WORKING WITH MONGODB

I. CREATE DATABASE IN MONGODB.

use myDB;

Confirm the existence of your database

```
test>
>>> use myDB;
switched to db myDB
myDB>
>>>
```

db;

To list all databases

show dbs;

```
>>> show dbs;
admin    102 kB
config  12.3 kB
local   73.7 kB
myDB>
>>>
```

II. CRUD (CREATE, READ, UPDATE, DELETE) OPERATIONS

1. To create a collection by the name “Student”. Let us take a look at the collection list prior to the creation of the new collection “Student”.

db.createCollection(“Student”); **=>** *sql equivalent* **CREATE TABLE STUDENT(…);**

```
>>> db.createCollection("Student");
{ ok: 1 }
myDB>
>>>
```

2. To drop a collection by the name “Student”.

db.Student.drop();

3. Create a collection by the name “Students” and store the following data in it.

db.Student.insert({_id:1,StudName:"MichelleJacintha",Grade:"VII",Hobbies:"InternetSurfing"});

```
>>> db.Student.insertOne({ _id : 1, StudentName : "Bruce Wayne", Grade :
"7" , Hobbies : "Training"});
{ acknowledged: true, insertedId: 1 }
```

4. Insert the document for “AryanDavid” in to the Students 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 his Hobbies from “Skating” to “Chess”.) Use “Update else insert” (if there is an existing document, it will attempt to update it, if there is no existing document then it will insert it).

db.Student.update({_id:3,StudName:"AryanDavid",Grade:"VII"},{$set:{Hobbies:"Skating"}},{upsert:true});

1.
```
>>> db.Student.find();
[
  {
    _id: 1,
    StudentName: 'Bruce Wayne',
    Grade: '7',
    Hobbies: 'Training'
  },
  { _id: 2, StudentName: 'Clark Kent', Grade: '7', Hobbies: 'Skating' }
]
```

2.
```
>>> db.Student.updateOne({_id : 2, StudentName : "Clark Kent", Grade :
"7"},{$set : {Hobbies : "Chess"}},{upset : true});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

3.
```
>>> db.Student.find();
[
  {
    _id: 1,
    StudentName: 'Bruce Wayne',
    Grade: '7',
    Hobbies: 'Training'
  },
  { _id: 2, StudentName: 'Clark Kent', Grade: '7', Hobbies: 'Chess' }
]
```

1.

5. FIND METHOD

A. To search for documents from the “Students” collection based on certain search criteria.

db.Student.find({StudName:"Aryan David"});
({cond..},{columns.. column:1, columnname:0})

```
myDB>
>>> db.Student.find({StudentName : "Bruce Wayne"});
[
  {
    _id: 1,
    StudentName: 'Bruce Wayne',
    Grade: '7',
    Hobbies: 'Training'
  }
]
```

B. To display only the StudName and Grade from all the documents of the Students collection. The identifier_id should be suppressed and NOT displayed.

db.Student.find({},{StudName:1,Grade:1,_id:0});

```
myDB>
>>> db.Student.find({},{StudentName : 1, Grade : 1, _id :0});
[
  { StudentName: 'Bruce Wayne', Grade: '7' },
  { StudentName: 'Clark Kent', Grade: '7' }
]
myDB>
```

C. To find those documents where the Grade is set to ‘VII’

db.Student.find({Grade:{$eq:'VII'}}).pretty();

```
myDB>
>>> db.Student.find({Grade : {$eq : "7"}});
[
  {
    _id: 1,
    StudentName: 'Bruce Wayne',
    Grade: '7',
    Hobbies: 'Training'
  },
  { _id: 2, StudentName: 'Clark Kent', Grade: '7', Hobbies: 'Chess' }
]
myDB>
```

D. To find those documents from the Students collection where the Hobbies is set to either ‘Chess’ or is set to ‘Skating’.

db.Student.find({Hobbies :{ $in: ['Chess','Skating']}}).pretty ();

