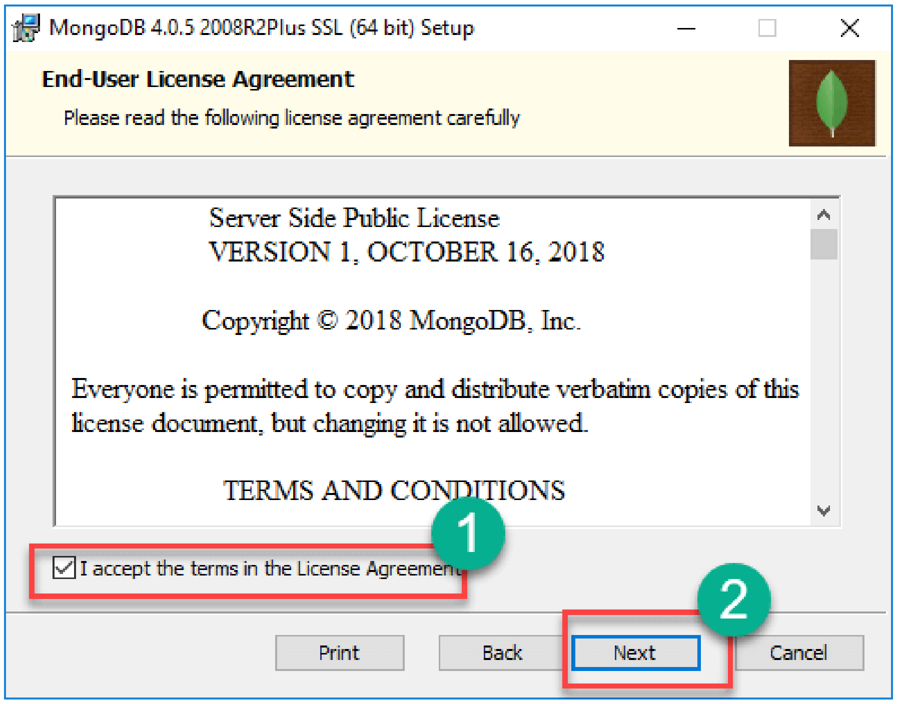

MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup

Welcome to the MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup Wizard

The Setup Wizard will install MongoDB 4.0.5 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Step 3) Accept the End-User License Agreement

1. Accept the End-User License Agreement
2. Click Next


MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup

End-User License Agreement

Please read the following license agreement carefully

Server Side Public License
VERSION 1, OCTOBER 16, 2018

Copyright © 2018 MongoDB, Inc.

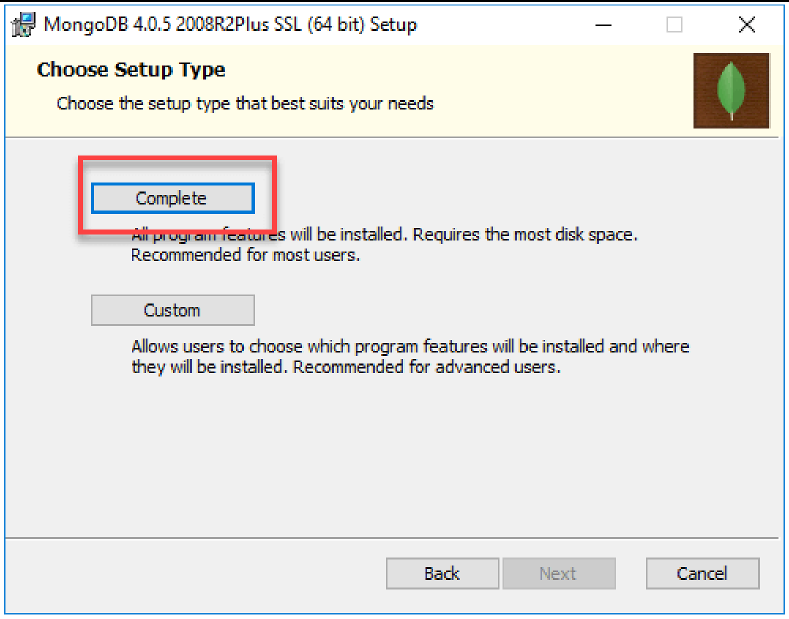
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

TERMS AND CONDITIONS

[✓] I accept the terms in the License Agreement

Step 4) Click on the "complete" button

Click on the "complete" button to install all of the components. The custom option can be used to install selective components or if you want to change the location of the installation.


MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup

Choose Setup Type

Choose the setup type that best suits your needs

Complete

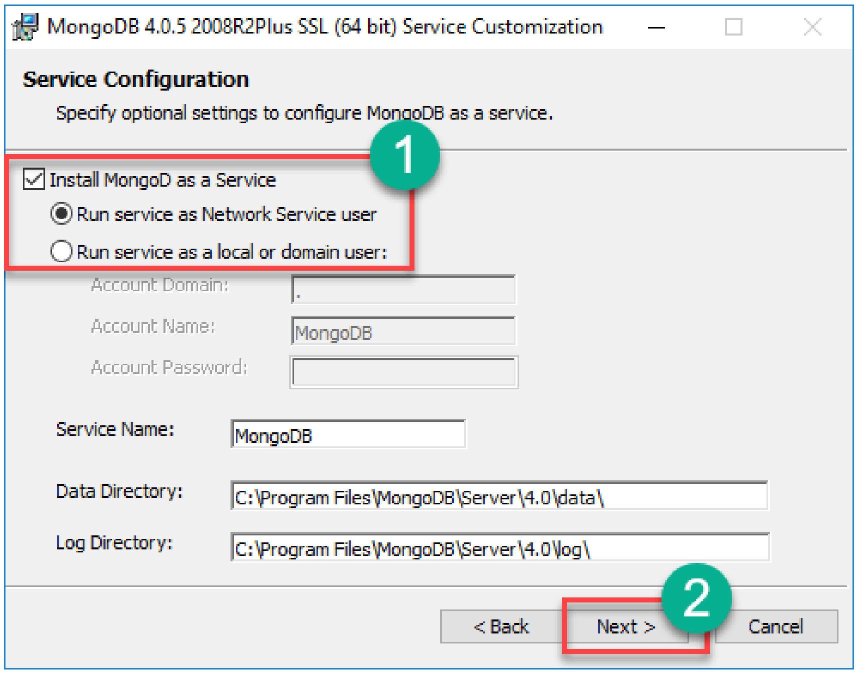
All program features will be installed. Requires the most disk space. Recommended for most users.

Custom

Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

Step 5) Service Configuration

1. Select "Run service as Network Service user". make a note of the data directory, we'll need this later.
2. Click Next


MongoDB 4.0.5 2008R2Plus SSL (64 bit) Service Customization

Service Configuration

Specify optional settings to configure MongoDB as a service.

[✓] Install MongoD as a Service

● Run service as Network Service user

○ Run service as a local or domain user:

Account Domain: .

Account Name: MongoDB

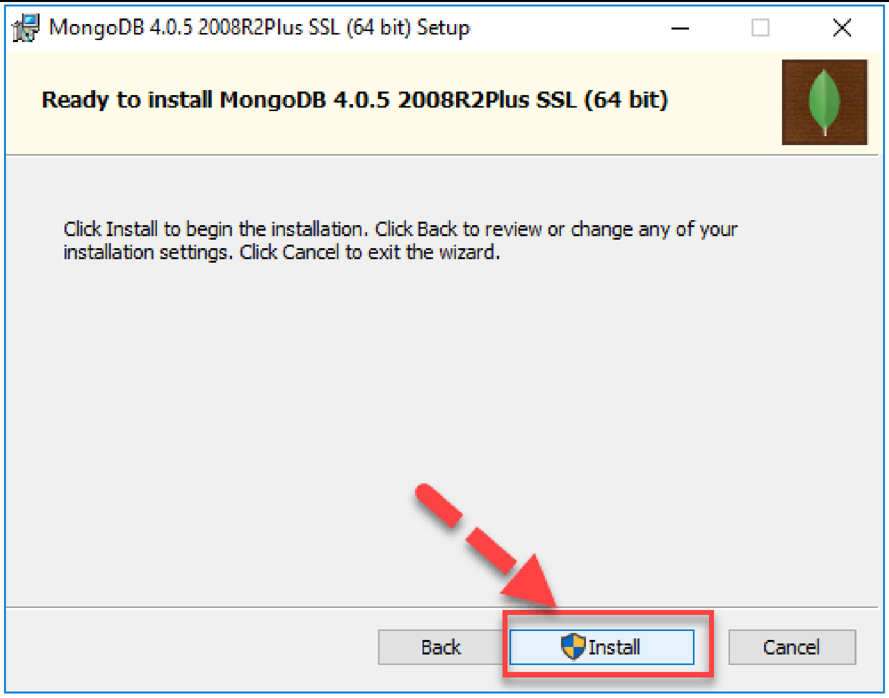
Account Password:

Service Name: MongoDB

Data Directory: C:\Program Files\MongoDB\Server\4.0\data\

Log Directory: C:\Program Files\MongoDB\Server\4.0\log\

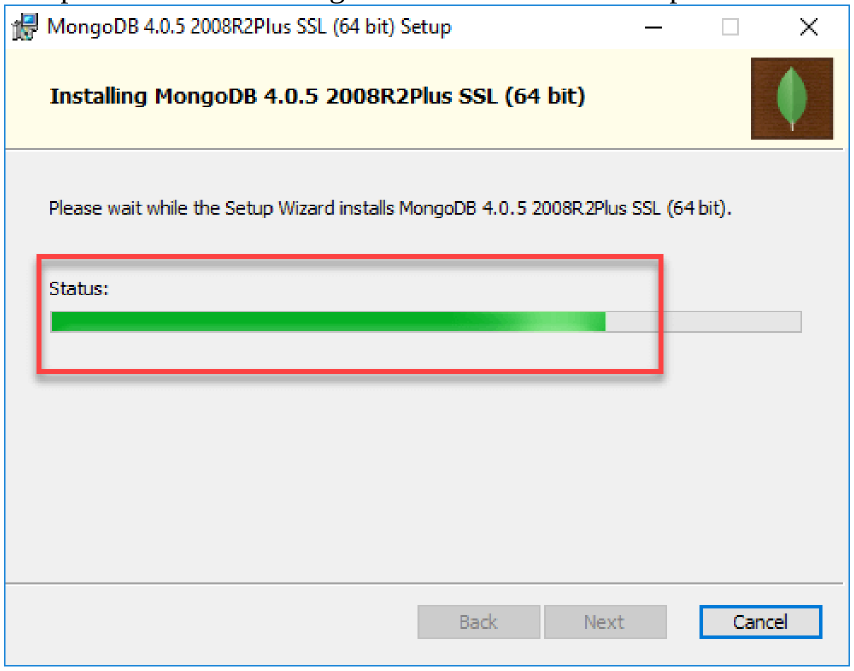
Step 6) Start installation process Click on the Install button to start the installation.


MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup

Ready to install MongoDB 4.0.5 2008R2Plus SSL (64 bit)

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Step 7) Click Next once completed Installation begins. Click Next once completed

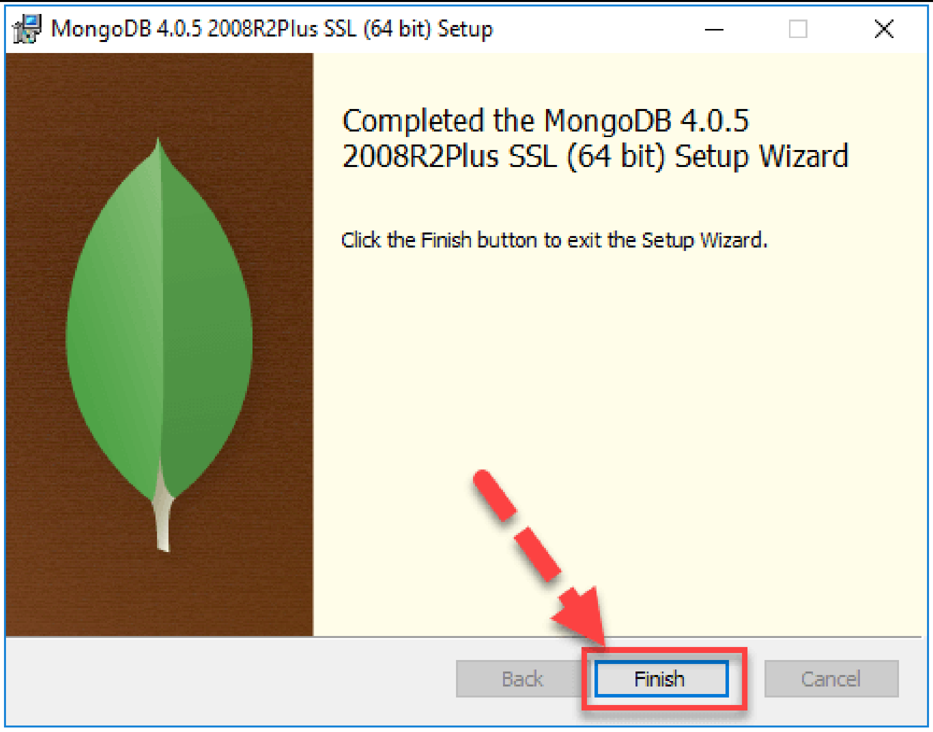

MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup

Installing MongoDB 4.0.5 2008R2Plus SSL (64 bit)

Please wait while the Setup Wizard installs MongoDB 4.0.5 2008R2Plus SSL (64 bit).

