COMPLETE NOTES ON

MongoDB

COPYRIGHTED BY :-

Instagram :- @codes.learning

Youtube :- Point of Programming

Telegram :- codelearning TG

WRITTEN BY :-

Mayuri Kharche (Software Engineer)

Index :-

No	Topic	Page No
1.	MongoDB Overview	1
2	MongoDB Advantages	2
3.	Data Modeling	3
4	Create Database	5
5.	Drop Database	6
6.	Create Collection	7
7.	Drop Collection	10
8.	Data types	11
9.	Insert Document	13
10.	Query Document	18
11.	Update Document	27
12.	Delete Document	30
13	Projection	31

14	Limiting Records	38
15	Sorting Records	34
16.	Indexing	35
17.	Aggregation	38
18	Replication	41
19.	Sharding	44
20.	Create backup	45
21	Relationships	47
22	Covered Queries	52
23	Cassandra Vs MongoDB	54
24	Analyzing Queries	55
25	Map Reduce	60
26	Regular Expression	64
27	Capped Collection	66
28	Redis Vs MongoDB	71
29	MongoDB Cloud	72

MongoDB

MongoDB is a cross-platform, document oriented database that provides high performance, high-availability and easy scalability. MongoDB works on concept of collection and document

Database:-

Database is physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple interface., database.

Collection:-

Collection is a group of MongoDB document. It is an eqivalent to RDBMS table. A collection exists within a single database. Collections do not enforce a schema document within a collection can have diffrent fields.

Document:-

A document is a set of key-value pair. Document have dynamic schema. Dynamic schema means that document in the same collection do not need to have the same set of fields and structure and common fields in a collection document may hold diffrent types of data.

Advantages of MongoDB over RDBMS :-

- Schema less :- MongoDB is a document database in which one collection holds diffrent documents Number of fields, content and size of the document can differ from one document to another.
- Structure of a single object is a clear
- No complex join
- Deep query-ability. MongoDB supports dynamic queries on document using a document-based query language that nearly as powerfull as SQL
- Tuning
- Conversion/

Why use MongoDB:

- Document Oriented Storaye.
- Index on any attribute
- Replication and availability
- Auto-Sharding
- Rich Queries
- Fast in-place updates
- Proffesional support by MongoDB

Where to use MongoDB:

(1) MongoDB -> Big Data
Content Management and Delivery
Mobile and Social Infrastructure
User Data Management
Data hub

Start MongoDB

sudo service mongodb start

Stop MongoDB

Sudo service mongodb stop

Restart MongoDB

Sudo service mongodb restart

MongoDB - Data Modelling.

Data in MongoDB has flexible schema documents in the same collection. They do not need to have the same set of fields or structure common fields in the collections document may hold diffrent types of data

Data Model Design:-

MongoDB provides 2 types of data models.

① Embedded Data Model

In this model, you can have all the related data in a single document. It is also known as de-normalized data model.

Example:-

```
{
    _id:,
    Emp_ID: "10025AE336"
    Personal_details: {
            First_Name : "Radhika",
```

```
        Last_Name : "Sharma",
        Dato_of_Birth : "1995-09-26"
    },
    Contact : {
        email : "radhika_sharma.123@gmail.com",
        phone : "9848022338"
    },
    Address : {
        city : "Hydrabad".
        Area : "MadApur",
        State : "Telangana"
```

Normalized Data Model :-

In this model, you can refer the sub-documents in the original document, using referances.

Example :-

```
Employee :-
    {
        _id : <ObjectId01>,
        Emp_ID : "10025AE336"
    }

Personal_details :
    {
        _id : <ObjectId102>
        empDOCID : "ObjectId101",
        First_Name : "Radhika",
        Last_Name : "Sharma",
        Date_of_Birth = "1995_09_26"
    }
```

Contact:

```
{
  _id : <ObjectId103>,
  empDocID : "ObjectId101",
  email : "radhika_sharma.123@gmail.com",
  phone : "9848022338"
}
```

Address:-

```
{
    _id:-<ObjectId104>,
    empDocID : "ObjectId101",
    City : "Hydrabad",
    Area : "Maclapur",
    State : "Telangana"
```

Considerations while designing Schema in MongoDB

- Design your schema according to user requirement
- Combine objects into one document if you will use them together.
- Do joins while write, not on read.
- Optimize your schema for most frequent use cases
- Do Complex aggregation in schema.
- Duplicate the data because disk space is cheap as compared to compute time.

MongoDB - Create Database

The use Command

MongoDB use DataBASE_NAME is used to create database. The command will create a new database if it doesn't exit, otherwise it will create a new database it will return existing database

Syntax:-
Basic Syntax of use DATABASE statement as follows:-
use DATABASE_NAME

Example:-
If you want to use a database with name <mydb>, then use DATABASE statement
>use mydb
switched to db mydb

To check your currently selected database, use the command db
>db
mydb

If you want to check your database list, use the command show dbs
>Show dbs
local 0.78125GB
test 0.23012 GB

Your created database is not present in list
To display database you need to insert atleast

One document into it.

```
>db.movie.insert({"name": "---."})
>show dbs
  local:      0.78125 GB
  mydb        0.23012 GB
  test        0.23012 GB
```

MongoDB - Drop Database

The dropDatabase() method.
MongoDB db.dropDatabase() command is used to drop a existing database.