```
myDB>
>>> db.Student.find({Hobbies : {$in : ["Chess","Skating"] }});
[ { _id: 2, StudentName: 'Clark Kent', Grade: '7', Hobbies: 'Chess' } ]
myDB>
```

E. To find documents from the Students collection where the StudName begins with “M”.

db.Student.find({StudName:/^M/}).pretty();

```
myDB>
>>> db.Student.find({StudentName: /^B/});
[
  {
    _id: 1,
    StudentName: 'Bruce Wayne',
    Grade: '7',
    Hobbies: 'Training'
  }
]
```

F. To find documents from the Students collection where the StudName has an “e” in any position.

db.Student.find({StudName:/e/}).pretty();

```
myDB>
>>> db.Student.find({StudentName: /e/});
[
  {
    _id: 1,
    StudentName: 'Bruce Wayne',
    Grade: '7',
    Hobbies: 'Training'
  },
  { _id: 2, StudentName: 'Clark Kent', Grade: '7', Hobbies: 'Chess' }
]
myDB>
```

G. To find the number of documents in the Students collection.

db.Student.count();

```
myDB>
>>> db.Student.countDocuments();
2
myDB>
```

H. To sort the documents from the Students collection in the descending order of StudName.

db.Student.find().sort({StudName:-1}).pretty();

```
myDB>
>>> db.Student.find().sort({StudentName: -1});
[
  { _id: 2, StudentName: 'Clark Kent', Grade: '7', Hobbies: 'Chess' },
  {
    _id: 1,
    StudentName: 'Bruce Wayne',
    Grade: '7',
    Hobbies: 'Training'
  }
]
myDB>
```

III. Import data from a CSV file

Given a CSV file “sample.txt” in the D:drive, import the file into the MongoDB collection, “SampleJSON”. The collection is in the database “test”.

mongoimport --db Student --collection airlines --type csv –headerline --file /home/hduser/Desktop/airline.csv

IV. Export data to a CSV file

This command used at the command prompt exports MongoDB JSON documents from "Customers" collection in the "test" database into a CSV file "Output.txt" in the D:drive.

mongoexport --host localhost --db Student --collection airlines --csv --out /home/hduser/Desktop/output.txt –fields "Year","Quarter"

V. Save Method :

Save() method will insert a new document, if the document with the _id does not exist. If it exists it will replace the exisiting document.

db.Students.save({StudName:"Vamsi", Grade:"VI"})

VI. Add a new field to existing Document:

db.Students.update({_id:4},{$set:{Location:"Network"}})

```
myDB>
>>> db.Student.update({_id : 1},{$set : {Location : "Gotham City"}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
myDB>
>>> db.Student.find({_id:{$eq: 1}});
[
  {
    _id: 1,
    StudentName: 'Bruce Wayne',
    Grade: '7',
    Hobbies: 'Training',
    Location: 'Gotham City'
  }
]
myDB>
```

VII. Remove the field in an existing Document

db.Students.update({_id:4},{$unset:{Location:"Network"}})

```
myDB>
>>> db.Student.update({_id : 1},{$unset : {Location : "Gotham City"}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
myDB>
```

VIII. Finding Document based on search criteria suppressing few fields

db.Student.find({_id:1},{StudName:1,Grade:1,_id:0});

```
myDB>
>>> db.Student.find({_id : 1}, {StudentName : 1, Grade : 1, _id : 0});
[ { StudentName: 'Bruce Wayne', Grade: '7' } ]
myDB>
```

To find those documents where the Grade is not set to 'VII'

db.Student.find({Grade:{$ne:'VII'}}).pretty();

```
>>> db.Student.find({Grade : {$ne : "7"}});
[
  {
    _id: ObjectId("6277c3e0bd8f013c5c3f84d1"),
    StudentName: 'Diana Prince',
    Grade: '8'
  }
]
myDB>
```

To find documents from the Students collection where the StudName ends with s.

db.Student.find({StudName:/s$/}).pretty();

