

Group C:
MongoDB

Assignment No. 1

Date:

Title: Create a database with suitable example using MongoDB and implement Inserting and saving document (batch insert, insert validation)

Removing document

Updating document (document replacement, using modifiers, upsets, updating multiple documents, returning updated documents)

Remarks:

Aim: -: Create a database with suitable example using MongoDB and implement Inserting, updating, removing and saving document

Objective: Perform CRUD operation on MongoDB Database

What is MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

Document Database

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

Figure shows a MongoDB document.

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```



The diagram illustrates the structure of the MongoDB document shown in the code block. It consists of four field-value pairs, each with a blue arrow pointing from the text 'field: value' to the corresponding field in the document:

- name: "sue",
- age: 26,
- status: "A",
- groups: ["news", "sports"]

The advantages of using documents are:

- Documents (i.e. objects) correspond to native data types in many programming languages.
- Embedded documents and arrays reduce need for expensive joins.
- Dynamic schema supports fluent polymorphism.

Key Features

High Performance

MongoDB provides high performance data persistence. In particular, • Support for embedded data models reduces I/O activity on database system.

- Indexes support faster queries and can include keys from embedded documents and arrays.

High Availability

To provide high availability, MongoDB's replication facility, called replica sets, provide:

- Automatic failover.
- Data redundancy.

A replica set is a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability.

Automatic Scaling

MongoDB provides horizontal scalability as part of its core functionality.

- Automatic sharding distributes data across a cluster of machines.
- Replica sets can provide eventually-consistent reads for low-latency high throughput deployments.

Objective:

- ❑ In this Assignment, we are creating Teacher Database. Which contain the information of Teacher_id, name of a teacher, department of a teacher, salary and status of a teacher? Here status is whether teacher is approved by the university or not.
- ❑ Our main aim is to implement all the DDL & DML queries on the Teacher Database and difference between SQL Commands and mongodb commands.

SQL Vs MongoDB

SQL Concepts	MongoDB Concepts
database	database
table	Collection
Row	Document Or BSON Document
Column	Field
Index	Index
Table Join	Embedded documents & Linking
Primary key	Primary Key
Specify any unique column or column combination as primary key.	In MongoDB, the primary key is automatically set to the <u>_id</u> field.
aggregation (e.g. group by)	aggregation pipeline

Executables

	Oracle	MySQL	MongoDB
Database Server	oracle	mysqld	mongod
Database Client	sqlplus	mysql	mongo

MongoDB: Creation of Document

```
{
  Teacher_id: "Pic001",
  Teacher_Name: "Ravi",
  Dept_Name: "IT",
  Sal: 30000,
  status: 'A'
}
```

OR

```
db.createCollection("Teacher_info")
```

Insert Command:

```
db.Teacher_info.insert( { Teacher_id: "Pic001", Teacher_Name: "Ravi",Dept_Name: "IT",
Sal:30000, status: "A" } )
```

```
db.Teacher_info.insert( { Teacher_id: "Pic002", Teacher_Name: "Ravi",Dept_Name: "IT",
Sal:20000, status: "A" } )
```

```
db.Teacher_info.insert( { Teacher_id: "Pic003", Teacher_Name: "Akshay",Dept_Name:
"Comp", Sal:25000, status: "N" } )
```

Retrieving data from MongoDB:

- ▣ > db.Teacher_info.find()
- ▣ { "_id" : ObjectId("53a2d8ac8404f005f1acc666"), "Teacher_id" : "pic001", "Teacher_Name" : "Ravi", "Dept_name" : "IT", "sal" : 20000, "status" : "A" }
- ▣ { "_id" : ObjectId("53a2d8fc8404f005f1acc667"), "Teacher_id" : "pic001", "Teacher_Name" : "Ravi", "Dept_name" : "IT", "sal" : 20000, "status" : "A" }
- ▣ { "_id" : ObjectId("53a2d91b8404f005f1acc668"), "Teacher_id" : "pic003", "Teacher_Name" : "Akshay", "Dept_name" : "IT", "sal" : 25000, "status" : "N" }
- ▣ { "_id" : ObjectId("53a2da038404f005f1acc669"), "Teacher_id" : "pic003", "Teacher_Name" : "Akshay", "Dept_name" : "IT", "sal" : 25000, "status" : "N" }

SQL & MongoDB Commands

SQL SELECT Statements	MongoDB find() Statements
SELECT * FROM Teacher_info;	db.Teacher_info.find()
SELECT * FROM Teacher_info WHERE sal = 25000;	db.Teacher_info.find({sal: 25000})
SELECT Teacher_id FROM Teacher_info WHERE teacher_id 1;	db.Teacher_info.find({Teacher_id: "pic001"})
SELECT * FROM Teacher_info WHERE status != "A";	db.Teacher_info.find({status:{\$ne:"A"}})
SELECT * FROM Teacher_info WHERE status = "A" AND sal = 20000;	db.Teacher_info.find({status:"A", sal:20000})
SELECT * FROM Teacher_info WHERE status = "A" OR sal = 50000;	> db.Teacher_info.find({ \$or: [{ status: "A" } , { sal:50000 }] })
SELECT * FROM Teacher_info WHERE sal > 40000	db. Teacher_info.find({ sal: { \$gt: 40000 } })
SELECT * FROM Teacher_infoWHERE sal < 30000	db. Teacher_info.find({ sal: { \$gt: 30000 } })
SELECT * FROM Teacher_info WHERE status = "A" ORDER BY SAL ASC	db. Teacher_info.find({ status: "A" }).sort({ sal: 1 })
SELECT * FROM users WHERE status = "A" ORDER	db. Teacher_info.find({ status: "A" }).sort({sal: -1 })

BY SAL DESC	
SELECT COUNT(*) FROM Teacher_info;	db. Teacher_info.count() <i>or</i> db. Teacher_info.find().count()
SELECT DISTINCT(Dept_name) FROM Teacher_info;	db. Teacher_info.distinct("Dept_name")

Update Records

UPDATE Teacher_info SET Dept_name = "ETC" WHERE sal > 250000	db. Teacher_info.update({ sal: { \$gt: 25000 } }, { \$set: { Dept_name: "ETC" } }, { multi: true })
UPDATE Teacher_info SET sal = sal + 10000 WHERE status = "A"	db. Teacher_info.update({ status: "A" } , { \$inc: { sal: 10000 } }, { multi: true })

Delete Records

DELETE FROM Teacher_info WHERE Teacher_id = "pic001"	db.Teacher_info.remove({Teacher_id: "pic001"});
DELETE FROM Teacher_info;	db. Teacher_info.remove({})

Alter Table in Oracle & MongoDB

Oracle:

ALTER TABLE Teacher_info ADD join_date DATETIME

MongoDb:

At the document level, [update\(\)](#) operations can add fields to existing documents using the [\\$set](#) operator.

Ex:

db.Teacher_info.update({ }, { \$set: { join_date: new Date() } }, { multi: true })

Drop Command

Oracle:

DROP TABLE Teacher_info

Mongo:

```
db.Teacher_info.drop()
```

1) Finding all the records in the collection

```
> db.college.find()
```

2) Finding a particular record

```
> db.college.find("name":"pict")
```

```
... { "_id" : ObjectId("531abcc1fd853871fff162e8"), "name" : "pict" }  
... { "_id" : ObjectId("531ac014fd853871fff162e9"), "name" : "pict ", "rno":4 }  
... { "_id" : ObjectId("531ac014fd853871fff162e9"), "name" : "pict ", "rno":5 }  
... { "_id" : ObjectId("531ac014fd853871fff162e9"), "name" : "pict ", "rno":6 }  
... { "_id" : ObjectId("531ac04dfd853871fff162ef"), "name" : 7 }  
... { "_id" : ObjectId("531ac04dfd853871fff162f0"), "name" : 8 }
```

3) Updating a record

```
>db.college.update({"name":"hsaifjdas"},{$addToSet:{"dept":"mech"}},{ 'multi':true})
```

4) Removing a record

```
> db.college.remove({"name":"hsaifjdas"})
```

5) Ensuring the index

```
> db.events.ensure_index('path')
```

Conclusion: -Understand and execute MongoDB query

Department of Information Technology

Subject: Software Laboratory-I

Assignment No:-1

Problem Statement

Create employee collection in company database and insert at least five documents in employee collection containing eid,ename,designation,hiredate,salary and city. And execute following queries on company database.

```
use company;
switched to db company
> db.createCollection("emp");
{ "ok" : 1 }
> db.emp.insert({eid:1,ename:'Vishal',designation:'CEO',hiredate:'29-08-1997',salary:'90000',city:'pune'})
> db.emp.insert({eid:12,ename:'Rajesh',designation:'Manager',hiredate:'19-11-1990',salary:'60000',city:'pune'})
> db.emp.update({ename:'Rajesh'},{$set :{eid:2}})
> db.emp.insert({eid:3,ename:'Amruta',designation:'salesman',hiredate:'05-03-1980',salary:'20000',city:'goa'})
> db.emp.insert({eid:4,ename:'Yash',designation:'salesman',hiredate:'10-12-2000',salary:'20000',city:'mumbai'})
> db.emp.insert({eid:5,ename:'Rajnish',designation:'Manager',hiredate:'01-12-2000',salary:'50000',city:'ranchi'})
> db.emp.find().pretty();
{
  "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),
  "eid" : 1,
  "ename" : "Vishal",
  "designation" : "CEO",
  "hiredate" : "29-08-1997",
  "salary" : "90000",
  "city" : "pune"
}
{
  "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),
  "eid" : 2,
```

```

    "ename" : "Rajesh",
    "designation" : "Manager",
    "hiredate" : "19-11-1990",
    "salary" : "60000",
    "city" : "pune"
  }
  {
    "_id" : ObjectId("59ba6030c56c901ec584c5c4"),
    "eid" : 3,
    "ename" : "Amruta",
    "designation" : "salesman",
    "hiredate" : "05-03-1980",
    "salary" : "20000",
    "city" : "goa"
  }
  {
    "_id" : ObjectId("59ba6078c56c901ec584c5c5"),
    "eid" : 4,
    "ename" : "Yash",
    "designation" : "salesman",
    "hiredate" : "10-12-2000",
    "salary" : "20000",
    "city" : "mumbai"
  }
  {
    "_id" : ObjectId("59ba60d9c56c901ec584c5c6"),
    "eid" : 5,
    "ename" : "Rajnish",
    "designation" : "Manager",
    "hiredate" : "01-12-2000",
    "salary" : "50000",
    "city" : "ranchi"
  }
}

```

1) Add three more documents using batch insert commands.

```

> db.emp.insert([ {eid:6,ename:'Parimal',designation:'analyst',hiredate:'05-07-1970',salary:70000,city:'satara'}, {eid:7,ename:'Harsh',designation:'peon',hiredate:'14-06-1998',salary:10000,city:'akola'}, {eid:8,ename:'Aniket',designation:'Chairman',hiredate:'15-06-2013',salary:90000,city:'pune'} ])

```

```

> db.emp.find().pretty();
{
  "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),
  "eid" : 1,
  "ename" : "Vishal",
  "designation" : "CEO",

```

```

    "hiredate" : "29-08-1997",
    "salary" : "90000",
    "city" : "pune"
  }
  {
    "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),
    "eid" : 2,
    "ename" : "Rajesh",
    "designation" : "Manager",
    "hiredate" : "19-11-1990",
    "salary" : "60000",
    "city" : "pune"
  }
  {
    "_id" : ObjectId("59ba6030c56c901ec584c5c4"),
    "eid" : 3,
    "ename" : "Amruta",
    "designation" : "salesman",
    "hiredate" : "05-03-1980",
    "salary" : "20000",
    "city" : "goa"
  }
  {
    "_id" : ObjectId("59ba6078c56c901ec584c5c5"),
    "eid" : 4,
    "ename" : "Yash",
    "designation" : "salesman",
    "hiredate" : "10-12-2000",
    "salary" : "20000",
    "city" : "mumbai"
  }
  {
    "_id" : ObjectId("59ba60d9c56c901ec584c5c6"),
    "eid" : 5,
    "ename" : "Rajnish",
    "designation" : "Manager",
    "hiredate" : "01-12-2000",
    "salary" : "50000",
    "city" : "ranchi"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c7"),
    "eid" : 6,
    "ename" : "Parimal",
    "designation" : "analyst",
    "hiredate" : "05-07-1970",
    "salary" : 70000,

```

```

    "city" : "satara"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c8"),
    "eid" : 7,
    "ename" : "Harsh",
    "designation" : "peon",
    "hiredate" : "14-06-1998",
    "salary" : 10000,
    "city" : "akola"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c9"),
    "eid" : 8,
    "ename" : "Aniket",
    "designation" : "Chairman",
    "hiredate" : "15-06-2013",
    "salary" : 90000,
    "city" : "pune"
  }
}

```

2) Update the salary of all employee whose city is Pune.

```

> db.emp.update({city:'pune'},{$set :{salary:80000}},{multi:true})
> db.emp.find().pretty();
{
  "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),
  "city" : "pune",
  "designation" : "CEO",
  "eid" : 1,
  "ename" : "Vishal",
  "hiredate" : "29-08-1997",
  "salary" : 80000
}
{
  "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),
  "city" : "pune",
  "designation" : "Manager",
  "eid" : 2,
  "ename" : "Rajesh",
  "hiredate" : "19-11-1990",
  "salary" : 80000
}
{
  "_id" : ObjectId("59ba6030c56c901ec584c5c4"),
  "eid" : 3,
  "ename" : "Amruta",

```

```

        "designation" : "salesman",
        "hiredate" : "05-03-1980",
        "salary" : "20000",
        "city" : "goa"
    }
    {
        "_id" : ObjectId("59ba6078c56c901ec584c5c5"),
        "eid" : 4,
        "ename" : "Yash",
        "designation" : "salesman",
        "hiredate" : "10-12-2000",
        "salary" : "20000",
        "city" : "mumbai"
    }
    {
        "_id" : ObjectId("59ba60d9c56c901ec584c5c6"),
        "eid" : 5,
        "ename" : "Rajnish",
        "designation" : "Manager",
        "hiredate" : "01-12-2000",
        "salary" : "50000",
        "city" : "ranchi"
    }
    {
        "_id" : ObjectId("59ba66b8c56c901ec584c5c7"),
        "eid" : 6,
        "ename" : "Parimal",
        "designation" : "analyst",
        "hiredate" : "05-07-1970",
        "salary" : 70000,
        "city" : "satara"
    }
    {
        "_id" : ObjectId("59ba66b8c56c901ec584c5c8"),
        "eid" : 7,
        "ename" : "Harsh",
        "designation" : "peon",
        "hiredate" : "14-06-1998",
        "salary" : 10000,
        "city" : "akola"
    }
    {
        "_id" : ObjectId("59ba66b8c56c901ec584c5c9"),
        "eid" : 8,
        "ename" : "Aniket",
        "designation" : "Chairman",
        "hiredate" : "15-06-2013",