Syntax :-
Basic syntax of dropDatabase() command is as follow

```
db.dropDatabase()
```

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database

Example:-
Check the list of available databases by using the command, show dbs

```
>Show dbs
local   0.78125GB
mydb    0.23012SGB
test    0.23012 GB
>
```

If you want to delete new database <mydb> then dropDatabase() command would be follows

```
>use mydb
switched to db mydb
>db.dropDatabase()
>{"dropped":"mydb", "ok":1}
>
```

MongoDB - Create Collection

The createCollection() Method
MongoDB db.createCollection(name, options) is used to create collection.

Syntax:-
Basic syntax of createCollection() command

db.createCollection(name, options)
In the command, name is name of collection to be created Options is a document and is used to specify configure of collection.

Parameter	Type	Description
Name	String	Name of the collection to be created
Options	Document	Specify options about memory size and indexing

Options parameter is optional, so you need to specify only name of collection.

Field	Type	Description
Capped	Boolean	If true, enables a caped collection. Capped collection is a fixed size collection that automatically overwrite its oldest entries. when it reaches to its maximum size. If you specify true, you need to specify size parameter also
autoIndexed	Boolean	If true, automatically create index on _id fields Default Value is false
Size	number	Specifies a maximum size in bytes for a capped collection If capped is true then you need to specify this field also.
max	number	Specifies the maximum number of document allowed in the capped collection

While Inserting document, MongoDB first checks size field of capped collection, then it checks max field.

Example:-
Basic syntax of createCollection() method without option is as follows.

```
>use test
switched to db test
>db.createCollection("mycollection")
{ "Ok" : 1 }
>
```

You can check the created collection by using the command show collections.

```
> Show collections
mycollections
system.indexes
```

The following example shows the syntax of create collection() method will few important options.-

```
>db.createCollection("mycol", { capped : true,
        autoIndexID: true, size: 6142800,
        max:10000 }) {
            "Ok": 0,
    "errmsg": "BSON field 'create.autoIndexID'
        is an unknown field", "code": 40415,
  "codeName":"Location 40415" }
  >
```

In MongoDB, you don't need to create collection, MongoDB create collection automatically.

```
>db.tutorialspoint.insert({"name:"tutorialspoint"})
WriteResult({"nInserted":1})
> Show collections
 mycol
 mycollection
 System.Indexes
 tutorialspoint
 >
```

MongoDB - Drop Collection.

The drop() Method.
MongoDB's db.collection.drop() is used to drop collection from database.

Syntax:-
Basic syntax of drop command

```
db.COLLECTION_NAME.drop()
```

Example.

```
> Use mydb
 Switched to db mydb
 > Show collections
 mycol
 mycollection
 System.indexes
 tutorialspoint
 >
```

Now drop the collection with the name mycollection.

```
>db.mycollection.drop()
true
>
```

Again check the list of collection into database.

```
>show collections
mycol
System.Indexes.
tutorialspoint
>
```

drop() method will return true. If the selected collection is dropped successfully, otherwise it will return false.

MongoDB - Datatypes.

- String
  This is most commonly used datatypes to store the data. String MongoDB must be UTF-8 valid.

- Integer
  This type is used to store a numerical value Integer can be 32 bit or 64 bit depending upon server.

- Boolean
  This type is used to store boolean value.

- Double
  This type is used to store floating point values.

- Min/max keys -
  This type is used to compare a value again- the lowest and highest BSON elements.

- Arrays :-
  This type is used to store arrays or list or multiple values in one key.

- Timestamp :-
  ctimestamp. This can be handy for recording when a document has been modified or added.

- Object :-
  This datatype is used for embedded document

- Null :-
  This type is used to store Null value

- Symbol :-
  This datatype is used identically to a string its generally reserved for languages that use a specific symbol type.

- Date :-
  This datatype is used to store the current date or time in UNIX time format. You can specify your own time by creating object of Date and passing day, month, year onto it

- Object ID-
  This datatype is used to store document ID.

- Binary data :-
  This datatype is used to store binary data.

- Code :-
  This datatype is used to store javascript code into the document.

- Regular Expression :-
  This datatype is used to store regular Expression

MongoDB - Insert Document

The insert () method :-
To insert data into MongoDB collection, you need to use MongoDB's insert () or save () method

Syntax :-
```
> db.COLLECTION_NAME.insert (document)
```

Example :-
```
> db.users.insert ({
... _id : ObjectID("507f191e810c19729de860"),
... title : "MongoDB Overview",
... description : "MongoDB is noSQL Database",
... by : "tutorial point",
... url : "http://www.tutorialspoint.com",
... tags: ['mongodb', 'database', 'NoSQL'],
... likes : 100
... })
WriteResult ({"nInserted" : 1})
>
```

Here mycol is our collection name, as created in this. If the collection does'nt exist in database then MongoDB will create this collection and then insert document into it.