Status:

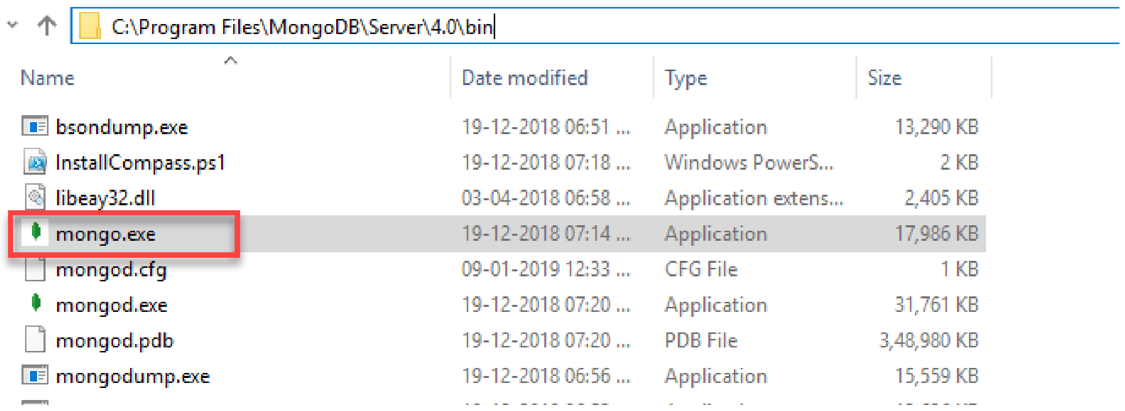
Step 8) Click on the Finish button
Final step, Once complete the installation, Click on the Finish button


MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup

Completed the MongoDB 4.0.5 2008R2Plus SSL (64 bit) Setup Wizard

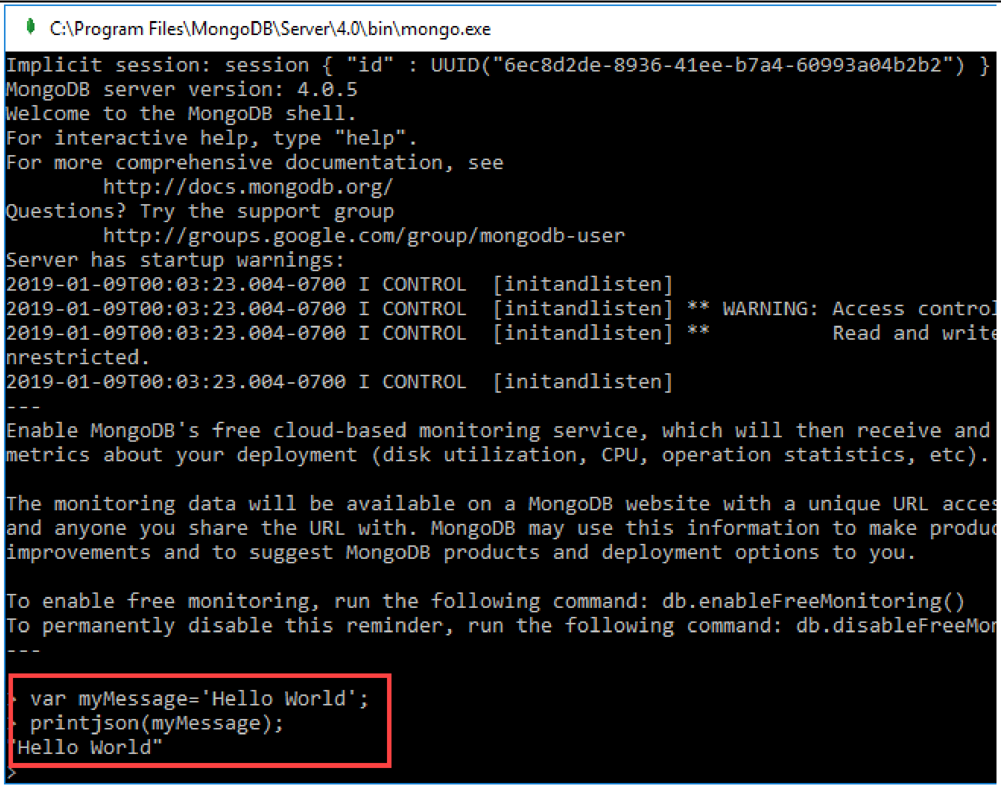
Click the Finish button to exit the Setup Wizard.

Hello World MongoDB: JavaScript Driver
Drivers in MongoDB are used for connectivity between client applications and the database. For example, if you had Java program and required it to connect to MongoDB then you would require to download and integrate the Java driver so that the program can work with the MongoDB database.
The driver for JavaScript comes out of the box. The MongoDB shell which is used to work with MongoDB database is actually a javascript shell. To access it
Step 1) Go to " C:\Program Files\MongoDB\Server\4.0\bin" and double click on mongo.exe. Alternatively, you can also click on the MongoDB desktop item


C:\Program Files\MongoDB\Server\4.0\bin

Name	Date modified	Type	Size
bsondump.exe	19-12-2018 06:51 ...	Application	13,290 KB
InstallCompass.ps1	19-12-2018 07:18 ...	Windows PowerS...	2 KB
libeay32.dll	03-04-2018 06:58 ...	Application extens...	2,405 KB
mongo.exe	19-12-2018 07:14 ...	Application	17,986 KB
mongod.cfg	09-01-2019 12:33 ...	CFG File	1 KB
mongod.exe	19-12-2018 07:20 ...	Application	31,761 KB
mongod.pdb	19-12-2018 07:20 ...	PDB File	3,48,980 KB
mongodump.exe	19-12-2018 06:56 ...	Application	15,559 KB

Step 2) Enter following program into shell
var myMessage='Hello World';
printjson(myMessage);



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
Implicit session: session { "id" : UUID("6ec8d2de-8936-41ee-b7a4-60993a04b2b2") }
MongoDB server version: 4.0.5
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
Server has startup warnings:
2019-01-09T00:03:23.004-0700 I CONTROL  [initandlisten]
2019-01-09T00:03:23.004-0700 I CONTROL  [initandlisten] ** WARNING: Access control
2019-01-09T00:03:23.004-0700 I CONTROL  [initandlisten] **          Read and write
nrestricted.
2019-01-09T00:03:23.004-0700 I CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL acces
and anyone you share the URL with. MongoDB may use this information to make produc
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMor
---

> var myMessage='Hello World';
> printjson(myMessage);
"Hello World"
>
```

Code Explanation:

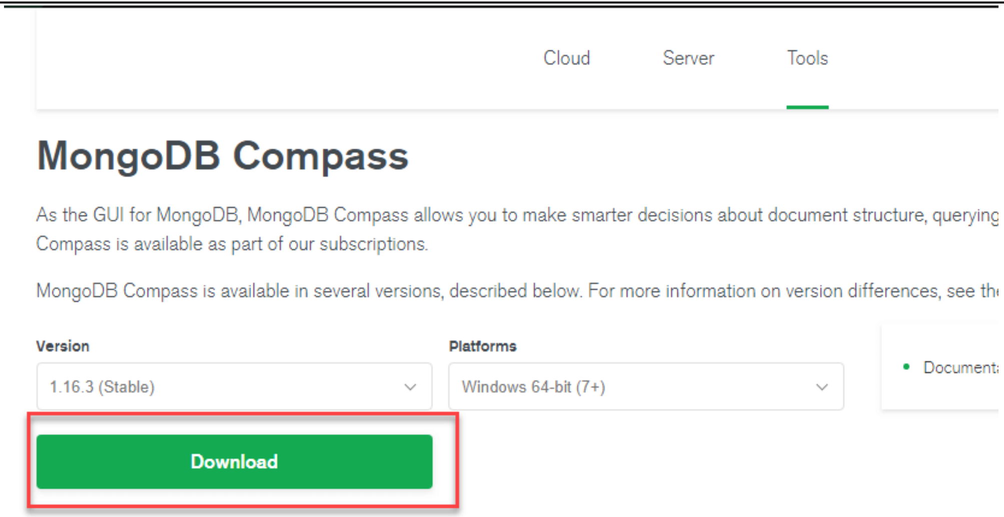
1. We are just declaring a simple Javascript variable to store a string called ‘Hello World.’
2. We are using the printjson method to print the variable to the screen.

Install Python Driver
Step 1) Ensure Python is installed on the system
Step 2) Install the mongo related drivers by issuing the below command pip install pymongo
Install Ruby Driver
Step 1) Ensure Ruby is installed on the system
Step 2) Ensure gems is updated by issuing the commandgem update -system
Step 3) Install the mongo related drivers by issuing the below command gem install mong

Install MongoDB Compass- MongoDB Management Tool
There are tools in the market which are available for managing MongoDB. One such non-commercial tool is MongoDB Compass.
Some of the features of Compass are given below:

1. Full power of the Mongoshell
2. Multiple shells
3. Multiple results

Step 1) Go to link and click download



MongoDB Compass

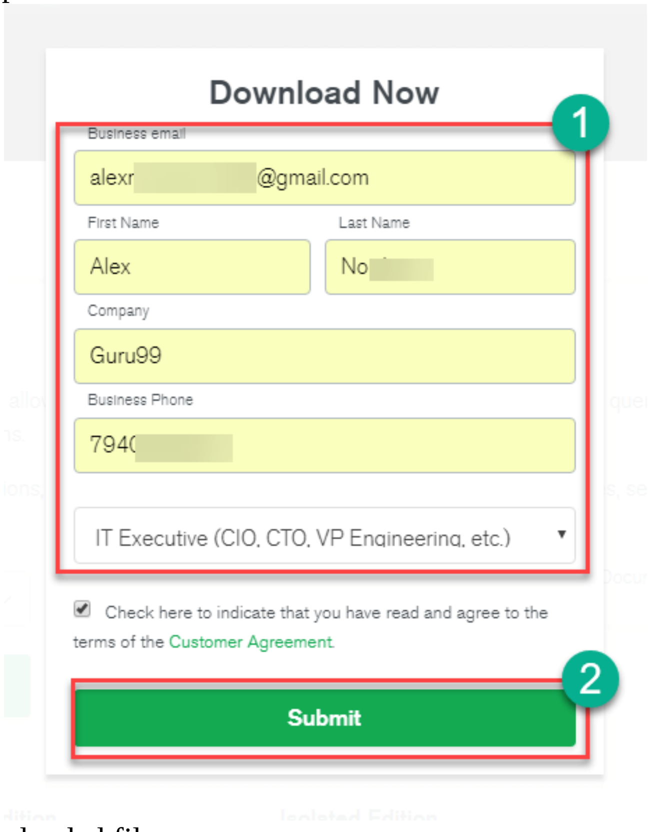
As the GUI for MongoDB, MongoDB Compass allows you to make smarter decisions about document structure, querying
Compass is available as part of our subscriptions.

MongoDB Compass is available in several versions, described below. For more information on version differences, see th

Version: 1.16.3 (Stable)

Platforms: Windows 64-bit (7+)

Step 2) Enter details in the popup and click submit



Download Now

Business email: alexr [redacted] @gmail.com

First Name: Alex

Last Name: No [redacted]

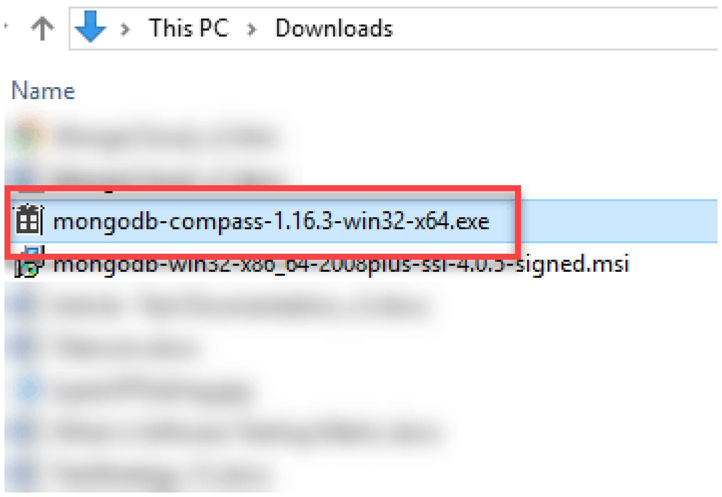
Company: Guru99

Business Phone: 794 [redacted]

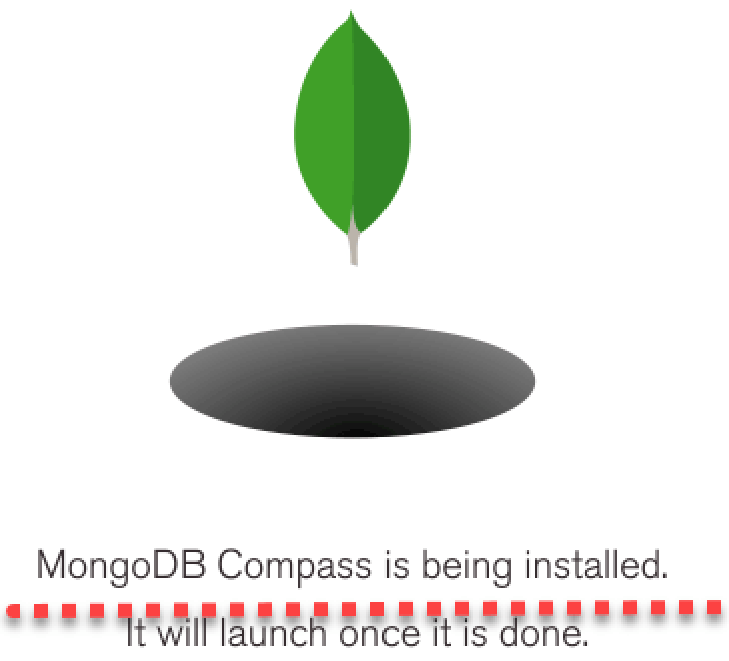
IT Executive (CIO, CTO, VP Engineering, etc.)

☑ Check here to indicate that you have read and agree to the terms of the Customer Agreement.

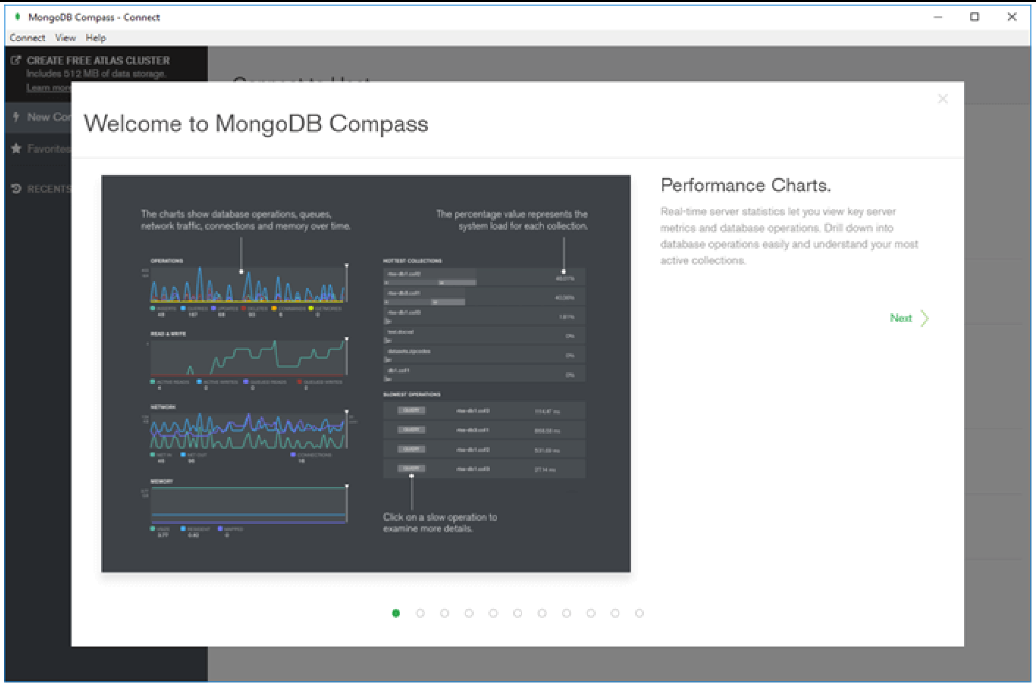
Step 3) Double click on the downloaded file


This PC › Downloads
Name
mongodb-compass-1.16.3-win32-x64.exe
mongodb-win32-x86_64-2008plus-ssl-4.0.5-signed.msi

Step 4) Installation will auto-start


MongoDB Compass is being installed.
It will launch once it is done.

Step 5) Compass will launch with a Welcome screen


MongoDB Compass - Connect
Connect View Help
CREATE FREE ATLAS CLUSTER
Includes 512 MB of data storage.
New Con
Favorites
RECENTS
Next

Welcome to MongoDB Compass

Performance Charts.

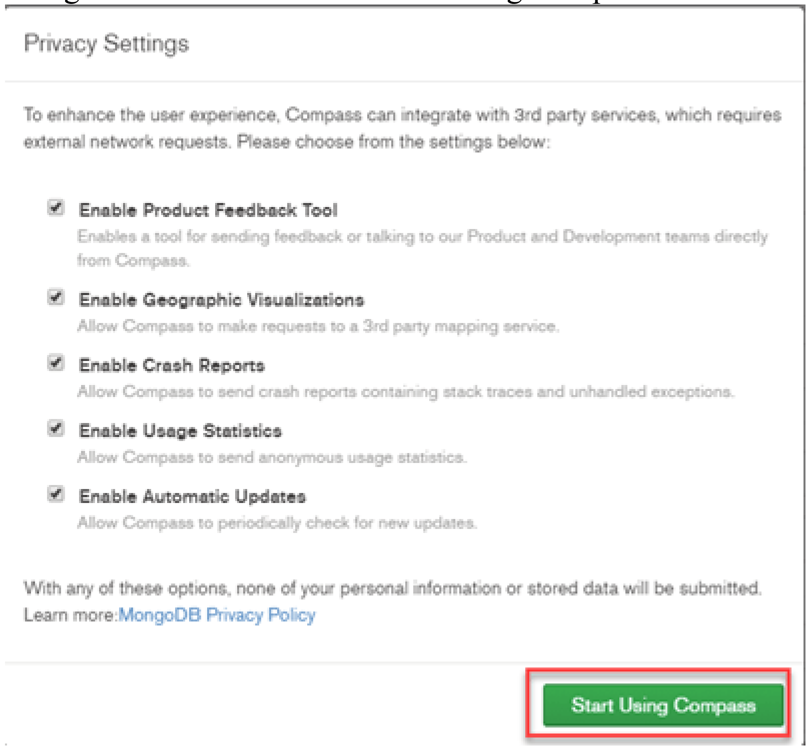
Real-time server statistics let you view key server metrics and database operations. Drill down into database operations easily and understand your most active collections.

The charts show database operations, queues, network traffic, connections and memory over time.

The percentage value represents the system load for each collection.

Click on a slow operation to examine more details.

Step 6) Keep the privacy settings as default and Click "Start Using Compass"



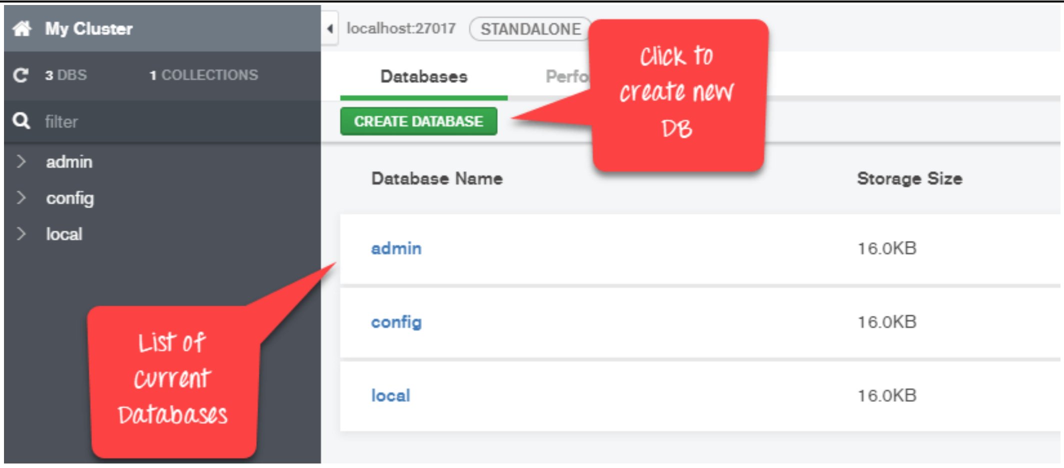
Privacy Settings

To enhance the user experience, Compass can integrate with 3rd party services, which requires external network requests. Please choose from the settings below:

- ✓ **Enable Product Feedback Tool**
  Enables a tool for sending feedback or talking to our Product and Development teams directly from Compass.
- ✓ **Enable Geographic Visualizations**
  Allow Compass to make requests to a 3rd party mapping service.
- ✓ **Enable Crash Reports**
  Allow Compass to send crash reports containing stack traces and unhandled exceptions.
- ✓ **Enable Usage Statistics**
  Allow Compass to send anonymous usage statistics.
- ✓ **Enable Automatic Updates**
  Allow Compass to periodically check for new updates.

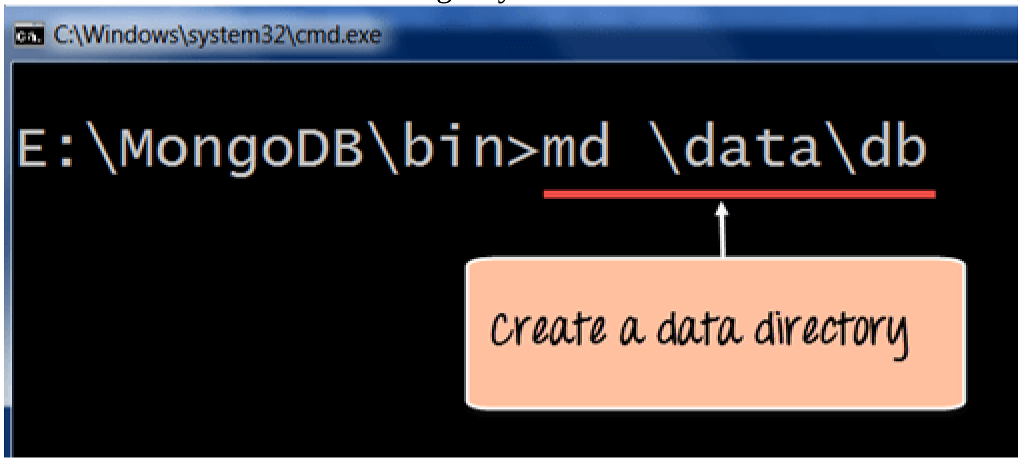
With any of these options, none of your personal information or stored data will be submitted.
Learn more: MongoDB Privacy Policy

Step 7) You will see home screen with list of current databases.



MongoDB Configuration, Import, and Export

Before starting the MongoDB server, the first key aspect is to configure the data directory where all the MongoDB data will be stored. This can be done in the following way