```

```
"salary" : 80000,  
"city" : "pune"
```

3) Add the skills of all employee using array concept in MongoDB.

```
> db.emp.update({}, {$set : {"skills":'Reading'}},{multi:true})
```

```
> db.emp.find().pretty();
```

```
{  
  "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),  
  "city" : "pune",  
  "designation" : "CEO",  
  "eid" : 1,  
  "ename" : "Vishal",  
  "hiredate" : "29-08-1997",  
  "salary" : 80000,  
  "skills" : "Reading"  
}  
{  
  "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),  
  "city" : "pune",  
  "designation" : "Manager",  
  "eid" : 2,  
  "ename" : "Rajesh",  
  "hiredate" : "19-11-1990",  
  "salary" : 80000,  
  "skills" : "Reading"  
}  
{  
  "_id" : ObjectId("59ba6030c56c901ec584c5c4"),  
  "city" : "goa",  
  "designation" : "salesman",  
  "eid" : 3,  
  "ename" : "Amruta",  
  "hiredate" : "05-03-1980",  
  "salary" : "20000",  
  "skills" : "Reading"  
}  
{  
  "_id" : ObjectId("59ba6078c56c901ec584c5c5"),  
  "city" : "mumbai",  
  "designation" : "salesman",  
  "eid" : 4,  
  "ename" : "Yash",  
  "hiredate" : "10-12-2000",  
  "salary" : "20000",  
  "skills" : "Reading"  
}
```

```

{
  "_id" : ObjectId("59ba60d9c56c901ec584c5c6"),
  "city" : "ranchi",
  "designation" : "Manager",
  "eid" : 5,
  "ename" : "Rajnish",
  "hiredate" : "01-12-2000",
  "salary" : "50000",
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba66b8c56c901ec584c5c7"),
  "city" : "satara",
  "designation" : "analyst",
  "eid" : 6,
  "ename" : "Parimal",
  "hiredate" : "05-07-1970",
  "salary" : 70000,
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba66b8c56c901ec584c5c8"),
  "city" : "akola",
  "designation" : "peon",
  "eid" : 7,
  "ename" : "Harsh",
  "hiredate" : "14-06-1998",
  "salary" : 10000,
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba66b8c56c901ec584c5c9"),
  "city" : "pune",
  "designation" : "Chairman",
  "eid" : 8,
  "ename" : "Aniket",
  "hiredate" : "15-06-2013",
  "salary" : 80000,
  "skills" : "Reading"
}

```

4) Delete all employee from employee collection whose hiredate is 01/12/2000.

```

> db.emp.remove({hiredate:'01-12-2000'})
> db.emp.find().pretty();
{

```

```

    "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),
    "city" : "pune",
    "designation" : "CEO",
    "eid" : 1,
    "ename" : "Vishal",
    "hiredate" : "29-08-1997",
    "salary" : 80000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),
    "city" : "pune",
    "designation" : "Manager",
    "eid" : 2,
    "ename" : "Rajesh",
    "hiredate" : "19-11-1990",
    "salary" : 80000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba6030c56c901ec584c5c4"),
    "city" : "goa",
    "designation" : "salesman",
    "eid" : 3,
    "ename" : "Amruta",
    "hiredate" : "05-03-1980",
    "salary" : "20000",
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba6078c56c901ec584c5c5"),
    "city" : "mumbai",
    "designation" : "salesman",
    "eid" : 4,
    "ename" : "Yash",
    "hiredate" : "10-12-2000",
    "salary" : "20000",
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c7"),
    "city" : "satara",
    "designation" : "analyst",
    "eid" : 6,
    "ename" : "Parimal",
    "hiredate" : "05-07-1970",
    "salary" : 70000,

```



```

    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c8"),
    "city" : "akola",
    "designation" : "peon",
    "eid" : 7,
    "ename" : "Harsh",
    "hiredate" : "14-06-1998",
    "salary" : 10000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c9"),
    "city" : "pune",
    "designation" : "Chairman",
    "eid" : 8,
    "ename" : "Aniket",
    "hiredate" : "15-06-2013",
    "salary" : 80000,
    "skills" : "Reading"
  }
}

```

5) Delete skills of all employee whose designation is salesman.

```
> db.emp.update({designation:'salesman'},{$unset: {city:' '}},{multi:true})
```

```
> db.emp.find().pretty();
```

```

{
  "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),
  "city" : "pune",
  "designation" : "Manager",
  "eid" : 2,
  "ename" : "Rajesh",
  "hiredate" : "19-11-1990",
  "salary" : 80000,
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba6078c56c901ec584c5c5"),
  "designation" : "salesman",
  "eid" : 4,
  "ename" : "Yash",
  "hiredate" : "10-12-2000",
  "salary" : "20000",
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba66b8c56c901ec584c5c7"),

```

```

        "city" : "satara",
        "designation" : "analyst",
        "eid" : 6,
        "ename" : "Parimal",
        "hiredate" : "05-07-1970",
        "salary" : 70000,
        "skills" : "Reading"
    }
    {
        "_id" : ObjectId("59ba66b8c56c901ec584c5c8"),
        "city" : "akola",
        "designation" : "peon",
        "eid" : 7,
        "ename" : "Harsh",
        "hiredate" : "14-06-1998",
        "salary" : 10000,
        "skills" : "Reading"
    }
    {
        "_id" : ObjectId("59ba66b8c56c901ec584c5c9"),
        "city" : "pune",
        "designation" : "Chairman",
        "eid" : 8,
        "ename" : "Aniket",
        "hiredate" : "15-06-2013",
        "salary" : 80000,
        "skills" : "Reading"
    }
    {
        "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),
        "city" : "pune",
        "designation" : "CEO",
        "eid" : 1,
        "ename" : "Vishal",
        "hiredate" : "29-08-1997",
        "salary" : 80000,
        "skills" : [
            "Java",
            "python"
        ]
    }
}

```

6) Update the employee skills of employee id is 1 with java and python.

```

> db.emp.update({eid:1},{ $set :{skills:['Java','python']}})
> db.emp.find().pretty();

```

```

{
  "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),
  "city" : "pune",
  "designation" : "Manager",
  "eid" : 2,
  "ename" : "Rajesh",
  "hiredate" : "19-11-1990",
  "salary" : 80000,
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba6030c56c901ec584c5c4"),
  "city" : "goa",
  "designation" : "salesman",
  "eid" : 3,
  "ename" : "Amruta",
  "hiredate" : "05-03-1980",
  "salary" : "20000",
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba6078c56c901ec584c5c5"),
  "city" : "mumbai",
  "designation" : "salesman",
  "eid" : 4,
  "ename" : "Yash",
  "hiredate" : "10-12-2000",
  "salary" : "20000",
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba66b8c56c901ec584c5c7"),
  "city" : "satara",
  "designation" : "analyst",
  "eid" : 6,
  "ename" : "Parimal",
  "hiredate" : "05-07-1970",
  "salary" : 70000,
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba66b8c56c901ec584c5c8"),
  "city" : "akola",
  "designation" : "peon",
  "eid" : 7,
  "ename" : "Harsh",
  "hiredate" : "14-06-1998",

```

```

        "salary" : 10000,
        "skills" : "Reading"
    }
    {
        "_id" : ObjectId("59ba66b8c56c901ec584c5c9"),
        "city" : "pune",
        "designation" : "Chairman",
        "eid" : 8,
        "ename" : "Aniket",
        "hiredate" : "15-06-2013",
        "salary" : 80000,
        "skills" : "Reading"
    }
    {
        "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),
        "city" : "pune",
        "designation" : "CEO",
        "eid" : 1,
        "ename" : "Vishal",
        "hiredate" : "29-08-1997",
        "salary" : 80000,
        "skills" : [
            "Java",
            "python"
        ]
    }
}

```

7) Delete all employee who has joined in the company in month of march.

```

> db.emp.remove({'hiredate':/^([0-9])[0-9]\-03-/)})
> db.emp.find().pretty();
{
  "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),
  "city" : "pune",
  "designation" : "Manager",
  "eid" : 2,
  "ename" : "Rajesh",
  "hiredate" : "19-11-1990",
  "salary" : 80000,
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba6078c56c901ec584c5c5"),
  "city" : "mumbai",
  "designation" : "salesman",
  "eid" : 4,
  "ename" : "Yash",
  "hiredate" : "10-12-2000",
}

```

```

    "salary" : "20000",
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c7"),
    "city" : "satara",
    "designation" : "analyst",
    "eid" : 6,
    "ename" : "Parimal",
    "hiredate" : "05-07-1970",
    "salary" : 70000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c8"),
    "city" : "akola",
    "designation" : "peon",
    "eid" : 7,
    "ename" : "Harsh",
    "hiredate" : "14-06-1998",
    "salary" : 10000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c9"),
    "city" : "pune",
    "designation" : "Chairman",
    "eid" : 8,
    "ename" : "Aniket",
    "hiredate" : "15-06-2013",
    "salary" : 80000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),
    "city" : "pune",
    "designation" : "CEO",
    "eid" : 1,
    "ename" : "Vishal",
    "hiredate" : "29-08-1997",
    "salary" : 80000,
    "skills" : [
      "Java",
      "python"
    ]
  }
}

```

8) Remove City of all employees whose designation is salesman.

```
> db.emp.update({designation:'salesman'},{$unset: {city:' '}})
```

```
> db.emp.find().pretty();
```

```
{
  "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),
  "city" : "pune",
  "designation" : "Manager",
  "eid" : 2,
  "ename" : "Rajesh",
  "hiredate" : "19-11-1990",
  "salary" : 80000,
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba6078c56c901ec584c5c5"),
  "designation" : "salesman",
  "eid" : 4,
  "ename" : "Yash",
  "hiredate" : "10-12-2000",
  "salary" : "20000",
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba66b8c56c901ec584c5c7"),
  "city" : "satara",
  "designation" : "analyst",
  "eid" : 6,
  "ename" : "Parimal",
  "hiredate" : "05-07-1970",
  "salary" : 70000,
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba66b8c56c901ec584c5c8"),
  "city" : "akola",
  "designation" : "peon",
  "eid" : 7,
  "ename" : "Harsh",
  "hiredate" : "14-06-1998",
  "salary" : 10000,
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba66b8c56c901ec584c5c9"),
  "city" : "pune",
  "designation" : "Chairman",
```

```

    "eid" : 8,
    "ename" : "Aniket",
    "hiredate" : "15-06-2013",
    "salary" : 80000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),
    "city" : "pune",
    "designation" : "CEO",
    "eid" : 1,
    "ename" : "Vishal",
    "hiredate" : "29-08-1997",
    "salary" : 80000,
    "skills" : [
      "Java",
      "python"
    ]
  }
}

```

9) Modify the existing documents with new key-value pairs.

```

> db.emp.update({} , {$set : {"contact":'9145313295'}},{multi:true})
> db.emp.find().pretty();
{
  "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),
  "city" : "pune",
  "contact" : "9145313295",
  "designation" : "Manager",
  "eid" : 2,
  "ename" : "Rajesh",
  "hiredate" : "19-11-1990",
  "salary" : 80000,
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba6078c56c901ec584c5c5"),
  "contact" : "9145313295",
  "designation" : "salesman",
  "eid" : 4,
  "ename" : "Yash",
  "hiredate" : "10-12-2000",
  "salary" : "20000",
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba66b8c56c901ec584c5c7"),

```

```

    "city" : "satara",
    "contact" : "9145313295",
    "designation" : "analyst",
    "eid" : 6,
    "ename" : "Parimal",
    "hiredate" : "05-07-1970",
    "salary" : 70000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c8"),
    "city" : "akola",
    "contact" : "9145313295",
    "designation" : "peon",
    "eid" : 7,
    "ename" : "Harsh",
    "hiredate" : "14-06-1998",
    "salary" : 10000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c9"),
    "city" : "pune",
    "contact" : "9145313295",
    "designation" : "Chairman",
    "eid" : 8,
    "ename" : "Aniket",
    "hiredate" : "15-06-2013",
    "salary" : 80000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),
    "city" : "pune",
    "contact" : "9145313295",
    "designation" : "CEO",
    "eid" : 1,
    "ename" : "Vishal",
    "hiredate" : "29-08-1997",
    "salary" : 80000,
    "skills" : [
      "Java",
      "python"
    ]
  }
}

```

10) Display all databases present in MongoDB database Management System.


```
> show dbs;
company      0.203125GB
local        0.078125GB
session0     0.203125GB
session1     0.203125GB
session2     0.203125GB
session3     0.203125GB
```

11) Display all collections that are present in company collection.

```
> db.emp.find()
{ "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"), "city" : "pune", "contact" : "9145313295",
  "designation" : "Manager", "eid" : 2, "ename" : "Rajesh", "hiredate" : "19-11-1990", "salary" :
  80000, "skills" : "Reading" }
{ "_id" : ObjectId("59ba6078c56c901ec584c5c5"), "contact" : "9145313295", "designation" :
  "salesman", "eid" : 4, "ename" : "Yash", "hiredate" : "10-12-2000", "salary" : "20000", "skills" :
  "Reading" }
{ "_id" : ObjectId("59ba66b8c56c901ec584c5c7"), "city" : "satara", "contact" : "9145313295",
  "designation" : "analyst", "eid" : 6, "ename" : "Parimal", "hiredate" : "05-07-1970", "salary" :
  70000, "skills" : "Reading" }
{ "_id" : ObjectId("59ba66b8c56c901ec584c5c8"), "city" : "akola", "contact" : "9145313295",
  "designation" : "peon", "eid" : 7, "ename" : "Harsh", "hiredate" : "14-06-1998", "salary" : 10000,
  "skills" : "Reading" }
{ "_id" : ObjectId("59ba66b8c56c901ec584c5c9"), "city" : "pune", "contact" : "9145313295",
  "designation" : "Chairman", "eid" : 8, "ename" : "Aniket", "hiredate" : "15-06-2013", "salary" :
  80000, "skills" : "Reading" }
{ "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"), "city" : "pune", "contact" : "9145313295",
  "designation" : "CEO", "eid" : 1, "ename" : "Vishal", "hiredate" : "29-08-1997", "salary" : 80000,
  "skills" : [ "Java", "python" ] }
```