In the inserted document, if we don't specify the _id parameter then MongoDB assign a unique ObjectId for this document.

_id is 12 bytes hexadecimal number unique for every document in a collection 12 bytes are divided as follows

_id : ObjectId (4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)

You can also pass an array of document into the Insert() method as shown below.

```
>db.createCollection("post")
>db.post.insert([
    {
      titele :"MongoDB Overiview",
      description:"MongoDB is no SQL Database",
      by: "Tutorials Point"
      url: "http//www.tutorialpoint.com")
      tags : ["mongodb","database","NoSQL"],
      likes : 100
    },
    {
    title : "NoSQL Database",
    description:"NoSQL database doesn't have table",
```

```
tags : ["Mongodb", "database", "NoSQL"],
likes : 20,
Comments : [
    {
        user: "user1",
        message : "My first comment",
        dateCreated : new Date(2013,11,10,2,35),
        like = 0
        }
    ]
 }
])
BulkWriteResult({
    "writeErrors" : [],
    "writeConcernErrors" : [],
    "nInserted" : 2,
    "nUpserted" : 0,
    "nMatched" : 0,
    "nModified" : 0,
    "nRemoved" : 0,
    "upserted" : [],
    }]
>
```

To insert the document you can use db.post.save(document) also. If you don't specify _id in the document then save() method will work same as insert() method. If you specify _id then. It will replace whole data of document containing _id specified in save() method.

The insertOne() method.
If you need to insert only one document into a collection you can use this method

Syntax:-
```
>db.COLLECTION_NAME.insertOne(document)
```

Example:-
```
>db.createCollection("empDetails")
 {"ok" : 1}

> db.empDetails.insertOne(
      {
        First_Name :"Radhika",
        Last-Name :- "Sharma",
        Date_Of_Birth " :"1995-09-26",
        email :"radhika_sharma.123@gmail.com",
        phone :"9848022338"
      })
  {
   "acknowledged" : true,
   "insertedId" : ObjectId("5dd62b4070fb13eec")
  }
>
```

The insertMany() method :-
You can insert multiple document using the insertMany() method. To this method you need to pass an array of documents.

Example :-

```
>db.empDetails.insertMany(
  [
    {
        First-Name : "Radhika",
        Last-Name : "Sharma"
      Date_of_Birth : "1995-09-26",
      email : "radhika-sharma.123@gmail.com",
      phone : "900012345"
    },
    {
        First_Name : "Rachel",
        Last_Name : "Christopher",
      Date_of_Birth : "1990-02-16",
      email : "Rachel_Christopher.123@gmail.com"
      phone : "900005 4321"
    },
    {
        First_Name : "Fathima",
        Last_Name : "Sheik",
      Date_of_Birth : "1990-02-16"
      email : "Fathima_Sheik.123@gmail.com",
      phone : "900005 4321"
    }
  ]
)
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5dd631f270fb13eec396"),
        ObjectId("5dd631f270fb13eec396"),
        ObjectId("5dd631f270fb13eecbef"),
    ]
}
>
```

MongoDB - Query Document

The find() method
To query data from MongoDB collection, you need to use MongoDB find() method.

Syntax :-
> db.COLLECTION_NAME.find()

find() method will display all the document in non-structured way.

Example :-
Assume we created collection named mycol as

```
> use sampleDB
Switched to db sampleDB
> db.createCollection("mycol")
  {"OK" : 1}
>
```

And inserted 3 documents in it using the insert() method as shown.

```
> db.mycol.insert([
    {
      title : "MongoDB Overview",
      description : "MongoDB is no SQL database",
      by : "tutorials point"
    url : "http://www.tutorialspoint.com",
    tags:["mongodb", "database", "NoSQL"],
      likes : 100
    },
```

```
t{
    title : "NoSQL Database",
  description : "NoSQL database doesn't have tables",
    by : "tutorials point",
    url : "http:// www.tutorialspoint.com",
  tags : ["mongodb","database," NoSQL"],
    like : 20,
    comments: [
        {
            user : "user 1",
            message :" my first comment",
          datacreated: new Date (2013, 11,10,2,35),
          like : 0
        }
      ]
    }
  ])
```

Following method retrives all the documents in the collection.

```
> db.mycol.find()
{ "_id" : ObjectId ("5dd4e2cc0082ld3b446075"),
  "title", " MongoDB Overview", "description",:
  "MongoDB is no SQL database", "by" : "tutorials
  point", "url": " http:// www.tutorialspoint.
  com", "tays" : ["mongodb", "database", "NoSQL"
  , "likes" ; 100 }
```

The pretty () Method.

To display the results in formatted way, you can use pretty () method.

