Lab Task -1 CRUD operations in MongoDB

Chapter · February 2023

CITATIONS
0

READS
187

1 author:

Bal Krishna Nyaupane
Tribhuvan University
33 PUBLICATIONS **9** CITATIONS

SEE PROFILE

Lab Task -1
CRUD operations in MongoDB

Bal Krishna Nyaupane

Assistant Professor

Department of Electronics and Computer Engineering

Paschimanchal Campus, IOE

bkn@wrc.edu.np

MongoDB

- MongoDB is powerful but easy to get started with. Some of the basic concepts of MongoDB:
- A ***document is the basic unit of data for MongoDB*** *and is roughly* ***equivalent to a row in a relational database management system*** (but much more expressive).
- A ***collection is a group of documents***. If a document is the MongoDB analog of a row in a relational database, then ***a collection can be thought of as the analog to a table.***
- A single instance of MongoDB can host multiple independent *databases, each of* which can have its own collections.
- ***Every document has a special key, "_id***", that is unique within a collection.
- MongoDB comes with a simple but powerful ***JavaScript shell, which is useful for the administration of MongoDB instances and data manipulation***.
- ***At the heart of MongoDB is the document: an ordered set of keys with associated values.***

MongoDB

- MongoDB is ***type-sensitive and case-sensitive***. For example, these documents are distinct***: {"foo" : 3} and {"foo" : "3"}.***
- A final important thing to note is that ***documents in MongoDB cannot contain duplicate keys.*** For example, the following is ***not a legal document***:

  {"greeting" : "Hello, world!", "greeting" : "Hello, MongoDB!"}.

- ***Dynamic Schemas:*** Collections have *dynamic schemas. This means that the* ***documents within a single collection can have any number of different "shapes."***
- ***For example***, both of the following documents ***could be stored in a single collection:***

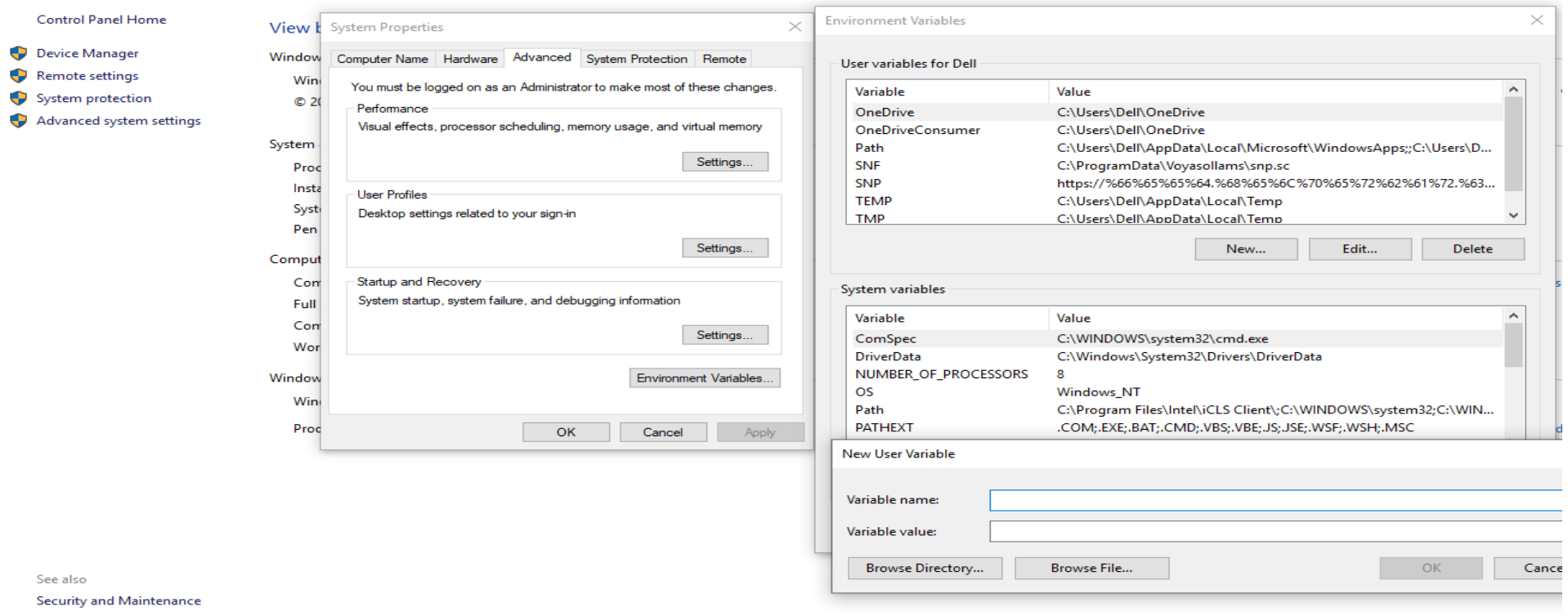
  {"greeting" : "Hello, world!"}
  {"foo" : 5}

MongoDB – Installation (Windows)

- Download MongoDB from following link

  https://fastdl.mongodb.org/windows/mongodb-windows-x86_64-5.0.14-signed.msi

- Install and set Path Variable (***C:\Program Files\MongoDB\Server\4.0\bin)***
- MongoDB requires a data folder to store its files. The default location for the MongoDB data directory is ***c:\data\db.***
- Open command prompt and ***execute the following command***.
  - mongod
  - mongo



MongoDB - Create Database

- To check your currently selected database, use the command ***db***
- If you want to check your databases list, use the command ***show dbs.***

```
> db
test
> show dbs
admin     0.000GB
config    0.000GB
local     0.000GB
mydb      0.000GB
>
```

- In MongoDB ***default database is test***. If you didn't create any database, then collections will be stored in test database.
- If you want to use database with name <***db_name***>, then ***use db_name*** statement.
- If Database does not exists then it will create a new Database ***db_name.***

```
> use admin
switched to db admin
```

MongoDB - Drop Database

- To exit from MongoDB shell:

```
> exit
bye

C:\Users\balkr>
```

- An alternative way to close the shell is to press ***CTRL + C***
- MongoDB **db.dropDatabase()** command is used to drop a existing database.