12) Delete skills of employee id is 5 from employee collection.

```
> db.emp.update({eid:3},{ $unset: {skills:"} })
> db.emp.find().pretty();
{
  "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),
  "city" : "pune",
  "contact" : "9145313295",
  "designation" : "Manager",
  "eid" : 2,
  "ename" : "Rajesh",
  "hiredate" : "19-11-1990",
  "salary" : 80000,
  "skills" : "Reading"
}
{
```

```

    "_id" : ObjectId("59ba6078c56c901ec584c5c5"),
    "contact" : "9145313295",
    "designation" : "salesman",
    "eid" : 4,
    "ename" : "Yash",
    "hiredate" : "10-12-2000",
    "salary" : "20000",
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c7"),
    "city" : "satara",
    "contact" : "9145313295",
    "designation" : "analyst",
    "eid" : 6,
    "ename" : "Parimal",
    "hiredate" : "05-07-1970",
    "salary" : 70000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c8"),
    "city" : "akola",
    "contact" : "9145313295",
    "designation" : "peon",
    "eid" : 7,
    "ename" : "Harsh",
    "hiredate" : "14-06-1998",
    "salary" : 10000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c9"),
    "city" : "pune",
    "contact" : "9145313295",
    "designation" : "Chairman",
    "eid" : 8,
    "ename" : "Aniket",
    "hiredate" : "15-06-2013",
    "salary" : 80000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),

```

```

> db.emp.drop()
true

```

```

    "city" : "pune",
    "contact" : "9145313295",
    "designation" : "CEO",
    "eid" : 1,
    "ename" : "Vishal",
    "hiredate" : "29-08-1997",
    "salary" : 80000,
    "skills" : [
        "Java",
        "python"
    ]
}
13) Update the city of all employees.

> db.emp.update({},{$set: {city:'pune'}},{multi:true})
> db.emp.find().pretty();
{
  "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),
  "city" : "pune",
  "contact" : "9145313295",
  "designation" : "Manager",
  "eid" : 2,
  "ename" : "Rajesh",
  "hiredate" : "19-11-1990",
  "salary" : 80000,
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba66b8c56c901ec584c5c7"),
  "city" : "pune",
  "contact" : "9145313295",
  "designation" : "analyst",
  "eid" : 6,
  "ename" : "Parimal",
  "hiredate" : "05-07-1970",
  "salary" : 70000,
  "skills" : "Reading"
}
{
  "_id" : ObjectId("59ba66b8c56c901ec584c5c8"),
  "city" : "pune",
  "contact" : "9145313295",
  "designation" : "peon",
  "eid" : 7,
  "ename" : "Harsh",
  "hiredate" : "14-06-1998",
  "salary" : 10000,

```

```

    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba66b8c56c901ec584c5c9"),
    "city" : "pune",
    "contact" : "9145313295",
    "designation" : "Chairman",
    "eid" : 8,
    "ename" : "Aniket",
    "hiredate" : "15-06-2013",
    "salary" : 80000,
    "skills" : "Reading"
  }
  {
    "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),
    "city" : "pune",
    "contact" : "9145313295",
    "designation" : "CEO",
    "eid" : 1,
    "ename" : "Vishal",
    "hiredate" : "29-08-1997",
    "salary" : 80000,
    "skills" : [
      "Java",
      "python"
    ]
  }
  {
    "_id" : ObjectId("59ba6078c56c901ec584c5c5"),
    "city" : "pune",
    "contact" : "9145313295",
    "designation" : "salesman",
    "eid" : 4,
    "ename" : "Yash",
    "hiredate" : "10-12-2000",
    "salary" : "20000",
    "skills" : "Reading"
  }
>

```

14) Display all employee information in JSON format.

```
> db.emp.find().pretty();
```

```

{
  "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),
  "city" : "pune",
  "contact" : "9145313295",
  "designation" : "Manager",
  "eid" : 2,

```

```

        "ename" : "Rajesh",
        "hiredate" : "19-11-1990",
        "salary" : 80000,
        "skills" : "Reading"
    }
    {
        "_id" : ObjectId("59ba6078c56c901ec584c5c5"),
        "contact" : "9145313295",
        "designation" : "salesman",
        "eid" : 4,
        "ename" : "Yash",
        "hiredate" : "10-12-2000",
        "salary" : "20000",
        "skills" : "Reading"
    }
    {
        "_id" : ObjectId("59ba66b8c56c901ec584c5c7"),
        "city" : "satara",
        "contact" : "9145313295",
        "designation" : "analyst",
        "eid" : 6,
        "ename" : "Parimal",
        "hiredate" : "05-07-1970",
        "salary" : 70000,
        "skills" : "Reading"
    }
    {
        "_id" : ObjectId("59ba66b8c56c901ec584c5c8"),
        "city" : "akola",
        "contact" : "9145313295",
        "designation" : "peon",
        "eid" : 7,
        "ename" : "Harsh",
        "hiredate" : "14-06-1998",
        "salary" : 10000,
        "skills" : "Reading"
    }
    {
        "_id" : ObjectId("59ba66b8c56c901ec584c5c9"),
        "city" : "pune",
        "contact" : "9145313295",
        "designation" : "Chairman",
        "eid" : 8,> db.emp.drop()

```

true

```

        "ename" : "Aniket",
        "hiredate" : "15-06-2013",

```

```

        "salary" : 80000,
        "skills" : "Reading"
    }
    {
        "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),
        "city" : "pune",
        "contact" : "9145313295",
        "designation" : "CEO",
        "eid" : 1,
        "ename" : "Vishal",
        "hiredate" : "29-08-1997",
        "salary" : 80000,
        "skills" : [
            "Java",
            "python"
        ]
    }
}
> db.emp.update({},{$set: {city:'pune'}},{multi:true})
> db.emp.find().pretty();
{
    "_id" : ObjectId("59ba5f8cc56c901ec584c5c3"),
    "city" : "pune",
    "contact" : "9145313295",
    "designation" : "Manager",
    "eid" : 2,
    "ename" : "Rajesh",
    "hiredate" : "19-11-1990",
    "salary" : 80000,
    "skills" : "Reading"
}
{
    "_id" : ObjectId("59ba66b8c56c901ec584c5c7"),
    "city" : "pune",
    "contact" : "9145313295",
    "designation" : "analyst",
    "eid" : 6,
    "ename" : "Parimal",
    "hiredate" : "05-07-1970",
    "salary" : 70000,
    "skills" : "Reading"
}
{
    "_id" : ObjectId("59ba66b8c56c901ec584c5c8"),
    "city" : "pune",
    "contact" : "9145313295",
    "designation" : "peon",
    "eid" : 7,

```

```

        "ename" : "Harsh",
        "hiredate" : "14-06-1998",
        "salary" : 10000,
        "skills" : "Reading"
    }
    {
        "_id" : ObjectId("59ba66b8c56c901ec584c5c9"),
        "city" : "pune",
        "contact" : "9145313295",
        "designation" : "Chairman",
        "eid" : 8,
        "ename" : "Aniket",
        "hiredate" : "15-06-2013",
        "salary" : 80000,
        "skills" : "Reading"
    }
    {
        "_id" : ObjectId("59ba5f5ec56c901ec584c5c2"),
        "city" : "pune",
        "contact" : "9145313295",
        "designation" : "CEO",
        "eid" : 1,
        "ename" : "Vishal",
        "hiredate" : "29-08-1997",
        "salary" : 80000,
        "skills" : [
            "Java",
            "python"
        ]
    }
    {
        "_id" : ObjectId("59ba6078c56c901ec584c5c5"),
        "city" : "pune",
        "contact" : "9145313295",
        "designation" : "salesman",
        "eid" : 4,
        "ename" : "Yash",
        "hiredate" : "10-12-2000",
        "salary" : "20000",
        "skills" : "Reading"
    }
}

```

15) Delete employee collection from company database.

```

> db.emp.drop()
true

```

Assignment No.2**Date:**

Title: Execute at least 10 queries on any suitable MongoDB database that demonstrates following querying techniques:

find and findOne (specific values)

Query criteria (Query conditionals, OR queries, \$not, Conditional semantics)

Type-specific queries (Null, Regular expression, Querying arrays)

Remarks:

Aim: -: Execute at least 10 queries on any suitable MongoDB database that demonstrates following querying techniques:

find and findOne (specific values)

Query criteria (Query conditionals, OR queries, \$not, Conditional semantics)

Type-specific queries (Null, Regular expression, Querying arrays)

Introduction to find

The find method is used to perform queries in MongoDB. Querying returns a subset of documents in a collection, from no documents at all to the entire collection. Which documents get returned is determined by the first argument to find, which is a document specifying the query to be performed. An empty query document (i.e., {}) matches everything in the collection. If find isn't given a query document, it defaults to {}. For example, the following:

```
> db.c.find()
```

returns everything in the collection c. When we start adding key/value pairs to the query document, we begin restricting our search. This works in a straightforward way for most types. Integers match integers, Booleans match Booleans, and strings match strings. Querying for a simple type is as easy as specifying the value that you are looking for.

For example, to find all documents where the value for "age" is 27, we can add that key/value pair to the query document:

```
> db.users.find({"age" : 27})
```

If we have a string we want to match, such as a "username" key with the value "joe", we use that key/value pair instead: > db.users.find({"username" : "joe"}) Multiple conditions can be strung together by adding more key/value pairs to the query document, which gets interpreted as "condition1 AND condition2 AND ... AND conditionN." For instance, to get all users who are 27-year-olds with the username "joe," we can query for the following: > db.users.find({"username" : "joe", "age" : 27}) Specifying Which Keys to Return Sometimes, you do not need all of the key/value pairs in a document returned.

If this is the case, you can pass a second argument to find (**or findOne**) specifying the keys you want. This reduces both the amount of data sent over the wire and the time and memory used to decode documents on the client side.

For example, if you have a user collection and you are interested only in the "user name" and "email" keys, you could return just those keys with the following query:

```
> db.users.find({}, {"username" : 1, "email" : 1})
```

```
{ "_id" : ObjectId("4ba0f0dfd22aa494fd523620"), "username" : "joe", "email" : "joe@example.com" }
```

As you can see from the previous output, the "_id" key is always returned, even if it isn't specifically listed. You can also use this second parameter to exclude specific key/value pairs from the results of a query. For instance, you may have documents with a variety of keys, and the only thing you know is that you never want to return the "fatal weakness" key:

```
> db.users.find({}, {"fatal_weakness" : 0})
```

This can even prevent "_id" from being returned:

```
> db.users.find({}, {"username" : 1, "_id" : 0}) { "username" : "joe", }
```

Query Criteria Queries can go beyond the exact matching described in the previous section; they can match more complex criteria, such as ranges, OR-clauses, and negation. Query Conditionals "\$lt", "\$lte", "\$gt", and "\$gte" are all comparison operators, corresponding to <=, >, and >=, respectively. They can be combined to look for a range of values.

For example, to look for users who are between the ages of 18 and 30 inclusive, we can do this:

```
> db.users.find({"age" : {"$gte" : 18, "$lte" : 30}})
```

These types of range queries are often useful for dates.

For example, to find people who registered before January 1, 2007, we can do this:

```
> start = new Date ("01/01/2007")
```

```
> db.users.find({"registered" : {"$lt" : start}})
```

An exact match on a date is less useful, because dates are only stored with millisecond precision. Often you want a whole day, week, or month, making a range query necessary. To query for documents where a key's value is not equal to a certain value, you must use another conditional operator, "\$ne", which stands for "not equal." If you want to find all users who do not have the username "joe," you can query for them using this:

```
> db.users.find({"username" : {"$ne" : "joe"}})
```

"\$ne" can be used with any type. OR Queries There are two ways to do an OR query in MongoDB. "\$in" can be used to query for a variety of values for a single key. "\$or" is more general; it can be used to query for any of the given values across multiple keys. If you have more than one possible value to match for a single key, use an array of criteria with "\$in". For instance, suppose we were running a raffle and the winning ticket numbers were 725, 542, and 390. To find all three of these documents, we can construct the following query

```
> db.raffle.find({"ticket_no" : {"$in" : [725, 542, 390]}})
```

"\$in" is very flexible and allows you to specify criteria of different types as well as values. For example, if we are gradually migrating our schema to use usernames instead of user ID numbers, we can query for either by using this:

```
> db.users.find({"user_id" : {"$in" : [12345, "joe"]}})
```

This matches documents with a "user_id" equal to 12345, and documents with a "user_id" equal to "joe". If "\$in" is given an array with a single value, it behaves the same as directly matching the value. For instance, {ticket_no : { \$in : [725] }} matches the same documents as {ticket_no : 725}. The opposite of "\$in" is "\$nin", which returns documents that don't match any of the criteria in the array. If we want to return all of the people who didn't win anything in the raffle, we can query for them with this: > db.raffle.find({"ticket_no" : {"\$nin" : [725, 542, 390]}}) This query returns everyone who did not have tickets with those numbers. "\$in" gives you an OR query for a single key, but what if we need to find documents where "ticket_no" is 725 or "winner" is true? For this type of query, we'll need to use the "\$or" conditional. "\$or" takes an array of possible criteria. In the raffle

case, using "\$or" would look like this: `> db.raffle.find({"$or" : [{"ticket_no" : 725}, {"winner" : true}]})` "\$or" can contain other conditionals. If, for example, we want to match any of the three "ticket_no" values or the "winner" key, we can use this:

`> db.raffle.find({"$or" : [{"ticket_no" : {"$in" : [725, 542, 390]}}, {"winner" : true}]})` With a normal AND-type query, you want to narrow your results down as far as possible in as few arguments as possible. OR-type queries are the opposite: they are most efficient if the first arguments match as many documents as possible.

\$not "\$not" is a metaconditional: it can be applied on top of any other criteria. As an example, let's consider the modulus operator, "\$mod". "\$mod" queries for keys whose values, when divided by the first value given, have a remainder of the second value: `> db.users.find({"id_num" : {"$mod" : [5, 1]})` The previous query returns users with "id_num"s of 1, 6, 11, 16, and so on. If we want, instead, to return users with "id_num"s of 2, 3, 4, 5, 7, 8, 9, 10, 12, and so on, we can use "\$not": `> db.users.find({"id_num" : {"$not" : {"$mod" : [5, 1]}}})` "\$not" can be particularly useful in conjunction with regular expressions to find all documents that don't match a given pattern

Conditional semantics

In the query, "\$lt" is in the inner document; in the update, "\$inc" is the key for the outer document. This generally holds true: conditionals are an inner document key, and modifiers are always a key in the outer document. Multiple conditions can be put on a single key. For example, to find all users between the ages of 20 and 30, we can query for both "\$gt" and "\$lt" on the "age" key:

```
> db.users.find({"age" : {"$lt" : 30, "$gt" : 20}})
```

Any number of conditionals can be used with a single key. Multiple update modifiers cannot be used on a single key, however. For example, you cannot have a modifier document such as `{"$inc" : {"age" : 1}, "$set" : {age : 40}}` because it modifies "age" twice. With query conditionals, no such rule applies.

