St. Francis Institute of Technology, Mumbai-400103

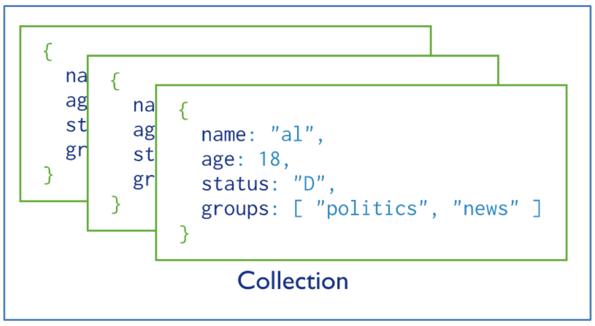
Class: BE-CMPN-A; Semester: VII; A.Y.: 2019-2020
Subject: BIG DATA ANALYTICS
Experiment No.: 03

AIM: Implementation of NoSQL (MongoDB) commands.

THEORY:

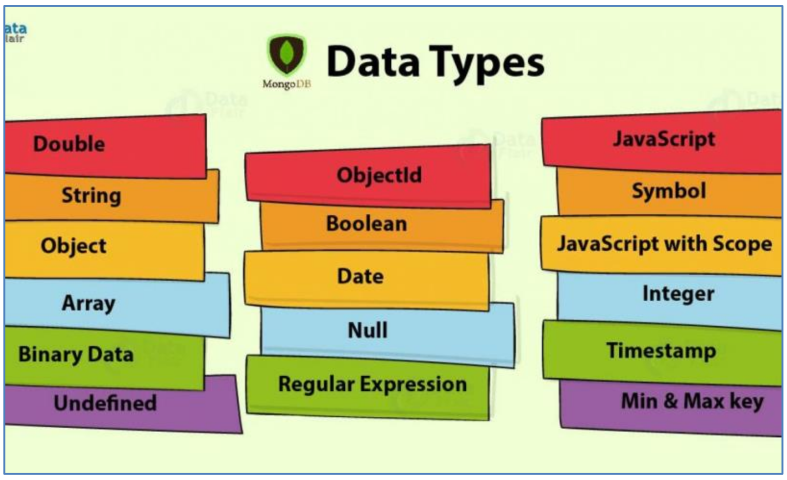
1. What is MongoDB?
   Ans: MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. The data objects are stored as separate documents inside a collection — instead of storing the data into the columns and rows of a traditional relational database
2. Why MongoDB?
   Ans: The document data model is a powerful way to store and retrieve data that allows developers to move fast. MongoDB's horizontal, scale-out architecture can support huge volumes of both data and traffic.
   MongoDB enables such iteration. More than any other NoSQL database, and dramatically more than any relational database, MongoDB's document-oriented data model makes it exceptionally easy to add or change fields, among other things. Because MongoDB enables profound developer agility through its flexible data model.
3. What is a database in MongoDB?
   Ans: MongoDB stores data records as documents (specifically BSON documents) which are gathered together in collections. A database stores one or more collections of documents. MongoDB stores documents in collections. Collections are analogous to tables in relational databases.

4. What are collections in MongoDB?
   Ans: MongoDB stores documents in collections. Collections are analogous to tables in relational databases.
   If a collection does not exist, MongoDB creates the collection when you first store data for that collection.
   MongoDB provides the db.createCollection() method to explicitly create a collection with various options, such as setting the maximum size or the documentation validation rules. If you are not specifying these options, you do not need to explicitly create the collection since MongoDB creates new collections when you first store data for the collections.



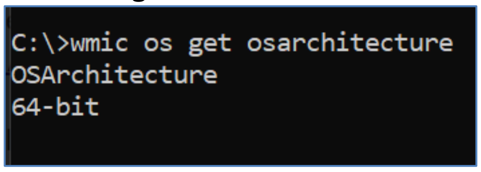
5. What are the different datatypes in MongoDB?
   Ans: MongoDB has 16 various data types out of which few a listed below

- Double: The double data type is used to store the floating-point values. ...
- Boolean: The boolean data type is used to store either true or false. ...
- Null: The null data type is used to store the null value. ...
- Array: The Array is the set of values. ...
- Object: Object data type stores embedded documents.
- Symbol: This data type similar to the string data type
- JavaScript: This datatype is used to store JavaScript code into the document without the scope.
- JavaScript with Scope: This MongoDB data type store JavaScript data with a scope.
- Binary Data: This datatype is used to store binary data.
- Undefined: This data type stores the undefined values.



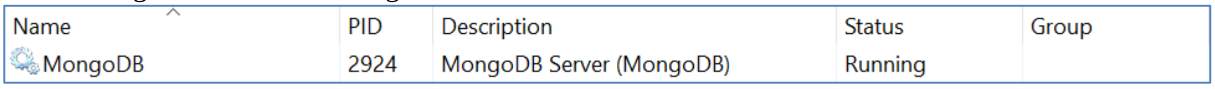
OUTPUT:

1. To get your windows version –
   `wmic os get osarchitecture`



```
C:\>wmic os get osarchitecture
OSArchitecture
64-bit
```

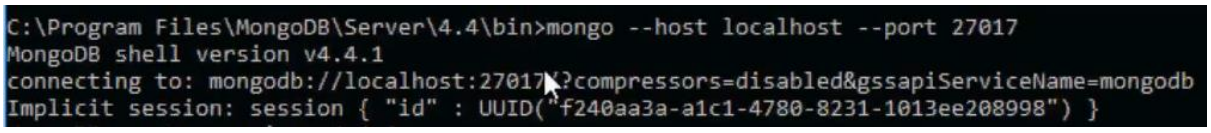
2. Connecting to Server –
   c:\Program Files\MongoDB\Server\5.0\bin –dppath c:\Program Files\MongoDB\Server\5.0\data
   Or
   Task Manager-> Services->MongoDB->Start Server



Name	PID	Description	Status	Group
MongoDB	2924	MongoDB Server (MongoDB)	Running	

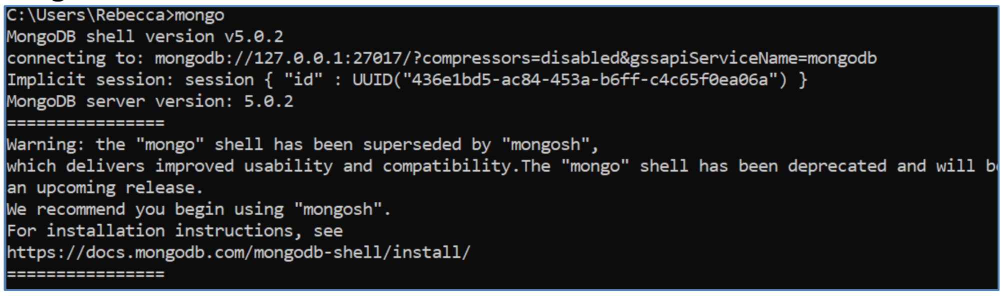
In MongoDB Compass
`mongodb://localhost`

```
Mongo -host localhost -port 27017
```



```
C:\Program Files\MongoDB\Server\4.4\bin>mongo --host localhost --port 27017
MongoDB shell version v4.4.1
connecting to: mongodb://localhost:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("f240aa3a-a1c1-4780-8231-1013ee208998") }
```

3. Connecting to Client –
   c:\mongodb\bin\mongo.exe
   mongo

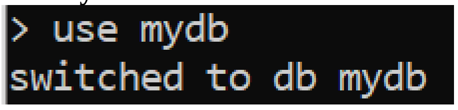


```
C:\Users\Rebecca>mongo
MongoDB shell version v5.0.2
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("436e1bd5-ac84-453a-b6ff-c4c65f0ea06a") }
MongoDB server version: 5.0.2
================
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will b
an upcoming release.
We recommend you begin using "mongosh".
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
================
```

Database creation commands

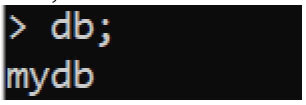
4. Create database –

   > **use mydb**



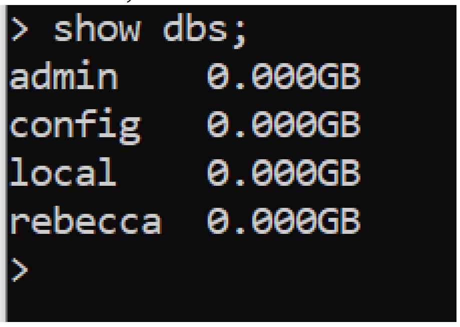
```
> use mydb
switched to db mydb
```

5. Confirm the existence of your database
   >db;



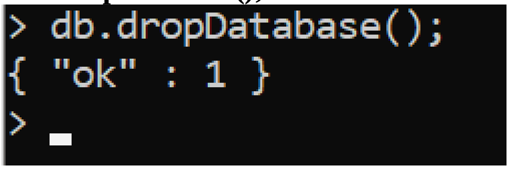
```
> db;
mydb
```

6. To get a list of all databases
   >show dbs;



```
> show dbs;
admin     0.000GB
config    0.000GB
local     0.000GB
rebecca   0.000GB
>
```