Syntax: ***db.dropDatabase()***

- If you want to delete new database **mydb**, then **dropDatabase()** command would be as follows:

```
> use mydb
switched to db mydb
> db.dropDatabase()
{ "dropped" : "mydb", "ok" : 1 }
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
>
```

Useful Commands

```
> db.help()
DB methods:
        db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs command [just calls db.runCommand(...)]
        db.aggregate([pipeline], {options}) - performs a collectionless aggregation on this database; returns a cursor
        db.auth(username, password)
        db.commandHelp(name) returns the help for the command
        db.createUser(userDocument)
        db.createView(name, viewOn, [{$operator: {...}}, ...], {viewOptions})
        db.currentOp() displays currently executing operations in the db
        db.dropDatabase(writeConcern)
        db.dropUser(username)
        db.eval() - deprecated
        db.fsyncLock() flush data to disk and lock server for backups
        db.fsyncUnlock() unlocks server following a db.fsyncLock()
        db.getCollection(cname) same as db['cname'] or db.cname
        db.getCollectionInfos([filter]) - returns a list that contains the names and options of the db's collections
        db.getCollectionNames()
        db.getLastError() - just returns the err msg string
        db.getLastErrorObj() - return full status object
        db.getLogComponents()
        db.getMongo() get the server connection object
        db.getMongo().setSecondaryOk() allow queries on a replication secondary server
        db.getName()
        db.getProfilingLevel() - deprecated
        db.getProfilingStatus() - returns if profiling is on and slow threshold
        db.getReplicationInfo()
        db.getSiblingDB(name) get the db at the same server as this one
        db.getWriteConcern() - returns the write concern used for any operations on this db, inherited from server object if set
        db.hostInfo() get details about the server's host
        db.isMaster() check replica primary status
        db.hello() check replica primary status
        db.killOp(opid) kills the current operation in the db
        db.listCommands() lists all the db commands
```

MongoDB – Create Collection

- MongoDB **db.createCollection(name, options)** is used to create collection.

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

- In the command, **name** is name ***of collection*** to be created. **Options** is a document and is used ***to specify configuration of collection***. *Options parameter is optional*, so you need to specify only the name of the collection.
- You can check the created collection by using the command **show collections**.

```
> use mydb
switched to db mydb
> db.createCollection("MyCollection")
{ "ok" : 1 }
> show collections
MyCollection
>
```

MongoDB – Drop Collection

- In MongoDB, ***you don't need to create collection***. ***MongoDB creates collection automatically***, when you insert some document.

```
> db.exCollection.insert({"Name": "ExampleCollection"})
WriteResult({ "nInserted" : 1 })
> show collections
MyCollection
exCollection
>
```

- MongoDB's **db.collection.drop()** is used to drop a collection from the database.

Syntax: ***db.Collection_Name.drop()***

```
> db.exCollection.drop()
true
> show collections
MyCollection
>
```

MongoDB – Data Type

- **String** − String in MongoDB must be UTF-8 valid.
- **Integer** − Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** − This type is used to store a boolean (true/ false) value.
- **Double** − This type is used to store floating point values.
- **Min/ Max keys** − This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays** − This type is used to store arrays or list or multiple values into one key.
- **Timestamp** − ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object** − This data type is used for embedded documents.
- **Null** − This type is used to store a Null value.

MongoDB – Data Type

- **Symbol** − This data type is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- **Date** − This data type is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID** − This data type is used to store the document's ID.
- **Binary data** − This data type is used to store binary data.
- **Code** − This data type is used to store JavaScript code into the document.
- **Regular expression** − This data type is used to store regular expression.

Inserting and Updating Documents

- Create a Collection *by the name "student" and insert document*. The documents contains following information: *id, Name, Age, Mobile, Batch, and Address.*
  - *To view the documents has been inserted into the collection:* ***use find() method.***
  - *To format the result, add pretty()method.*

```
> db.student.insert(
... {
... _id:1,
... Name: " Dinesh",
... Age:25,
... Mobile: 123456779,
... Batch:2073,
... Address: "Lamachaur"
... }
... )
```

```
> db.student.find().pretty()
{
        "_id" : ObjectId("5d0354ade63b95a06fb9cda8"),
        "Name" : " Dinesh",
        "Age" : 25,
        "Mobile" : 123456779,
        "Batch" : 2073,
        "Address" : "Lamachaur"
}
{
        "_id" : 1,
        "Name" : " Dinesh",
        "Age" : 25,
        "Mobile" : 123456779,
        "Batch" : 2073,
        "Address" : "Lamachaur"
}
```

```
> db.student.find()
{ "_id" : ObjectId("5d0354ade63b95a06fb9cda8"), "Name" : " Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur" }
{ "_id" : 1, "Name" : " Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur" }
```