```
E:\MongoDB\bin>md \data\db
```

The above command 'md \data\db' makes a directory called \data\db in your current location.

MongoDB will automatically create the databases in this location, because this is the default location for MongoDB to store its information. We are just ensuring the directory is present, so that MongoDB can find it when it starts.

The import of data into MongoDB is done using the "mongoimport" command. The following example shows how this can be done.

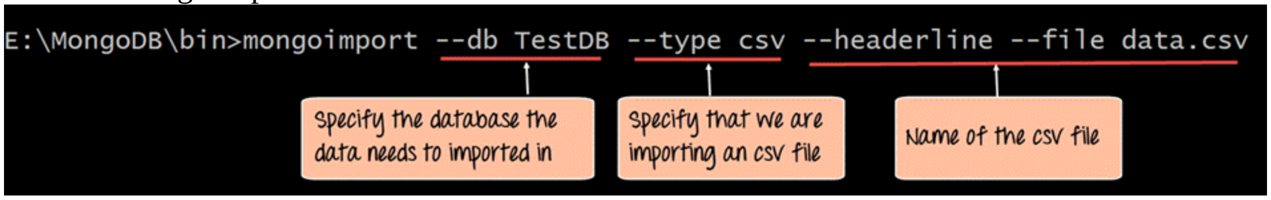
Step 1) Create a CSV file called data.csv and put the following data in it

Employeeid, Employee Name

1. Guru99
2. Mohan
3. Smith

So in the above example, we are assuming we want to import 3 documents into a collection called data. The first row is called the header line which will become the Field names of the collection.

Step 2) Issue the mongo import command



```
E:\MongoDB\bin>mongoimport --db TestDB --type csv --headerline --file data.csv
```

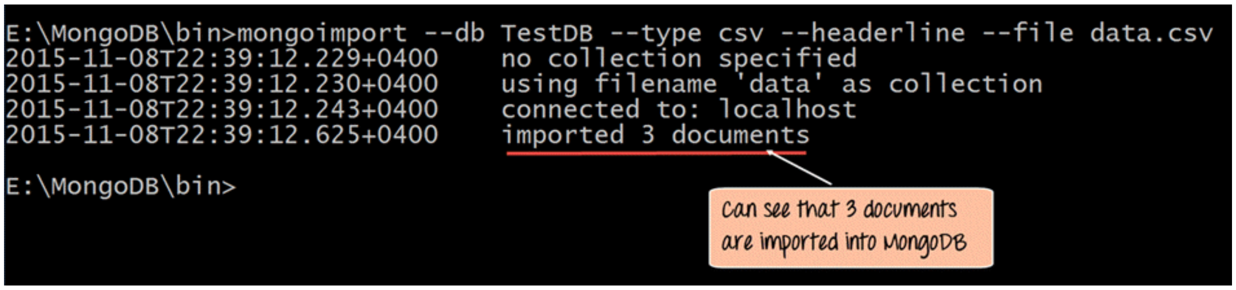
Specify the database the data needs to imported in
Specify that we are importing an csv file
Name of the csv file

Code Explanation:

1. We are specifying the db option to say which database the data should be imported to
2. The type option is to specify that we are importing a csv file

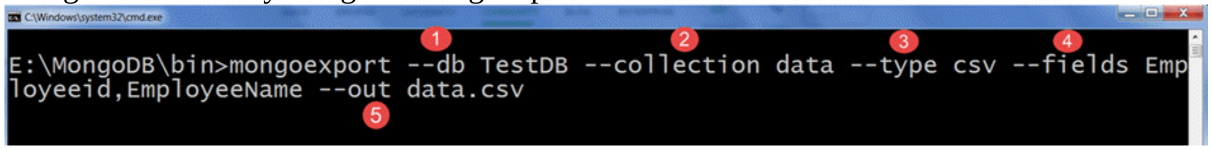
3. Remember that the first row is called the header line which will become the Field names of the collection, that is why we specify the –headerline option. And then we specify our data.csv file.

Output



```
E:\MongoDB\bin>mongoimport --db TestDB --type csv --headerline --file data.csv
2015-11-08T22:39:12.229+0400    no collection specified
2015-11-08T22:39:12.230+0400    using filename 'data' as collection
2015-11-08T22:39:12.243+0400    connected to: localhost
2015-11-08T22:39:12.625+0400    imported 3 documents

E:\MongoDB\bin>
```

The output clearly shows that 3 documents were imported into MongoDB.
Exporting MongoDB is done by using the mongoexport command

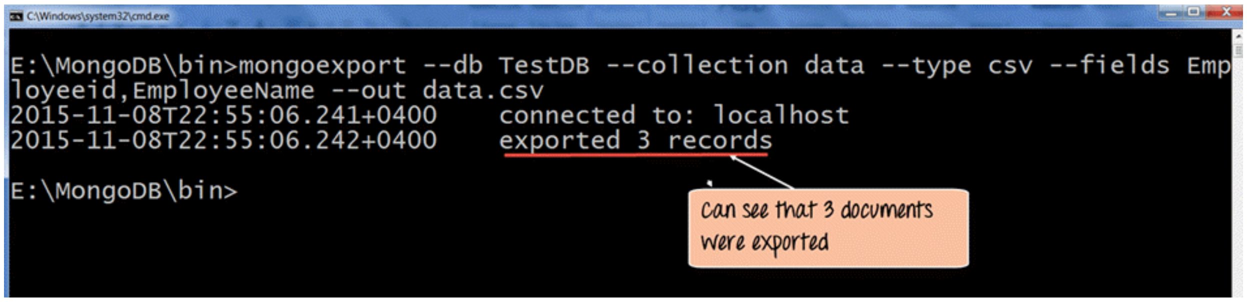


```
E:\MongoDB\bin>mongoexport --db TestDB --collection data --type csv --fields Emp
loyeeid,EmployeeName --out data.csv
```

Code Explanation:

1. We are specifying the db option to say which database the data should be exported from.
2. We are specifying the collection option to say which collection to use
3. The third option is to specify that we want to export to a csv file
4. The fourth is to specify which fields of the collection should be exported.
5. The –out option specifies the name of the csv file to export the data to.

Output



```
E:\MongoDB\bin>mongoexport --db TestDB --collection data --type csv --fields Emp
loyeeid,EmployeeName --out data.csv
2015-11-08T22:55:06.241+0400    connected to: localhost
2015-11-08T22:55:06.242+0400    exported 3 records