Type-Specific Queries

MongoDB has a wide variety of types that can be used in a document. Some of these behave specially in queries. null null behaves a bit strangely. It does match itself, so if we have a collection with the following documents:

```
> db.c.find() { "_id" : ObjectId("4ba0f0dfd22aa494fd523621"), "y" : null }
```

```
{ "_id" : ObjectId("4ba0f0dfd22aa494fd523622"), "y" : 1 }  
{ "_id" : ObjectId("4ba0f148d22aa494fd523623"), "y" : 2 }
```

we can query for documents whose "y" key is null in the expected way: `> db.c.find({"y" : null})` { "_id" : ObjectId("4ba0f0dfd22aa494fd523621"), "y" : null } However, null not only

matches itself but also matches “does not exist.” Thus, querying for a key with the value null will return all documents lacking that key:

```
> db.c.find({"z" : null})
```

```
{ "_id": ObjectId("4ba0f0dfd22aa494fd523621"), "y": null }  
{ "_id": ObjectId("4ba0f0dfd22aa494fd523622"), "y": 1 }  
{ "_id": ObjectId("4ba0f148d22aa494fd523623"), "y": 2 }
```

If we only want to find keys whose value is null, we can check that the key is null and exists using the "\$exists" conditional: `> db.c.find({"z" : {"$in" : [null], "$exists" : true}})` Unfortunately, there is no "\$eq" operator, which makes this a little awkward, but "\$in" with one element is equivalent.

Regular Expressions

Regular expressions are useful for flexible string matching. For example, if we want to find all users with the name Joe or joe, we can use a regular expression to do caseinsensitive matching:

```
> db.users.find({"name" : /joe/i})
```

 Regular expression flags (i) are allowed but not required.

If we want to match not only various capitalizations of joe, but also joey, we can continue to improve our regular expression:

```
> db.users.find({"name" : /joey?/i})
```

 MongoDB uses the Perl Compatible Regular Expression (PCRE) library to match regular expressions; any regular expression syntax allowed by PCRE is allowed in MongoDB. It is a good idea to check your syntax with the JavaScript shell before using it in a query to make sure it matches what you think it matches. Regular expressions can also match themselves.

Very few people insert regular expressions into the database, but if you insert one, you can match it with itself:

```
> db.foo.insert({"bar" : /baz/})  
> db.foo.find({"bar" : /baz/})
```

```
{ "_id" : ObjectId("4b23c3ca7525f35f94b60a2d"), "bar" : /baz/ }
```

Querying Arrays

Querying for elements of an array is simple. An array can mostly be treated as though each element is the value of the overall key.

For example, if the array is a list of fruits, like this: `> db.food.insert({"fruit" : ["apple", "banana", "peach"]})` the following query: `> db.food.find({"fruit" : "banana"})` will successfully match the document. We can query for it in much the same way as though we had a document that looked like the (illegal) document: `{"fruit" : "apple", "fruit" : "banana", "fruit" : "peach"}`. `$all` If you need to match arrays by more than one element, you can use `"$all"`. This allows you to match a list of elements. For example, suppose we created a collection with three elements:

```
> db.food.insert({"_id" : 1, "fruit" : ["apple", "banana", "peach"]})
> db.food.insert({"_id" : 2, "fruit" : ["apple", "kumquat", "orange"]})
> db.food.insert({"_id" : 3, "fruit" : ["cherry", "banana", "apple"]})
```

Then we can find all documents with both "apple" and "banana" elements by querying with `"$all"`: `> db.food.find({fruit : {$all : ["apple", "banana"]}})`

```
{"_id" : 1, "fruit" : ["apple", "banana", "peach"]}
{"_id" : 3, "fruit" : ["cherry", "banana", "apple"]}
```

Order does not matter. Notice "banana" comes before "apple" in the second result. Using a one-element array with `"$all"` is equivalent to not using `"$all"`. For instance, `{fruit : {$all : ['apple']}}` will match the same documents as `{fruit : 'apple'}`. You can also query by exact match using the entire array. However, exact match will not match a document if any elements are missing or superfluous.

For example, this will match the first document shown previously:

```
> db.food.find({"fruit" : ["apple", "banana", "peach"]})
```

But this will not: `> db.food.find({"fruit" : ["apple", "banana"]})` and neither will this:

```
> db.food.find({"fruit" : ["banana", "apple", "peach"]})
```

If you want to query for a specific element of an array, you can specify an index using the syntax `key.index`:

```
> db.food.find({"fruit.2" : "peach"})
```

Arrays are always 0-indexed, so this would match the third array element against the string "peach".

`$size` A useful conditional for querying arrays is `"$size"`, which allows you to query for arrays of a given size. Here's an example:

```
> db.food.find({"fruit" : {"$size" : 3}})
```

One common query is to get a range of sizes. `"$size"` cannot be combined with another `$` conditional (in this example, `"$gt"`), but this query can be accomplished by adding a "size" key to the document. Then, every time you add an element to the array, increment the value of "size". If the original update looked like this:

```
> db.food.update({"$push" : {"fruit" : "strawberry"}}) it can simply be changed to this:
> db.food.update({"$push" : {"fruit" : "strawberry"}, "$inc" : {"size" : 1}})
```

Incrementing is extremely fast, so any performance penalty is negligible. Storing documents like this allows you to do queries such as this:

```
> db.food.find({"size" : {"$gt" : 3}}) Unfortunately, this technique doesn't work as well with the "$addToSet" operator.
```

The \$slice operator the optional second argument to find specifies the keys to be returned. The special "\$slice" operator can be used to return a subset of elements for an array key. For example, suppose we had a blog post document and we wanted to return the first 10 comments:

```
> db.blog.posts.findOne(criteria, {"comments" : {"$slice" : 10}}) Alternatively, if we wanted the last 10 comments, we could use -10:
```

```
> db.blog.posts.findOne(criteria, {"comments" : {"$slice" : -10}})
```

"\$slice" can also return pages in the middle of the results by taking an offset and the number of elements to return:

```
> db.blog.posts.findOne(criteria, {"comments" : {"$slice" : [23, 10]}})
```

 This would skip the first 23 elements and return the 24th through 34th. If there are fewer than 34 elements in the array, it will return as many as possible.

Unless otherwise specified, all keys in a document are returned when "\$slice" is used. This is unlike the other key specifiers, which suppress unmentioned keys from being returned. For instance, if we had a blog post document that looked like this:

```
{ "_id" : ObjectId("4b2d75476cc613d5ee930164"), "title" : "A blog post", "content" : "...", "comments" : [ { "name" : "joe", "email" : "joe@example.com", "content" : "nice post." }, { "name" : "bob", "email" : "bob@example.com", "content" : "good post." } ] } and we did a "$slice" to get the last comment, we'd get this:
```

```
> db.blog.posts.findOne(criteria, {"comments" : {"$slice" : -1}}) { "_id" : ObjectId("4b2d75476cc613d5ee930164"), "title" : "A blog post", "content" : "...", "comments" : [ { "name" : "bob", "email" : "bob@example.com", "content" : "good post." } ] } Both "title" and "content" are still returned, even though they weren't explicitly included in the key specifier.
```

Conclusion: - Executed queries on MongoDB database that demonstrates querying techniques: like find and findOne, Query conditionals, OR queries, \$not, Conditional semantics, Null, Regular expression, Querying arrays

Department of Information Technology

Software Laboratory-I

Assignment No:-2 (Part C)

Problem Statement

Create employee collection in company database and insert at least 10 employee information in employee collection containing eid,ename,designation,hiredate,salary,hobbies and department.

And Execute following queries on employee collection.

```
> db.EMP.find().pretty()
```

```
{
  "_id" : ObjectId("59ca886e14f50a016f3b5450"),
  "EID" : 1,
  "ENAME" : "Jon",
  "Designation" : "Lead Developer",
  "Hiredate" : ISODate("2013-02-15T00:00:00Z"),
  "Salary" : 15000,
  "Hobbies" : [
    "Reading",
    "Coding"
  ],
  "Department" : "Production"
}
{
  "_id" : ObjectId("59ca88cd14f50a016f3b5451"),
  "EID" : 2,
  "ENAME" : "Tirrion",
  "Designation" : "Analyst",
  "Hiredate" : ISODate("2012-05-25T00:00:00Z"),
```

```
    "Salary" : 16000,
    "Hobbies" : [
        "Reading",
        "Braingames"
    ],
    "Department" : "Testing"
}
{
    "_id" : ObjectId("59ca893614f50a016f3b5452"),
    "EID" : 3,
    "ENAME" : "Jamie",
    "Designation" : "Salesman",
    "HiredDate" : ISODate("2015-08-03T00:00:00Z"),
    "Salary" : 9000,
    "Hobbies" : [
        "Speaking",
        "Planning"
    ],
    "Department" : "Sales"
}
{
    "_id" : ObjectId("59ca8a1414f50a016f3b5453"),
    "EID" : 4,
    "ENAME" : "Dany",
    "Designation" : "CEO",
    "HiredDate" : ISODate("2009-06-03T00:00:00Z"),
    "Salary" : 25000,
    "Hobbies" : [
        "Reading",
        "Planning"
    ],
    "Department" : "Management"
}
{
```



```
    "_id" : ObjectId("59ca8a5e14f50a016f3b5454"),
    "EID" : 5,
    "ENAME" : "Arya",
    "Designation" : "Developer",
    "Hiredate" : ISODate("2017-10-17T00:00:00Z"),
    "Salary" : 10000,
    "Hobbies" : [
        "Painting",
        "Coding"
    ],
    "Department" : "Production"
}
{
    "_id" : ObjectId("59ca8ada14f50a016f3b5455"),
    "EID" : 6,
    "ENAME" : "Sansa",
    "Designation" : "Manager",
    "Hiredate" : ISODate("2014-12-02T00:00:00Z"),
    "Salary" : 14000,
    "Hobbies" : [
        "Music",
        "Planning"
    ],
    "Department" : "HR"
}
{
    "_id" : ObjectId("59ca8b2b14f50a016f3b5456"),
    "EID" : 7,
    "ENAME" : "Cersie",
    "Designation" : "Sr Manager",
    "Hiredate" : ISODate("2012-02-08T00:00:00Z"),
    "Salary" : 17000,
    "Hobbies" : [
        "Dancing",
```

```
        "Planning"
    ],
    "Department" : "HR"
}
{
    "_id" : ObjectId("59ca8b9314f50a016f3b5457"),
    "EID" : 8,
    "ENAME" : "Dahrio",
    "Designation" : "Analyst",
    "Hiredate" : ISODate("2016-08-12T00:00:00Z"),
    "Salary" : 13000,
    "Hobbies" : [
        "MAnagement",
        "Games"
    ],
    "Department" : "Testing"
}
{
    "_id" : ObjectId("59ca8c2514f50a016f3b5458"),
    "EID" : 9,
    "ENAME" : "Theon",
    "Designation" : "Jr Developer",
    "Hiredate" : ISODate("2017-08-19T00:00:00Z"),
    "Salary" : 9000,
    "Hobbies" : [
        "Reading",
        "Games"
    ],
    "Department" : "Production"
}
{
    "_id" : ObjectId("59ca8d4f14f50a016f3b5459"),
    "EID" : 10,
    "ENAME" : "Bronn",
```

```

    "Designation" : "Salesman",
    "Hiredate" : ISODate("2014-05-29T00:00:00Z"),
    "Salary" : 11000,
    "Hobbies" : [
        "Speaking",
        "Reading"
    ],
    "Department" : "Sales"
}

```

a) List the names of analysts and salesmen.

```

> db.EMP.find({Designation:{$in:['Analyst','Salesman']}},{EName:1,_id:0})
{ "EName" : "Tirion" }
{ "EName" : "Jamie" }
{ "EName" : "Dahrio" }
{ "EName" : "Bronn" }
>

```

b) List details of employees who have joined before 1 Mar 2015.

```

> db.EMP.find({Hiredate:{$lt:ISODate("2015-03-01T00:00:00Z")}}).pretty()
{
  "_id" : ObjectId("59ca886e14f50a016f3b5450"),
  "EID" : 1,
  "EName" : "Jon",
  "Designation" : "Lead Developer",
  "Hiredate" : ISODate("2013-02-15T00:00:00Z"),
  "Salary" : 15000,
  "Hobbies" : [
    "Reading",
    "Coding"
  ],
  "Department" : "Production"
}

```

```
{
  "_id" : ObjectId("59ca88cd14f50a016f3b5451"),
  "EID" : 2,
  "ENAME" : "Tirrion",
  "Designation" : "Analyst",
  "Hiredate" : ISODate("2012-05-25T00:00:00Z"),
  "Salary" : 16000,
  "Hobbies" : [
    "Reading",
    "Braingames"
  ],
  "Department" : "Testing"
}
```

```
{
  "_id" : ObjectId("59ca8a1414f50a016f3b5453"),
  "EID" : 4,
  "ENAME" : "Dany",
  "Designation" : "CEO",
  "Hiredate" : ISODate("2009-06-03T00:00:00Z"),
  "Salary" : 25000,
  "Hobbies" : [
    "Reading",
    "Planning"
  ],
  "Department" : "Management"
}
```

```
{
  "_id" : ObjectId("59ca8ada14f50a016f3b5455"),
  "EID" : 6,
  "ENAME" : "Sansa",
  "Designation" : "Manager",
  "Hiredate" : ISODate("2014-12-02T00:00:00Z"),
  "Salary" : 14000,
  "Hobbies" : [
```

```

        "Music",
        "Planning"
    ],
    "Department" : "HR"
}
{
    "_id" : ObjectId("59ca8b2b14f50a016f3b5456"),
    "EID" : 7,
    "ENAME" : "Cersie",
    "Designation" : "Sr Manager",
    "Hiredate" : ISODate("2012-02-08T00:00:00Z"),
    "Salary" : 17000,
    "Hobbies" : [
        "Dancing",
        "Planning"
    ],
    "Department" : "HR"
}
{
    "_id" : ObjectId("59ca8d4f14f50a016f3b5459"),
    "EID" : 10,
    "ENAME" : "Bronn",
    "Designation" : "Salesman",
    "Hiredate" : ISODate("2014-05-29T00:00:00Z"),
    "Salary" : 11000,
    "Hobbies" : [
        "Speaking",
        "Reading"
    ],
    "Department" : "Sales"
}