Inserting and Updating Documents

- *Add few documents into the student collection..*

```
> db.student.find().pretty()
{
        "_id" : 2,
        "Name" : "Ramesh",
        "Age" : 22,
        "Mobile" : 124356779,
        "Batch" : 2073,
        "Address" : "Bagar",
        "Hobbies" : "Baseball"
}
{
        "_id" : 3,
        "Name" : "Shyam",
        "Age" : 23,
        "Mobile" : 123457879,
        "Batch" : 2072,
        "Address" : "New Road"
}
{
        "_id" : 4,
        "Name" : "Santosh",
        "Age" : 21,
        "Mobile" : 987456779,
        "Batch" : 2073,
        "Address" : "Batulechaur"
}
{
        "_id" : 5,
        "Name" : "Biplav",
        "Age" : 23,
        "Mobile" : 773456779,
        "Batch" : 2072,
        "Address" : "Halan Chowk"
}
{
        "_id" : 1,
        "Name" : "Dinesh",
        "Age" : 25,
        "Mobile" : 123456779,
        "Batch" : 2073,
        "Address" : "Lamachaur"
}
>
```

Bulk Inserting

- *To insert multiple documents in a single query, you can pass an array of documents in insert() command.*

```
})
> db.post.insert([ {Name: "Dinesh", Age:25 },{Name:"Ram", Age:33},{Name:"Sita",Age:24}])
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
> db.post.find()
{ "_id" : ObjectId("5d0498362c381963c62985af"), "Name" : "Dinesh", "Age" : 25 }
{ "_id" : ObjectId("5d0498362c381963c62985b0"), "Name" : "Ram", "Age" : 33 }
{ "_id" : ObjectId("5d0498612c381963c62985b1"), "Name" : "Dinesh", "Age" : 25 }
{ "_id" : ObjectId("5d0498612c381963c62985b2"), "Name" : "Ram", "Age" : 33 }
{ "_id" : ObjectId("5d0498612c381963c62985b3"), "Name" : "Sita", "Age" : 24 }
>
```

Update() Method

- Insert the document for ***"Bhimesh" into "student" collection*** only if it does not already exist in the collection. However, if it is already present in the collection, then update the document with new values. ***(Update Hobbies from Basketball to Chess.)*** ***Use Update else insert.***

```
> db.student.update(
... {
... _id:6,
... Name: "Bhimesh",
... Age:24,
... Mobile:7777856779,
... Batch:2073,
... Address: "Halan Chowk"
... },
... {$set:{Hobbies:"BAsketball"}},
... {upsert:true}
...         )
```

```
> db.student.find()
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22, "Mobile" : 124356779, "Batch" : 2073, "Address" : "Bagar", "Hobbies" : "Baseball" }
{ "_id" : 3, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073, "Address" : "Batulechaur" }
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "Address" : "Halan Chowk" }
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur" }
{ "_id" : 6, "Address" : "Halan Chowk", "Age" : 24, "Batch" : 2073, "Mobile" : 7777856779, "Name" : "Bhimesh", "Hobbies" : "BAsketball" }
```

Update() Method

- *(Update Hobbies from Basketball to Chess.)* ***Use Update else insert.***

```
> db.student.update(
... {
... _id:6,
... Name: "Bhimesh",
... Age:24,
... Mobile:7777856779,
... Batch:2073,
... Address: "Halan Chowk"
... },
... {$set:{Hobbies:"Chess"}},
... {upsert:true}
...         )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.student.find()
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22, "Mobile" : 124356779, "Batch" : 2073, "Address" : "Bagar", "Hobbies" : "Baseball" }
{ "_id" : 3, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073, "Address" : "Batulechaur" }
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "Address" : "Halan Chowk" }
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur" }
{ "_id" : 6, "Address" : "Halan Chowk", "Age" : 24, "Batch" : 2073, "Mobile" : 7777856779, "Name" : "Bhimesh", "Hobbies" : "Chess" }
```

Save() Method

- *The **save () method** will **insert a new document** if the document with the specified **_id does not exist**. However, **if a document with the specified id exist**, it **replace** the existing document **with the new one.***

```
> db.student.save(
... {
... _id:7,
... Name:"Bhim",
... Batch:2072,
... Address: "Halan Chowk"
... }
...  )
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 7 })
> db.student.find()
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22, "Mobile" : 124356779, "Batch" : 2073, "Address" : "Bagar", "Hobbies" : "Baseball" }
{ "_id" : 3, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073, "Address" : "Batulechaur" }
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "Address" : "Halan Chowk" }
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur" }
{ "_id" : 6, "Address" : "Halan Chowk", "Age" : 24, "Batch" : 2073, "Mobile" : 7777856779, "Name" : "Bhimesh", "Hobbies" : "Chess" }
{ "_id" : 7, "Name" : "Bhim", "Batch" : 2072, "Address" : "Halan Chowk" }
```

Adding a new field to an Existing Document

- *To **add a new field "Gender" with value "Male"** to the document **(_id:1)** of students.*

```
> db.student.update(
... {_id:1},
... {$set:{Gender:"Male"}}
...       )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find()
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22, "Mobile" : 124356779, "Batch" : 2073, "Address" : "Bagar", "Hobbies" : "Baseball" }
{ "_id" : 3, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073, "Address" : "Batulechaur" }
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "Address" : "Halan Chowk" }
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur", "Gender" : "Male" }
```

Remove() Method

- ***Remove _id:6 documents from the collection.***

```
> db.student.remove( {_id:6} )
WriteResult({ "nRemoved" : 1 })
> db.student.find()
{ "_id" : 3, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073, "Address" : "Batulechaur" }
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "Address" : "Halan Chowk" }
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur", "Gender" : "Male" }
```

- *To **Remove a field "Gender"** with value **"Male"** to the document **(_id:1)** of students.*

```
> db.student.update(
... {_id:1},
... {$unset:{Gender:"Male"}}
...         )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find()
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073, "Address" : "Batulechaur" }
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "Address" : "Halan Chowk" }
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur" }
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
```