Syntax:-

db.COLLECTION_NAME.find().pretty()

Example:-

```
>db.mycol.find().pretty()
 {
    "_id" :ObjectId("54dde2cc0821d3b44607534c"),
   "title" : "MongoDB Overview",
   "description" : "MongoDB is no SQL database",
    "by" : "tutorials point",
    "url" : "http://www.tutorialspoint.com",
    "tags" : [
              "mongodb",
              "database",
              "NoSQL"
        ],
     "likes" : 100
   }
   {
    "_id" : ObjectId("sdd4e2cc0821d3b44607534d"),
    "title" : "NoSQL Database",
   "description": "NoSQL database doesn't have
          tables",
    "by" : "tutorials point",
   "url" : "http:// www.tutorialpoint.com",
    "tags" : [
              "mongodb",
              "database",
              "NoSQL"
     ],
     "likes" : 20,
```

```
"comments":[
        {
        "user": "user1",
        "message": "My first comment",
       "dateCreated": ISODate("2013-12-09T21:05:
                                                 00Z"),
        "like": 0
    }
 ]
```

The findOne() method.

Apart from the find() method, there is findone () method, that returns only one document.

Syntax:-

```
>db.COLLECTIONNAME.findOne()
```

Example:-

```
>db.mycol.findOne({title:"MongoDB Overview"})
   {
     "_id": ObjectId("5dde 5421 70fb13 eec 39c3bf"),
     "title": "MongoDB Overview",
    "description": "MongoDB is noSQL database"
     "by": "tutorials point",
    "url" : "http://www.tutorialspoint.com",
    "tags" : [
              "mongodb",
              "database",
              "NoSQL"
          ],
         "likes" : 100
   }
```

RDBMS Where Clause Equivalents in MongoDB

To query the document on the basis of some condition, you can use this operation.

① Operation :- Equality
Syntax :- {<key>:{$eq; <value>}}
Example :- db.mycol.find({"by": "tutorial point"
}).pretty()
RDBMS Equivalent :-
where by = 'tutorials point'

② Operation :- Less than
Syntax :- {<key>:{$lt: <value>}}
Example :- db.mycol.find({"likes": {$lt: 50}}.pretty
RDBMS Equivalent :-
where likes < 50

③ Operation :- Less Than Equals
Syntax :- {<key>:{$lte: <value>}}
Example :- db.mycol.find({"likes": {$lte: 50}}).
pretty()
RDBMS Equivalent :-
where likes <= 50

④ Operation :- Greater Than
Syntax :- {<key>:{$gt: <value>}}
Example :- db.mycol.find({"likes": {$gt: 50}}).
pretty()
RDBMS Equivalent :-
where likes > 50

5 Operations:- Greater Than Equals.
Syntax:- {<key>:{$gte: <value>}}
Example:- db.mycol.find({"likes": {$gte: 50}}).
pretty()
RDBMS Equivalent:-
where likes >= 50

6. Operations:- Not Equals.
Syntax:- {<key>:{$gte: <value>}}
Example:- db.mycol.find({"likes": {$ne: 50}}).pretty()
RDBMS Equivalent:-
where likes != 50

7. Operations:- Values in an array.
Syntax:- {<key>:{$in: [<value1>, <value2> .....
<value N]}}
Example:- db.mycol.find({"name": {$in: ["Raj",
"Ram", "Raghu"]}}).pretty()
RDBMS Equivalent:-
Where name matches any one value in:
["Raj", "Ram", "Raghu"]

8. Operations:- Values not in an array.
Syntax:- {<key>:{$nin: <value>]}
Example:- db.mycol.find({"name": {$nin: ["Ramu",
"Raghav"]}}).pretty()
RDBMS Equivalent:-
Where name values is not in the array
["Ramu", "Raghav"] or, doesn't exit at all

AND in MongoDB.

Syntax:-

To query documents based on the AND condition you need to use $and keyword.

```
>db.mycol.find ({$and : [{<key1>:<value 1>},
    {<key 2>:<value 2>}]})
```

Example:-

```
db.mycol.find ({$and:[{"by":"tutorrals point"},
  {title:"MongoDB Overview"}]}).pretty()
{
    "_id":ObjectId ("54dde2cc0821d34460704c"),
    "title" : "MongoDB Overview",
   "description": "MongoDB is no SQL database",
   "url" : "https//www.javatpoint.com",
   "tags": [
            "mongoDB",
            "database",
            "NoSQL"
     ],
     "likes" : 100
   }
  ]
```

Equivalent where clause will be 'where by = 'tutorials point" And title "MongoDB Overview" You can pass any number of key, values pair in find clause.