7. To drop a database
   >**use mydb;**
   >db.dropDatabase();

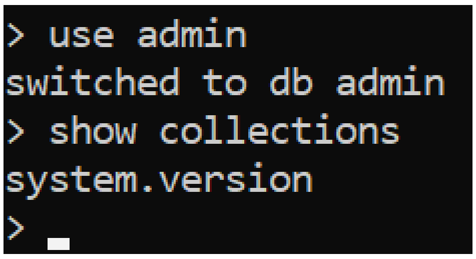


```
> db.dropDatabase();
{ "ok" : 1 }
>
```

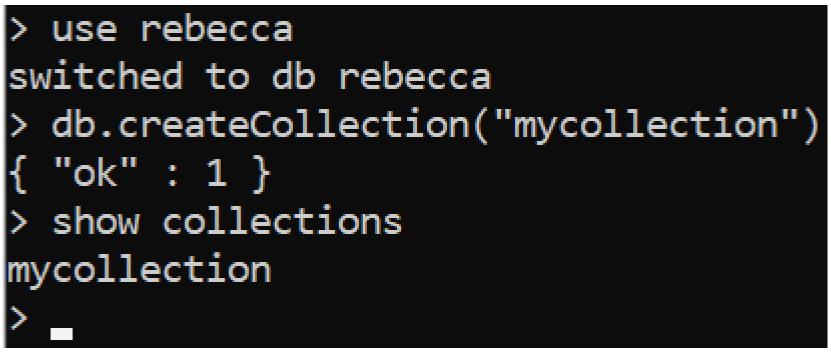
Creation of collections

8. To display the list of collections
   > Create collection
   > db.createCollection(“myCollection”)

>**show collections**



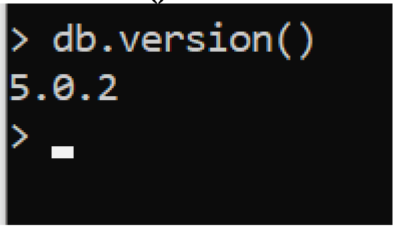
```
> use admin
switched to db admin
> show collections
system.version
>
```



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

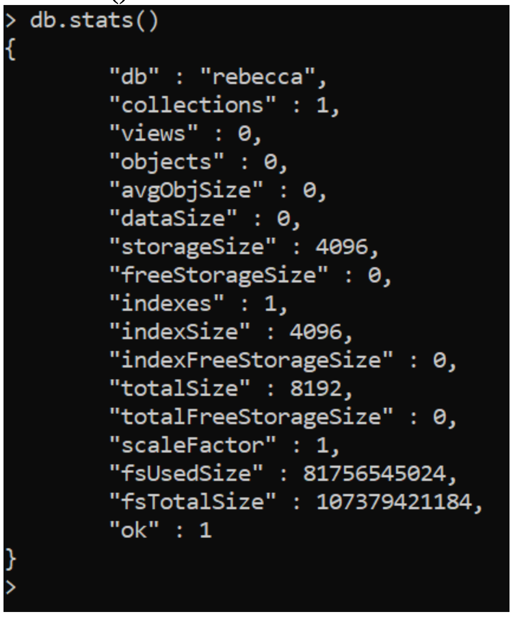
Other commands

9. To display the current version of MongoDB Server
>db.version()



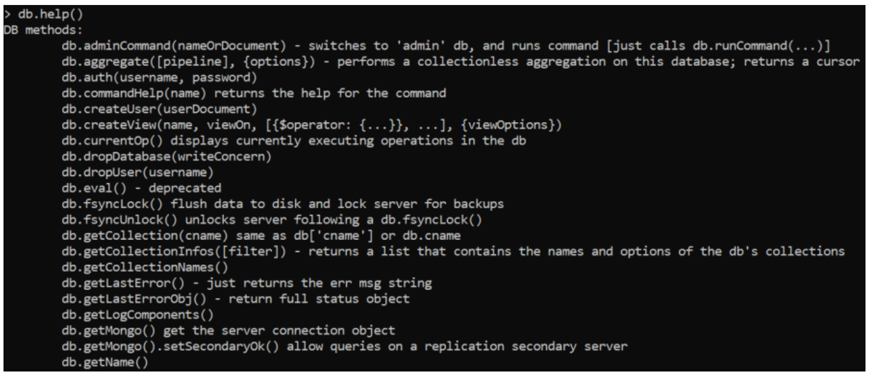
```
> db.version()
5.0.2
>
```

10. To display the statistics that reflect the use state of the database
>db.stats()



```
> db.stats()
{
        "db" : "rebecca",
        "collections" : 1,
        "views" : 0,
        "objects" : 0,
        "avgObjSize" : 0,
        "dataSize" : 0,
        "storageSize" : 4096,
        "freeStorageSize" : 0,
        "indexes" : 1,
        "indexSize" : 4096,
        "indexFreeStorageSize" : 0,
        "totalSize" : 8192,
        "totalFreeStorageSize" : 0,
        "scaleFactor" : 1,
        "fsUsedSize" : 81756545024,
        "fsTotalSize" : 107379421184,
        "ok" : 1
}
>
```

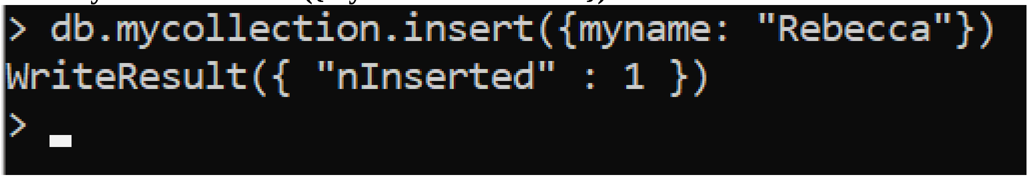
11. To display the list of commands
>db.help()



```
> 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()
```

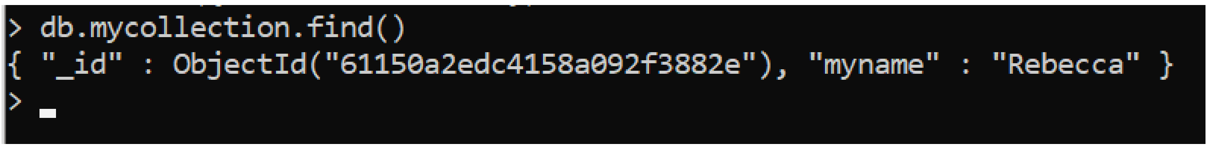
Manipulation commands

12. <u>Insert –</u>
    > **db.mycollection.insert({myname: "Rebecca"})**



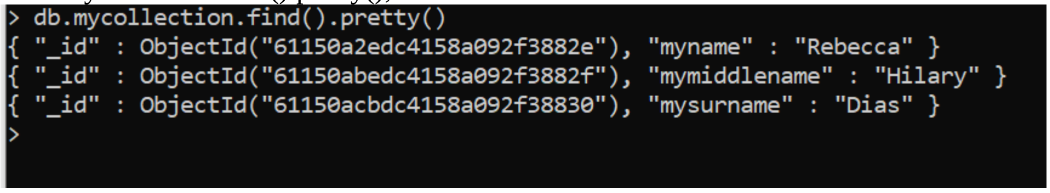
```
> db.mycollection.insert({myname: "Rebecca"})
WriteResult({ "nInserted" : 1 })
> 
```