Remove() Method

- If there are ***multiple records*** and you ***want to delete only the first record***, then set just **One parameter** in **remove()** method.

```
> db.student.find()
{ "_id" : 3, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073, "Address" : "Batulechaur" }
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "Address" : "Halan Chowk" }
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur", "Gender" : "Male" }
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
> db.student.remove(
... {Name:"Shyam"},1
... )
WriteResult({ "nRemoved" : 1 })
> db.student.find()
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073, "Address" : "Batulechaur" }
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "Address" : "Halan Chowk" }
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur", "Gender" : "Male" }
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
>
```

Find() Method

- Find the documents where Name has value "Dinesh".

```
> db.student.find({Name:"Dinesh"})
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "
> db.student.find({Name:"Dinesh"}).pretty()
{
        "_id" : 1,
        "Name" : "Dinesh",
        "Age" : 25,
        "Mobile" : 123456779,
        "Batch" : 2073,
        "Address" : "Lamachaur"
}
```

- *To display only the Name from all the documents of student's collection.* ***The identifier "_id" should be suppressed and NOT displayed.***

```
> db.student.find({},{Name:1,_id:0}).pretty()
{ "Name" : "Santosh" }
{ "Name" : "Biplav" }
{ "Name" : "Dinesh" }
{ "Name" : "Shyam" }
```

```
> db.student.find({},{Name:1}).pretty()
{ "_id" : 4, "Name" : "Santosh" }
{ "_id" : 5, "Name" : "Biplav" }
{ "_id" : 1, "Name" : "Dinesh" }
{ "_id" : 6, "Name" : "Shyam" }
```

Find() Method

- *To display Name, Batch, and Address from all the documents of student's collection.* ***The identifier "_id" should be suppressed and NOT displayed.***

```
> db.student.find({},{Name:1,Batch:1,Address:1,_id:0}).pretty()
{ "Name" : "Santosh", "Batch" : 2073, "Address" : "Batulechaur" }
{ "Name" : "Biplav", "Batch" : 2072, "Address" : "Halan Chowk" }
{ "Name" : "Dinesh", "Batch" : 2073, "Address" : "Lamachaur" }
{ "Name" : "Shyam", "Batch" : 2072, "Address" : "New Road" }
```

- *To display* ***Name, Address as well as the identifier, _id****, from all the documents of student's collection* ***where the Batch is 2073.***

```
> db.student.find({Batch:2073},{Name:1,Address:1}).pretty()
{ "_id" : 4, "Name" : "Santosh", "Address" : "Batulechaur" }
{ "_id" : 1, "Name" : "Dinesh", "Address" : "Lamachaur" }
>
```

Relational operators in the Search Criteria

- *To display Name, Batch, and Address from all the documents of student's collection.* ***The identifier "_id" should be suppressed and NOT displayed.***

Operator	Meaning
$eq	Equal to
$ne	Not equal to
$gte	Greater than or equal to
$lte	Less than or equal to
$gt	Greater than
$lt	Less than

Relational operators in the Search Criteria

- *To find those documents where the **Age is set to 23.***

```
> db.student.find({Age:{$eq:23}})
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "Address" : "Halan Chowk" }
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
{ "_id" : 3, "Name" : "Nitesh", "Age" : 23, "Mobile" : 893457879, "Batch" : 2073, "Address" : "Simpani" }
```

- *To find those documents from the student's collection where the Address is set to either "Bagar" or is set to "New Road".*

```
> db.student.find({Address:{$in:["Bagar","New Road"]}})
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22, "Mobile" : 124356779, "Batch" : 2073, "Address" : "Bagar", "Hobbies" : "Baseball" }
```

- *To find those documents from the student's collection where the Address is set to neither to" Bagar" nor is set to "New Road".*

```
> db.student.find({Address:{$nin:["Bagar","New Road"]}})
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073, "Address" : "Batulechaur" }
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "Address" : "Halan Chowk" }
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur" }
{ "_id" : 3, "Name" : "Nitesh", "Age" : 23, "Mobile" : 893457879, "Batch" : 2073, "Address" : "Simpani" }
```

Relational operators in the Search Criteria

- *To find documents where the Name begins with "B".*

```
> db.student.find({Name:/^B/}).pretty()
```

```
> db.student.find({Name:{$regex:"^B"}}).pretty()
```

- *To find documents where the Name ends with "v".*

```
> db.student.find({Name:/v$/}).pretty()
```

```
> db.student.find({Name:{$regex:"v$"}}).pretty()
```

- *To find documents where the Name has an "e" in any position.*

```
> db.student.find({Name:{$regex:"e"}}).pretty()
```

```
> db.student.find({Name:/e/}).pretty()
```

Dealing with Null Values

- *Update the documents with NULL values in the "Address" column whose _id is set to 3 and 4.*

```
> db.student.update( {_id:4}, {$set:{Address:null}} )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

- *To display the documents with NULL values in the "Address" column .*

```
> db.student.find(
... {Address:{$eq:null}}
... )
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073, "Address" : null }
{ "_id" : 3, "Name" : "Nitesh", "Age" : 23, "Mobile" : 893457879, "Batch" : 2073, "Address" : null }
```

- *To remove "Address" field having NULL values from the documents(_id:3)*

```
> db.student.update( {_id:3}, {$unset:{Address:null}} )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find()
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073 }
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "Address" : "Halan Chowk" }
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur" }
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22, "Mobile" : 124356779, "Batch" : 2073, "Address" : "Bagar", "Hobbies" : "Baseball" }
{ "_id" : 3, "Name" : "Nitesh", "Age" : 23, "Mobile" : 893457879, "Batch" : 2073 }
```

Dealing with Count, Limit, Sort, and Skip

- *Count the **number of documents** in student collection.*