OR in MongoDB

Syntax:-

To query documents based on the OR condition you need to use $or keyword.

```
>db.mycol.find (
    {
      $or : [
        {key1:value1}, {key2:value2}
      ]
    }
  ). pretty()
```

Example:-

```
>db.mycol.find({$or:[{"by": "tutorials point"},
    {"title": "MongoDB Overview"
{
  "_id" : ObjectId("54dd4e2cc08213b460"),
  "title": "MongoDB"
  "description" : "MongoDB Overview",
  "tags": [
            "mongodb",
            "database"
            "NoSQL"
        ],
        "likes" : 100
}
>
```

Using AND or and OR Together

"Where likes >10 AND (by="tutorials point" OR title="MongoDB Overview")"

```
>db.mycol.find ({"likes": {"$gt :10}, {$or: [ {
        "by" : "tutorials point" },
    { "title" : "MongoDB Overview"}]}).pretty()
{
    "_id" :ObjectId (7df78ad8902c),
    "title" : "MongoDB Overview"
    "description" : "MongoDB is no SQL database",
     "by": 'tutorials point",
    "url": "http:// www.tutorialspoint.com",
    "tags": ["mongodb", "database", "NoSQL"],
    "likes" : "100"
    }
>
```

NOR in MongoDB

Syntax:-

To query documents based on the NOT condition you need to us $not keyword.

```
>db.COLLECTION_NAME.find (
        {
            $not : [
                {key1 : value1}, {key 2 : value2}
            ]
        }
    )
```

NOT in MongoDB.

Syntax:-

To query documents based on NOT condition you need to use $not condition in the basic syntax of NOT -

Example:-

```
> db.empDetails.find({"Age": {$not : {$gt : "25"}
  }})
{
    "_id" : ObjectId("5dd66368 70fb 13eec396"),
    "First_Name" : "Fathima",
    "Last_Name": "Sheik",
    "Age" : "24"
    "e-mail" : "Fathima_Sheikh.123@gmail.com",
    "Phone": "9000054321"
}
```

MongoDB - Update Document

MongoDB's update() and save() methods are used to update document into a collection. The update() method updates the values in the existing document while the save() method replace the existing document with the document passed in save() method.

MongoDB Update() Method

The update() method updates the value in existing document.

Syntax:-

```
> db.COLLECTION_NAME.update(SELECTION_
    CRITERIA, UPDATE_DATA)
```

Example:-

```
{"_id": ObjectId(5983148783ad45ec5), "title":
    "MongoDB Overview"}
{"_id : ObjectId(5983148783ad45ec5), "title":
```

"NoSQL Overview"}

MongoDB Save() Method

The save() method replace the existing document with the new document passed in the save method.

Syntax :-

```
>db.COLLECTION_NAME({_id:ObjectId(), NEW_DATA})
```

Example :-

```
>db.mycol.save(
  {
 "_id" : ObjectId("507f191e810c19729de860ea"),
        "title": "Tutorials Point New Topic",
      "by": "Tutorials Point"
    } )
 WriteResult({
        "nMatched" : 0,
       "nUpserted" : 1,
       "nModified" : 0,
      "_id" : ObjectId("507f191e810c19729de860ea")
 })
>db.mycol.find()
 { "_id" : ObjectId("507f191e810c19729de860e6")
      "title" : "Tutorials Point NewTopic",
      "by" : "Tutorials Point" }
{ "_id" : ObjectId("507f191e810c19729d8606"),
      "title" : "NoSQL Overview" }
  >
```

MongoDB findOneAndUpdate() method

The findOneAndUpdate() method updates the values in the existing document.

Syntax:-

```
>db.COLLECTION_NAME.findOneUpdate(SELECTION_
   CRITERIA, UPDATED_DATA)
```

Example:-

```
>db.empDetails.insertMany(
   [
      {  First_Name : 'Radhika",
         Last_Name :"Sharma",
         Age :"26",
         email :"radhika_sharma.123@gmail.com",
         phone: "9000012345"
      }
   ]
```

Following example update age and email values

```
>db.empDetails.findOneAndUpdate(
   {First_Name : "Radhika"},
   {$set :{Age: '30', email : "radhika@gmail.com"
                                                 }}
   )
{
   "_id" : ObjectId("5dd663687 0fb13eec396"),
   "First_Name" : "Radhika",
   "Last_Name" : "Sharma",
   "Age" : "30",
   "email" : "Radhika_newemail@gmail.com",
   "phone" : "9000123 45"
}
```

MongoDB updateOne() method

This method updates a single document which matches the given filter.

Syntax:-

```
>db.COLLECTION-NAME.updateOne(<filter>,<update>)
```

Example:-

```
>db.empDetails.updateOne(
    { First_Name : 'Radhika'},
  {$set: {Age: "30", email: 'radhika_newemail@gmail.com"
    }}
  )
{"acknowledged" : true, "matchedCount" : 1,
    "modifiedCount" : 0}
```

MongoDB updateMany() method.

```
>db.COLLECTION-NAME.update(<filter>,
            <update>)
```

MongoDB - Delete Document.

① The remove() Method.

MongoDB's remove() method is used to remove a document from the collection. remove() method accepts 2 parameters. One is deletion creteria and second is justOne flag.

① deletion criteria:- deletion criteria according to documents will be removed.

② justone - if set to true or 1, then remove only 1 document.