13. To check if the document is successfully inserted. Use find() method
    > db.mycollection.find();



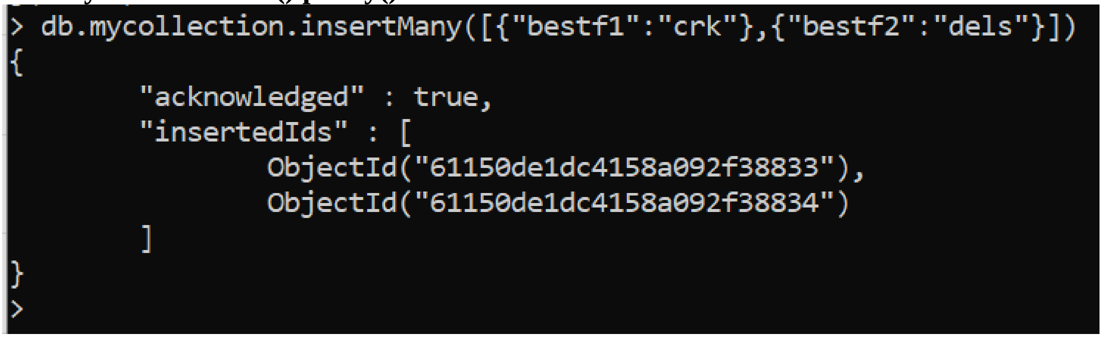
```
> db.mycollection.find()
{ "_id" : ObjectId("61150a2edc4158a092f3882e"), "myname" : "Rebecca" }
> 
```

14. To check if the document is successfully inserted and displayed in an organized manner.
    > db.mycollection.find().pretty();

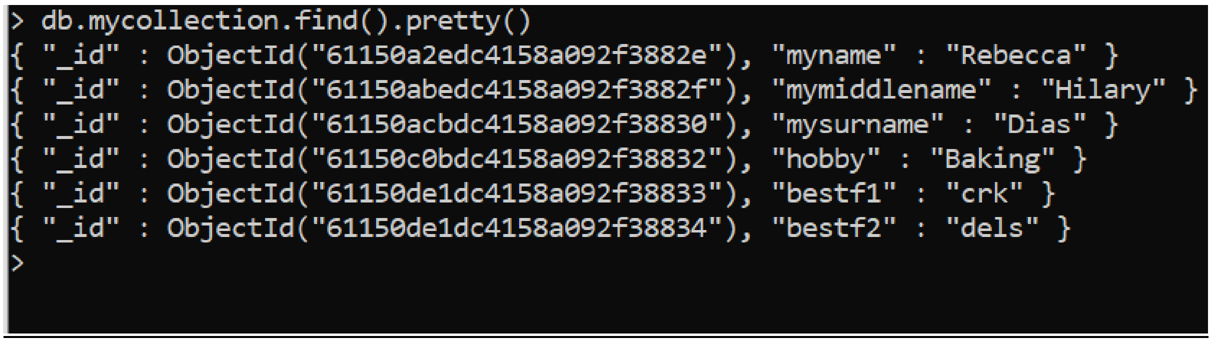


```
> db.mycollection.find().pretty()
{ "_id" : ObjectId("61150a2edc4158a092f3882e"), "myname" : "Rebecca" }
{ "_id" : ObjectId("61150abedc4158a092f3882f"), "mymiddlename" : "Hilary" }
{ "_id" : ObjectId("61150acbdc4158a092f38830"), "mysurname" : "Dias" }
>
```

15. Add multiple records
    >db.mycollection.insertMany([{"bestf1":"crk"},{"bestf2":"dels"}])
    >db.mycollection.find().pretty()



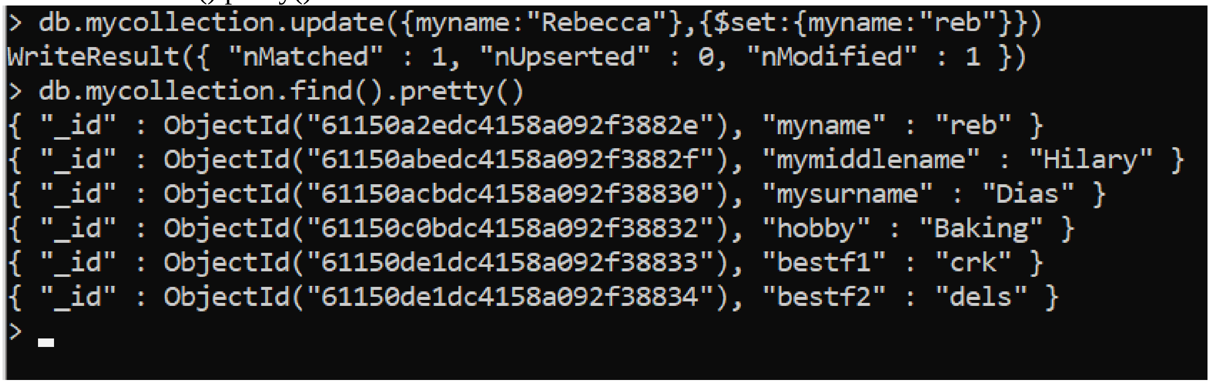
```
> db.mycollection.insertMany([{"bestf1":"crk"},{"bestf2":"dels"}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("61150de1dc4158a092f38833"),
                ObjectId("61150de1dc4158a092f38834")
        ]
}
>
```



```
> db.mycollection.find().pretty()
{ "_id" : ObjectId("61150a2edc4158a092f3882e"), "myname" : "Rebecca" }
{ "_id" : ObjectId("61150abedc4158a092f3882f"), "mymiddlename" : "Hilary" }
{ "_id" : ObjectId("61150acbdc4158a092f38830"), "mysurname" : "Dias" }
{ "_id" : ObjectId("61150c0bdc4158a092f38832"), "hobby" : "Baking" }
{ "_id" : ObjectId("61150de1dc4158a092f38833"), "bestf1" : "crk" }
{ "_id" : ObjectId("61150de1dc4158a092f38834"), "bestf2" : "dels" }
>
```

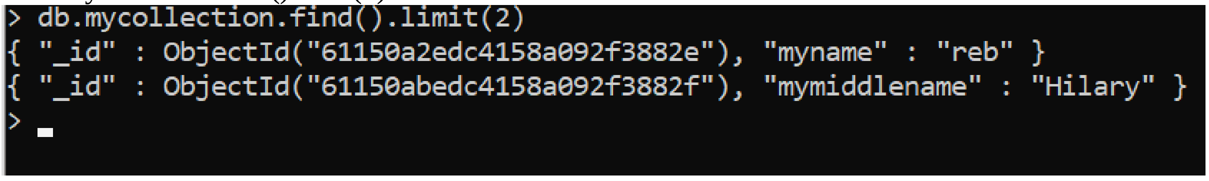
16. Update –

> db.mycollection.update({myname:"Rebecca"},{$set:{myname:"reb"}})});
>db.student.find().pretty()



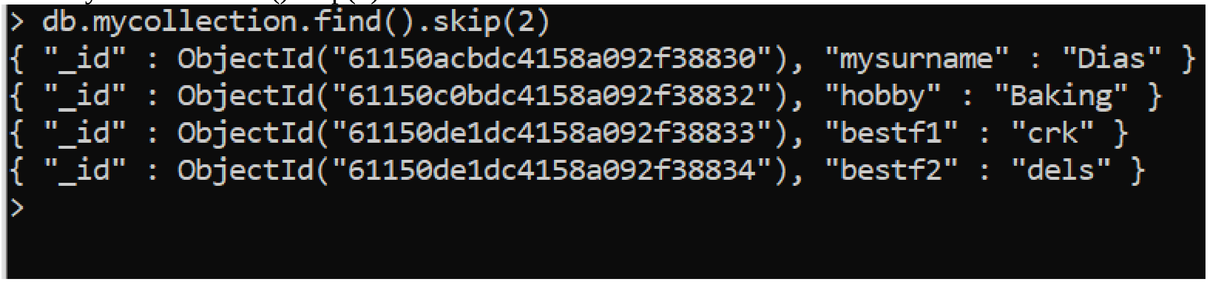
```
> db.mycollection.update({myname:"Rebecca"},{$set:{myname:"reb"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.mycollection.find().pretty()
{ "_id" : ObjectId("61150a2edc4158a092f3882e"), "myname" : "reb" }
{ "_id" : ObjectId("61150abedc4158a092f3882f"), "mymiddlename" : "Hilary" }
{ "_id" : ObjectId("61150acbdc4158a092f38830"), "mysurname" : "Dias" }
{ "_id" : ObjectId("61150c0bdc4158a092f38832"), "hobby" : "Baking" }
{ "_id" : ObjectId("61150de1dc4158a092f38833"), "bestf1" : "crk" }
{ "_id" : ObjectId("61150de1dc4158a092f38834"), "bestf2" : "dels" }
>
```