```
> db.student.count()
6
```

- *Count the **number of document** in student collection **wherein the Batch is 2073.***

```
> db.student.count({Batch:2073})
4
```

- *Retrieve the **first 2 documents** from the student collection wherein the Batch is 2073.*

```
> db.student.find({Batch:2073}).limit(2)
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073 }
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur" }
```

Dealing with Count, Limit, Sort, and Skip

- ***Sort the documents*** *from the student collection in the* ***ascending order of Name****.*
- ***Sort the documents*** *from the student collection in the* ***descending order of Name****.*

```
> db.student.find().sort({Name:1})
{ "_id" : 5, "Name" : "Biplav", "Age" : 23,
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25,
{ "_id" : 3, "Name" : "Nitesh", "Age" : 23,
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22,
{ "_id" : 4, "Name" : "Santosh", "Age" : 21,
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "
>
```

```
> db.student.find().sort({Name:-1})
{ "_id" : 6, "Name" : "Shyam", "Age" :
{ "_id" : 4, "Name" : "Santosh", "Age"
{ "_id" : 2, "Name" : "Ramesh", "Age"
{ "_id" : 3, "Name" : "Nitesh", "Age"
{ "_id" : 1, "Name" : "Dinesh", "Age"
{ "_id" : 5, "Name" : "Biplav", "Age"
>
```

- ***Sort the documents*** *from the student collection first on* ***Age in Ascending*** *order and then on* ***Name in Descending order.***

```
> db.student.find().sort({Age:1,Name:-1})
{ "_id" : 4, "Name" : "Santosh", "Age" : 21,
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22,
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "
{ "_id" : 3, "Name" : "Nitesh", "Age" : 23,
{ "_id" : 5, "Name" : "Biplav", "Age" : 23,
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25,
```

Dealing with Count, Limit, Sort, and Skip

- ***Skip the first two documents*** from the student collection.

```
> db.student.find()
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073 }
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "Address" : "Halan Chowk" }
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur" }
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22, "Mobile" : 124356779, "Batch" : 2073, "Address" : "Bagar", "Hobbies"
{ "_id" : 3, "Name" : "Nitesh", "Age" : 23, "Mobile" : 893457879, "Batch" : 2073 }
> db.student.find().skip(2)
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur" }
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22, "Mobile" : 124356779, "Batch" : 2073, "Address" : "Bagar", "Hobbies"
{ "_id" : 3, "Name" : "Nitesh", "Age" : 23, "Mobile" : 893457879, "Batch" : 2073 }
```

- ***Sort*** *the documents from the student collection in the* ***ascending order of Name*** *and* ***skip the first Three*** *documents from the output.*

```
> db.student.find().skip(3).sort({Name:1})
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22, "Mobile" : 124356779, "Batch" : 2073, "Address" : "Bagar", "Ho
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073 }
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
>
```

Dealing with Count, Limit, Sort, and Skip

- ***Display the last Two records*** from the student collection.

```
> db.student.find()
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073 }
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "A
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22, "Mobile" : 124356779, "Batch" : 2073, "
{ "_id" : 3, "Name" : "Nitesh", "Age" : 23, "Mobile" : 893457879, "Batch" : 2073 }
> db.student.find().skip(db.student.count()-2)
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22, "Mobile" : 124356779, "Batch" : 2073, "
{ "_id" : 3, "Name" : "Nitesh", "Age" : 23, "Mobile" : 893457879, "Batch" : 2073 }
```

- ***Display the Fourth and Fifth document from the student collection.***

```
> db.student.find()
{ "_id" : 4, "Name" : "Santosh", "Age" : 21, "Mobile" : 987456779, "Batch" : 2073 }
{ "_id" : 5, "Name" : "Biplav", "Age" : 23, "Mobile" : 773456779, "Batch" : 2072, "Address" : "Halan Chowk" }
{ "_id" : 1, "Name" : "Dinesh", "Age" : 25, "Mobile" : 123456779, "Batch" : 2073, "Address" : "Lamachaur" }
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22, "Mobile" : 124356779, "Batch" : 2073, "Address" : "Bagar", "Hobbies
{ "_id" : 3, "Name" : "Nitesh", "Age" : 23, "Mobile" : 893457879, "Batch" : 2073 }
> db.student.find().skip(3).limit(2)
{ "_id" : 6, "Name" : "Shyam", "Age" : 23, "Mobile" : 123457879, "Batch" : 2072, "Address" : "New Road" }
{ "_id" : 2, "Name" : "Ramesh", "Age" : 22, "Mobile" : 124356779, "Batch" : 2073, "Address" : "Bagar", "Hobbies
```

Dealing with Arrays

- Create a collection ***"food" and*** then insert documents into ***"food".*** Each documents should have ***a "fruits " array.***

```
> db.food.insert({_id:2,fruits:['Orange','Mango','Grapes']})
WriteResult({ "nInserted" : 1 })
> db.food.insert({_id:3,fruits:['Pineapple','Strawberry','Grapes']})
WriteResult({ "nInserted" : 1 })
> db.food.insert({_id:4,fruits:['Orange','Grapes']})
WriteResult({ "nInserted" : 1 })
> db.food.find()
{ "_id" : 1, "fruits" : [ "Banana", "Apple", "Cherry" ] }
{ "_id" : 2, "fruits" : [ "Orange", "Mango", "Grapes" ] }
{ "_id" : 3, "fruits" : [ "Pineapple", "Strawberry", "Grapes" ] }
{ "_id" : 4, "fruits" : [ "Orange", "Grapes" ] }
```

- *Find those documents from "food" collection which has the **"fruit" array having "orange" as an element.***