```
myDB>
>>> db.Student.find({StudentName: /e$/});
[
  {
    _id: 1,
    StudentName: 'Bruce Wayne',
    Grade: '7',
    Hobbies: 'Training'
  },
  {
    _id: ObjectId("6277c3e0bd8f013c5c3f84d1"),
    StudentName: 'Diana Prince',
    Grade: '8'
  }
]
myDB>
```

IX. to set a particular field value to NULL

db.Students.update({_id:3},{$set:{Location:null}})

```
>>> db.Student.updateOne({_id : 1}, {$set : {Location : null}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
myDB>
>>> db.Student.find();
[
  {
    _id: 1,
    StudentName: 'Bruce Wayne',
    Grade: '7',
    Hobbies: 'Training',
    Location: null
  },
```

X. Count the number of documents in Student Collections

db.Students.count()

```
>>> db.Student.count() ;
3
```

XI. Count the number of documents in Student Collections with grade :VII

db.Students.count({Grade:"VII"})

```
myDB>
>>> db.Student.count({Grade:"7"});
1
```

retrieve first 3 documents

db.Students.find({Grade:"VII"}).limit(3).pretty();

```
>>> db.Student.find({Grade:"7"}).limit(3);
[ { _id: 1, StudentName: 'Bruce Wayne', Grade: '7' } ]
myDB>
```

Sort the document in Ascending order

db.Students.find().sort({StudName:1}).pretty();

```
myDB>
>>> db.Student.find().sort({StudentName:1});
[
  { _id: 1, StudentName: 'Bruce Wayne', Grade: '7' },
  { _id: 2, StudentName: 'Clark Kent', Grade: '9' },
  { _id: 3, StudentName: 'Diana Prince', Grade: '10' }
]
myDB>
```

Note:
for desending order : db.Students.find().sort({StudName:-1}).pretty();

```
myDB>
>>> db.Student.find().sort({StudentName:-1});
[
  { _id: 3, StudentName: 'Diana Prince', Grade: '10' },
  { _id: 2, StudentName: 'Clark Kent', Grade: '9' },
  { _id: 1, StudentName: 'Bruce Wayne', Grade: '7' }
]
```

to Skip the 1st two documents from the Students Collections

db.Students.find().skip(2).pretty()

```
>>> db.Student.find().skip(2);
[ { _id: 3, StudentName: 'Diana Prince', Grade: '10' } ]
myDB>
```

XII. Create a collection by name “food” and add to each document add a “fruits” array

db.food.insert({ _id:1, fruits:['grapes','mango','apple'] })
db.food.insert({ _id:2, fruits:['grapes','mango','cherry'] })
db.food.insert({ _id:3, fruits:['banana','mango'] })

```
>>> db.createCollection("food");
{ ok: 1 }
test>
>>> db.food.insertOne({_id : 1, fruits : ["Apple","Mango","Jack
Fruit"]});
{ acknowledged: true, insertedId: 1 }
test>
>>> db.food.insertOne({_id : 2, fruits : ["Cherry","Orange","Butter
Fruit"]});
{ acknowledged: true, insertedId: 2 }
test>
>>> db.food.insertOne({_id : 3, fruits : ["Banana","Water Melon"]});
{ acknowledged: true, insertedId: 3 }
test>
>>>
```

To find those documents from the “food” collection which has the “fruits array” constitute of “grapes”, “mango” and “apple”.

db.food.find ({fruits: ['grapes','mango','apple'] }). pretty().

```
test>
>>> db.food.find({fruits:["Banana","Water Melon"]});
[ { _id: 3, fruits: [ 'Banana', 'Water Melon' ] } ]
test>
>>>
```

To find in “fruits” array having “mango” in the first index position.

db.food.find ({'fruits.1':'grapes'})

```
test>
>>> db.food.find({ 'fruits.0' : 'Banana'});
[ { _id: 3, fruits: [ 'Banana', 'Water Melon' ] } ]
test>
>>>
```

To find those documents from the "food" collection where the size of the array is two.

db.food.find ({"fruits": {$size:2}})

To find the document with a particular id and display the first two elements from the array "fruits"

db.food.find({_id:1},{"fruits":{$slice:2}})