```

c) List names of employees who are not managers.

```
> db.EMP.find({Designation:{$nin:['Manager']}}, {ENAME:1, _id:0})
{ "ENAME" : "Jon" }
{ "ENAME" : "Tirrion" }
{ "ENAME" : "Jamie" }
{ "ENAME" : "Dany" }
{ "ENAME" : "Arya" }
{ "ENAME" : "Cersie" }
{ "ENAME" : "Dahrio" }
{ "ENAME" : "Theon" }
{ "ENAME" : "Bronn" }
```

d) List the names of employees whose employee numbers are 1,3,7,9.

```
> db.EMP.find({EID:{$in:[1,3,7,9]}}, {EID:1, ENAME:1, _id:0})
{ "EID" : 1, "ENAME" : "Jon" }
{ "EID" : 3, "ENAME" : "Jamie" }
{ "EID" : 7, "ENAME" : "Cersie" }
{ "EID" : 9, "ENAME" : "Theon" }
```

e) List employees not belonging to department Testing, Production or Sales.

```
> db.EMP.find({Department:{$nin:['Testing','Sales','Production']}},
{ENAME:1, Department:1, _id:0})
{ "ENAME" : "Dany", "Department" : "Management" }
{ "ENAME" : "Sansa", "Department" : "HR" }
{ "ENAME" : "Cersie", "Department" : "HR" }
```

f) List employee names for those who have joined between 30 June and 31 Dec 2015.

```
> db.EMP.find({"Hiredate":{$gt:ISODate("2015-06-30T00:00:00Z"),
$lt:ISODate("2015-12-31T00:00:00Z")}}).pretty()
{
  "_id" : ObjectId("59ca893614f50a016f3b5452"),
  "EID" : 3,
  "ENAME" : "Jamie",
```

```

    "Designation" : "Salesman",
    "HiredDate" : ISODate("2015-08-03T00:00:00Z"),
    "Salary" : 9000,
    "Hobbies" : [
        "Speaking",
        "Planning"
    ],
    "Department" : "Sales"
}

```

g) List the different designations in the company.

```

> db.EMP.distinct("Designation")
[
    "Lead Developer",
    "Analyst",
    "Salesman",
    "CEO",
    "Developer",
    "Manager",
    "Sr Manager",
    "Jr Developer"
]
>

```

h) List the eid,ename,salary of all employees whose salary is less than 10000.

```

> db.EMP.find({"Salary":{"$lt:10000}}),
  {EID:1,ENAME:1,Salary:1,_id:0}).pretty()
{ "EID" : 3, "ENAME" : "Jamie", "Salary" : 9000 }
{ "EID" : 9, "ENAME" : "Theon", "Salary" : 9000 }

```

i) List the name and designation of the employee who works in production department.

```
> db.EMP.find({Department:"Production"},{ENAME:1,Designation:1,_id:0})
{ "ENAME" : "Jon", "Designation" : "Lead Developer" }
{ "ENAME" : "Arya", "Designation" : "Developer" }
{ "ENAME" : "Theon", "Designation" : "Jr Developer" }
>
```

j) List the all employees whose name start with "A" letter.

```
> db.EMP.find({ENAME:/^A/},{ENAME:1,_id:0})
{ "ENAME" : "Arya" }
```

k) List the all employees whose name containing "on" string.

```
db.EMP.find({ENAME:/on/},{ENAME:1,_id:0})
{ "ENAME" : "Jon" }
{ "ENAME" : "Tirrion" }
{ "ENAME" : "Theon" }
{ "ENAME" : "Bronn" }
>
```

l) List the all employees whose names either start or end with "S".

```
db.EMP.find({$or:[{ENAME:/^S/},{ENAME:/$S/}]},{ENAME:1,_id:0})
{ "ENAME" : "Sansa" }
>
```

m) List names of employees whose names have "i" as the second character.

```
> db.EMP.find({ENAME:/[A-Z]i/},{ENAME:1,_id:0})
{ "ENAME" : "Tirrion" }
>
```

n) List the number of employees working in sales department.

```
> db.EMP.count({Department:'Sales'})
```


o) List the number of designations available in the EMP collections.

```
> db.EMP.distinct("Designation").length
```

```
8
```

p) List the eid,ename,salary of all employees whose salary in between 10000 to 20000.

```
> db.EMP.find({Salary:{$gte:10000,$lte:20000}}, {EID:1,ENAME:1,Salary:1,_id:0})
```

```
{ "EID" : 1, "ENAME" : "Jon", "Salary" : 15000 }
{ "EID" : 2, "ENAME" : "Tirion", "Salary" : 16000 }
{ "EID" : 5, "ENAME" : "Arya", "Salary" : 10000 }
{ "EID" : 6, "ENAME" : "Sansa", "Salary" : 14000 }
{ "EID" : 7, "ENAME" : "Cersie", "Salary" : 17000 }
{ "EID" : 8, "ENAME" : "Dahrio", "Salary" : 13000 }
{ "EID" : 10, "ENAME" : "Bronn", "Salary" : 11000 }
>
```

q) List the eid,ename of all employees whose salary is greater than or equal to 15000.

```
> db.EMP.find({Salary:{$gte:15000}}, {EID:1,ENAME:1,Salary:1,_id:0})
```

```
{ "EID" : 1, "ENAME" : "Jon", "Salary" : 15000 }
{ "EID" : 2, "ENAME" : "Tirion", "Salary" : 16000 }
{ "EID" : 4, "ENAME" : "Dany", "Salary" : 25000 }
{ "EID" : 7, "ENAME" : "Cersie", "Salary" : 17000 }
>
```

r) List details of employees whose department is Sales and salary is 11000.

```
> db.EMP.find({Department:"Sales",Salary:11000})
```

```
{ "_id" : ObjectId("59ca8d4f14f50a016f3b5459"), "EID" : 10, "ENAME" :
"Bronn", "Designation" : "Salesman", "HiredDate" : ISODate("2014-05-
29T00:00:00Z"), "Salary" : 11000, "Hobbies" : [ "Speaking", "Reading" ],
"Department" : "Sales" }
>
```

s) List the number of employees present in employee collection.

```
> db.EMP.count()
```

```
10
```

t) List the names of employees whose department is not HR.

```
> db.EMP.find({Department:{$nin:['HR']}},{EName:1,Department:1,_id:0})
```

```
{ "EName" : "Jon", "Department" : "Production" }
```

```
{ "EName" : "Tirrion", "Department" : "Testing" }
```

```
{ "EName" : "Jamie", "Department" : "Sales" }
```

```
{ "EName" : "Dany", "Department" : "Management" }
```

```
{ "EName" : "Arya", "Department" : "Production" }
```

```
{ "EName" : "Dahrio", "Department" : "Testing" }
```

```
{ "EName" : "Theon", "Department" : "Production" }
```

```
{ "EName" : "Bronn", "Department" : "Sales" }
```

u) List the eid,ename and salary from employee collection.

```
> db.EMP.find({}, {EID:1,EName:1,Salary:1,_id:0})
```

```
{ "EID" : 1, "EName" : "Jon", "Salary" : 15000 }
```

```
{ "EID" : 2, "EName" : "Tirrion", "Salary" : 16000 }
```

```
{ "EID" : 3, "EName" : "Jamie", "Salary" : 9000 }
```

```
{ "EID" : 4, "EName" : "Dany", "Salary" : 25000 }
```

```
{ "EID" : 5, "EName" : "Arya", "Salary" : 10000 }
```

```
{ "EID" : 6, "EName" : "Sansa", "Salary" : 14000 }
```

```
{ "EID" : 7, "EName" : "Cersie", "Salary" : 17000 }
```

```
{ "EID" : 8, "EName" : "Dahrio", "Salary" : 13000 }
```

```
{ "EID" : 9, "EName" : "Theon", "Salary" : 9000 }
```

```
{ "EID" : 10, "ENAME" : "Bronn", "Salary" : 11000 }
```

```
>
```

Assignment No. 3

Date:

Title: Execute at least 10 queries on any suitable MongoDB database that demonstrates following:
\$ where queries
Cursors (Limits, skips, sorts, advanced query options)
Database commands

Remarks:

Aim: - Execute at least 10 queries on any suitable MongoDB database that demonstrates following:

\$ where queries

Cursors (Limits, skips, sorts, advanced query options)

Database commands

Theory: -

\$where Queries

Key/value pairs are a fairly expressive way to query, but there are some queries that they cannot represent. For queries that cannot be done any other way, there are "\$where" clauses, which allow you to execute arbitrary JavaScript as part of your query. This allows you to do (almost) anything within a query. The most common case for this is wanting to compare the values for two keys in a document, for instance, if we had a list of items and wanted to return documents where any two of the values are equal.

Here's an example:

```
> db.foo.insert({"apple" : 1, "banana" : 6, "peach" : 3})
> db.foo.insert({"apple" : 8, "spinach" : 4, "watermelon" : 4})
```

In the second document, "spinach" and "watermelon" have the same value, so we'd like that document returned. It's unlikely MongoDB will ever have a \$ conditional for this, so we can use a "\$where" clause to do it with JavaScript:

```
> db.foo.find({"$where" : function () { ... for (var current in this) { ... for (var other in this) { ...
if (current != other && this[current] == this[other]) { ... return true; ... } ... } ... return false;
... }});
```

If the function returns true, the document will be part of the result set; if it returns false, it won't be.

We used a function earlier, but you can also use strings to specify a "\$where" query; the following two "\$where" queries are equivalent:

```
> db.foo.find({"$where" : "this.x + this.y == 10"})
> db.foo.find({"$where" : "function() { return this.x + this.y == 10; }"})
```

"\$where" queries should not be used unless strictly necessary: they are much slower than regular queries. Each document has to be converted from BSON to a JavaScript object and then run through the "\$where" expression. Indexes cannot be used to satisfy a "\$where", either.

Hence, you should use "\$where" only when there is no other way of doing the query. You can cut down on the penalty by using other query filters in combination with "\$where". If possible, an index will be used to filter based on the non- \$where clauses; the "\$where" expression will be used only to fine-tune the results.

Cursors

The database returns results from find using a cursor. The client-side implementations of cursors generally allow you to control a great deal about the eventual output of a query. You can limit the number of results, skip over some number of results, sort results by any combination of keys in any direction, and perform a number of other powerful operations.

To create a cursor with the shell, put some documents into a collection, do a query on them, and assign the results to a local variable (variables defined with "var" are local). Here, we create a very simple collection and query it, storing the results in the cursor variable: > for(i=0; i var cursor = db.collection.find();

The advantage of doing this is that you can look at one result at a time. If you store the results in a global variable or no variable at all, the MongoDB shell will automatically iterate through and display the first couple of documents. This is what we've been seeing up until this point, and it is often the behavior you want for seeing what's in a collection but not for doing actual programming with the shell. To iterate through the results, you can use the next method on the cursor. You can use hasNext to check whether there is another result.

A typical loop through results looks like the following:

```
> while (cursor.hasNext()) { ... obj = cursor.next(); ... // do stuff ... }
cursor.hasNext() checks that the next result exists, and cursor.next() fetches it.
```

The cursor class also implements the iterator interface, so you can use it in a forEach loop:

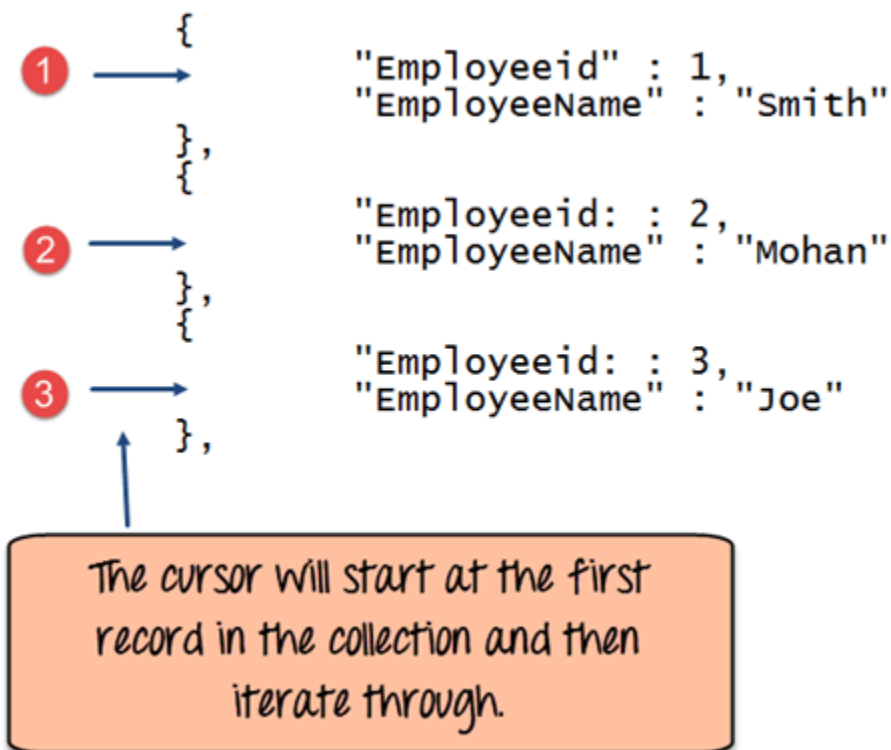
```
> var cursor = db.people.find(); > cursor.forEach(function(x) { ... print(x.name); ... }); adam
matt zak
```

When you call find, the shell does not query the database immediately. It waits until you actually start requesting results to send the query, which allows you to chain additional options onto a query before it is performed. Almost every method on a cursor object returns the cursor itself so that you can chain them in any order. For instance, all of the following are equivalent: > var cursor = db.foo.find().sort({"x" : 1}).limit(1).skip(10); > var cursor = db.foo.find().limit(1).sort({"x" : 1}).skip(10); > var cursor = db.foo.find().skip(10).limit(1).sort({"x" : 1}); At this point, the query has not been executed yet. All of these functions merely build the query. Now, suppose we call the following: > cursor.hasNext() At this point, the query will be sent to the server. The shell fetches the first 100 results or first 4MB of results (whichever is smaller) at once so that the next calls to next or hasNext will not have to make trips to the server. After the client has run through the first set of results, the shell will again contact the database and ask for more results. This process continues until the cursor is exhausted and all results have been returned.

Example MongoDB cursor

When the **db.collection.find ()** function is used to search for documents in the collection, the result returns a pointer to the collection of documents returned which is called a cursor.

By default, the cursor will be iterated automatically when the result of the query is returned. But one can also explicitly go through the items returned in the cursor one by one. If you see the below example, if we have 3 documents in our collection, the cursor will point to the first document and then iterate through all of the documents of the collection.