```
db.food.find({fruits:'Orange'})
"_id" : 2, "fruits" : [ "Orange", "Mango", "Grapes" ] }
"_id" : 4, "fruits" : [ "Orange", "Grapes" ] }
```

Dealing with Arrays

- *Find those documents from "food" collection which has the **"fruit" array having "orange" in the ZERO index position.***

```
> db.food.find({'fruits.0':'Orange'})
{ "_id" : 2, "fruits" : [ "Orange", "Mango", "Grapes" ] }
{ "_id" : 4, "fruits" : [ "Orange", "Grapes" ] }
```

- *Find those documents from "food" collection where the **size of the array is two.***

```
> db.food.find({'fruits':{$size:2}})
{ "_id" : 4, "fruits" : [ "Orange", "Grapes" ] }
```

- *Find the documents with _id:1 from the food collection and display the first two elements from the array "fruits".*

```
> db.food.find({_id:1},{"fruits":{$slice:2}})
{ "_id" : 1, "fruits" : [ "Banana", "Apple" ] }
```

Dealing with Arrays

- *Find all documents from "food" collection which have **elements "orange" and "Grapes" in the array "fruits".***

```
> db.food.find({"fruits":{$all:["orange","Grapes"]}})
> db.food.find({"fruits":{$all:["Orange","Grapes"]}})
{ "_id" : 2, "fruits" : [ "Orange", "Mango", "Grapes" ] }
{ "_id" : 4, "fruits" : [ "Orange", "Grapes" ] }
```

- *Find the documents **with _id:1** from the food collection and display the **two elements from the array** "fruits", starting with the element **at 0th index position.***

```
> db.food.find({_id:1},{"fruits":{$slice:[0,1]}})
{ "_id" : 1, "fruits" : [ "Banana" ] }
> db.food.find({_id:1},{"fruits":{$slice:[0,2]}})
{ "_id" : 1, "fruits" : [ "Banana", "Apple" ] }
```

Dealing with Arrays

- *Update the document **with (_id:4)** and replace the element present in the **First index position** of the "**fruits" array with "apple".***

```
> db.food.update({_id:4},{$set:{'fruits.1':"apple"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.food.find()
{ "_id" : 1, "fruits" : [ "Banana", "Apple", "Cherry" ] }
{ "_id" : 2, "fruits" : [ "Orange", "Mango", "Grapes" ] }
{ "_id" : 3, "fruits" : [ "Pineapple", "Strawberry", "Grapes" ] }
{ "_id" : 4, "fruits" : [ "Orange", "apple" ] }
```

- *Update the document **with (_id:3)** and replace the element "**Grapes**" of the "**fruits" array with "Apple".***

```
> db.food.update({_id:3,'fruits':'Grapes'},{$set:{'fruits.$':"Apple"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.food.find()
{ "_id" : 1, "fruits" : [ "Banana", "Apple", "Cherry" ] }
{ "_id" : 2, "fruits" : [ "Orange", "Mango", "Grapes" ] }
{ "_id" : 3, "fruits" : [ "Pineapple", "Strawberry", "Apple" ] }
{ "_id" : 4, "fruits" : [ "Orange", "apple" ] }
```

Dealing with Arrays

- *Update the document **with (_id:2)** and **push new key value pairs** in the "fruits" array. **price:{orange:70, Grapes:300,mango:150}***

```
> db.food.update({_id:2},{$push:{price:{orange:70, Grapes:300,mango:150}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.food.find()
{ "_id" : 1, "fruits" : [ "Banana", "Apple", "Cherry" ] }
{ "_id" : 3, "fruits" : [ "Pineapple", "Strawberry", "Apple" ] }
{ "_id" : 4, "fruits" : [ "Orange", "apple" ] }
{ "_id" : 2, "fruits" : [ "Orange", "Mango", "Grapes" ], "price" : [ { "orange" : 70, "Grapes" : 300, "mango" : 150 } ] }
```

- *Update the documents **with "_id:4"** by **adding an element "Banana"** to the list of elements in the array "Fruit"*

```
> db.food.update({_id:4},{$addToSet:{fruits:"Banana"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.food.find({_id:4})
{ "_id" : 4, "fruits" : [ "Orange", "Grapes", "Banana" ] }
```

Dealing with Arrays

- *Update the documents with "_id:3" by poping an element from the list of elements present in the array "fruits". **The element popped is the one from the end of the array. Pop Function expects 1 or -1.***

```
> db.food.update({_id:3},{$pop:{fruits:1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.food.find({_id:3})
{ "_id" : 3, "fruits" : [ "Pineapple", "Strawberry" ] }
```

- *Update the documents with "_id:2" by poping an element from the list of elements present in the array "fruits". **The element popped is the one from the beginning of the array.***

```
> db.food.update({_id:2},{$pop:{fruits:-1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.food.find({_id:2})
{ "_id" : 2, "fruits" : [ "Mango", "Grapes" ], "price" : [ { "orange"
```

Dealing with Arrays

- *Update the documents **with "_id:4"** by **poping Two elements "Orange" and "Apple"** from the list of the elements present in the array "fruits".*

```
> db.food.find({_id:4})
{ "_id" : 4, "fruits" : [ "Orange", "Grapes", "Banana", "Apple" ] }
> db.food.update({_id:4},{$pullAll:{fruits:["Orange","Apple"]}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.food.find({_id:4})
{ "_id" : 4, "fruits" : [ "Grapes", "Banana" ] }
```

- *Update the documents **having "Banana" as an element** in the array "fruits" and **pop out the element "Banana" from the document**.*