E:\MongoDB\bin>
```

C:\Windows\system32\cmd.exe
Can see that 3 documents were exported

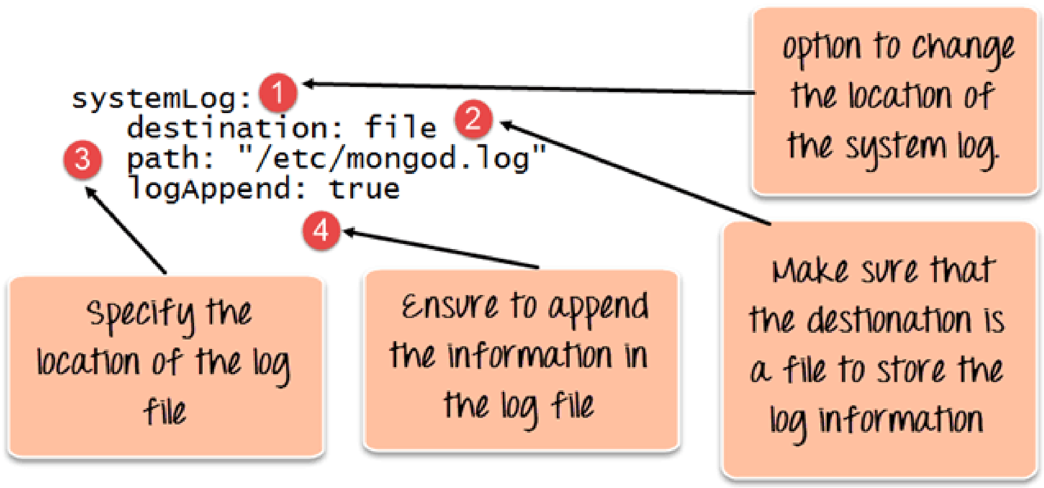
The output clearly shows that 3 records were exported from MongoDB.

Configuring MongoDB server with configuration file

One can configure the mongod server instance to startup with a configuration file. The configuration file contains settings that are equivalent to the mongod command-line options.

For example, supposed you wanted MongoDB to store all its logging information to a custom location then follow the below steps

Step 1) Create a file called, "mongod.conf" and store the below information in the file

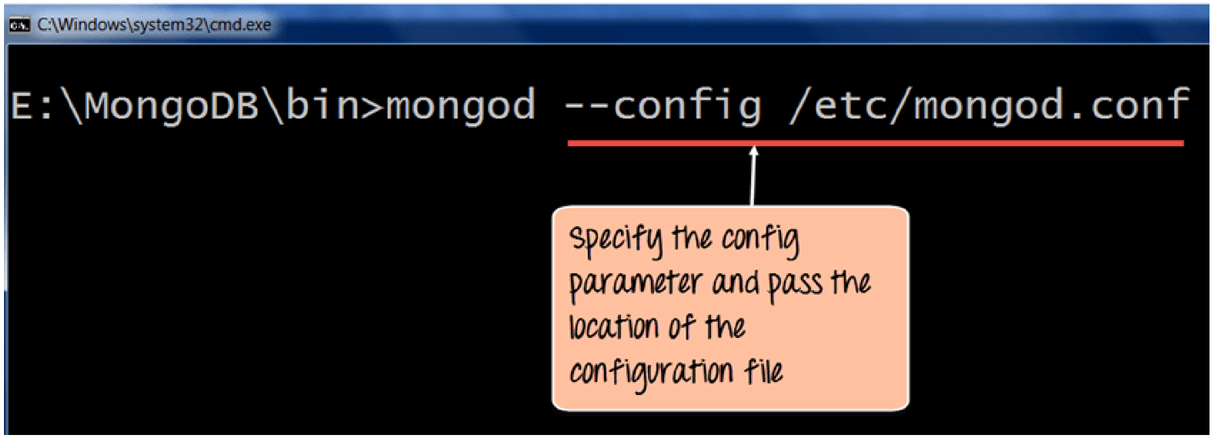


```
systemLog:
   destination: file
   path: "/etc/mongod.log"
   logAppend: true
```

1
2
3
4
option to change the location of the system log.
Make sure that the destionation is a file to store the log information
Specify the location of the log file
Ensure to append the information in the log file

1. The first line of the file specifies that we want to add configuration for the system log file, that is where the information about what the server is doing in a custom log file.
2. The second option is to mention that the location will be a file.
3. This mentions the location of the log file
4. The logAppend: "true" means to ensure that the log information keeps on getting added to the log file. If you put the value as "false", then the file would be deleted and created fresh whenever the server starts again.

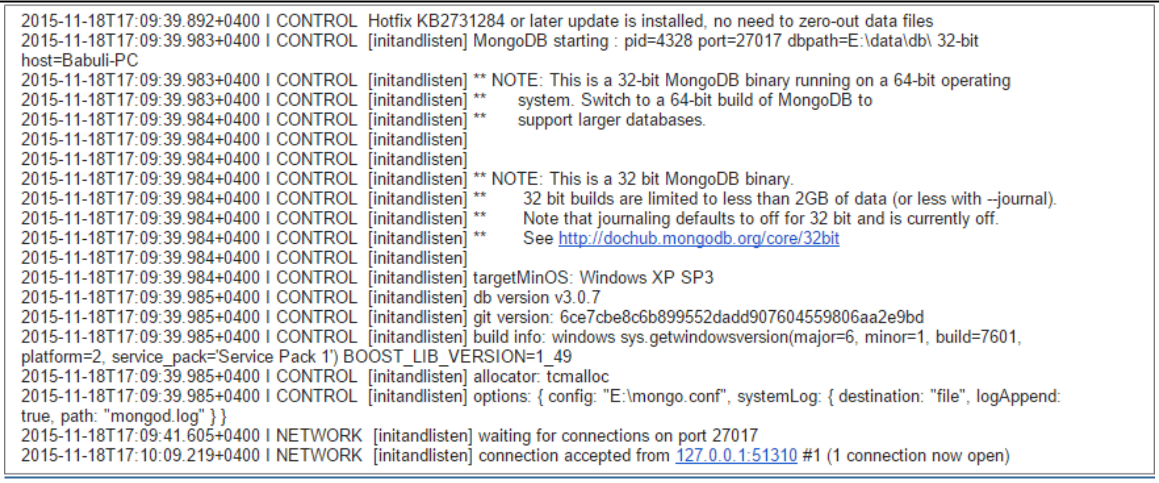
Step 2) Start the mongod server process and specify the above created configuration file as a parameter. The screenshot of how this can be done is shown below



```
E:\MongoDB\bin>mongod --config /etc/mongod.conf
```

C:\Windows\system32\cmd.exe
Specify the config parameter and pass the location of the configuration file

Once the above command is executed, the server process will start using this configuration file, and if you go to the /etc. directory on your system, you will see the mongod.log file created.
The below snapshot shows an example of what a log file would look like.


```
2015-11-18T17:09:39.892+0400 I CONTROL  Hotfix KB2731284 or later update is installed, no need to zero-out data files
2015-11-18T17:09:39.983+0400 I CONTROL  [initandlisten] MongoDB starting : pid=4328 port=27017 dbpath=E:\data\db\ 32-bit
host=Babuli-PC
2015-11-18T17:09:39.983+0400 I CONTROL  [initandlisten] ** NOTE: This is a 32-bit MongoDB binary running on a 64-bit operating
2015-11-18T17:09:39.983+0400 I CONTROL  [initandlisten] **       system. Switch to a 64-bit build of MongoDB to
2015-11-18T17:09:39.984+0400 I CONTROL  [initandlisten] **       support larger databases.
2015-11-18T17:09:39.984+0400 I CONTROL  [initandlisten]
2015-11-18T17:09:39.984+0400 I CONTROL  [initandlisten]
2015-11-18T17:09:39.984+0400 I CONTROL  [initandlisten] ** NOTE: This is a 32 bit MongoDB binary.
2015-11-18T17:09:39.984+0400 I CONTROL  [initandlisten] **       32 bit builds are limited to less than 2GB of data (or less with --journal).
2015-11-18T17:09:39.984+0400 I CONTROL  [initandlisten] **       Note that journaling defaults to off for 32 bit and is currently off.
2015-11-18T17:09:39.984+0400 I CONTROL  [initandlisten] **       See http://dochub.mongodb.org/core/32bit
2015-11-18T17:09:39.984+0400 I CONTROL  [initandlisten]
2015-11-18T17:09:39.984+0400 I CONTROL  [initandlisten] targetMinOS: Windows XP SP3
2015-11-18T17:09:39.985+0400 I CONTROL  [initandlisten] db version v3.0.7
2015-11-18T17:09:39.985+0400 I CONTROL  [initandlisten] git version: 6ce7cbe8c6b899552dadd907604559806aa2e9bd
2015-11-18T17:09:39.985+0400 I CONTROL  [initandlisten] build info: windows sys.getwindowsversion(major=6, minor=1, build=7601,
platform=2, service_pack='Service Pack 1') BOOST_LIB_VERSION=1_49
2015-11-18T17:09:39.985+0400 I CONTROL  [initandlisten] allocator: tcmalloc
2015-11-18T17:09:39.985+0400 I CONTROL  [initandlisten] options: { config: "E:\mongo.conf", systemLog: { destination: "file", logAppend:
true, path: "mongod.log" } }
2015-11-18T17:09:41.605+0400 I NETWORK  [initandlisten] waiting for connections on port 27017
2015-11-18T17:10:09.219+0400 I NETWORK  [initandlisten] connection accepted from 127.0.0.1:51310 #1 (1 connection now open)
```

Experiment 2:

Perform / execute below sets of basic commands on MongoDB lab environment.

- Login to Lab
- Show all Databases
- Select database to work with
- Authenticate and Log out from databases
- List down Collections, Users, Roles
- Create Collection

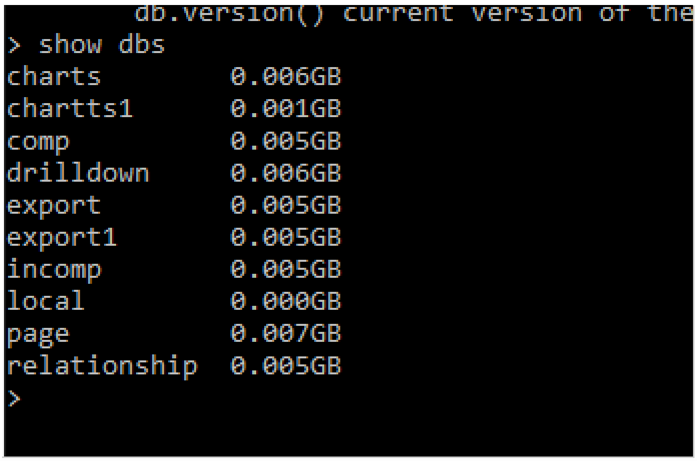
1. Log into MongoDB

The following command can be used to log into the MongoDB database. Make sure that the user with credentials such as *username* and *password* exist in the database mentioned in place of `dbname`.

```
mongo -u <username> -p <password> --authenticationDatabase <dbname>
```

2. Show All Databases

Use below command to get list of all databases.
`show dbs`


```
        db.version() current version of the
> show dbs
charts         0.006GB
chartts1       0.001GB
comp           0.005GB
drilldown      0.006GB
export         0.005GB
export1        0.005GB
incomp         0.005GB
local          0.000GB
page           0.007GB
relationship   0.005GB
>
```

3. Select Database to Work With

To start working with a particular database, the following command can be executed:

```
use databaseName
```

4. **Authenticate and Log Out From Database**

When switching to a different database using the `use dbName` command, the user is required to authenticate using a valid database user for that database. The following command can be used for authentication:

```
//
// Authenticate
//
db.auth("username", "password");
//
// Logout
//
db.logout()
```

5. **List Down Collections, Users, Roles, etc.**

The following commands can be used to check existing collections, users, etc.

```
//
// List down collections of the current database
//
show collections;
db.getCollectionNames();
//
// List down all the users of current database
//
show users;
db.getUsers();
//
```

```
// List down all the roles
//
show roles
```

```
6. Create a Collection
```

The following command can be used to create a collection

```
db.createCollection("collectionName");
```

Experiment 3:

Perform / execute below sets of basic commands on MongoDB lab environment.

- Insert Document
- Save Document
- Update Document
- Display Collection Records
- Drop Function

Insert a Document in a Collection

Once a collection is created, the next step is to insert one or more documents. Following is a sample command for inserting a document in a collection.

```
// Insert single document
db.<collectionName>.insert({field1: "value", field2: "value"})
// Insert multiple documents
db.<collectionName>.insert([{field1: "value1"}, {field1: "value2"}])

db.<collectionName>.insertMany([{field1: "value1"}, {field1: "value2"}])
```

Save or Update Document

The `save` command can be used to either update an existing document or insert a new one depending on the document parameter passed to it. If the `_id` passed matches an existing document, the document is updated. Otherwise, a new document is created. Internally, the `save` method uses either the `insert` or the `update` command.

```
// Matching document will be updated; In case, no document matching the ID is found, a new
document is created

db.<collectionName>.save({"_id": new ObjectId("jhgsdjhgdsf"), field1: "value", field2:
"value"});
```

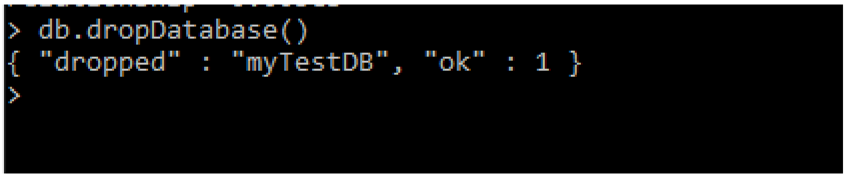
Display Collection Records

The following commands can be used to retrieve collection records:

```
// Retrieve all records
db.<collectionName>.find();

// Retrieve limited number of records; Following command will print 10 results;
db.<collectionName>.find().limit(10);

// Retrieve records by id
db.<collectionName>.find({"_id": ObjectId("someid")});

// Retrieve values of specific collection attributes by passing an object having
// attribute names assigned to 1 or 0 based on whether that attribute value needs
// to be included in the output or not, respectively.

db.<collectionName>.find({"_id": ObjectId("someid")}, {field1: 1, field2: 1});

db.<collectionName>.find({"_id": ObjectId("someid")}, {field1: 0}); // Exclude field1

db.<collectionName>.count();
```

- ## Drop database

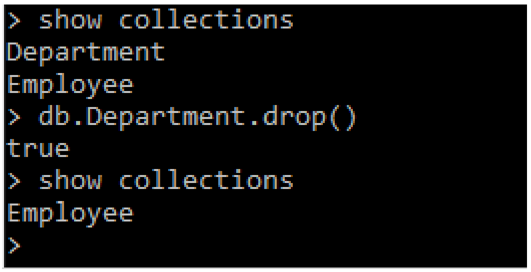
  To drop the database execute following command, this will drop the selected database
  `db.dropDatabase()`



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

- ## Drop collection

  To drop the selected collection execute the following command
  `db.COLLECTION_NAME.drop()`



```
> show collections
Department
Employee
> db.Department.drop()
true
> show collections
Employee
>
```

Experiment 4:

Perform / execute below sets of advanced commands on MongoDB lab environment.

- Administrative Commands
- Projection
- Limit Method
- Skip Method
- Sort Records
- Indexing
- Aggregation
- Interacting with cursors

Administrative Commands

Following are some of the administrative commands that can be helpful in finding collection details such as storage size, total size, and overall statistics.

```
// Get the collection statistics
db.<collectionName>.stats()

db.printCollectionStats()

// Latency statistics for read, writes operations including average time taken for reads,
writes
// and related umber of operations performed
db.<collectionName>.latencyStats()

// Get collection size for data and indexes

db.<collectionName>.dataSize() // Size of the collection

db.<collectionName>.storageSize() // Total size of document stored in the collection

db.<collectionName>.totalSize() // Total size in bytes for both collection data and indexes

db.<collectionName>.totalIndexSize() // Total size of all indexes in the collection
```

MongoDB Projection

In mongodb projection meaning is selecting only necessary data rather than selecting whole of the data of a document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them.

`The find() Method`

MongoDB's find() method, explained in MongoDB Query Document accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB when you execute find() method, then it displays all fields of a document.

To limit this you need to set list of fields with value 1 or 0. 1 is used to show the filed while 0 is used to hide the field.