The following example shows how this can be done.

```
var myEmployee = db.Employee.find( { Employeeid : { $gt:2 } });  
  
while(myEmployee.hasNext())  
{  
    print(tojson(myCursor.next()));  
}
```

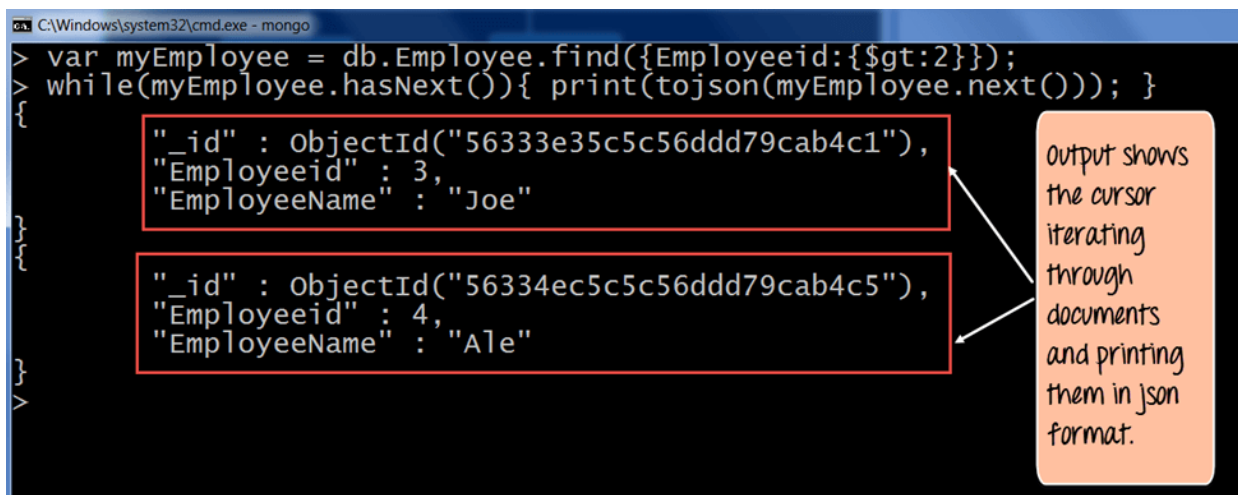
}

Code Explanation:

1. First we take the result set of the query which finds the Employee's whose id is greater than 2 and assign it to the [JavaScript](#) variable 'myEmployee'
2. Next we use the while loop to iterate through all of the documents which are returned as part of the query.
3. Finally, for each document, we print the details of that document in JSON readable format.

If the command is executed successfully, the following Output will be shown

Output:



```
C:\Windows\system32\cmd.exe - mongo
> var myEmployee = db.Employee.find({Employeeid:{$gt:2}});
> while(myEmployee.hasNext()){ print(tojson(myEmployee.next())); }
{
  "_id" : ObjectId("56333e35c5c56ddd79cab4c1"),
  "Employeeid" : 3,
  "EmployeeName" : "Joe"
}
{
  "_id" : ObjectId("56334ec5c5c56ddd79cab4c5"),
  "Employeeid" : 4,
  "EmployeeName" : "Ale"
}
```

Output shows the cursor iterating through documents and printing them in json format.

Limits, Skips, and Sorts

The most common query options are limiting the number of results returned, skipping a number of results, and sorting. All of these options must be added before a query is sent to the database. To set a limit, chain the limit function onto your call to find.

For example, to only return three results, use this:

> db.c.find().limit(3) If there are fewer than three documents matching your query in the collection, only the number of matching documents will be returned; limit sets an upper limit, not a lower limit. skip works similarly to limit:

> db.c.find().skip(3) This will skip the first three matching documents and return the rest of the matches. If there are less than three documents in your collection, it will not return any

documents. sort takes an object: a set of key/value pairs where the keys are key names and the values are the sort directions.

Sort direction can be 1 (ascending) or -1 (descending). If multiple keys are given, the results will be sorted in that order. For instance, to sort the results by "username" ascending and "age" descending, we do the following:

> db.c.find().sort({username : 1, age : -1}) These three methods can be combined. This is often handy for pagination. For example, suppose that you are running an online store and someone searches for mp3. If you want 50 results per page sorted by price from high to low, you can do the following:

> db.stock.find({"desc" : "mp3"}).limit(50).sort({"price" : -1}) If they click Next Page to see more results, you can simply add a skip to the query, which will skip over the first 50 matches (which the user already saw on page 1): > db.stock.find({"desc" : "mp3"}).limit(50).skip(50).sort({"price" : -1}) However, large skips are not very performant, so there are suggestions on avoiding them in a moment

Advanced Query Options

There are two types of queries: wrapped and plain. A plain query is something like this:

> var cursor = db.foo.find({"foo" : "bar"}) There are a couple options that “wrap” the query. For example, suppose we perform a sort:

> var cursor = db.foo.find({"foo" : "bar"}).sort({"x" : 1}) Instead of sending {"foo" : "bar"} to the database as the query, the query gets wrapped in a larger document. The shell converts the query from {"foo" : "bar"} to {"\$query" : {"foo" : "bar"}, "\$orderby" : {"x" : 1}}. Most drivers provide helpers for adding arbitrary options to queries.

Other helpful options include the following:

\$maxscan: integer Specify the maximum number of documents that should be scanned for the query.

\$min: document Start criteria for querying.

\$max: document End criteria for querying.

\$hint: document Tell the server which index to use for the query.

\$explain: Boolean Get an explanation of how the query will be executed (indexes used, number of results, how long it takes, etc.), instead of actually doing the query.

\$snapshot: Boolean Ensure that the query’s results will be a consistent snapshot from the point in time when the query was executed. See the next section for details

Database Commands

MongoDB supports a wide range of advanced operations that are implemented as commands. Commands implement all of the functionality that doesn’t fit neatly into “create, read, update, delete.” We’ve already seen a couple of commands in the previous chapters; for instance, we used the getLastError command in Chapter 3 to check the number of documents affected by an update:

```
> db.count.update({x : 1}, {$inc : {x : 1}}, false, true)
> db.runCommand({getLastError : 1}) { "err" : null, "updatedExisting" : true, "n" : 5, "ok" :
true }
```

We'll also describe some of the most useful commands that are supported by MongoDB. How Commands Work One example of a database command that you are probably familiar with is

drop: to drop a collection from the shell, we run `db.test.drop()`. Under the hood, this function is actually running the drop command—we can perform the exact same operation using `runCommand`: `> db.runCommand({"drop" : "test"})`;

```
{ "nIndexesWas" : 1, "msg" : "indexes dropped for collection", "ns" : "test.test", "ok" : true }
```

The document we get as a result is the command response, which contains information about whether the command was successful, as well as any other information that the command might provide. The command response will always contain the key "ok". If "ok" is true, the command was successful, and if it is false, the command failed for some reason.

If "ok" is false, then an additional key will be present, "errmsg". The value of "errmsg" is a string explaining why the command failed.

As an example, let's try running the drop command again, on the collection that we just dropped: `> db.runCommand({"drop" : "test"})`; { "errmsg" : "ns not found", "ok" : false } Commands in MongoDB are actually implemented as a special type of query that gets performed on the \$cmd collection. `runCommand` just takes a command document and performs the equivalent query, so our drop call becomes the following: `db.$cmd.findOne({"drop" : "test"})`; When the MongoDB server gets a query on the \$cmd collection, it handles it using special logic, rather than the normal code for handling queries. Almost all MongoDB drivers provide a helper method like `runCommand` for running commands, but commands can always be run using a simple query if necessary.

Some commands require administrator access and must be run on the admin database. If such a command is run on any other database, it will return an "access denied" error.

Conclusion: -Execute queries on suitable MongoDB database that demonstrates following:

- \$ where queries, Cursors (Limits, skips, sorts, advanced query options), Database commands

Department of Information Technology

Software Laboratory-I

Assignment No:-3 (Part C)

Problem Statement

Create employee collection in company database and insert at least 10 employee information in employee collection containing eid,ename,designation,hiredate,salary,department.

Execute following queries on employee collection.

```
> db.EMP2.find()
{ "_id" : ObjectId("59ca886e14f50a016f3b5450"), "EID" : 1, "ENAME" : "Jon",
"Designation" : "Lead Developer", "Hiredate" : ISODate("2013-02-
15T00:00:00Z"), "Salary" : 15000, "Department" : "Production" }
{ "_id" : ObjectId("59ca88cd14f50a016f3b5451"), "EID" : 2, "ENAME" :
"Tirion", "Designation" : "Analyst", "Hiredate" : ISODate("2012-05-
25T00:00:00Z"), "Salary" : 16000, "Department" : "Testing" }
{ "_id" : ObjectId("59ca893614f50a016f3b5452"), "EID" : 3, "ENAME" :
"Jamie", "Designation" : "Salesman", "Hiredate" : ISODate("2015-08-
03T00:00:00Z"), "Salary" : 9000, "Department" : "Sales" }
{ "_id" : ObjectId("59ca8a1414f50a016f3b5453"), "EID" : 4, "ENAME" :
"Dany", "Designation" : "CEO", "Hiredate" : ISODate("2009-06-
03T00:00:00Z"), "Salary" : 25000, "Department" : "Management" }
{ "_id" : ObjectId("59ca8a5e14f50a016f3b5454"), "EID" : 5, "ENAME" :
"Arya", "Designation" : "Developer", "Hiredate" : ISODate("2017-10-
17T00:00:00Z"), "Salary" : 10000, "Department" : "Production" }
{ "_id" : ObjectId("59ca8ada14f50a016f3b5455"), "EID" : 6, "ENAME" :
"Sansa", "Designation" : "Manager", "Hiredate" : ISODate("2014-12-
02T00:00:00Z"), "Salary" : 14000, "Department" : "HR" }
```

```
{ "_id" : ObjectId("59ca8b2b14f50a016f3b5456"), "EID" : 7, "ENAME" :
"Cersie", "Designation" : "Sr Manager", "Hiredate" : ISODate("2012-02-
08T00:00:00Z"), "Salary" : 17000, "Department" : "HR" }
{ "_id" : ObjectId("59ca8b9314f50a016f3b5457"), "EID" : 8, "ENAME" :
"Dahrio", "Designation" : "Analyst", "Hiredate" : ISODate("2016-08-
12T00:00:00Z"), "Salary" : 13000, "Department" : "Testing" }
{ "_id" : ObjectId("59ca8c2514f50a016f3b5458"), "EID" : 9, "ENAME" :
"Theon", "Designation" : "Jr Developer", "Hiredate" : ISODate("2017-08-
19T00:00:00Z"), "Salary" : 9000, "Department" : "Production" }
{ "_id" : ObjectId("59ca8d4f14f50a016f3b5459"), "EID" : 10, "ENAME" :
"Bronn", "Designation" : "Salesman", "Hiredate" : ISODate("2014-05-
29T00:00:00Z"), "Salary" : 11000, "Department" : "Sales" }
>
```

a) List the total salaries paid to the employees.

```
> db.EMP2.aggregate({$group:{_id:"EID",Sum:{$sum:"$Salary"}}})
{ "_id" : "EID", "Sum" : 139000 }
>
```

b) List the maximum, minimum and average salary in the company.

```
> db.EMP2.aggregate({$group:{_id:"EID",maxSal:{$max:"$Salary"},minSal:
{$min:"$Salary"},avgSal:{$avg:"$Salary"}}})
{ "_id" : "EID", "maxSal" : 25000, "minSal" : 9000, "avgSal" : 13900 }
```

c) List details of employees whose department is Sales and salary is 9000.

```
> db.EMP2.find({Department:"Sales",Salary:9000}).pretty()
{
  "_id" : ObjectId("59ca893614f50a016f3b5452"),
  "EID" : 3,
  "ENAME" : "Jamie",
  "Designation" : "Salesman",
  "Hiredate" : ISODate("2015-08-03T00:00:00Z"),
  "Salary" : 9000,
  "Department" : "Sales"
}
```

>

d) List the total salary of all employees whose salary is greater than 15000.

```
> db.EMP2.aggregate({$match:{Salary:{$gt:15000}}},{ $group:
{ _id:"EID",SumSal:{$sum:"$Salary"}}})
{ "_id" : "EID", "SumSal" : 58000 }
```

e) List the details of employee whose salary is maximum in the company.

```
> db.EMP2.find().sort({Salary:-1}).limit(1).pretty()
{
  "_id" : ObjectId("59ca8a1414f50a016f3b5453"),
  "EID" : 4,
  "ENAME" : "Dany",
  "Designation" : "CEO",
  "Hiredate" : ISODate("2009-06-03T00:00:00Z"),
  "Salary" : 25000,
  "Department" : "Management"
}
```

f) List the details of employees whose salary is 4th & 5th top most in the company.

```
> db.EMP2.find().sort({Salary:-1}).skip(3).limit(2).pretty()
{
  "_id" : ObjectId("59ca886e14f50a016f3b5450"),
  "EID" : 1,
  "ENAME" : "Jon",
  "Designation" : "Lead Developer",
  "Hiredate" : ISODate("2013-02-15T00:00:00Z"),
  "Salary" : 15000,
  "Department" : "Production"
}
{
  "_id" : ObjectId("59ca8ada14f50a016f3b5455"),
  "EID" : 6,
  "ENAME" : "Sansa",
```

```
    "Designation" : "Manager",
    "HiredDate" : ISODate("2014-12-02T00:00:00Z"),
    "Salary" : 14000,
    "Department" : "HR"
}
```

g) List the 2nd lowermost salary paid employee from company.

```
> db.EMP2.find().sort({Salary:1}).skip(1).limit(1).pretty()
{
  "_id" : ObjectId("59ca8c2514f50a016f3b5458"),
  "EID" : 9,
  "ENAME" : "Theon",
  "Designation" : "Jr Developer",
  "HiredDate" : ISODate("2017-08-19T00:00:00Z"),
  "Salary" : 9000,
  "Department" : "Production"
}
>
```

h) List the top three salary paid employee from the company.

```
> db.EMP2.find().sort({Salary:-1}).limit(3).pretty()
{
  "_id" : ObjectId("59ca8a1414f50a016f3b5453"),
  "EID" : 4,
  "ENAME" : "Dany",
  "Designation" : "CEO",
  "HiredDate" : ISODate("2009-06-03T00:00:00Z"),
  "Salary" : 25000,
  "Department" : "Management"
}
{
  "_id" : ObjectId("59ca8b2b14f50a016f3b5456"),
```

```

      "EID" : 7,
      "ENAME" : "Cersie",
      "Designation" : "Sr Manager",
      "Hiredate" : ISODate("2012-02-08T00:00:00Z"),
      "Salary" : 17000,
      "Department" : "HR"
    }
  {
    "_id" : ObjectId("59ca88cd14f50a016f3b5451"),
    "EID" : 2,
    "ENAME" : "Tirrion",
    "Designation" : "Analyst",
    "Hiredate" : ISODate("2012-05-25T00:00:00Z"),
    "Salary" : 16000,
    "Department" : "Testing"
  }
>