```
> db.food.find()
{ "_id" : 1, "fruits" : [ "Apple", "Cherry" ] }
{ "_id" : 3, "fruits" : [ "Pineapple", "Strawberry" ] }
{ "_id" : 2, "fruits" : [ "Mango", "Grapes" ], "price" :
{ "_id" : 4, "fruits" : [ "Grapes", "Banana" ] }
> db.food.update({fruits:"Banana"},{$pull:{fruits:"Banana
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified
> db.food.find()
{ "_id" : 1, "fruits" : [ "Apple", "Cherry" ] }
{ "_id" : 3, "fruits" : [ "Pineapple", "Strawberry" ] }
{ "_id" : 2, "fruits" : [ "Mango", "Grapes" ], "price" :
{ "_id" : 4, "fruits" : [ "Grapes" ] }
```

Aggregate Function

- *Insert the Number of documents in Employee collection.*

```
> db.Employee.insert([
... {_id:1,ename:"Ramesh",Salary:50000,Post:"Coordinator",Department:"Computer"},
... {_id:2,ename:"Hari",Salary:45000,Post:"Coordinator",Department:"Computer"},
... {_id:3,ename:"Laxmi",Salary:55000,Post:"Manager",Department:"Electronics"},
... {_id:4,ename:"Padam",Salary:50000,Post:"Manager",Department:"Admin"},
... {_id:5,ename:"Dinesh",Salary:37000,Post:"Coordinator",Department:"Admin"},
... {_id:6,ename:"Dipesh",Salary:35000,Post:"Coordinator",Department:"Electronics"},
... {_id:7,ename:"Dipak",Salary:55000,Post:"Manager",Department:"Computer"},
... {_id:8,ename:"Laxman",Salary:45000,Post:"Manager",Department:"Admin"}
...  ]  )
```

```
> db.Employee.find()
{ "_id" : 1, "ename" : "Ramesh", "Salary" : 50000, "Post" : "Coordinator", "Department" : "Computer" }
{ "_id" : 2, "ename" : "Hari", "Salary" : 45000, "Post" : "Coordinator", "Department" : "Computer" }
{ "_id" : 3, "ename" : "Laxmi", "Salary" : 55000, "Post" : "Manager", "Department" : "Electronics" }
{ "_id" : 4, "ename" : "Padam", "Salary" : 50000, "Post" : "Manager", "Department" : "Admin" }
{ "_id" : 5, "ename" : "Dinesh", "Salary" : 37000, "Post" : "Coordinator", "Department" : "Admin" }
{ "_id" : 6, "ename" : "Dipesh", "Salary" : 35000, "Post" : "Coordinator", "Department" : "Electronics" }
{ "_id" : 7, "ename" : "Dipak", "Salary" : 55000, "Post" : "Manager", "Department" : "Computer" }
{ "_id" : 8, "ename" : "Laxman", "Salary" : 45000, "Post" : "Manager", "Department" : "Admin" }
```

Aggregate Function

Different expressions used by Aggregate function

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

Aggregate Function

- *List **all Employee name** with their salary **who works in Computer Department**.*

```
> db.Employee.aggregate([{$match:{Department:"Computer"}},{$project:{ename:1,Salary:1,_id:0}}])
{ "ename" : "Ramesh", "Salary" : 50000 }
{ "ename" : "Hari", "Salary" : 45000 }
{ "ename" : "Dipak", "Salary" : 55000 }
```

- *List **all Employee name, Post and Department** who works in **Admin Department** and Post is Manager.*

```
> db.Employee.aggregate([{$match:{Department:"Admin",Post:"Manager"}},{$project:{ename:1,Department:1,Post:1,_id:0}}])
{ "ename" : "Padam", "Post" : "Manager", "Department" : "Admin" }
{ "ename" : "Laxman", "Post" : "Manager", "Department" : "Admin" }
```

Aggregate Function

- *Find the **Minimum salary** of Each **Department**.*

```
> db.Employee.aggregate([{$group:{_id:"$Department",Minimum:{$min:"$Salary"}}}])
{ "_id" : "Electronics", "Minimum" : 35000 }
{ "_id" : "Admin", "Minimum" : 37000 }
{ "_id" : "Computer", "Minimum" : 45000 }
```

- *Find the **Total salary** of Each **Department**.*

```
> db.Employee.aggregate([{$group:{_id:"$Department",totalSalary:{$sum:"$Salary"}}}])
{ "_id" : "Electronics", "totalSalary" : 90000 }
{ "_id" : "Admin", "totalSalary" : 132000 }
{ "_id" : "Computer", "totalSalary" : 150000 }
```

- *Find the total salary of Manger department wise.*

```
> db.Employee.aggregate([{$match:{Post:"Manager"}},{$group:{_id:"$Department",Total_Salary:{$sum:"$Salary"}}}])
{ "_id" : "Computer", "Total_Salary" : 55000 }
{ "_id" : "Admin", "Total_Salary" : 95000 }
{ "_id" : "Electronics", "Total_Salary" : 55000 }
```

Aggregate Function

- *Find the total **salary of Manger department wise**. And then display the Department that have the total **salary greater than 40000**.*

```
> db.Employee.aggregate([{$match:{Post:"Manager"}},{$group:{_id:"$Department",
Total_Salary:{$sum:"$Salary"}}},{$match:{Total_Salary:{$gt:40000}}}])
{ "_id" : "Computer", "Total_Salary" : 55000 }
{ "_id" : "Admin", "Total_Salary" : 95000 }
{ "_id" : "Electronics", "Total_Salary" : 55000 }
```

- *Calculate the Annual salary of each Employee.*