17. Limit – this limits the display of output
> db.mycollection.find().limit(2)



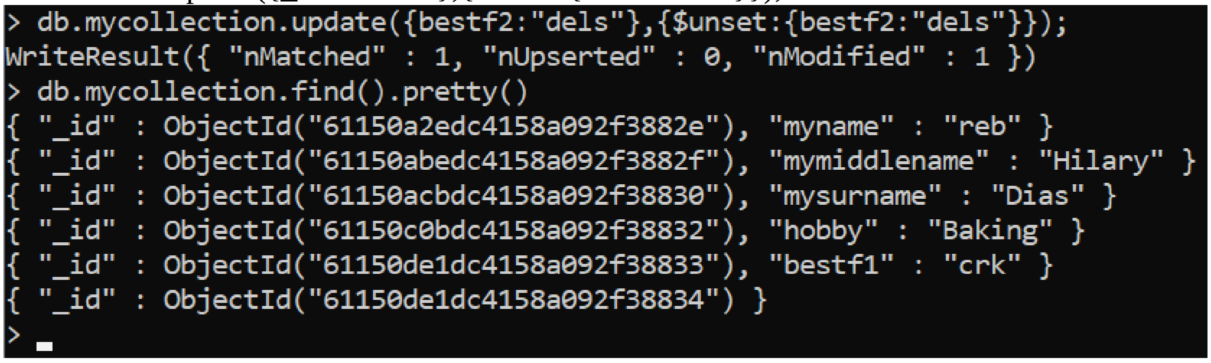
```
> db.mycollection.find().limit(2)
{ "_id" : ObjectId("61150a2edc4158a092f3882e"), "myname" : "reb" }
{ "_id" : ObjectId("61150abedc4158a092f3882f"), "mymiddlename" : "Hilary" }
>
```

18. Skip – This skips the first n records.
> db.mycollection.find().skip(2)



```
> db.mycollection.find().skip(2)
{ "_id" : ObjectId("61150acbdc4158a092f38830"), "mysurname" : "Dias" }
{ "_id" : ObjectId("61150c0bdc4158a092f38832"), "hobby" : "Baking" }
{ "_id" : ObjectId("61150de1dc4158a092f38833"), "bestf1" : "crk" }
{ "_id" : ObjectId("61150de1dc4158a092f38834"), "bestf2" : "dels" }
>
```

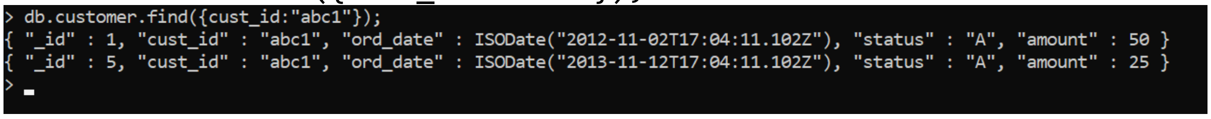
19. <u>To remove an attribute use unset</u>
> db.student.update({_bestf2:dels},{$unset:{bestf2:'dels'}});



```
> db.mycollection.update({bestf2:"dels"},{$unset:{bestf2:"dels"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.mycollection.find().pretty()
{ "_id" : ObjectId("61150a2edc4158a092f3882e"), "myname" : "reb" }
{ "_id" : ObjectId("61150abedc4158a092f3882f"), "mymiddlename" : "Hilary" }
{ "_id" : ObjectId("61150acbdc4158a092f38830"), "mysurname" : "Dias" }
{ "_id" : ObjectId("61150c0bdc4158a092f38832"), "hobby" : "Baking" }
{ "_id" : ObjectId("61150de1dc4158a092f38833"), "bestf1" : "crk" }
{ "_id" : ObjectId("61150de1dc4158a092f38834") }
> _
```

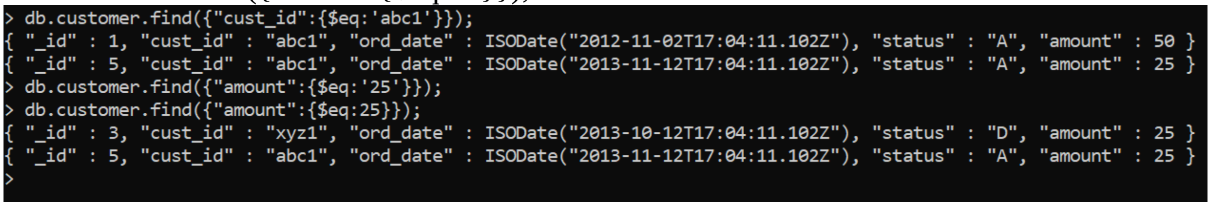
Queries using Mongodb

20. <u>Finding documents based on search criteria-find method</u>
> `db.customer.find({cust_id:'abc1'});`



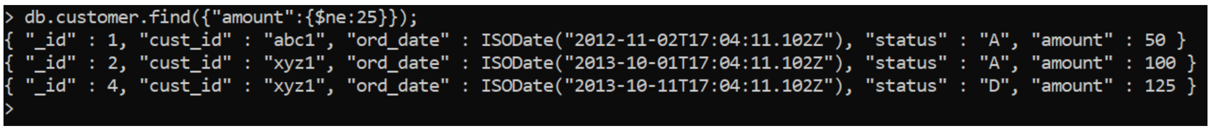
```
> db.customer.find({cust_id:"abc1"});
{ "_id" : 1, "cust_id" : "abc1", "ord_date" : ISODate("2012-11-02T17:04:11.102Z"), "status" : "A", "amount" : 50 }
{ "_id" : 5, "cust_id" : "abc1", "ord_date" : ISODate("2013-11-12T17:04:11.102Z"), "status" : "A", "amount" : 25 }
> _
```

21. <u>Finding records with same matching criteria</u>. (Equivalent to "=" clause)
> db.customer.find({amount:{$eq:25}});



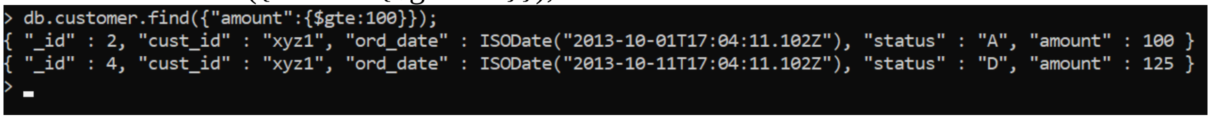
```
> db.customer.find({"cust_id":{$eq:'abc1'}});
{ "_id" : 1, "cust_id" : "abc1", "ord_date" : ISODate("2012-11-02T17:04:11.102Z"), "status" : "A", "amount" : 50 }
{ "_id" : 5, "cust_id" : "abc1", "ord_date" : ISODate("2013-11-12T17:04:11.102Z"), "status" : "A", "amount" : 25 }
> db.customer.find({"amount":{$eq:'25'}});
> db.customer.find({"amount":{$eq:25}});
{ "_id" : 3, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-12T17:04:11.102Z"), "status" : "D", "amount" : 25 }
{ "_id" : 5, "cust_id" : "abc1", "ord_date" : ISODate("2013-11-12T17:04:11.102Z"), "status" : "A", "amount" : 25 }
>
```

$ne→not equal to
> db.customer.find({amount:{$ne:25}});



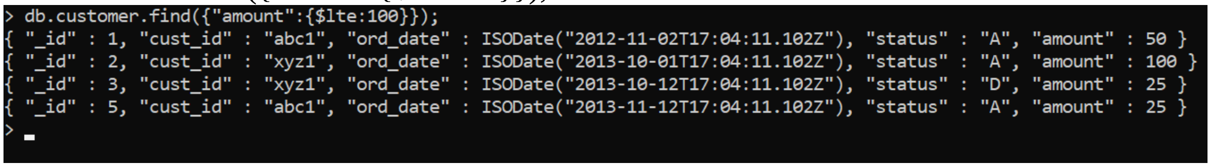
```
> db.customer.find({"amount":{$ne:25}});
{ "_id" : 1, "cust_id" : "abc1", "ord_date" : ISODate("2012-11-02T17:04:11.102Z"), "status" : "A", "amount" : 50 }
{ "_id" : 2, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-01T17:04:11.102Z"), "status" : "A", "amount" : 100 }
{ "_id" : 4, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-11T17:04:11.102Z"), "status" : "D", "amount" : 125 }
>
```

$gte→greater than or equal to
> db.customer.find({amount:{$gte:100}});