```

i) List the total number of employee working in each departments.

```

> db.EMP2.aggregate([{$group:{$_id:"$Department",total:{$sum:1}}}])
{ "_id" : "Production", "total" : 3 }
{ "_id" : "Management", "total" : 1 }
{ "_id" : "Testing", "total" : 2 }
{ "_id" : "HR", "total" : 2 }
{ "_id" : "Sales", "total" : 2 }
>

```

j) List the maximum,minimum and average salary of each departments.

```

> db.EMP2.aggregate([{$group:{$_id:"$Department",MaxSal:
{$max:"$Salary"},MinSal:{$min:"$Salary"},AvgSal:{$avg:"$Salary"}}}])
{ "_id" : "Production", "MaxSal" : 15000, "MinSal" : 9000, "AvgSal" :
11333.333333333334 }
{ "_id" : "Management", "MaxSal" : 25000, "MinSal" : 25000, "AvgSal" :
25000 }

```

```
{ "_id" : "Testing", "MaxSal" : 16000, "MinSal" : 13000, "AvgSal" : 14500 }
{ "_id" : "HR", "MaxSal" : 17000, "MinSal" : 14000, "AvgSal" : 15500 }
{ "_id" : "Sales", "MaxSal" : 11000, "MinSal" : 9000, "AvgSal" : 10000 }
>
```

k) List the all employees in ascedeing order according to ename.

```
> db.EMP2.find().sort({ENAME:1})
{ "_id" : ObjectId("59ca8a5e14f50a016f3b5454"), "EID" : 5, "ENAME" :
"Arya", "Designation" : "Developer", "Hiredate" : ISODate("2017-10-
17T00:00:00Z"), "Salary" : 10000, "Department" : "Production" }
{ "_id" : ObjectId("59ca8d4f14f50a016f3b5459"), "EID" : 10, "ENAME" :
"Bronn", "Designation" : "Salesman", "Hiredate" : ISODate("2014-05-
29T00:00:00Z"), "Salary" : 11000, "Department" : "Sales" }
{ "_id" : ObjectId("59ca8b2b14f50a016f3b5456"), "EID" : 7, "ENAME" :
"Cersie", "Designation" : "Sr Manager", "Hiredate" : ISODate("2012-02-
08T00:00:00Z"), "Salary" : 17000, "Department" : "HR" }
{ "_id" : ObjectId("59ca8b9314f50a016f3b5457"), "EID" : 8, "ENAME" :
"Dahrio", "Designation" : "Analyst", "Hiredate" : ISODate("2016-08-
12T00:00:00Z"), "Salary" : 13000, "Department" : "Testing" }
{ "_id" : ObjectId("59ca8a1414f50a016f3b5453"), "EID" : 4, "ENAME" :
"Dany", "Designation" : "CEO", "Hiredate" : ISODate("2009-06-
03T00:00:00Z"), "Salary" : 25000, "Department" : "Management" }
{ "_id" : ObjectId("59ca893614f50a016f3b5452"), "EID" : 3, "ENAME" :
"Jamie", "Designation" : "Salesman", "Hiredate" : ISODate("2015-08-
03T00:00:00Z"), "Salary" : 9000, "Department" : "Sales" }
{ "_id" : ObjectId("59ca886e14f50a016f3b5450"), "EID" : 1, "ENAME" : "Jon",
"Designation" : "Lead Developer", "Hiredate" : ISODate("2013-02-
15T00:00:00Z"), "Salary" : 15000, "Department" : "Production" }
{ "_id" : ObjectId("59ca8ada14f50a016f3b5455"), "EID" : 6, "ENAME" :
"Sansa", "Designation" : "Manager", "Hiredate" : ISODate("2014-12-
02T00:00:00Z"), "Salary" : 14000, "Department" : "HR" }
{ "_id" : ObjectId("59ca8c2514f50a016f3b5458"), "EID" : 9, "ENAME" :
"Theon", "Designation" : "Jr Developer", "Hiredate" : ISODate("2017-08-
```



```
19T00:00:00Z"), "Salary" : 9000, "Department" : "Production" }  
{ "_id" : ObjectId("59ca88cd14f50a016f3b5451"), "EID" : 2, "ENAME" :  
"Tirrion", "Designation" : "Analyst", "Hiredate" : ISODate("2012-05-  
25T00:00:00Z"), "Salary" : 16000, "Department" : "Testing" }  
>
```

Assignment No. 4

Date:

Title: Implement Map reduce example with suitable example

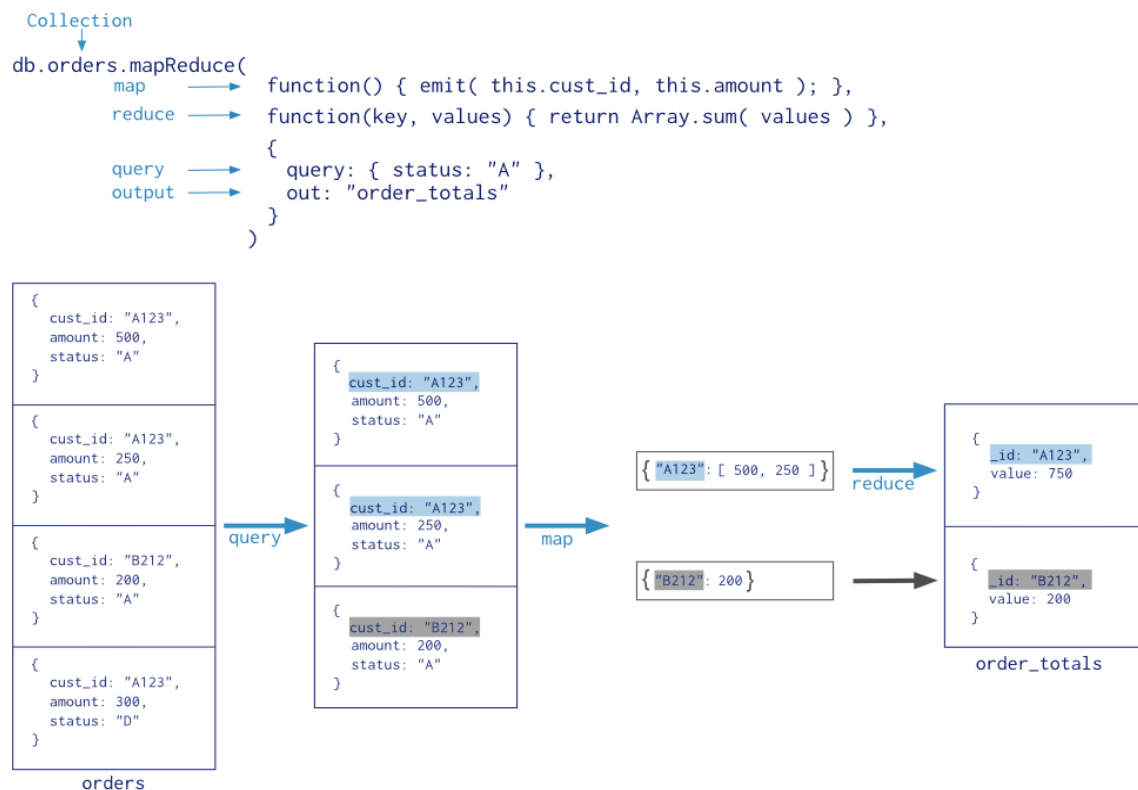
Remarks:

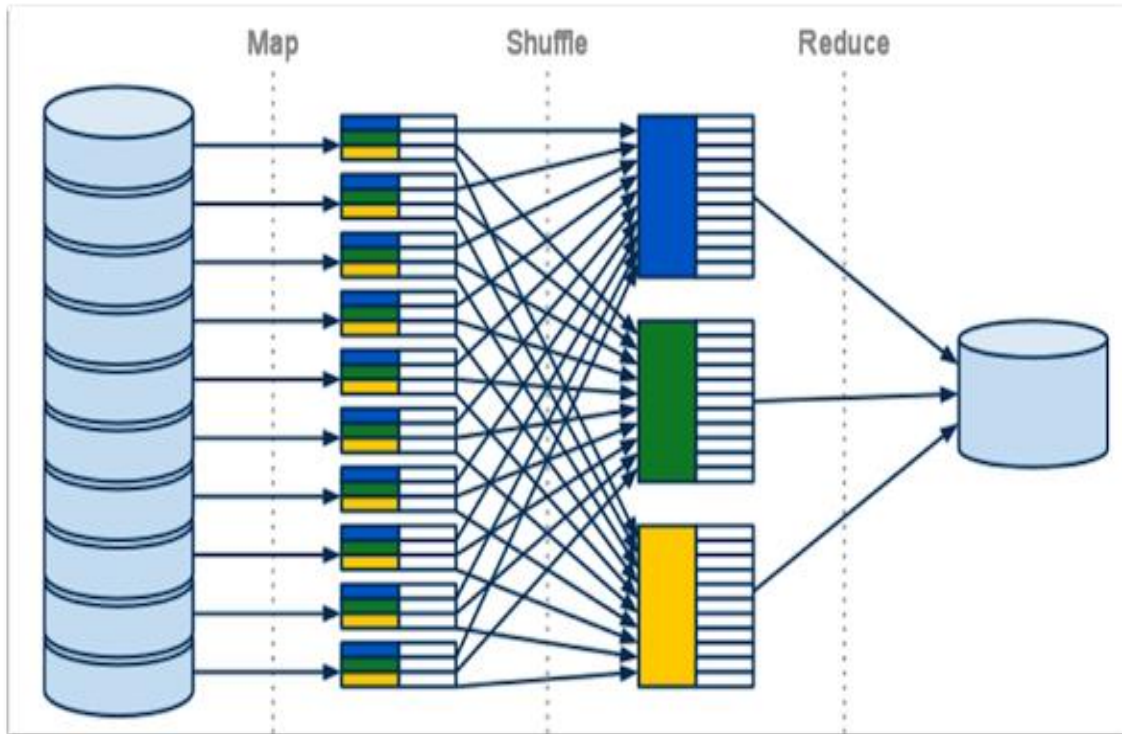
Aim: Implement MapReduce operations with suitable example using MongoDB.

Objective: To learn MapReduce operations

Theory:

- MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.
- A MapReduce program is composed of a Map() procedure that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a Reduce() procedure that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies).
- Map-reduce is a data processing paradigm for condensing large volumes of data into useful *aggregated* results.
- For map-reduce operations, MongoDB provides the [mapReduce](#) database command.





In map Reduce we have to write 3 functions.

1. Map Function (Ex. Person to each city to count population).
2. Reduce Function (Ex. Reducing the total population count to single value.)
3. Map Reduce Function (it will create a new collection it contains the total population)

Step 1: Map

```
var mapFunction1 = function() {
    emit(this.cust_id, this.price);
};
```

Map function to process each input document:

In the function, this refers to the document that the map-reduce operation is processing.

The function maps the price to the cust_id for each document and emits the cust_id and price pair.

Step 2: Reduce

```
var reduceFunction1 = function(keyCustId, valuesPrices) { return Array.sum(valuesPrices);
};
```

Define the corresponding reduce function with two arguments keyCustId and valuesPrices:

The valuesPrices is an array whose elements are the price values emitted by the map function and grouped by keyCustId. The function reduces the valuesPrice array to the sum of its elements.

Step 3: Map Reduce

```
db.orders.mapReduce(  
  mapFunction1,  
  reduceFunction1,  
  { out: "map_example" }  
)
```

Perform the map-reduce on all documents in the orders collection using the mapFunction1 map function and the reduceFunction1 reduce function. This operation outputs the results to a collection named map_example. If the map_example collection already exists, the operation will replace the contents with the results of this map-reduce operation:

Map-Reduce

MongoDB also provides [map-reduce](#) operations to perform aggregation. In general, map-reduce operations have two phases: a map stage that processes each document and emits one or more objects for each input document, and reduce phase that combines the output of the map operation. Optionally, map-reduce can have a finalize stage to make final modifications to the result. Like other aggregation operations, map-reduce can specify a query condition to select the input documents as well as sort and limit the results.

Map-reduce uses custom JavaScript functions to perform the map and reduce operations, as well as the optional finalize operation. While the custom JavaScript provide great flexibility compared to the aggregation pipeline, in general, mapreduce is less efficient and more complex than the aggregation pipeline. Additionally, map-reduce operations can have output sets that exceed the 16 megabyte output limitation of the aggregation pipeline.

Conclusion: Understand and implement Map Reduced Operation

Department of Information Technology

Software Laboratory-I

Assignment No:-4

Problem Statement

Implement Map Reduce operation with following example using MongoDB. From the collection books for which the document structure as shown

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

```
db.books.find();
```

```
{ "_id" : ObjectId("5426522c517b30434f6a2bd7"), "name" : "Understanding JAVA", "pages" : 100 }
```

```
{ "_id" : ObjectId("54265231517b30434f6a2bd8"), "name" : "Understanding JSON", "pages" : 200 }
```

```
{ "_id" : ObjectId("54265249517b30434f6a2bd9"), "name" : "Understanding XML", "pages" : 300 }
```

```
{ "_id" : ObjectId("5426525e517b30434f6a2bda"), "name" : "Understanding Web Services", "pages" : 400 }
```

```
{ "_id" : ObjectId("54265272517b30434f6a2bdb"), "name" : "Understanding Axis2", "pages" : 150 }
```

Using Map Reduce function find the number of books having pages less than 250 pages and greater than or equal to 250 pages.

```
> use Books;
```

switched to db Books

```
> db.createCollection("BOOKS");
```

```
{ "ok" : 1 }
```

```
> db.BOOKS.insert({NAME:'Understanding JAVA',PAGES:100});
```

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

```
> db.BOOKS.insert({NAME:'Understanding JSON',PAGES:200});
```

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

```
> db.BOOKS.insert({NAME:'Understanding XML',PAGES:300});
```

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

```
> db.BOOKS.insert({NAME:'Understanding Web Services',PAGES:400});
```

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

```
> db.BOOKS.insert({NAME:'Understanding Axis2',PAGES:150});
```

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

```
> db.BOOKS.find();
```

```
{ "_id" : ObjectId("5f9d70cdce22f2286268e2a2"), "NAME" : "Understanding JAVA", "PAGES" :  
100 }
```

```
{ "_id" : ObjectId("5f9d70dbce22f2286268e2a3"), "NAME" : "Understanding JSON", "PAGES" :  
200 }
```

```
{ "_id" : ObjectId("5f9d70e8ce22f2286268e2a4"), "NAME" : "Understanding XML", "PAGES" :  
300 }
```

```
{ "_id" : ObjectId("5f9d70f5ce22f2286268e2a5"), "NAME" : "Understanding Web Services",  
"PAGES" : 400 }
```

```
{ "_id" : ObjectId("5f9d7105ce22f2286268e2a6"), "NAME" : "Understanding Axis2", "PAGES" :  
150 }
```

```

> var Map=function()
{
var category;
if(this.PAGES>=250)
    category='Big';
else
    category='Small';
emit(category,{Name:this.NAME});
};

> var Reduce=function(key,values)
{
var cnt=0;
values.forEach(function(doc){cnt+=1;});
return {Books:cnt};
};

> db.BOOKS.mapReduce(Map,Reduce,{out:"ResultCollectionCategory"});
{
  "result" : "ResultCollectionCategory",
  "timeMillis" : 1737,
  "counts" : {
    "input" : 5,
    "emit" : 5,
    "reduce" : 2,
    "output" : 2
  },
  "ok" : 1
}

> db.ResultCollectionCategory.find();
{ "_id" : "Big", "value" : { "Books" : 2 } }
{ "_id" : "Small", "value" : { "Books" : 3 } }

```


Assignment No. 5

Date:

Title: Implement the aggregation and indexing with suitable example in MongoDB. Demonstrate the following:

1. Aggregation framework
2. Create and drop different types of indexes and explain () to show the advantage of the indexes.