```
Syntax:
Basic syntax of find() method with projection is as follows
>db.COLLECTION_NAME.find({},{KEY:1})
```

```
Example
Consider the collection myycol has the following data
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}


Following example will display the title of the document while quering the document.
>db.mycol.find({},{"title":1,_id:0})
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
{"title":"Tutorials Point Overview"}
>
```

Please note **_id** field is always displayed while executing **find ()** method, if you don't want this field, then you need to set it as 0.

MongoDB Limit Records

The Limit () Method

To limit the records in MongoDB, you need to use limit () method.
limit () method accepts one number type argument, which is number of documents that you want to displayed.

Syntax:
Basic syntax of **limit()** method is as follows

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

```
Example
Consider the collection myycol has the following data
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

Following example will display only 2 documents while quering the document.
>db.mycol.find({},{"title":1,_id:0}).limit(2)
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
>
```

If you don't specify number argument in limit() method then it will display all documents from the collection.

MongoDB Skip() Method

Apart from limit() method there is one more method skip() which also accepts number type argument and used tob skip number of documents.

Syntax:
Basic syntax of **skip()** method is as follows

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

```
Example:
Following example will only display only second document.
>db.mycol.find({},{"title":1,_id:0}).limit(1).skip(1)
{"title":"NoSQL Overview"}
```

```
>
```
Please note default value in skip() method is 0

MongoDB Sort Documents

The sort() Method

To sort documents in MongoDB, you need to use sort() method. sort() method accepts a document containing list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

```
Syntax:
Basic syntax of sort() method is as follows
>db.COLLECTION_NAME.find().sort({KEY:1})

Example
Consider the collection myycol has the following data
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

Following example will display the documents sorted by title in descending order.
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Tutorials Point Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>

Please note if you don't specify the sorting preference, then sort() method will display
documents in ascending order
```

MongoDB Indexing

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and require the mongod to process a large volume of data. Indexes are special data structures, that store a small portion of the data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in index.

The ensureIndex() Method

To create an index you need to use ensureIndex() method of mongodb.

Syntax:

Basic syntax of **ensureIndex()** method is as follows()

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

Here key is the name of filed on which you want to create index and 1 is for ascending
order. To create index in descending order you need to use -1.

Example
>db.mycol.ensureIndex({"title":1})
>
In ensureIndex() method you can pass multiple fields, to create index on multiple fields.
>db.mycol.ensureIndex({"title":1,"description":-1})
>
```

MongoDB supports the following types of the index for running a query.

1. Single Field Index

MongoDB supports user-defined indexes like single field index. A single field index is used to create an index on the single field of a document. With single field index, MongoDB can traverse in ascending and descending order. By default, each collection has a single field index automatically created on the `_id` field, the primary key.

Example

```
{
  "_id": 1,
  "person": { name: "Alex", surname: "K" },
  "age": 29,
  "city": "New York"
}
```

We can define, a single field index on the age field.

```
db.people.createIndex( {age : 1} ) // creates an ascending index

db.people.createIndex( {age : -1} ) // creates a descending index
```

With this kind of index we can improve all the queries that find documents with a condition and the age field, like the following:

```
db.people.find( { age : 20 } )
db.people.find( { name : "Alex", age : 30 } )
db.people.find( { age : { $gt : 25} } )
```

2. Compound Index

A compound index is an index on multiple fields. Using the same people collection we can create a compound index combining the city and age field.

```
db.people.createIndex( {city: 1, age: 1, person.surname: 1  } )
```

In this case, we have created a compound index where the first entry is the value of the city field, the second is the value of the age field, and the third is the person.name. All the fields here are defined in ascending order.

Queries such as the following can benefit from the index:

```
db.people.find( { city: "Miami", age: { $gt: 50 } } )
db.people.find( { city: "Boston" } )
db.people.find( { city: "Atlanta", age: {$lt: 25}, "person.surname": "Green" } )
```

3. Multikey Index

This is the index type for arrays. When creating an index on an array, MongoDB will create an index entry for every element.

Example

```
{
   "_id": 1,
   "person": { name: "John", surname: "Brown" },
   "age": 34,
   "city": "New York",
   "hobbies": [ "music", "gardening", "skiing" ]
 }
```

The multikey index can be created as:

```
db.people.createIndex( { hobbies: 1} )
```

Queries such as these next examples will use the index:

```
db.people.find( { hobbies: "music" } )
db.people.find( { hobbies: "music", hobbies: "gardening" } )
```

4. Geospatial Index

GeoIndexes are a special index type that allows a search based on location, distance from a point and many other different features. To query geospatial data, MongoDB supports two types of indexes – `2d indexes` and `2d sphere indexes`. 2d indexes use planar geometry when returning results and 2dsphere indexes use spherical geometry to return results.

5. Text Index

It is another type of index that is supported by MongoDB. Text index supports searching for string content in a collection. These index types do not store language-specific stop words (e.g. "the", "a", "or"). Text indexes restrict the words in a collection to only store root words.

Example

Let's insert some sample documents.

```
var entries = db.people("blogs").entries;
entries.insert( {
  title : "my blog post",
  text : "i am writing a blog. yay",
  site: "home",
  language: "english" });
entries.insert( {
  title : "my 2nd post",
  text : "this is a new blog i am typing. yay",
  site: "work",
  language: "english" });
entries.insert( {
  title : "knives are Fun",
  text : "this is a new blog i am writing. yay",
  site: "home",
  language: "english" });
```

Let's define create the text index.

```
var entries = db.people("blogs").entries;
```

```
entries.ensureIndex({title: "text", text: "text"}, { weights: {
    title: 10,
    text: 5
  },
  name: "TextIndex",
  default_language: "english",
  language_override: "language" });
```

Queries such as these next examples will use the index:

```
var entries = db.people("blogs").entries;
entries.find({$text: {$search: "blog"}, site: "home"})
```

6. Hashed Index

MongoDB supports hash-based sharding and provides hashed indexes. These indexes are the hashes of the field value. Shards use hashed indexes and create a hash according to the field value to spread the writes across the sharded instances.

MongoDB Aggregation

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In sql count(*) and with group by is an equivalent of mongodb aggregation.

The aggregate() Method

For the aggregation in mongodb you should use **aggregate()** method.

Syntax:

Basic syntax of **aggregate()** method is as follows

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Example:

In the collection you have the following data:

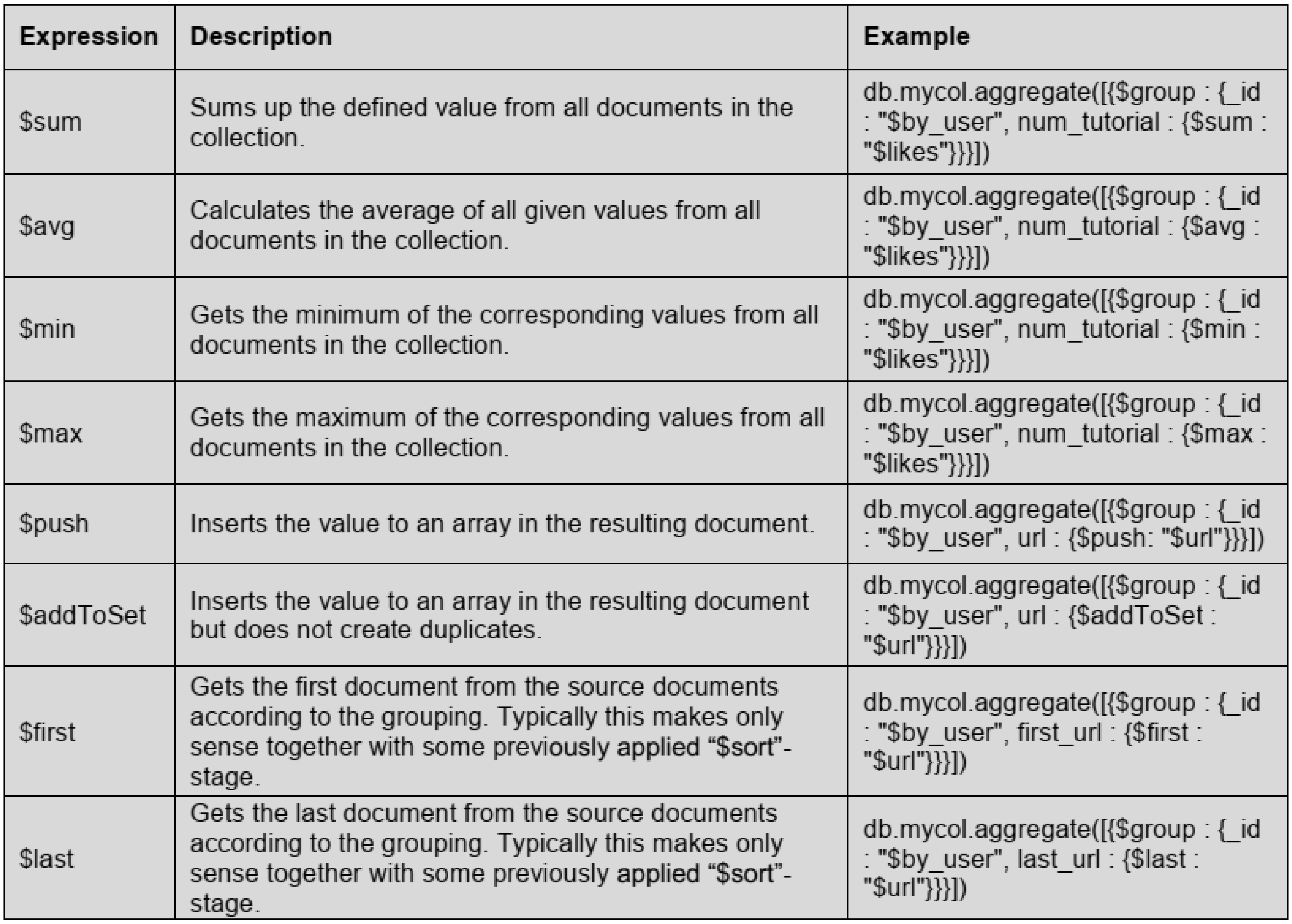
```
{
_id: ObjectId(7df78ad8902c)
title: 'MongoDB Overview',
description: 'MongoDB is no sql database',
by_user: 'tutorials point',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
},
{
_id: ObjectId(7df78ad8902d)
title: 'NoSQL Overview',
description: 'No sql database is very fast',
by_user: 'tutorials point',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 10
},
{
_id: ObjectId(7df78ad8902e)
title: 'Neo4j Overview',
description: 'Neo4j is no sql database',
by_user: 'Neo4j',
url: 'http://www.neo4j.com',
tags: ['neo4j', 'database', 'NoSQL'],
likes: 750
}
```

Now from the above collection if you want to display a list that how many tutorials are written by each user then you will use **aggregate()** method as shown below:

```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
```

```
{
"result" : [
{
"_id" : "tutorials point",
"num_tutorial" : 2
},
{
"_id" : "tutorials point",
"num_tutorial" : 1
}
],
"ok" : 1
}
```

Sql equivalent query for the above use case will be **select by_user, count(*) from mycol group by by_user**
In the above example we have grouped documents by field **by_user** and on each occurance of by_user previous value of sum is incremented. There is a list available aggregation expressions.



Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

Pipeline Concept

In UNIX command shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on. MongoDB also support same concept in aggregation framework. There is a set of possible stages and each of those is taken a set of documents as an input and is producing a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn again be used for the next stage an so on.

Possible stages in aggregation framework are following:

- **$project:** Used to select some specific fields from a collection.
- **$match:** This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.
- **$group:** This does the actual aggregation as discussed above.
- **$sort:** Sorts the documents.
- **$skip:** With this it is possible to skip forward in the list of documents for a given amount of documents.
- **$limit:** This limits the amount of documents to look at by the given number starting from the current position.s
- **$unwind:** This is used to unwind document that are using arrays. when using an array the data is kind of prejoinded and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

Interacting with cursors

When we compose a query, Mongo gives us back a cursor object from which we can get the values.

When we called limit and sort in the last section, we were actually calling methods on the cursor that was returned by find.

If we save the cursor in a variable we can call more methods on it.

```
var people = db.people.find( );
```

We can iterate over the the cursor using a simple while loop. We can check if the cursor has a next value, and can call cursor.next to get the value out.

```
var people = db.people.find();
while (people.hasNext()) {
print(tojson(people.next()));
}
```

Functional Loops - forEach and map

We can simplify the code above using functional style loops.

```
db.people.find().forEach(function(person) {
print(person.name);
});
```

We also have access to map, which will return an array of values.

```
var array = db.people.find().map(function(person) {
return person.name;
});
```

Cursor persistence

Cursors last for 10 minutes and are then garbage collected. This should be sufficient for most tasks. If you need a longer lasting cursor for some reason, you can create a long lasting cursor, however you should be aware that it will eventually go out of sync with the database.

If you want to know how to prevent this behaviour, see here:
http://docs.mongodb.org/manual/core/cursors/#closure-of-inactive-cursors

Exercise - Cursor methods

You can read all the cursor methods here:
http://docs.mongodb.org/manual/reference/method/js-cursor/

- Iterate over each of the people and output them.
- change the find function to find only the people with cats.
- Iterate over each of the people, outputting just the cat name and age each time.
- Use Map to generate an array containing all of the cat names.

Experiment 5

Execute below steps by inserting some data which we can work with.