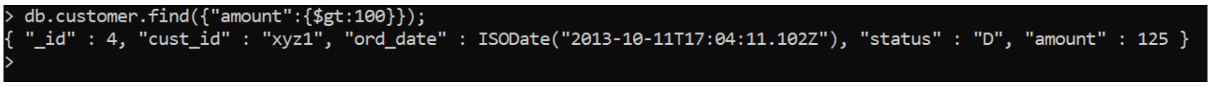
```
> db.customer.find({"amount":{$gte:100}});
{ "_id" : 2, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-01T17:04:11.102Z"), "status" : "A", "amount" : 100 }
{ "_id" : 4, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-11T17:04:11.102Z"), "status" : "D", "amount" : 125 }
> _
```

$lte→less than or equal to
> db.customer.find({amount:{$lte:100}});



```
> db.customer.find({"amount":{$lte:100}});
{ "_id" : 1, "cust_id" : "abc1", "ord_date" : ISODate("2012-11-02T17:04:11.102Z"), "status" : "A", "amount" : 50 }
{ "_id" : 2, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-01T17:04:11.102Z"), "status" : "A", "amount" : 100 }
{ "_id" : 3, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-12T17:04:11.102Z"), "status" : "D", "amount" : 25 }
{ "_id" : 5, "cust_id" : "abc1", "ord_date" : ISODate("2013-11-12T17:04:11.102Z"), "status" : "A", "amount" : 25 }
> _
```

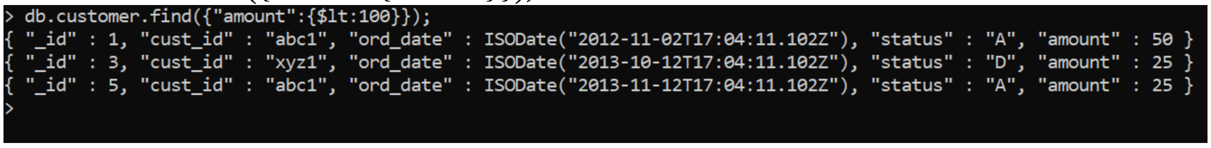
$gt→ greater than
> db.customer.find({amount:{$gt:100}});



```
> db.customer.find({"amount":{$gt:100}});
{ "_id" : 4, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-11T17:04:11.102Z"), "status" : "D", "amount" : 125 }
>
```

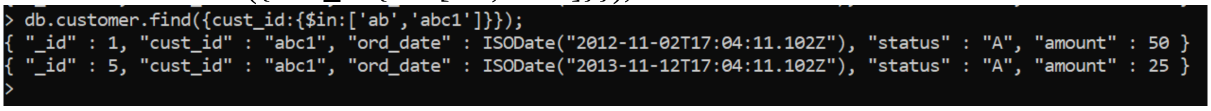
$lt→lesser than

> db.customer.find({amount:{$lt:100}});



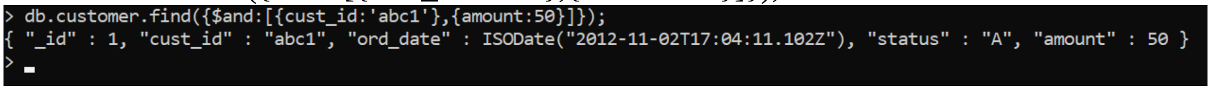
```
> db.customer.find({"amount":{$lt:100}});
{ "_id" : 1, "cust_id" : "abc1", "ord_date" : ISODate("2012-11-02T17:04:11.102Z"), "status" : "A", "amount" : 50 }
{ "_id" : 3, "cust_id" : "xyz1", "ord_date" : ISODate("2013-10-12T17:04:11.102Z"), "status" : "D", "amount" : 25 }
{ "_id" : 5, "cust_id" : "abc1", "ord_date" : ISODate("2013-11-12T17:04:11.102Z"), "status" : "A", "amount" : 25 }
>
```

22. Finding records based on the 'IN' operator. Similar to SQL
    > db.customer.find({cust_id:{$in:['ab','abc1']}});



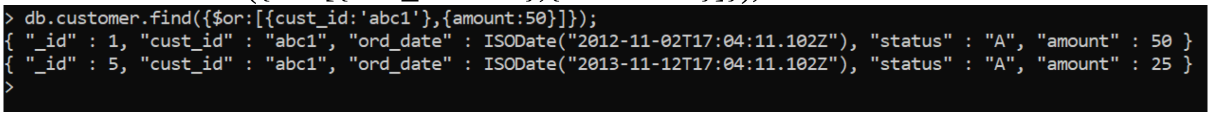
```
> db.customer.find({cust_id:{$in:['ab','abc1']}});
{ "_id" : 1, "cust_id" : "abc1", "ord_date" : ISODate("2012-11-02T17:04:11.102Z"), "status" : "A", "amount" : 50 }
{ "_id" : 5, "cust_id" : "abc1", "ord_date" : ISODate("2013-11-12T17:04:11.102Z"), "status" : "A", "amount" : 25 }
>
```

23. Finding records based on the AND Clause
    > db.customer.find({$and:[{cust_id:'abc1'},{amount:50}]});



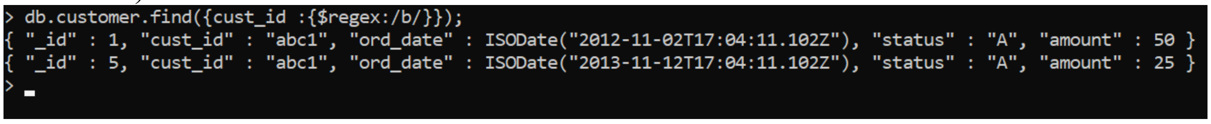
```
> db.customer.find({$and:[{cust_id:'abc1'},{amount:50}]});
{ "_id" : 1, "cust_id" : "abc1", "ord_date" : ISODate("2012-11-02T17:04:11.102Z"), "status" : "A", "amount" : 50 }
>
```

24. Finding records based on the OR clause
    > db.customer.find({$or:[{cust_id:'abc1'},{amount:50}]});



```
> db.customer.find({$or:[{cust_id:'abc1'},{amount:50}]});
{ "_id" : 1, "cust_id" : "abc1", "ord_date" : ISODate("2012-11-02T17:04:11.102Z"), "status" : "A", "amount" : 50 }
{ "_id" : 5, "cust_id" : "abc1", "ord_date" : ISODate("2013-11-12T17:04:11.102Z"), "status" : "A", "amount" : 25 }
>
```

25. Finding records based on matching patterns (Regex)
    > db.customer.find({cust_id :{$regex:/b/}}); All customers whose id contains the letter 'b')

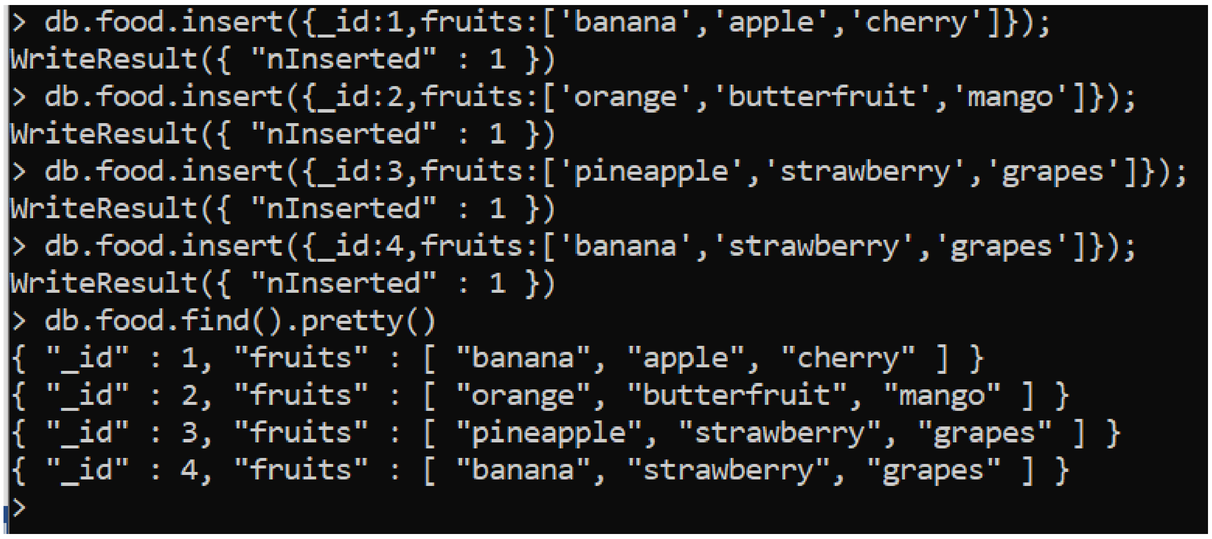


```
> db.customer.find({cust_id :{$regex:/b/}});
{ "_id" : 1, "cust_id" : "abc1", "ord_date" : ISODate("2012-11-02T17:04:11.102Z"), "status" : "A", "amount" : 50 }
{ "_id" : 5, "cust_id" : "abc1", "ord_date" : ISODate("2013-11-12T17:04:11.102Z"), "status" : "A", "amount" : 25 }
>
```