Syntax:-

```
>db.COLLECTION_NAME.remove(DELETION-CRITERIA)
```

Example:-

```
{_id : ObjectId("507f191e810c19729de861"), title :
        "MongoDB Overview"},
{_id : ObjectId("507f19e810c19729de862"), title:
        " NoSQL Overview"}
```

following example will remove all the documents whose title is 'MongoDB Overview"

```
>db.mycol.remove({"title":"MongoDB Overview"})
WriteResult({"nRemoved":1})
>db.mycol.find()
{"_id": ObjectId("507f19e810c19729de8602),
        "title": "NoSQL Overview"}
```

Removed All Documents

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. This is equivalent of SQL's truncate command.

```
>db.mycol.remove({})
WriteResult({"nRemoved": 2})
> db.mycol.find()
>
```

MongoDB - Projection

In MongoDB projection means selecting only one necessary data rather than selecting whole data.

The find() method

MongoDB's find() method, accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB, when you execute find() method, then it displays all fields of a document. To limits, you need to set a list of fields with value 1 or 0, 1 is used to show field while 0 is used to hide the fields.

Syntax:-

```
>db.COLLECTION-NAME.find({}, {KEY:1})
```

Example:-

```
{_id : ObjectId ("507f191e810c19729de860ea"), title:
        "MongoDB Overview"},
{_id : ObjectId ("507f191e810c19729de860eb"), title:
        "NoSQL Overview"}.
```

Following example will display the title of the document while querying the document.

```
>db.mycol.find({}, {"title":1, _id:0})
{"title": "MongoDB Overview"}
{"title": "NoSQL Overview"}
```

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

MongoDB - Limit Records.

The Limit () Method :-

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

Syntax :-

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

Example :-

```
{_id : ObjectId("507f191e810c19729de1"), title
      : "MongoDB Overview"},
{_id: ObjectId ("507f191e810c19729de2"), title
      : "NoSQL Overview"}
```

Following example will display only 2 document while querying the document.

```
>db.mycol.find({},{"title":1,_id:0}).limit(1)
  {"title": "MongoDB Overview"}
```

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

MongoDB Skip() Method :-

Apart from limit() method skip() which also accepts number type argument and is used to skip the number of documents

Syntax:-

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

Example:-

Following example will display only the second document.

```
>db.mycol.find({},{"title":1,_id:0}),
    limit(1).skip(1)
 {"title":"NoSQL Overview"}
>
```

please note, the default value in skip() method is 0.

MongoDB - Sort Records.

The sort() Method:-

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

Syntax:-

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

Example:-

if we use key:-1 it will sort in descending order.

MongoDB - Indexing.

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those document that match the query statement. This scan is highly inefficient and requires MongoDB to process a large volume data.

The CreateIndex() Method.

To create an index, you need to use CreateIndex() method of MongoDB.

Syntax:-

```
>db.COLLECTION_NAME.CreateIndex({Key:1})
```

here key is the same of field on which you want to create index and 1 is for ascending order, To create index in descending order you need to use -1.

Example.

```
>db.mycol.createIndex({"title":1})
{
    "createdCollectionAutomatically":false,
    "numIndexesBefore":1
    "numIndexesAfter":2,
    "ok":1
}
```

In createIndex() method you can pass multiple field, to create index on multiple field.

```
>db.mycol.createIndex({"title":1,"description":-1
})
```

Parameter	Type	Description.
bckground	Boolean	Builds the index in the background so that building an index does not block other database activities Specify true to build in the background. The default value is false.
unique	Boolean	Creates a unique index so that the collection will not accept insertion of documents where the index key or keys match an existing value in the index. Specify true to create unique index.
name	string	The name of the index If unspecified field, These indexes use less space but behave differently in some situations.
weights	document	The weight is a number ranging from 1 to 99,999 and denotes the significance of the field relative to the other indexed fields in term of the score.

The dropIndex() method.

You can drop a particular index using the dropindex() method of MongoDB

Syntax:-

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

Here "key" is the name of the file on which you want to remove or existing index. You can also specify the name of the index directly as.

```
dropIndex("name_of_the_Index")
```

Example:-

```
>db.mycol.dropIndex({"title":1})
{
    "ok":0,
    "errmsg":"can't find index with key:
        {title:1.0}",
    "code":27,
    "codeName":"IndexNotFound".
```

The dropIndexes() method.

This method deletes multiple indexes on a collection.

Syntax:-

```
>db.COLLECTION_Name.dropIndexs()
```

MongoDB- Aggregation:-

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

The aggregate() Method.

For the aggregation in MongoDB, you should use aggregate() method.

Syntax:-

```
>db.COLLECTION-NAME.aggregate(AGGREGATE-OPERATION)
```

Aggregation Expression

① $sum:-

Sums up defined value from all documents in the collection

Example:-

```
db.mycol.aggreatgation([$group{_id:"$by_user",
num_tutorial:{$SUM:"$likes"}}])
```

② $avg:-

Calculations the average of all given values from all given values from all documents in the collection.