Remarks:

AIM: Implement Aggregation and Indexing with suitable example using MongoDB.

Objective: To understand 1) Aggregation 2) To understand Indexing in MongoDB

Theory:

Indexes provide high performance read operations for frequently used queries. This section introduces indexes in MongoDB, describes the types and configuration options for indexes, and describes special types of indexing MongoDB supports. The section also provides tutorials detailing procedures and operational concerns, and providing information on how applications may use indexes. Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must scan every document in a collection to select those documents that match the query statement. These collection scans are inefficient because they require mongod to process a larger volume of data than an index for each operation. Indexes are special data structures ¹ that store a small portion of the collection's data set in an easy to traverse form.

The index stores the value of a specific field or set of fields, ordered by the value of the field. Fundamentally, indexes in MongoDB are similar to indexes in other database systems. MongoDB defines indexes at the collection level and supports indexes on any field or sub-field of the documents in a MongoDB collection. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect. In some cases, MongoDB can use the data from the index to determine which documents match a query. The following diagram illustrates a query that selects documents using an index.

Index Types

MongoDB provides a number of different index types to support specific types of data and queries.

Default _id

All MongoDB collections have an index on the `_id` field that exists by default. If applications do not specify a value for `_id` the driver or the mongod will create an `_id` field with an ObjectId value.

The `_id` index is unique, and prevents clients from inserting two documents with the same value for the `_id` field.

Single Field

In addition to the MongoDB-defined `_id` index, MongoDB supports user-defined indexes on a [single field of a document](#)

Consider the following illustration of a single-field index:

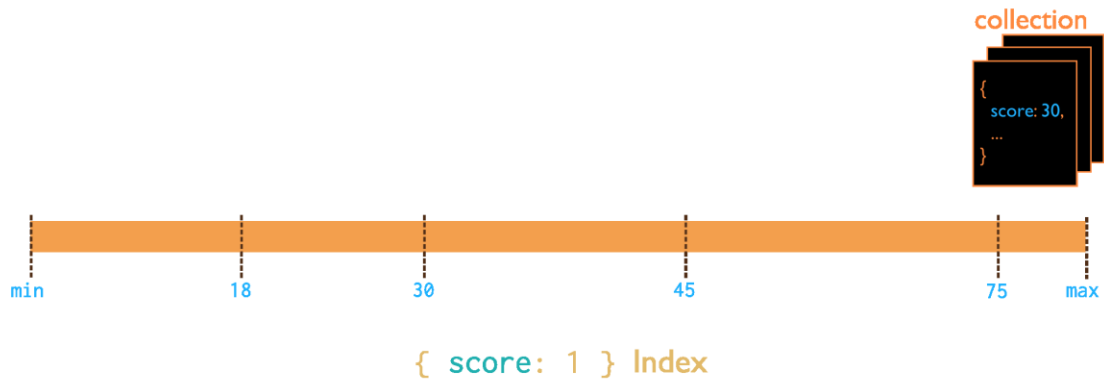


Diagram of an index on the score field (ascending).

Compound Index

MongoDB also supports user-defined indexes on multiple fields. These [compound indexes](#) behave like single-field indexes; however, the query can select documents based on additional fields. The order of fields listed in a compound index has significance. For instance, if a compound index consists of { userid: 1, score:-1 }, the index sorts first by userid and then, within each userid value, sort by score. Consider the following illustration of this compound index:

Multikey Index

MongoDB uses [multikey indexes](#) to index the content stored in arrays. If you index a field that holds an array value, MongoDB creates separate index entries for every element of the array. These [multikey indexes](#) allow queries to select documents that contain arrays by matching on element or elements of the arrays. MongoDB automatically determines whether to create a multikey index if the indexed field contains an array value; you do not need to explicitly specify the multikey type.

Geospatial Index

To support efficient queries of geospatial coordinate data, MongoDB provides two special indexes: [2d indexes](#) that uses planar geometry when returning results and [2sphere indexes](#) that use spherical geometry to return results.

Text Indexes

MongoDB provides a text index type that supports searching for string content in a collection. These text indexes do not store language-specific stop words (e.g. “the”, “a”, “or”) and stem the words in a collection to only store root words.

Hashed Indexes

To support [hash based sharding](#), MongoDB provides a [hashed index](#) type, which indexes the hash of the value of a field. These indexes have a more random distribution of values along their range, but only support equality matches and cannot support range-based queries.

Example Given the following document in the friends collection:

```
{ "_id" : ObjectId(...),  
  "name" : "Alice"  
  "age" : 27  
}
```

The following command creates an index on the name field:

```
db.friends.ensureIndex( { "name" : 1 } )
```

Indexes on Embedded Fields

You can create indexes on fields embedded in sub-documents, just as you can index top-level fields in documents. Indexes on embedded fields differ from [indexes on sub-documents](#), which include the full content up to the maximum index size of the sub-document in the index. Instead, indexes on embedded fields allow you to use a “dot notation,” to introspect into sub-documents.

Consider a collection named people that holds documents that resemble the following example document:

```
{ "_id": ObjectId(...)  
  "name": "John Doe"  
  "address": {  
    "street": "Main",  
    "zipcode": "53511",  
    "state": "WI"  
  }  
}
```

You can create an index on the address.zipcode field, using the following specification:

```
db.people.ensureIndex( { "address.zipcode": 1 } )
```

Aggregation

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.

MongoDB provides three ways to perform aggregation: the [aggregation pipeline](#), the [map-reduce function](#), and [single purpose aggregation methods and commands](#).

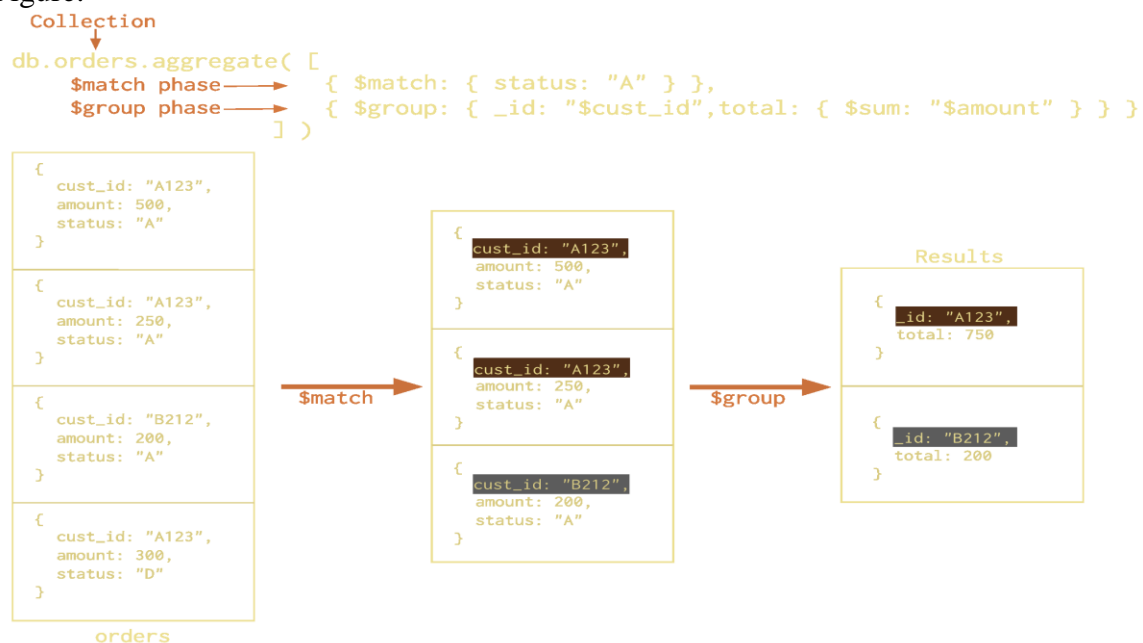
Aggregations are operations that process data records and return computed results. MongoDB provides a rich set of aggregation operations that examine and perform calculations on the data sets. Running data aggregation on the mongod instance simplifies application code and limits resource requirements. Like queries, aggregation operations in MongoDB use collections of documents as an input and return results in the form of one or more documents.

Aggregation Pipelines

MongoDB 2.2 introduced a new [aggregation framework](#), modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into an aggregated result.

The most basic pipeline stages provide filters that operate like queries and document transformations that modify the form of the output document. Other pipeline operations provide tools for grouping and sorting documents by specific field or fields as well as tools for aggregating the contents of arrays, including arrays of documents. In addition, pipeline stages can use provides efficient data aggregation using native operations within MongoDB, and is the preferred method for data aggregation in MongoDB.

Figure:



Conclusion: Understand and implement various Aggregation function and Indexing

Department of Information Technology

Software Laboratory-I

Assignment No:-5 (PART-C)

Problem Statement

Implement aggregation with customer collection using MongoDB. Customer collection consist of following data

```
> db.customer.find();
```

```
{ "_id" : ObjectId("54265694517b30434f6a2bdc"), "custID" : "A123", "Amount" : 500, "status" : "A" }
```

```
{ "_id" : ObjectId("542656eb517b30434f6a2bdd"), "custID" : "A123", "Amount" : 250, "status" : "A" }
```

```
{ "_id" : ObjectId("54265726517b30434f6a2bde"), "custID" : "B212", "Amount" : 200, "status" : "A" }
```

```
{ "_id" : ObjectId("54265757517b30434f6a2bdf"), "custID" : "A123", "Amount" : 300, "status" : "D" }
```

Execute following queries on employee collection.

PART-A

- a) Find the total amount of each customer.
- b) Find the total amount of each customer whose status is A.
- c) Find the minimum total amount of each customer whose Status is A.
- d) Find the maximum total amount of each customer whose Status is A.
- e) Find the average total amount of each customer whose Status is A.

PART-B

- f) Create index on custID.
- g) Execute getIndexes.
- h) Drop the index created.

```

> use Customer
switched to db Customer
> db.createCollection('Customer')
{ "ok" : 1 }
> db.Customer.insert({custID:'A123',Amount:500,Status:'A'})
WriteResult({ "nInserted" : 1 })
> db.Customer.insert({custID:'A123',Amount:250,Status:'A'})
WriteResult({ "nInserted" : 1 })
> db.Customer.insert({custID:'B212',Amount:200,Status:'A'})
WriteResult({ "nInserted" : 1 })
> db.Customer.insert({custID:'A123',Amount:300,Status:'D'})
WriteResult({ "nInserted" : 1 })

> db.Customer.find();
{ "_id" : ObjectId("59dd03f97ce97380afaf677a"), "custID" : "A123",
"Amount" : 500, "Status" : "A" }
{ "_id" : ObjectId("59dd040f7ce97380afaf677b"), "custID" : "A123",
"Amount" : 250, "Status" : "A" }
{ "_id" : ObjectId("59dd04207ce97380afaf677c"), "custID" : "B212",
"Amount" : 200, "Status" : "A" }
{ "_id" : ObjectId("59dd04347ce97380afaf677d"), "custID" : "A123",
"Amount" : 300, "Status" : "D" }

```

PART A

```

a)
> db.Customer.aggregate({$group:{_id:"$custID",totalAmount: {$sum:"$Amount"}}})

{ "_id" : "B212", "totalAmount" : 200 }
{ "_id" : "A123", "totalAmount" : 1050 }

```

b)

```
> db.Customer.aggregate({$match:{Status:'A'}},{ $group:
{_id:"$custID",totalAmount:{$sum:"$Amount"}}})
```

```
{ "_id" : "B212", "totalAmount" : 200 }
{ "_id" : "A123", "totalAmount" : 750 }
```

c)

```
> db.Customer.aggregate({$match:{Status:'A'}},{ $group:
{_id:"$custID",minTotalAmount:{$min:"$Amount"}}})
```

```
{ "_id" : "B212", "minTotalAmount" : 200 }
{ "_id" : "A123", "minTotalAmount" : 250 }
```

d)

```
> db.Customer.aggregate({$match:{Status:'A'}},{ $group:
{_id:"$custID",maxTotalAmount:{$max:"$Amount"}}})
{ "_id" : "B212", "maxTotalAmount" : 200 }
{ "_id" : "A123", "maxTotalAmount" : 500 }
```

e)

```
> db.Customer.aggregate({$match:{Status:'A'}},{ $group:
{_id:"$custID",avgTotalAmount:{$avg:"$Amount"}}})
{ "_id" : "B212", "avgTotalAmount" : 200 }
{ "_id" : "A123", "avgTotalAmount" : 375 }
```

PART B

```
f)> db.Customer.getIndexes()
```

```
[
  {
    "v" : 1,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
```



```

        "ns" : "Customer.Customer"
    }
]
> db.Customer.createIndex({custID:1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}

```

```

g)
> db.Customer.getIndexes()
[
  {
    "v" : 1,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "Customer.Customer"
  },
  {
    "v" : 1,
    "key" : {
      "custID" : 1
    },
    "name" : "custID_1",
    "ns" : "Customer.Customer"
  }
]

```

```

h)> db.Customer.dropIndex({custID:1})
{ "nIndexesWas" : 2, "ok" : 1 }

```

```
> db.Customer.getIndexes()
[
  {
    "v" : 1,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "Customer.Customer"
  }
]
```

EXTRA QUERIES :

```
> db.Customer.aggregate({$match:{Status:'A'}},{ $group:
{_id:"$custID",totalAmount:{$sum:"$Amount"}}},{ $sort:{totalAmount:1}},
{$limit:1})
{ "_id" : "B212", "totalAmount" : 200 }
```

```
> db.Customer.aggregate({$match:{Status:'A'}},{ $group:
{_id:"$custID",totalAmount:{$sum:"$Amount"}}},{ $sort:{totalAmount:-1}},
{$limit:1})
{ "_id" : "A123", "totalAmount" : 750 }
```