Other collection commands

1. To create arrays in a collection –

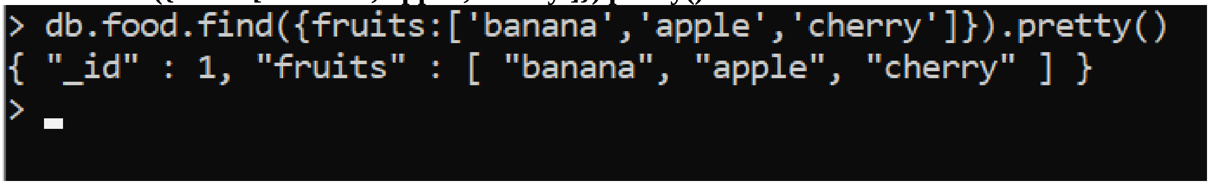
> db.food.insert({_id:1,fruits:['banana','apple','cherry']});
> db.food.insert({_id:2,fruits:['orange','butterfruit','mango']});
> db.food.insert({_id:3,fruits:['pineapple','strawberry','grapes']});
> db.food.insert({_id:4,fruits:['banana','strawberry','grapes']});



```
> db.food.insert({_id:1,fruits:['banana','apple','cherry']});
WriteResult({ "nInserted" : 1 })
> db.food.insert({_id:2,fruits:['orange','butterfruit','mango']});
WriteResult({ "nInserted" : 1 })
> db.food.insert({_id:3,fruits:['pineapple','strawberry','grapes']});
WriteResult({ "nInserted" : 1 })
> db.food.insert({_id:4,fruits:['banana','strawberry','grapes']});
WriteResult({ "nInserted" : 1 })
> db.food.find().pretty()
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
{ "_id" : 2, "fruits" : [ "orange", "butterfruit", "mango" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "strawberry", "grapes" ] }
>
```

2. To find certain elements in the array –

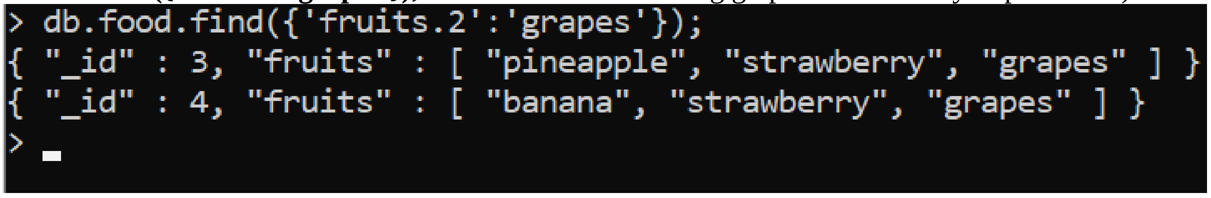
> **db.food.find({fruits:['banana','apple','cherry']}).pretty()**



```
> db.food.find({fruits:['banana','apple','cherry']}).pretty()
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
>
```

3. To find elements with certain array positions

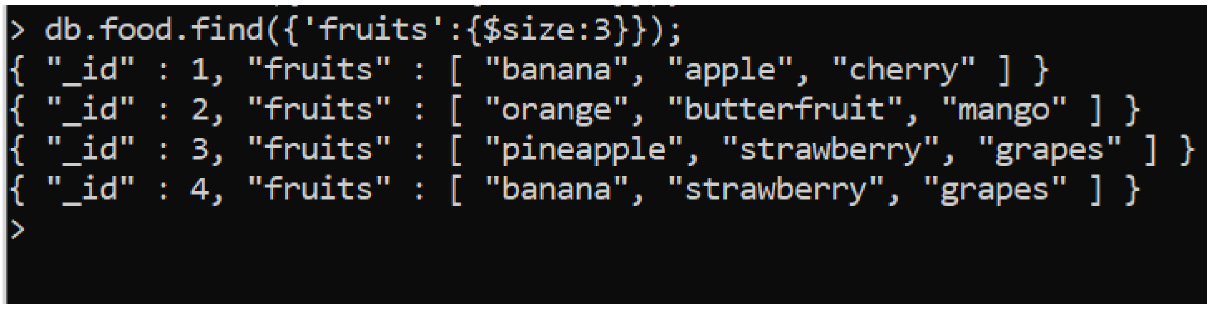
> **db.food.find({'fruits.2':'grapes'});** Find the elements having grapes in their array in position 2)



```
> db.food.find({'fruits.2':'grapes'});
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "strawberry", "grapes" ] }
>
```

4. To display documents having only two array elements –

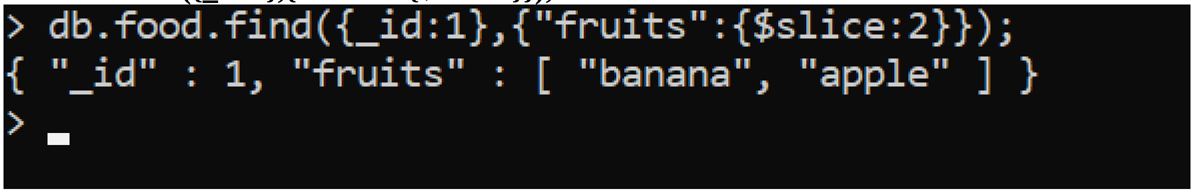
> **db.food.find({'fruits':{$size:3}});**



```
> db.food.find({'fruits':{$size:3}});
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
{ "_id" : 2, "fruits" : [ "orange", "butterfruit", "mango" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "strawberry", "grapes" ] }
>
```

5. To display selected array elements from a specific document

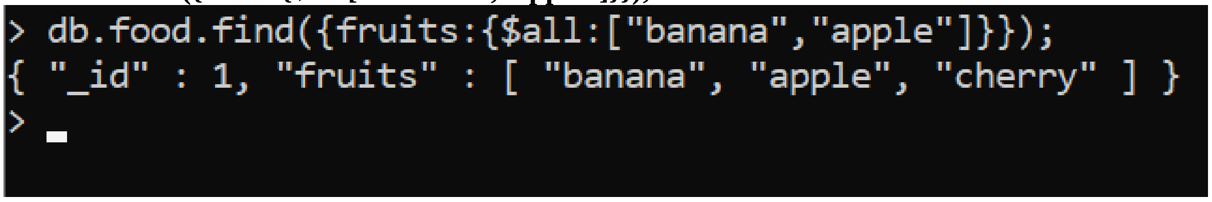
> **db.food.find({_id:1},{"fruits":{$slice:2}});**



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

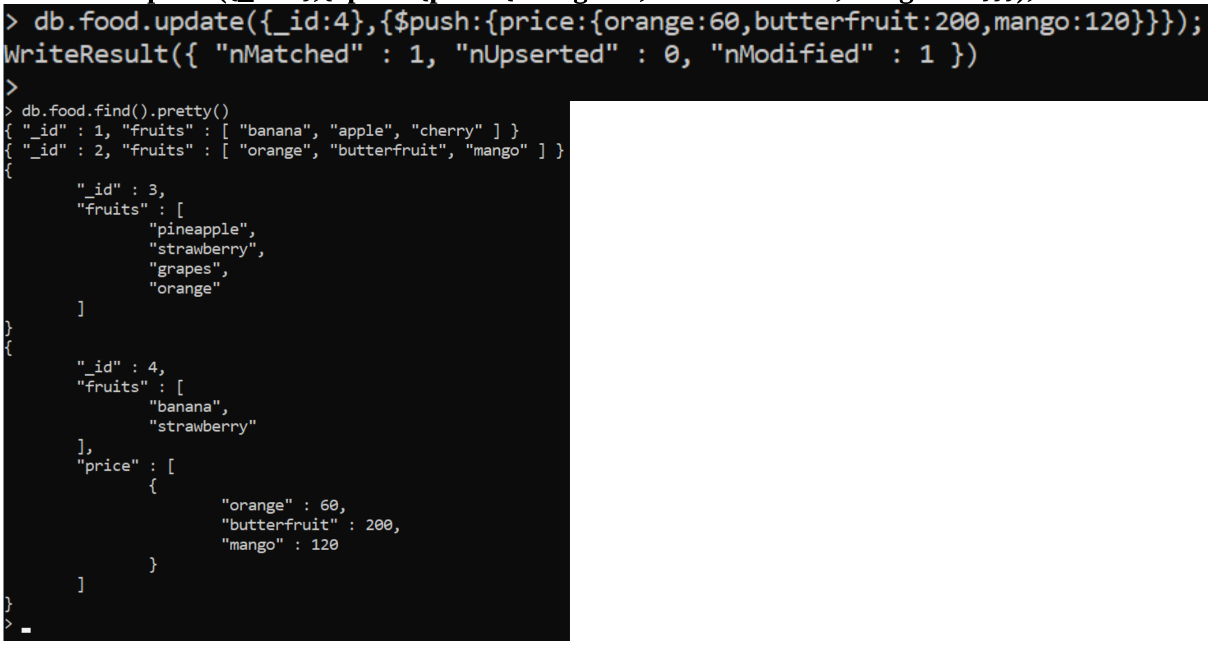
6. To display all documents which have the same array elements