Paste the following into your terminal to create a petshop with some pets in it

```
use petshop
db.pets.insert({name: "Mikey", species: "Gerbil"})
db.pets.insert({name: "Davey Bungooligan", species: "Piranha"})
db.pets.insert({name: "Suzy B", species: "Cat"})
db.pets.insert({name: "Mikey", species: "Hotdog"})
db.pets.insert({name: "Terrence", species: "Sausagedog"})
db.pets.insert({name: "Philomena Jones", species: "Cat"})
```

- Add another piranha, and a naked mole rat called Henry.

```
db.pets.insert({name:"Nemo"},{species:"Piranha"})
db.pets.insert({name: "Henry", species: "naked mole rat"})
```

Output

```
WriteResult({ "nInserted" : 1 })
WriteResult({ "nInserted" : 1 })
```

- Use find to list all the pets. Find the ID of Mikey the Gerbil.

```
db.pets.find({name:"Mikey",species:"Gerbil"})
```

Output

```
{"_id":"5caf07330e78250010b2fcff","name":"Mikey","species":"Gerbil"}
```

- Use find to find Mikey by id.

```
db.pets.find(ObjectId("5caf07330e78250010b2fcff"))
```

Output

```
{
"_id" : ObjectId("614447ed0c1b5b4fc0961776"),
"name" : "Mikey",
"species" : "Gerbil"
}
```

- Use find to find all the gerbils.

```
db.pets.find({species:"Gerbil"})
```

Output

```
{
"_id" : ObjectId("614447ed0c1b5b4fc0961776"),
"name" : "Mikey",
"species" : "Gerbil"
}
```

- Find all the creatures named Mikey.

```
db.pets.find({name:"Mikey"})
```

Output

```
     {
  "_id" : ObjectId("614447ed0c1b5b4fc0961776"),
  "name" : "Mikey",
  "species" : "Gerbil"
     }
     {
  "_id" : ObjectId("614448100c1b5b4fc0961778"),
  "name" : "Mikey",
  "species" : "Hotdog"
     }
```

- Find all the creatures named Mikey who are gerbils.
  `db.pets.find({name:"Mikey",species:"Gerbil"})`

  Output

```
  {
"_id" : ObjectId("614447ed0c1b5b4fc0961776"),
"name" : "Mikey",
"species" : "Gerbil"
  }
```

- Find all the creatures with the string "dog" in their species.
  `db.pets.find({ species: /dog/ })`

  Output

```
  {
"_id" : ObjectId("614448100c1b5b4fc0961778"),
"name" : "Mikey",
"species" : "Hotdog"
  }
  {
"_id" : ObjectId("6144481d0c1b5b4fc0961779"),
"name" : "Terrence",
"species" : "Sausagedog"
  }
  {
"_id" : ObjectId("614448310c1b5b4fc096177a"),
"name" : "Terrence",
"species" : "Sausagedog"
  }
```

Experiment 6:

AirPhone Corp is a famous telecom company. They have customers in all locations. Customers use AirPhone Corp's network to make calls. Government has brought in a regulation that all telecom companies should store call details of their customers. This is very important from a security point of view and all telecom companies have to retain this data for 15 years. AirPhone Corp already stores all customer details data, for their analytics team. But due to a surge in mobile users in recent years, their current database cannot handle huge amounts of data. Current database stores only six months of data. AirPhone Corp now wants to scale their database and wants to store 15 years of data.

Data contains following columns:
Source : Phone number of caller
Destination : Phone number of call receiver
Source_location : Caller's city
Destination_location : Call receiver's city
Call_duration : phone call duration
Roaming : Flag to check if caller is in roaming
Call_charge : Money charged for call

Sample Data:

```
{
source: "+919612345670",
destination: "+919612345671",
source_location: "Delhi",
destination_location: "Mumbai",
call_duration: 2.03,
roaming: false,
call_charge: 2.03
}
```

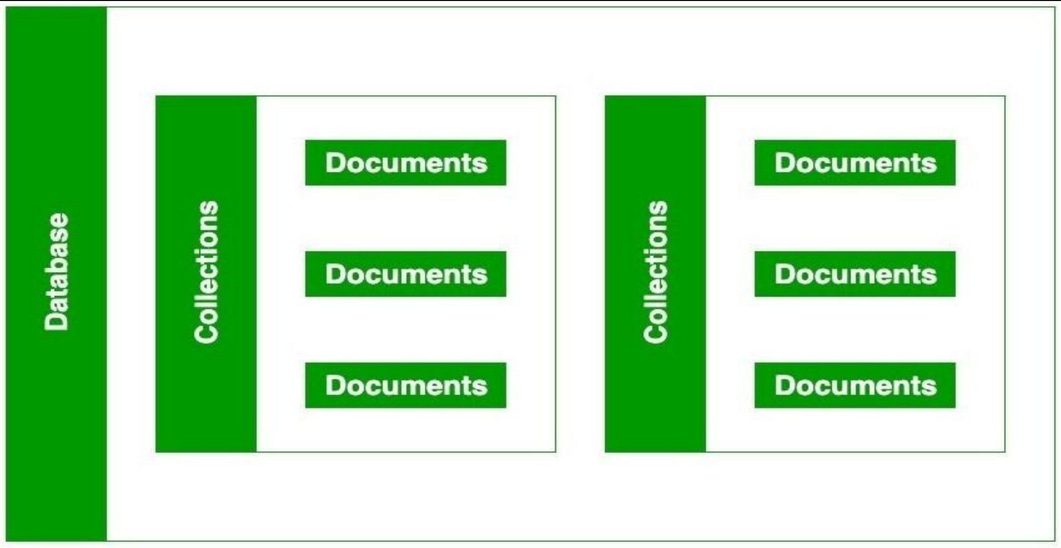
After discussing the requirements with database and architecture team, it has been decided that they should use MongoDb. You have been given the task to Setup a distributed system (database) such that data from different locations go to different nodes (to distribute the load)

- Import data to sharded collection
- Check data on each shard for distribution

Introduction to MongoDB Sharding

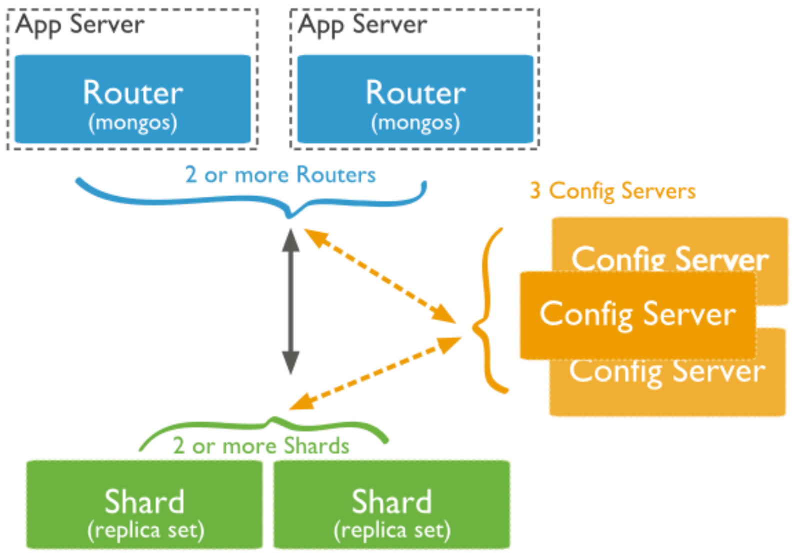
The main purpose of using a NoSQL Database for most organizations is the ability to deal with the storage and compute demands of storing and querying high volumes of data. MongoDB Sharding can be seen as the way in which MongoDB deals with high volumes of data. It can be seen as the process in which large datasets are split into smaller datasets that are stored across multiple MongoDB Instances. This is done because querying on large datasets could lead to high CPU utilization on the MongoDB Server.

The following image shows the structure of a MongoDB Database:



Each MongoDB Database consists of a large number of Collections. Each Collection is made up of a large number of Documents that store data as Key-Value pairs. MongoDB Sharding breaks up a large Collection into smaller Collections called Shards. Splitting up large Collections into Shards allows MongoDB to execute queries without putting much load on the Server.

MongoDB Sharding can be implemented by creating a Cluster of MongoDB Instances. The following image shows how MongoDB Sharding works in a Cluster.



The three main components of Sharded Cluster are as follows:

- Shard
- Config Servers
- Query Routers

1) Shard

Shard is the most basic unit of a Shared Cluster that is used to store a subset of the large dataset that has to be divided. Shards are designed in such a way that they are capable of providing high data availability and consistency.

2) Config Servers

Config Servers are supposed to store the metadata of the MongoDB Sharded Cluster. This metadata consists of information about what subset of data is stored in which Shard. This information can be used to direct user queries accordingly. Each Sharded Cluster is supposed to have exactly 3 Config Servers.

3) Query Routers

Query Routers can be seen as Mongo Instances that form an interface to the client applications. The Query Routers are responsible for forwarding user queries to the right Shard.

Benefits of MongoDB Sharding

MongoDB Sharding is important because of the following reasons:

- In a setup in which MongoDB Sharding has not been implemented, the Master nodes handle the potentially large number of write operations whereas the Slave Nodes are responsible for read operations and maintaining backups. Since MongoDB Sharding utilizes Replica Sets, queries are distributed equally among all nodes.
- The storage capacity of the Sharded Cluster can be increased without performing any complex hardware restructuring by simply adding additional Shards to the Cluster.
- If one or more Shards in the Cluster go down, other Shards will continue to operate which means that the data stored in those active Shards can be accessed without any issues.

Steps to Set up MongoDB Sharding

MongoDB Sharding can be set up by implementing the following steps:

- Step 1: Creating a Directory for Config Server
- Step 2: Starting MongoDB Instance in Configuration Mode
- Step 3: Starting Mongos Instance
- Step 4: Connecting to Mongos Instance
- Step 5: Adding Servers to Clusters
- Step 6: Enabling Sharding for Database

Step 1: Creating a Directory for Config Server

The first step to be performed in order to set up MongoDB Sharding would be to create a separate directory for Config Server. This can be done using the following command:

```
mkdir /data/configdb
```

Step 2: Starting MongoDB Instance in Configuration Mode

One Server has to be set up as the Configuration Server. Suppose you have a Server named “ConfServer” which would be used as the Configuration Server, the following command can be executed to perform that operation:

```
mongod –configdb ConfServer: 27019
```

Step 3: Starting Mongos Instance

Once the Configuration Server has been set up, the Mongos Instance can be started by executing the following command along with the name of your Configuration Server:

```
mongos –configdb ConfServer: 27019
```

Step 4: Connecting to Mongos Instance

A connection can be formed to the Mongos Instance by running the following command from the Mongo Shell:

```
mongo –host ConfServer –port 27017
```

Step 5: Adding Servers to Clusters

All Servers that have to be included in the Cluster can be added by the following command:

```
sh.addShard("SA:27017")
```

“SA” here has to be replaced with the name of your Server that has to be added to the Cluster. This command can be executed for all Servers that have to be added to the Cluster.

Step 6: Enabling Sharding for Database

Once the Sharded Cluster has been set up, Sharding for the required database has to be enabled. This can be done by the following command:

```
sh.enableSharding(db_test)
```

In the above command, “db_test” has to be replaced with the name of the database that you wish to Shard.

Limitations of MongoDB Sharding

The limitations of MongoDB Sharding are as follows:

- Setting up MongoDB Sharding is a complex operation and hence, careful planning and high maintenance is required.
- There are certain MongoDB operations that cannot be executed in a Sharded Cluster. For example, geoSpace command.
- Once a Collection in MongoDB has been sharded, there is no way to un-shard it and restore the Collection in the original format.

Experiment 7:

Execute below sets of problem by taking reference of Experiment Number 06 and find out:

- Add additional node to existing system (to test if we can add nodes easily when data increases)
- Check the behavior of cluster (data movement) on adding a share.
- Check the behavior of query for finding a document with source location Mumbai.

Experiment 8:

Anand Corp is a leading corporate training provider. A lot of prestigious organizations send their employees to Anand Corp for training on different skills. As a distinct training provider, Anand Corp has decided to share analysis report with their clients. This report will help their clients know the employees who have completed training and evaluation exam, what are their strengths, and what are the areas where employees need improvement. This is going to be a unique selling feature for the Anand Corp. As Anand Corp is already doing great business and they give training to a large number of people every month, they have huge amount of data to deal with. They have hired you as an expert and want your help to solve this problem.

Attributes of data:

Id: id of the person who was trained

Name: name of the person who was trained

Evaluation: evaluation term

Score: score achieved by the person for the specific term

A person can undergo multiple evaluations. Each evaluation will have a unique result score.

You can see the sample data below.

Sample Data

```
{
"_id":0,
"name":"Andy",
"results": [
{"evaluation":"term1","score":1.463179736705023},
{"evaluation":"term2","score":11.78273309957772},
{"evaluation":"term3","score":6.676176060654615}
]
}
```

PQR Corp has assigned the following tasks to you to analyze the results:

Create Sample data:

Create Database - "employee"

Collection - "training"

```
{"_id":0,"name":"Pavan","results":[{"evaluation":"term1","score":1.463179736705023},{"evaluation":"term2","score":11.78273309957772},{"evaluation":"term3","score":6.676176060654615}]}
{"_id":1,"name":"Vijay","results":[{"evaluation":"term1","score":60.06045071030959},{"evaluation":"term2","score":52.79790691903873},{"evaluation":"term3","score":71.76133439165544}]}
{"_id":2,"name":"amar","results":[{"evaluation":"term1","score":67.03077096065002},{"evaluation":"term2","score":6.301851677835235},{"evaluation":"term3","score":20.18160621941858}]}
{"_id":3,"name":"Girish","results":[{"evaluation":"term1","score":71.64343899778332},{"evaluation":"term2","score":24.80221293650313},{"evaluation":"term3","score":1.694720653897219}]}
{"_id":4,"name":"Shiv","results":[{"evaluation":"term1","score":78.68385091304332},{"evaluation":"term2","score":90.2963101368042},{"evaluation":"term3","score":34.41620148042529}]}
{"_id":5,"name":"Shobith","results":[{"evaluation":"term1","score":44.87186330181261},{"evaluation":"term2","score":25.72395114668016},{"evaluation":"term3","score":10.53058536508186}]}
{"_id":6,"name":"Bhavana","results":[{"evaluation":"term1","score":37.32285459166097},{"evaluation":"term2","score":28.32634976913737},{"evaluation":"term3","score":16.58341639738951}]}
{"_id":7,"name":"Monica","results":[{"evaluation":"term1","score":90.37826509157176},{"evaluation":"term2","score":42.48780666956811},{"evaluation":"term3","score":67.18328596625217}]}
```

```
package com.mongodb.customsearch;
import static java.util.Arrays.asList;
import java.util.ArrayList;
import java.util.Collection;
import org.bson.Document;
import com.mongodb.Block;
import com.mongodb.MongoClient;
import com.mongodb.client.AggregateIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
public class EmployeeTrainingScores {

 public static void main(String[] args) {
  try{
   // Creating mongo client default connection host:localhost port:12027
   //MongoClient mongoClient = new MongoClient();
   MongoClient mongoClient = new MongoClient("localhost", 27017);
   // fetching a database with name students
   MongoDatabase db = mongoClient.getDatabase("employee");
   // fetching collection scores from database students
   MongoCollection collection = db.getCollection("training");
   //find one record
   findFailedStudInTerm1(collection);
   //failed employees in aggregate (term1+term2+term3)
   failedInAggregate(collection);
   //average score of trainees in term1
   averageScoreTerm1(collection);
   //Average score of class for aggregate (term1 + term2 + term3)
   averageClassScore(collection);

   //count of employee failed in all three terms
   employeeCountFailInAlLTerms(collection);
   //count of employee failed in either of three terms
   employeeCountFailAtleastATerm(collection);
   mongoClient.close();
 }catch(Exception exception){
       System.err.println(exception.getClass().getName() + ":" + exception.getMessage());
    }
 }
```