Example:-

```
db.mycol.aggregate([$group:{_id:"$by-user",
num-tutorial:{$avg:$likes"}}})
```

$ min :-
Gets the minimum of corresponding values from all documents in the collection.
Example:-
db.mycol.aggregate([{$ group: {_id : "$by_user",
num_tutorial : {$ min : $ likes" }}}])

$ max:-
Gets the maximum of the corresponding values from all document in the collection.
Example :-
db.mycol.aggregate([{ group: {_id: "$by_user", num_
tutorials: {$ max : "$likes" }}}])

$ push
Inserts the value to an array in the resulting document but does not create duplicate.
Example:-
db. my col. aggregate ([{ $group : {_id :"$by_user.
user : ($ push : "$ url }}}])

$ add to set :-
Inserts the value to an array in the resulting document but does not create duplicate.
Example:-
db.mycol.aggregate ([{ group : {_id : "$by_user",
url : ($add To Set : "$ url " }}}])

$first
Gets the first document from the source document according to the grouping. Typically this makes only sense together with same previously "$ sort".

Pipeline Concept:-

In UNIX command, shell pipeline meanse the possibility to execute to an operation on some input and use the output as the input for the next command and so on. MongoDB also support same concept in aggregation framework. There is set of possible stage and each of those is taken as a set of documents as an input and produces a resulting set of document. This can in turn be used for the next stage and so on.

Stages in aggregation framework:-

① $project :-
Used to select some specific fields from a collection

② $match :-
This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.

③ $group:-
This does the actual aggregation as discussed above.

④ $sort :-
Sort the documents.

⑤ $skip :-
It is possible to skip forward in the list document for a given amount of documents.

$limit :-
The limits the amount of documents to look at by the given number starting from the current position.

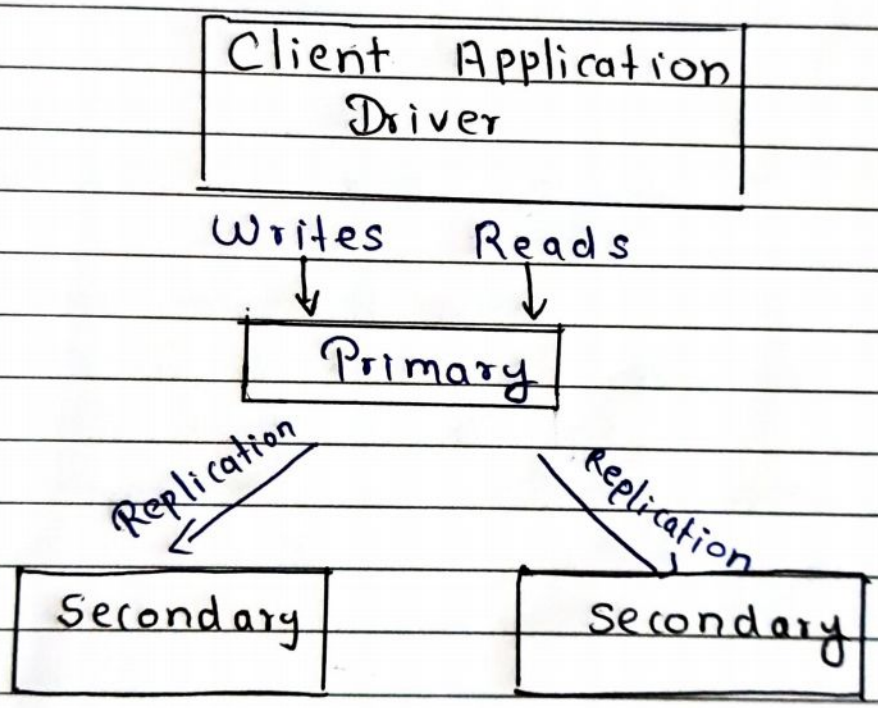
$unwind :-
This is used to unwind document that are using arrays. When using an array, the data is kind of pre-joined and this operations will be undone with this to have individual documents again

Replication:-
Replication is the process of synchronizing data across multiple server. Replication provides redundancy and increases data availability with multiple copies of data on diffrent servers. Replication protects a database from the loss of single server. Replication also allows you to recover from hardware failure and service intereptions. With additional copies of the data, you can dedicate one to disaster recovery, reporting or backup.

Why Replication.
- To keep your data safe.
- High (24&7) availability of data.
- Disaster Recovery.
- No downtime for maintainance.
- Read scaling
- Replica set is transperent to the application.

How Replication works in MongoDB

1] Replica set is a group of two or more nodes
2] In a replica set, one node and remaining nodes are secondary.
3] All data replicates from primary to secondary node
4] At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.
5] After the recovery of failed node, it again join the replica set and works as a secondary node.



Replica Set Features.

- A cluster of N node
- Any one node can be primary
- All write operation go to primary
- Automatic failover
- Automatic recovery
- Consensus election of primary.

Set up a Replica set.

We will convert standalone MongoDB instance to a replica set. To convert to replica set.

- Shutdown already running MongoDB server
- Start the MongoDB server by specifying - repset option.

monged --port "PORT" --dpath "Your_DB_DATA_PATH" --replset "REPLICA_SET_INSTANCE_NAME"

Example:-

mongod --port 27017 --dbpath "D:\set up\mongodb\data" --replset rs0

① It will start mongod instance with the name rs0, on port 27017

② Now start the command prompt and connect to this mongod instance.