```
> db.Employee.aggregate([{$project:{ename:1,_id:0,Annual_Salary:{$multiply:["$Salary",12]}}}])
{ "ename" : "Ramesh", "Annual_Salary" : 600000 }
{ "ename" : "Hari", "Annual_Salary" : 540000 }
{ "ename" : "Laxmi", "Annual_Salary" : 660000 }
{ "ename" : "Padam", "Annual_Salary" : 600000 }
{ "ename" : "Dinesh", "Annual_Salary" : 444000 }
{ "ename" : "Dipesh", "Annual_Salary" : 420000 }
{ "ename" : "Dipak", "Annual_Salary" : 660000 }
{ "ename" : "Laxman", "Annual_Salary" : 540000 }
```

Nested /Embedded Documents

```
db.Record.insert([
     {    "_id" : 1,
          "name" : "Class 1",
          "students" : [
               { "rollNo" : 10001, "name" : "Ram", "score" : 65 },
               { "rollNo" : 10002, "name" : "Shyam", "score" : 90 },
               { "rollNo" : 10003, "name" : "Mohan", "score" : 75 }
               ]
     },
    {
          "_id" : 2,
          "name" : "Class 2",
          "students" : [
               { "rollNo" : 20001, "name" : "Krishna", "score" : 88 },
               { "rollNo" : 20002, "name" : "Sohan", "score" : 91 },
               { "rollNo" : 20003, "name" : "Radhika", "score" : 82 },
               { "rollNo" : 20004, "name" : "Komal", "score" : 55 }
               ]
     },
     {
          "_id" : 3,
          "name" : "Class 3",
          "students" : [
               { "rollNo" : 30001, "name" : "Monika", "score" : 77 },
               { "rollNo" : 30002, "name" : "Rahul", "score" : 81 }
               ]
     }
])
```

Nested /Embedded Documents

- *Select students records only*

```
> db.Record.find({}, {_id:0, students: 1})
```

- Get list of student data (rollNo, name and score)

```
> db.Record.aggregate([
... { $unwind :'$students'},
... { $project : { _id:0, rollNo : '$students.rollNo', name : '$students.name', score : '$students.score' } }
... ])
{ "rollNo" : 10001, "name" : "Ram", "score" : 65 }
{ "rollNo" : 10002, "name" : "Shyam", "score" : 90 }
{ "rollNo" : 10003, "name" : "Mohan", "score" : 75 }
{ "rollNo" : 20001, "name" : "Krishna", "score" : 88 }
{ "rollNo" : 20002, "name" : "Sohan", "score" : 91 }
{ "rollNo" : 20003, "name" : "Radhika", "score" : 82 }
{ "rollNo" : 20004, "name" : "Komal", "score" : 55 }
{ "rollNo" : 30001, "name" : "Monika", "score" : 77 }
{ "rollNo" : 30002, "name" : "Rahul", "score" : 81 }
```

Nested /Embedded Documents

- **$unwind** deconstructs an array field from the input documents to output a document for each element. Each output document is the input document with the value of the array field replaced by the element.

```
> db.Record.aggregate([
... { $unwind :'$students'}
... ])
{ "_id" : 1, "name" : "Class 1", "students" : { "rollNo" : 10001, "name" : "Ram", "score" : 65 } }
{ "_id" : 1, "name" : "Class 1", "students" : { "rollNo" : 10002, "name" : "Shyam", "score" : 90 } }
{ "_id" : 1, "name" : "Class 1", "students" : { "rollNo" : 10003, "name" : "Mohan", "score" : 75 } }
{ "_id" : 2, "name" : "Class 2", "students" : { "rollNo" : 20001, "name" : "Krishna", "score" : 88 } }
{ "_id" : 2, "name" : "Class 2", "students" : { "rollNo" : 20002, "name" : "Sohan", "score" : 91 } }
{ "_id" : 2, "name" : "Class 2", "students" : { "rollNo" : 20003, "name" : "Radhika", "score" : 82 } }
{ "_id" : 2, "name" : "Class 2", "students" : { "rollNo" : 20004, "name" : "Komal", "score" : 55 } }
{ "_id" : 3, "name" : "Class 3", "students" : { "rollNo" : 30001, "name" : "Monika", "score" : 77 } }
{ "_id" : 3, "name" : "Class 3", "students" : { "rollNo" : 30002, "name" : "Rahul", "score" : 81 } }
```

Nested /Embedded Documents

- Get list of students having **score > 80.**

```
> db.Record.aggregate([
... { $unwind :'$students'},
... { $match : {'students.score': { $gt : 80} }},
... { $project : { _id:0, rollNo : '$students.rollNo', name : '$students.name', score : '$students.score' } }
... ])
{ "rollNo" : 10002, "name" : "Shyam", "score" : 90 }
{ "rollNo" : 20001, "name" : "Krishna", "score" : 88 }
{ "rollNo" : 20002, "name" : "Sohan", "score" : 91 }
{ "rollNo" : 20003, "name" : "Radhika", "score" : 82 }
{ "rollNo" : 30002, "name" : "Rahul", "score" : 81 }
```

- Get Roll number of students whose name is Monika.

```
> db.Record.aggregate([
... { $unwind :'$students'},
... { $match : {'students.name':"Monika"}},
... { $project : { _id:0, rollNo : '$students.rollNo', name : '$students.name'} }
... ])
{ "rollNo" : 30001, "name" : "Monika" }
```

Nested /Embedded Documents

- Get Roll number, score and Name of students whose name either Monika or Ram

```
> db.Record.aggregate([
... { $unwind :'$students'},
... { $match : {$or:[{'students.name':"Monika"},{'students.name':"Ram"}]}},
... { $project : { _id:0, rollNo : '$students.rollNo', name : '$students.name',score : '$students.score'} }
... ])
{ "rollNo" : 10001, "name" : "Ram", "score" : 65 }
{ "rollNo" : 30001, "name" : "Monika", "score" : 77 }
```

Thank You
???