Find count and percentage of employees who failed in term 1, the passing score being 37.

```
/**
 * function to find a record input : mongodb collection
 * 1. Find count and percentage of employees who failed in term 1, the passing score being 37
 */
 public static Long findFailedStudInTerm1(MongoCollection<Document> collection) {
  Long count = collection.count(new Document("results.evaluation", "term1").append("results.score", new
      Document("$lt",37)));
  Long totalStudents=collection.count();
```

```
        Long per_Stud=(count*100/totalStudents);
        System.out.println("############################### 1. Find count and percentage of employees who
           failed in term 1, the passing score being 37 ###############################");
        System.out.println("Number of students failed in exams in Term1: passing marks 37 ======> " + count);
        System.out.println("Percentage of students failed in exams  in Term1: passing marks 37 ======> " +
           per_Stud +" %");
        return per_Stud;
       }
```

Experiment 9

Execute below sets of problem by taking reference of Experiment Number 08 and find out:

- Find employees who failed in aggregate (term1 + term2 + term3).

```
/**
 * function to find a record input : mongodb collection
 * 2. Find employees who failed in aggregate (term1 + term2 + term3)
 */
 private static void failedInAggregate(MongoCollection collection) {
  System.out.println("");
  System.out.println("############################## 2. Find employees who failed in aggregate (term1 + term2 + term3) ##############################");
  //term1 + term2 + term3
  //37 + 37 + 37 = 111
  Collection<Document> employees = collection.aggregate(asList(new Document("$unwind","$results"),new Document("$group", new Document("_id","$name").append("total", new Document("$sum","$results.score"))),new Document("$match", new Document("total",new Document("$lt",111))))).into(new ArrayList<Document>());
  for (Document doc:employees){
   System.out.println("employees who failed in aggregate (term1+ term2 + term3): " + doc.toJson());
  }
 }
```

- Find the Average score of trainees for term1.

```
/**
 * function to find a record input : mongodb collection
 * 3. Find the Average score of trainees for term1
 */
 private static void averageScoreTerm1(MongoCollection collection) {
  System.out.println("");
  System.out.println("############################## 3. Find the Average score of trainees for term1 ##############################");
  Collection<Document> employees = collection.aggregate(asList( new Document("$unwind","$results"), new Document("$match",new Document("results.evaluation","term1")),new Document("$group",new Document("_id",null).append("Average", new Document("$avg","$results.score"))))).into(new ArrayList<Document>());
  for(Document doc:employees){
   System.out.println("Average score of trainees for term1 : " + doc.toJson());
  }
 }
```

Experiment 10:

Execute below sets of problem by taking reference of Experiment Number 08 and find out:

- Find the Average score of trainees for aggregate (term1 + term2 + term3).

```
/**
* function to find a record input : mongodb collection
* 4. Find the Average score of trainees for aggregate (term1 + term2 + term3)
*/
private static void averageClassScore(MongoCollection collection) {
 System.out.println("");
 System.out.println("############################## 4. Find the Average score of trainees for
     aggregate (term1 + term2 + term3)      ##############################");
 Collection<Document> employees = collection.aggregate(asList(new
     Document("$unwind","$results"),new Document("$group", new
     Document("_id","$name").append("Average", new Document("$avg","$results.score"))))).into(new
     ArrayList<Document>());
 for(Document doc:employees){
  System.out.println("Average score of trainees for aggregate (term1+ term2 + term3) : " + doc.toJson());
 }
}
```

- Find number of employees who failed in all the three (term1 + term2 + term3).

```
/**
 * function to find a record input : mongodb collection
 * 5. Find number of employees who failed in all the three (term1 + term2 + term3)
 */
 private static void employeeCountFailInAlLTerms(MongoCollection collection) {
  System.out.println("");
  System.out.println("############################## 5. Find number of employees who failed in all
     the three (term1 + term2 + term3) ##############################");
  Long count=collection.count(new Document("results.0.score",new
     Document("$lt",37)).append("results.1.score",new Document("$lt",37)).append("results.2.score",new
     Document("$lt",37)));
  System.out.println("Count of employees failing in all terms : " + count);
 }
```

- Find the number of employees who failed in any of the three (term1 + term2 + term3).

```
  **
  * function to find a record input : mongodb collection
  * 6. Find the number of employees who failed in any of the three (term1 + term2 + term3)
  */
  private static void employeeCountFailAtleastATerm(MongoCollection collection) {
   System.out.println("");
   System.out.println("############################## 6. Find the number of employees who failed in
     any of the three (term1 + term2 + term3) ##############################");
```

```
 Long count=collection.count(new Document("$or",asList(new Document("results.0.score",new
     Document("$lt",37)), new Document("results.1.score",new Document("$lt",37)),new
     Document("results.2.score",new Document("$lt",37)))));
 System.out.println("Count of employees failing in either of the terms : " + count);

 }
}
```

Lab Exp-8, 9 & 10(working with mongodb and python)

1. Anand Corp is a leading corporate training provider. A lot of prestigious organizations send their employees to Anand Corp for training on different skills. As a distinct training provider, Anand Corp has decided to share analysis report with their clients. This report will help their clients know the employees who have completed training and evaluation exam, what are their strengths, and what are the areas where employees need improvement. This is going to be a unique selling feature for the Anand Corp. As Anand Corp is already doing great business and they give training to a large number of people every month, they have huge amount of data to deal with. They have hired you as an expert and want your help to solve this problem.

   Attributes of data:
   Id : id of the person who was trained
   Name : name of the person who was trained
   Evaluation : evaluation term
   Score : score achieved by the person for the specific term
   A person can undergo multiple evaluations. Each evaluation will have a unique result score.
   You can see the sample data below.
   Sample Data

```
{
"_id":0,
"name":"Andy",
"results": [
{"evaluation":"term1","score":1.463179736705023},
{"evaluation":"term2","score":11.78273309957772},
{"evaluation":"term3","score":6.676176060654615}
]
}
```

   PQR Corp has assigned the following tasks to you to analyze the results:

   1. Find count and percentage of employees who failed in term 1, the passing score being 37
   2. Find employees who failed in aggregate (term1 + term2 + term3).
   3. Find the Average score of trainees for term1
   4. Find the Average score of trainees for aggregate (term1 + term2 + term3).
   5. Find number of employees who failed in all the three (term1 + term2 + term3).
   6. Find the number of employees who failed in any of the three (term1 + term2 + term3).

<u>Solution</u>

1. Configure the mongoDB in Jupiter note book

```
pip install pymongo
```

2. Import package

```
import pymongo
```

3. Make connection with mongoDB

```
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["employee"]
```

4. Display all the database in mongoDB

```
print(client.list_database_names())
```

5. Display the collections in employee Database

```
collist = db.list_collection_names()
print(collist)
```

6. Display the documents present in Training collection.

```
col = db['training']
for x in col.find():
  print(x)
```

Solution for first problem

```
# Find count and percentage of employees who failed in term 1, the passing score being 37
count=0
score=0
for x in col.find({},{"results":1}):
    #print(x["results"])
    for y in x["results"]:
        if y["evaluation"]=="term1" and y["score"]>=37:
            score=score+y["score"]
            count=count+1
print(count)
percentage=(score/count)
print(percentage)
```

Solution for the problem -2

```
# Find employees who failed in aggregate (term1 + term2 + term3)
count=0
score=[]
for x in col.find({},{"_id":1,"name":1,"results":1}):
    temp=x["results"]
    count=temp[0]["score"]+temp[1]["score"]+temp[2]["score"]
    per=count/3
    if per<=37:
        print(x["name"])


# Find the Average score of trainees for term1
count=0
score=[]
for x in col.find({},{"_id":1,"name":1,"results":1}):
    temp=x["results"]
    count=temp[0]["score"]
    print(x["_id"],x["name"],count)
```

Extra program for students to learn mongo DB

Practice Question Set-1

The Collection called “marketing” which stores data about a marketing campaign of a retail business. A document in this collection includes the following pieces of information.

```
{
 "_id" : ObjectId("6014dc988c628fa57a508088"),
 "Age" : "Middle",
 "Gender" : "Male",
 "OwnHome" : "Rent",
 "Married" : "Single",
 "Location" : "Close",
 "Salary" : 63600,
 "Children" : 0,
 "History" : "High",
 "Catalogs" : 6,
 "AmountSpent" : 1318
}
```

Query - 1

The find method retrieves all documents by default. We can use the limit method to display only a number of the documents. For instance, we can display the first document in the collection as follows:

```
> db.marketing.find().limit(1).pretty(){
 "_id" : ObjectId("6014dc988c628fa57a508088"),
 "Age" : "Middle",
 "Gender" : "Male",
 "OwnHome" : "Rent",
 "Married" : "Single",
 "Location" : "Close",
 "Salary" : 63600,
 "Children" : 0,
 "History" : "High",
 "Catalogs" : 6,
 "AmountSpent" : 1318
}
```

The db refers to the current database. We need to specify the collection name after the dot. The pretty method is to display the documents in a more structured way. Without the pretty method, the output looks like this:

```
{ "_id" : ObjectId("6014dc988c628fa57a508088"), "Age" : "Middle", "Gender" : "Male",
"OwnHome" : "Rent", "Married" : "Single", "Location" : "Close", "Salary" : 63600, "Children"
: 0, "History" : "High", "Catalogs" : 6, "AmountSpent" : 1318 }
```

Query-2

The find method allows for some basic filtering. We can specify the desired value for fields inside the find method as follows:

```
> db.marketing.find( {"Children": 1} ).limit(1).pretty(){
 "_id" : ObjectId("6014dc988c628fa57a50808a"),
 "Age" : "Middle",
 "Gender" : "Male",
 "OwnHome" : "Own",
 "Married" : "Married",
 "Location" : "Close",
 "Salary" : 85600,
 "Children" : 1,
 "History" : "High",
 "Catalogs" : 18,
 "AmountSpent" : 2436
}
```

However, this is not the best way for filtering. The aggregation pipeline of MongoDB provides a much more efficient way for filtering, transforming, and aggregating data as we will see in the following examples.

Query-3

We want to find out the average spent number of customers who received more than 10 catalogs. The aggregate pipeline can be used to accomplish this task as follows.

```
> db.marketing.aggregate([
... { $match : { Catalogs : {$gt : 10} } },
... { $group : { _id : null, avgSpent : {$avg : "$AmountSpent"} } }
... ]){ "_id" : null, "avgSpent" : 1418.9411764705883 }
```

The first step of the pipeline is the match stage which filters documents according to the given condition. The "$gt" expression stands for "greater than". The group stage performs the aggregation based on the given fields and aggregation function.

Query-4

In the previous example, we found the average spent amount of customers who received more than 10 catalogs. Let's take it one step further and find the same value for different age groups separately.

```
> db.marketing.aggregate([
... { $match : { Catalogs : {$gt : 10} } },
... { $group : { _id : "$Age", avgSpent : {$avg : "$AmountSpent"} } }
... ]){ "_id" : "Middle", "avgSpent" : 1678.3965087281795 }
{ "_id" : "Old", "avgSpent" : 1666.9056603773586 }
{ "_id" : "Young", "avgSpent" : 655.813829787234 }
```

The only change is on the "_id" value. It specifies the field to be used as the grouping field.

Query - 5

It is possible to perform multiple aggregations on multiple fields. For instance, we can calculate the average salary and the total spent amount for customers who have at least 1 child.

```
> db.marketing.aggregate([
... { $match : { Children : {$gt : 0} } },
... { $group: {
...             _id : null,
...             "avgSalary" : {$avg: "$Salary"},
...             "totalSpent" : {$sum: "$AmountSpent"}
...           }
... }
... ]){ "_id" : null, "avgSalary" : 57140.89219330855, "totalSpent" : 566902 }
```

Each aggregation is written as a separate entry in the group stage.

Query-6

The aggregate pipeline of MongoDB has a stage for sorting the query results. It is useful when the results consist of several entries. Moreover, a sorted set of data provides a more structured overview.

The following query returns the average salary of customers who spent more than 1000 dollars. The results are grouped based on the number of children and sorted by the average salary in descending order.

```
> db.marketing.aggregate([
... { $match: { AmountSpent : {$gt : 1000}}},
... { $group: { _id : "$Children", avgSalary : {$avg : "$Salary"}}},
... { $sort: { avgSalary : -1 }}
... ]){ "_id" : 3, "avgSalary" : 84279.48717948717 }
{ "_id" : 2, "avgSalary" : 83111.11111111111 }
{ "_id" : 1, "avgSalary" : 78855.97014925373 }
{ "_id" : 0, "avgSalary" : 71900.76045627377 }
```

In the sort stage, “-1” indicates descending and “1” indicates ascending order.

Query-7

We will see two new stages of the aggregation pipeline in this examples.

The project stage allows for selecting fields to be displayed. Since a typical document is likely to have many fields, displaying all of them may not be the optimal choice.

In the project stage, we select the fields to be displayed by the value 1. The result only displays the specified fields. It is important to note that the id field is displayed by default in all cases. We need to explicitly set it as 0 to exclude the id field from the result.

The limit stage in the aggregation pipeline puts a limit on the number of documents to be displayed. It is same as the limit keyword in SQL.

The following query filters the documents based on the spent amount and then sort by the salary. It only displays the salary and spent amount fields of the first 5 documents.

```
> db.marketing.aggregate([
... { $match : { AmountSpent : {$gt : 1500} } },
... { $sort : { Salary : -1 } },
... { $project : { _id : 0, Salary : 1, AmountSpent : 1 } },
... { $limit : 5 }
... ]){ "Salary" : 168800, "AmountSpent" : 1512 }
{ "Salary" : 140000, "AmountSpent" : 4894 }
{ "Salary" : 135700, "AmountSpent" : 2746 }
{ "Salary" : 134500, "AmountSpent" : 4558 }
{ "Salary" : 131500, "AmountSpent" : 2840 }
```

Practice Question - 2

Demonstrate how data can be retrieved from a MongoDB database.

We have a collection called “customer”. The documents in the customer collection contains customer name, age, gender, and the amount of the last purchase.

Here is a document in the customer collection:

```
{
 "_id" : ObjectId("600c1806289947de938c68ea"),
 "name" : "John",
 "age" : 32,
 "gender" : "male",
 "amount" : 32
}
```

The document is displayed in JSON format.

Query-1

Query documents that belong to a specific customer.

We use the find method to query documents from a MongoDB database. If used without any arguments or collections, find method retrieves all documents.