> db.food.find({fruits:{$all:["banana ","apple"]}});



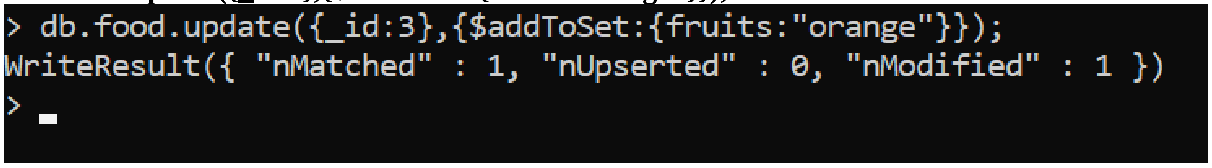
```
> db.food.find({fruits:{$all:["banana","apple"]}});
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
>
```

7. To update a specific document and include a new attribute for it
   > db.food.update({_id:4},{$push:{price:{orange:60,butterfruit:200,mango:120}}});



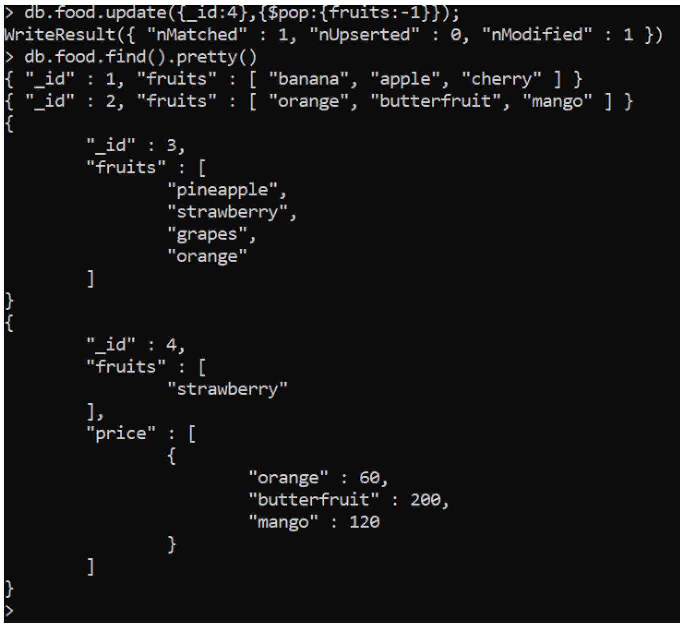
```
> db.food.update({_id:4},{$push:{price:{orange:60,butterfruit:200,mango:120}}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.food.find().pretty()
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
{ "_id" : 2, "fruits" : [ "orange", "butterfruit", "mango" ] }
{
        "_id" : 3,
        "fruits" : [
                "pineapple",
                "strawberry",
                "grapes",
                "orange"
        ]
}
{
        "_id" : 4,
        "fruits" : [
                "banana",
                "strawberry"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

8. To update a specific document by adding elements to the array
   > db.food.update({_id:3},{$addToSet:{fruits:"orange"}});



```
> db.food.update({_id:3},{$addToSet:{fruits:"orange"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

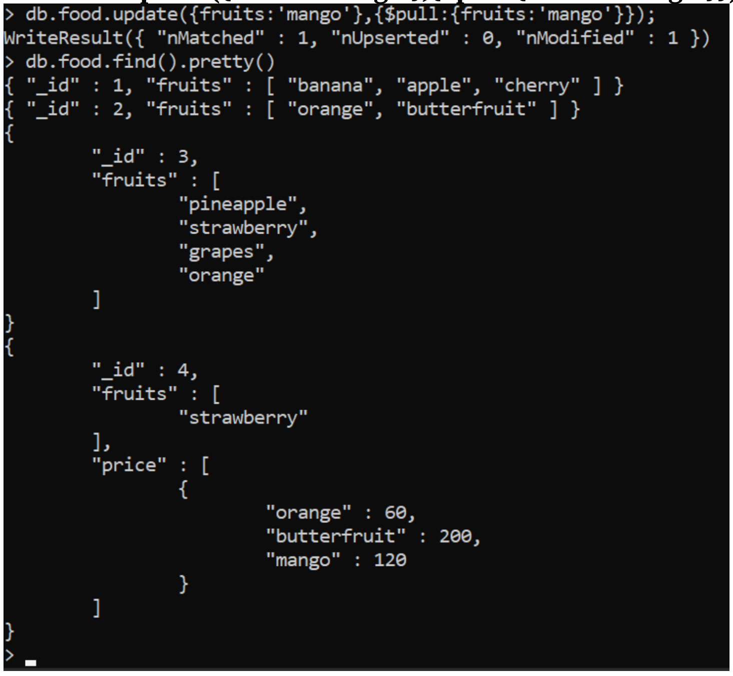
9. To update a specific document by removing elements to the array
   > db.food.update({_id:4},{$pop:{fruits:-1}});



```
> db.food.update({_id:4},{$pop:{fruits:-1}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.food.find().pretty()
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
{ "_id" : 2, "fruits" : [ "orange", "butterfruit", "mango" ] }
{
        "_id" : 3,
        "fruits" : [
                "pineapple",
                "strawberry",
                "grapes",
                "orange"
        ]
}
{
        "_id" : 4,
        "fruits" : [
                "strawberry"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

10. To remove an element from the entire set of documents

> db.food.update({fruits:'mango'},{$pull:{fruits:'mango'}});



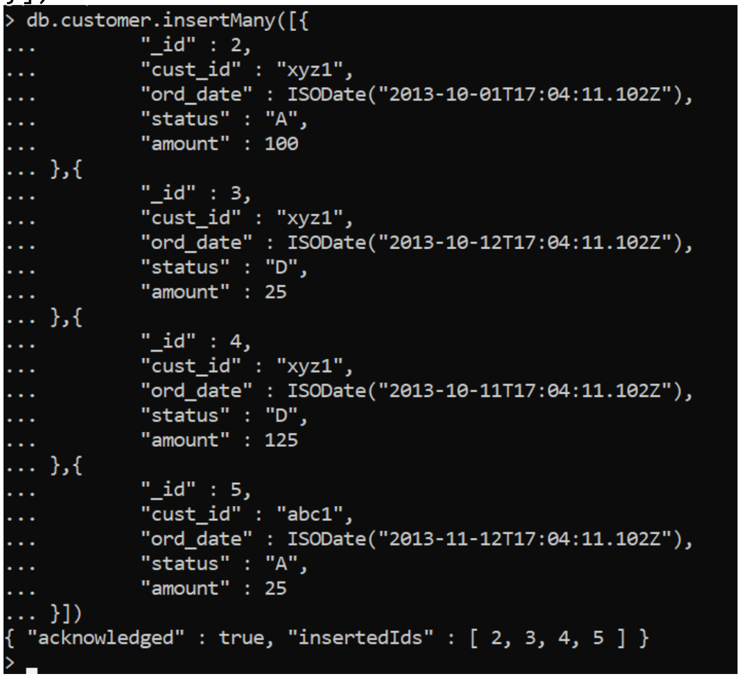
```
> db.food.update({fruits:'mango'},{$pull:{fruits:'mango'}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.food.find().pretty()
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
{ "_id" : 2, "fruits" : [ "orange", "butterfruit" ] }
{
        "_id" : 3,
        "fruits" : [
                "pineapple",
                "strawberry",
                "grapes",
                "orange"
        ]
}
{
        "_id" : 4,
        "fruits" : [
                "strawberry"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