```
test>
>>> db.food.find({ 'fruits' : {$size : 2}});
[ { _id: 3, fruits: [ 'Banana', 'Water Melon' ] } ]
test>
>>>
```

To find all the documets from the food collection which have elements mango and grapes in the array "fruits"

db.food.find({fruits:{$all:["mango","grapes"]}})

```
test>
>>> db.food.find({fruits:{$all:["Cherry","Orange"]}}) ;
[ { _id: 2, fruits: [ 'Cherry', 'Orange', 'Butter Fruit' ] } ]
test>
>>>
```

update on Array:
using particular id replace the element present in the 1st index position of the fruits array with apple

db.food.update({_id:3},{$set:{'fruits.1':'apple'}})

```
test>
>>> db.food.update({_id : 3}, {$set : {"fruits.1" : "Green Apple"}});
DeprecationWarning: Collection.update() is deprecated. Use updateOne,
updateMany, or bulkWrite.
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
test>
```

insert new key value pairs in the fruits array

db.food.update({_id:2},{$push:{price:{grapes:80,mango:200,cherry:100}}})

```
test>
>>> db.food.update({_id : 3}, {$push : {price : {Banana : 20, GreenApple
: 200}}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
test>
>>> db.food.find();
[
  { _id: 1, fruits: [ 'Apple', 'Mango', 'Jack Fruit' ] },
  { _id: 2, fruits: [ 'Cherry', 'Orange', 'Butter Fruit' ] },
  {
    _id: 3,
    fruits: [ 'Banana', 'Green Apple' ],
    price: [ {}, { Banana: 20, GreenApple: 200 } ]
  }
]
test>
```

Note: perform query operations using - pop, addToSet, pullAll and pull

XII. Aggregate Function :

Create a collection Customers with fields custID, AcctBal, AcctType.
Now group on "custID" and compute the sum of "AccBal".

```
>>> db.Customer.find();
{ "_id" : ObjectId("629449502b957d283eee6404"), "CustId" : 1, "AcctBal" : 1000, "AcctType" : "Savings" }
{ "_id" : ObjectId("629449872b957d283eee6405"), "CustId" : 1, "AcctBal" : 2000, "AcctType" : "Current" }
{ "_id" : ObjectId("6294499e2b957d283eee6406"), "CustId" : 2, "AcctBal" : 50000, "AcctType" : "Current" }
{ "_id" : ObjectId("629449d12b957d283eee6407"), "CustId" : 2, "AcctBal" : 5000, "AcctType" : "Savings" }
>>>
```

db.Customers.aggregate ({$group : { _id : "$custID",TotAccBal : {$sum:"$AccBal"} } });

```
>>> db.Customer.aggregate({$group : { _id : "$CustId", TotalAccBal :
{$sum : "$AcctBal"}}});
{ "_id" : 2, "TotalAccBal" : 55000 }
{ "_id" : 1, "TotalAccBal" : 3000 }
>>>
```

match on AcctType:"S" then group on "CustID" and compute the sum of "AccBal".

db.Customers.aggregate ({$match:{AcctType:"S"}},{$group : { _id : "$custID",TotAccBal : {$sum:"$AccBal"} } });

```
>>> db.Customer.aggregate( {$match:{AcctType:"Savings"}},{$group : { _id
: "$custID",TotalAccBal : {$sum:"$AcctBal"}}});
{ "_id" : null, "TotalAccBal" : 6000 }
```

match on AcctType:"S" then group on "CustID" and compute the sum of "AccBal" and total balance greater than 1200.

db.Customers.aggregate ({$match:{AcctType:"S"}},{$group : { _id : "$custID",TotAccBal : {$sum:"$AccBal"} } }, {$match:{TotAccBal:{$gt:1200}}});

```
>>> db.Customer.aggregate( {$match:{AcctType:"Savings"}},{$group : { _id
: "$custID",TotalAccBal : {$sum:"$AcctBal"}}},{$match:{TotalAccBal:
{$gt:1200}}});
{ "_id" : null, "TotalAccBal" : 6000 }
>>>
```