We want to see the document belongs to customer John so the name field needs to be specified in the find method.

```
> db.customer.find( {name: "John"} ){ "_id" : ObjectId("600c1806289947de938c68ea"), "name" :
"John", "age" : 32, "gender" : "male", "amount" : 32 }
```

We can attach pretty method to make the document seem more appealing.

```
> db.customer.find( {name: "John"} ).pretty(){
 "_id" : ObjectId("600c1806289947de938c68ea"),
 "name" : "John",
 "age" : 32,
 "gender" : "male",
 "amount" : 32
}
```

It is easier to read now.

Query-2

Query documents that belong to customers older than 40.

The condition is applied to age field using a logical operator. The “$gt” stands for “greater than” and is used as follows.

```
> db.customer.find( {age: {$gt:40}} ).pretty(){
 "_id" : ObjectId("600c19d2289947de938c68ee"),
 "name" : "Jenny",
 "age" : 42,
 "gender" : "female",
 "amount" : 36
}
```

Query 3

Query documents that belong to female customers who are younger than 25.

This example is like a combination of the previous two examples. Both conditions must be met so we use "and" logic to combine the conditions. It can be done by writing both conditions separated by comma.

```
> db.customer.find( {gender: "female", age: {$lt:25}} ).pretty(){
 "_id" : ObjectId("600c19d2289947de938c68f0"),
 "name" : "Samantha",
 "age" : 21,
 "gender" : "female",
 "amount" : 41
}{
 "_id" : ObjectId("600c19d2289947de938c68f1"),
 "name" : "Laura",
 "age" : 24,
 "gender" : "female",
 "amount" : 51
}
```

The "$lt" stands for "less than".

Query 4

In this example, we will repeat the previous example in a different way. Multiple conditions can also be combined with "and" logic as below.

```
> db.customer.find( {$and :[ {gender: "female", age: {$lt:25}} ]} ).pretty()
```

The logic used for combining the conditions is indicated at the beginning. The remaining part is same as the previous example but we need to put the conditions in a list ([ ]).

Query 5

Query customers who are either male or younger than 25.

This example requires a compound query with "or" logic. We just need to change "$and" to "$or".

```
> db.customer.find( { $or: [ {gender: "male"}, {age: {$lt: 22}} ] }){ "_id" :
ObjectId("600c1806289947de938c68ea"), "name" : "John", "age" : 32, "gender" : "male",
"amount" : 32 }{ "_id" : ObjectId("600c19d2289947de938c68ed"), "name" : "Martin", "age" :
28, "gender" : "male", "amount" : 49 }{ "_id" : ObjectId("600c19d2289947de938c68ef"), "name"
: "Mike", "age" : 29, "gender" : "male", "amount" : 22 }{ "_id" :
ObjectId("600c19d2289947de938c68f0"), "name" : "Samantha", "age" : 21, "gender" : "female",
"amount" : 41 }
```

Query 6

MongoDB allows for aggregating values while retrieving from the database. For instance, we can calculate the total purchase amount for males and females. The aggregate method is used instead of the find method.

```
> db.customer.aggregate([
... { $group: {_id: "$gender", total: {$sum: "$amount"} } }
... ]){ "_id" : "female", "total" : 198 }
{ "_id" : "male", "total" : 103 }
```

Let's elaborate on the syntax. We first group the documents by the gender column by selecting "$gender" as id. The next part specifies both the aggregation function which is "$sum" in our case and the column to be aggregated.

If you are familiar with Pandas, the syntax is quite similar to the groupby function.

Query 7

Let's take the previous example one step further and add a condition. Thus, we first select documents that "match" a condition and apply aggregation.

The following query is an aggregation pipeline which first selects the customers who are older than 25 and calculates the average purchase amount for males and females.

```
> db.customer.aggregate([
... { $match: { age: {$gt:25} } },
... { $group: { _id: "$gender", avg: {$avg: "$amount"} } }
... ]){ "_id" : "female", "avg" : 35.33 }
{ "_id" : "male", "avg" : 34.33 }
```

Query 8

The query in the previous example contains only two groups so it is not necessary to sort the results. However, we might have queries that return several values. In such cases, sorting the results is a good practice.

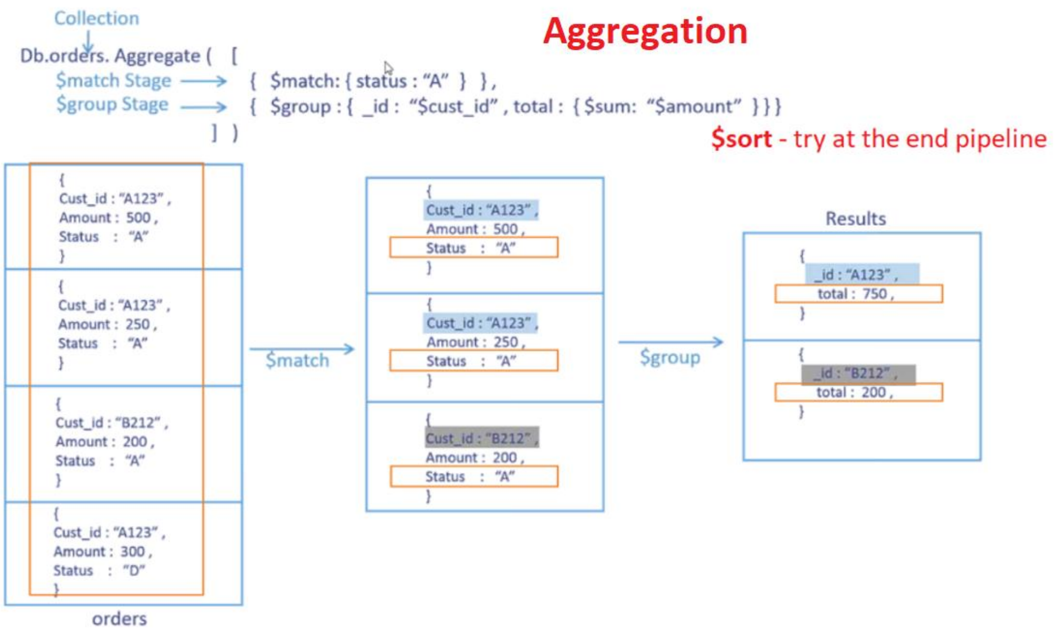
We can sort the results of the previous query by the average amount in ascending order.

```
> db.customer.aggregate([
... { $match: { age: {$gt:25} } },
... { $group: { _id: "$gender", avg: {$avg: "$amount"} } },
... { $sort: {avg: 1} }
... ]){ "_id" : "male", "avg" : 34.33 }
{ "_id" : "female", "avg" : 35.33 }
```

We have just added "$sort" in the aggregation pipeline. The field used for sorting is specified along with the sorting behavior. 1 means in ascending order and -1 means in descending order.

Practice Question - 3

Consider the below sample data to perform the aggregation


Collection
Aggregation
Db.orders. Aggregate ([
$match Stage → { $match: { status : "A" } },
$group Stage → { $group : { _id : "$cust_id" , total : { $sum: "$amount" } } }
])
$sort - try at the end pipeline
Cust_id : "A123" , Amount : 500 , Status : "A"
Cust_id : "A123" , Amount : 250 , Status : "A"
Cust_id : "B212" , Amount : 200 , Status : "A"
Cust_id : "A123" , Amount : 300 , Status : "D"
orders
$match
$group
Results
_id : "A123" , total : 750 ,
_id : "B212" , total : 200 ,

Aggregate ##

//Calculate the total transaction that customer did till now (consider only transaction >= 10000)
db.customer_info.aggregate([
 {$match:{"transaction": {$gte:100000}}}, //consider only records gte=100000
 {$group:{{_id: "Cust_Id", transaction: {$sum:"$amount"}}}}, //if Cust_Id is same add the transaction
 {$sort:"trasaction":1}}]);

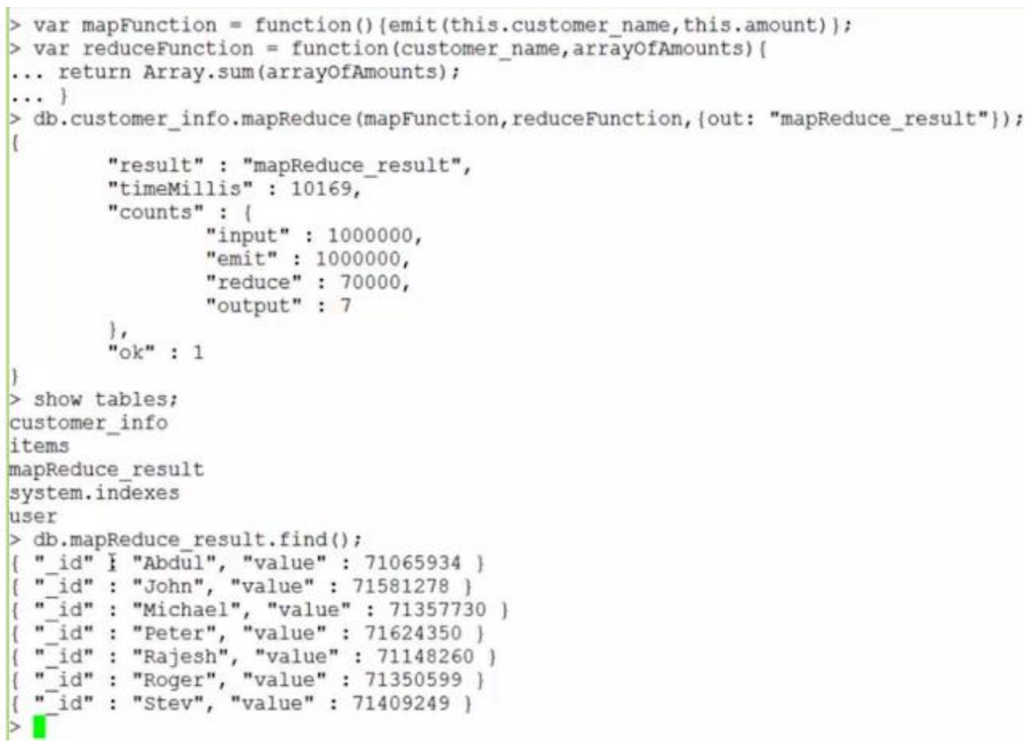
db.items.count(); //6
db.items.distinct("item"); //["pen", "pencil", "books"]
db.items.count({item: "pen"}); //2

MapReduce

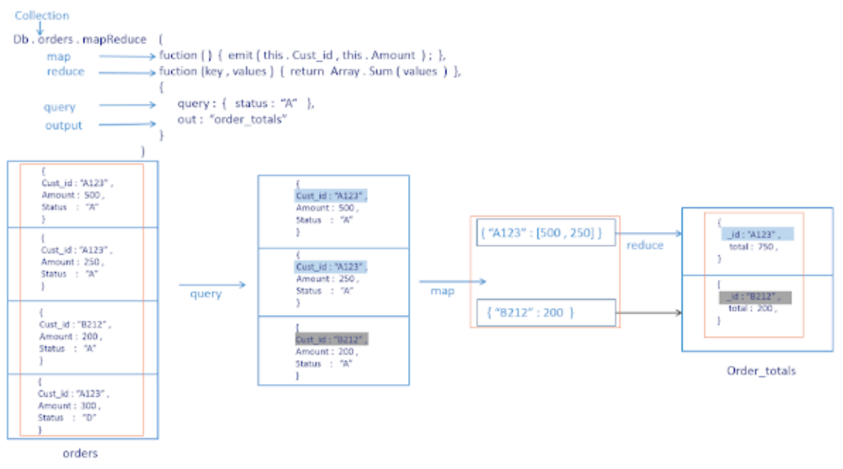
Map reduce: (Looks same as Aggregation, but this works parallely)
----
var mapFunction = function() {emit(this.customer_name,this.amount)}; // customer_name, amount are fields in customer_info.

var reduceFunction = function(customer_name,arrayOfAmounts){
return Array.sum(arrayOfAmounts);
}

//using **mapFunction** and **educeFunction** => it generates **mapReduce_result** table with final o/p
db.customer_info.mapReduce(mapFunction,reduceFunction,{out: "mapReduce_result"});
show tables; //you see - mapReduce_result
db.mapReduce_result.find(); //o/p will be reduced o/p



```
> var mapFunction = function(){emit(this.customer_name,this.amount)};
> var reduceFunction = function(customer_name,arrayOfAmounts){
... return Array.sum(arrayOfAmounts);
... }
> db.customer_info.mapReduce(mapFunction,reduceFunction,{out: "mapReduce_result"});
{
        "result" : "mapReduce_result",
        "timeMillis" : 10169,
        "counts" : {
                "input" : 1000000,
                "emit" : 1000000,
                "reduce" : 70000,
                "output" : 7
        },
        "ok" : 1
}
> show tables;
customer_info
items
mapReduce_result
system.indexes
user
> db.mapReduce_result.find();
{ "_id" : "Abdul", "value" : 71065934 }
{ "_id" : "John", "value" : 71581278 }
{ "_id" : "Michael", "value" : 71357730 }
{ "_id" : "Peter", "value" : 71624350 }
{ "_id" : "Rajesh", "value" : 71148260 }
{ "_id" : "Roger", "value" : 71350599 }
{ "_id" : "Stev", "value" : 71409249 }
>
```



```
Db . orders . mapReduce  (
                 fuction ( ) {  emit ( this . Cust_id , this . Amount  ) ;  },
                 fuction (key , values ) {  return  Array . Sum ( values  )  },
                 {
                    query : {  status :  "A"   },
                    out :  "order_totals"
                 }
            )
```

Collection
map
reduce
query
output
{ Cust_id : "A123" , Amount : 500 , Status : "A" }
{ Cust_id : "A123" , Amount : 250 , Status : "A" }
{ Cust_id : "B212" , Amount : 200 , Status : "A" }
{ Cust_id : "A123" , Amount : 300 , Status : "D" }
orders
query
{ Cust_id : "A123" , Amount : 500 , Status : "A" }
{ Cust_id : "A123" , Amount : 250 , Status : "A" }
{ Cust_id : "B212" , Amount : 200 , Status : "A" }
map
{ "A123" : [500 , 250] }
{ "B212" : 200 }
reduce
{ _id : "A123" , total : 750 , }
{ _id : "B212" , total : 200 , }
Order_totals

Group

//display totalAmount of particular customer_name = "John"

```
db.customer_info.group({key: {"customer_name":1},     //group on customer_name
                cond: {"customer_name": "John"},       //get only "John" customer
                reduce:function(curr,result)           //reduce: add all the amount
                        {result.amount += curr.amount;},
                intial: {amount:0}})
```

Course Outcomes:

- Summarize on the environment tools and systems necessary for open-source software practice.
- Demonstrate the usage of Sort command in Mongo DB.
- Write a query to retrieve database from different MongoDB operations.
- Demonstrate the different usage of applications.