Aggregate functions

1. Aggregate functions. Using map reduce

Create a customer Collection with the following values

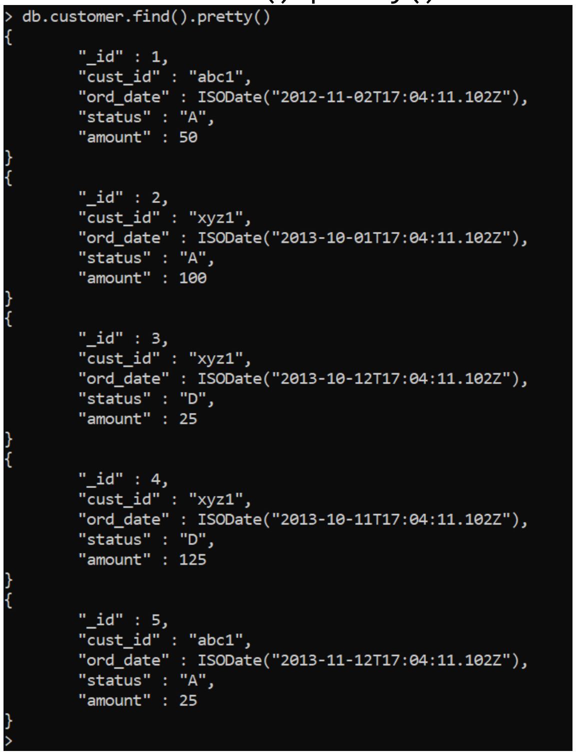
```
db.customer.insertMany([{
        "_id" : 1,
        "cust_id" : "abc1",
        "ord_date" : ISODate("2012-11-02T17:04:11.102Z"),
        "status" : "A",
        "amount" : 50
},{
        "_id" : 2,
        "cust_id" : "xyz1",
        "ord_date" : ISODate("2013-10-01T17:04:11.102Z"),
		"status" : "A",
		"amount" : 100
	},{
		"_id" : 3,
		"cust_id" : "xyz1",
		"ord_date" : ISODate("2013-10-12T17:04:11.102Z"),
		"status" : "D",
		"amount" : 25
	},{
		"_id" : 4,
		"cust_id" : "xyz1",
		"ord_date" : ISODate("2013-10-11T17:04:11.102Z"),
		"status" : "D",
		"amount" : 125
	},{
		"_id" : 5,
		"cust_id" : "abc1",
		"ord_date" : ISODate("2013-11-12T17:04:11.102Z"),
		"status" : "A",
		"amount" : 25
	}])
```



```
> db.customer.insertMany([{
...         "_id" : 2,
...         "cust_id" : "xyz1",
...         "ord_date" : ISODate("2013-10-01T17:04:11.102Z"),
...         "status" : "A",
...         "amount" : 100
... },{
...         "_id" : 3,
...         "cust_id" : "xyz1",
...         "ord_date" : ISODate("2013-10-12T17:04:11.102Z"),
...         "status" : "D",
...         "amount" : 25
... },{
...         "_id" : 4,
...         "cust_id" : "xyz1",
...         "ord_date" : ISODate("2013-10-11T17:04:11.102Z"),
...         "status" : "D",
...         "amount" : 125
... },{
...         "_id" : 5,
...         "cust_id" : "abc1",
...         "ord_date" : ISODate("2013-11-12T17:04:11.102Z"),
...         "status" : "A",
...         "amount" : 25
... }])
{ "acknowledged" : true, "insertedIds" : [ 2, 3, 4, 5 ] }
>
```

Confirm that the 5 documents exist in the collection customer

`db.customer.find().pretty()`

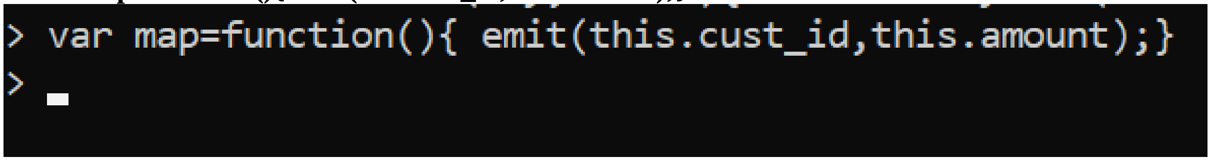


```
> db.customer.find().pretty()
{
        "_id" : 1,
        "cust_id" : "abc1",
        "ord_date" : ISODate("2012-11-02T17:04:11.102Z"),
        "status" : "A",
        "amount" : 50
}
{
        "_id" : 2,
        "cust_id" : "xyz1",
        "ord_date" : ISODate("2013-10-01T17:04:11.102Z"),
        "status" : "A",
        "amount" : 100
}
{
        "_id" : 3,
        "cust_id" : "xyz1",
        "ord_date" : ISODate("2013-10-12T17:04:11.102Z"),
        "status" : "D",
        "amount" : 25
}
{
        "_id" : 4,
        "cust_id" : "xyz1",
        "ord_date" : ISODate("2013-10-11T17:04:11.102Z"),
        "status" : "D",
        "amount" : 125
}
{
        "_id" : 5,
        "cust_id" : "abc1",
        "ord_date" : ISODate("2013-11-12T17:04:11.102Z"),
        "status" : "A",
        "amount" : 25
}
>
```

Map reduce in Mongodb

1. Creation of map function

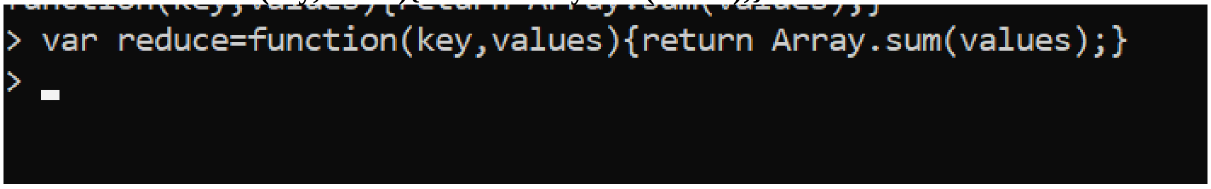
   > **var map=function(){ emit(this.cust_id,this.amount);}**



```
> var map=function(){ emit(this.cust_id,this.amount);}
>
```

2. Creation of reduce function

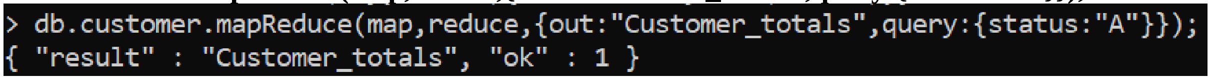
   var reduce=function(key,values){return Array.sum(values);}



```
> var reduce=function(key,values){return Array.sum(values);}
>
```

3. To execute the query.

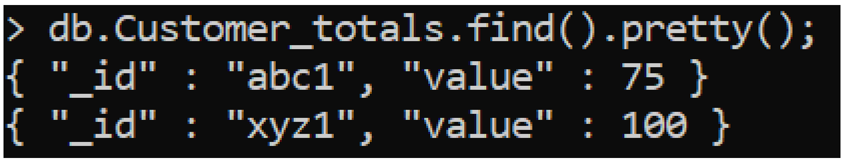
   > **db.customer.mapReduce(map,reduce,{out:"Customer_totals",query:{status:"A"}});**



```
> db.customer.mapReduce(map,reduce,{out:"Customer_totals",query:{status:"A"}});
{ "result" : "Customer_totals", "ok" : 1 }
```

4. To display the output

   db.Customer_totals.find().pretty();



```
> db.Customer_totals.find().pretty();
{ "_id" : "abc1", "value" : 75 }
{ "_id" : "xyz1", "value" : 100 }
```

Conclusion

MongoDB represents data as of JSON documents whereas MySQL represents data in tables and rows. In MongoDB, you don't need to define the schema while in MySQL you need to define your tables and columns. MongoDB doesn't support JOIN but MySQL supports JOIN operations NoSQL databases are non-relational databases. They come in a variety of flavours (mainly "document databases", "key-value stores", and "graph databases"), but you usually don't get any or all of the following "relational" features. A distributed systems design – allowing you to intelligently put data where you want it.

Successfully implemented NoSQL (MongoDB) commands and queries , output was shown as well.