③ In Mongo client, issue the command rs.initiate() to initiate a new replica set.

④ To check the replica set configuration, issue the command rs.conf() To check the status of replica set issue the command rs.status()

Add Members to Replica Set.

To add members of replica set, start mongod instanses on multiple machines. Now start a Mongo client and issue a command rs.add()

Syntax:-

>rs.add (HOST-NAME : PORT)

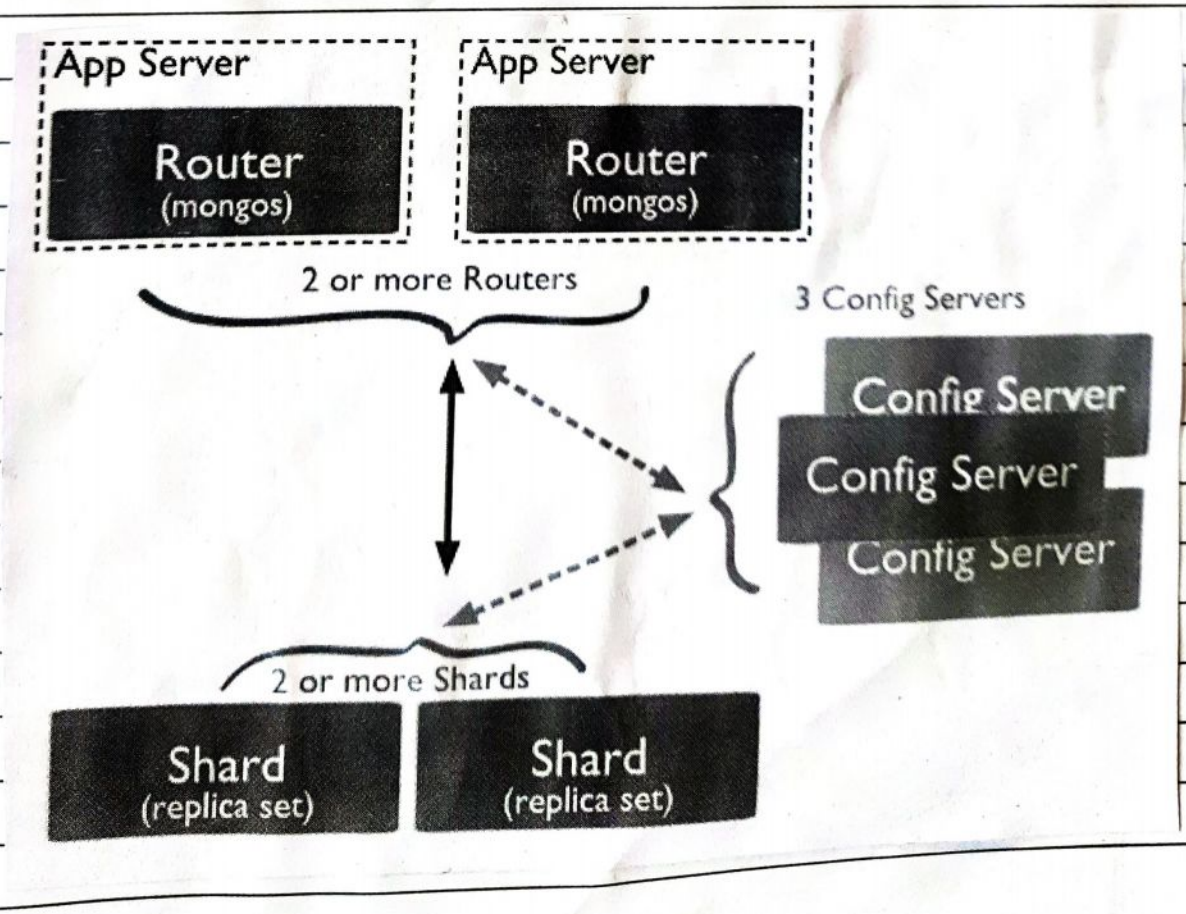
Sharding -

Sharding is the process of storing data records across multiple machines and it is MongoDB approch to meeting the demands of data growth. As the size of data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throught. Sharding solve the problem with horizontal scaling.

Why Sharding ?

- In replication, all writes go master node.
- Latency sensative queries still go to master
- Single replica set has limitation 12 nodes.
- Local disk is not big enough.

Sharding in MongoDB.



Shards:-

Shards are used to store data. They provide high availability and data consistency. In production envirnment, each shard is a Seperate replica set.

Config Server:-

Config servers store cluster's metadata. The data contains a mapping of the clusters data to the shards. The query router uses his metadata to target operations to specific shards. In Production envirnment shard clusters have exactly 3 config server.

Query Routers:-

Query routers are basically mongo instances. Interface with client application and direct operation to the appropriate shard. The query router process and targets the operations to shards and then returns results to the client. A sharded cluster can contain more than one query router to divide the client request load. A client sends requests to one query router. Generally, a sharded cluster have many query routers.

Create Backup:-

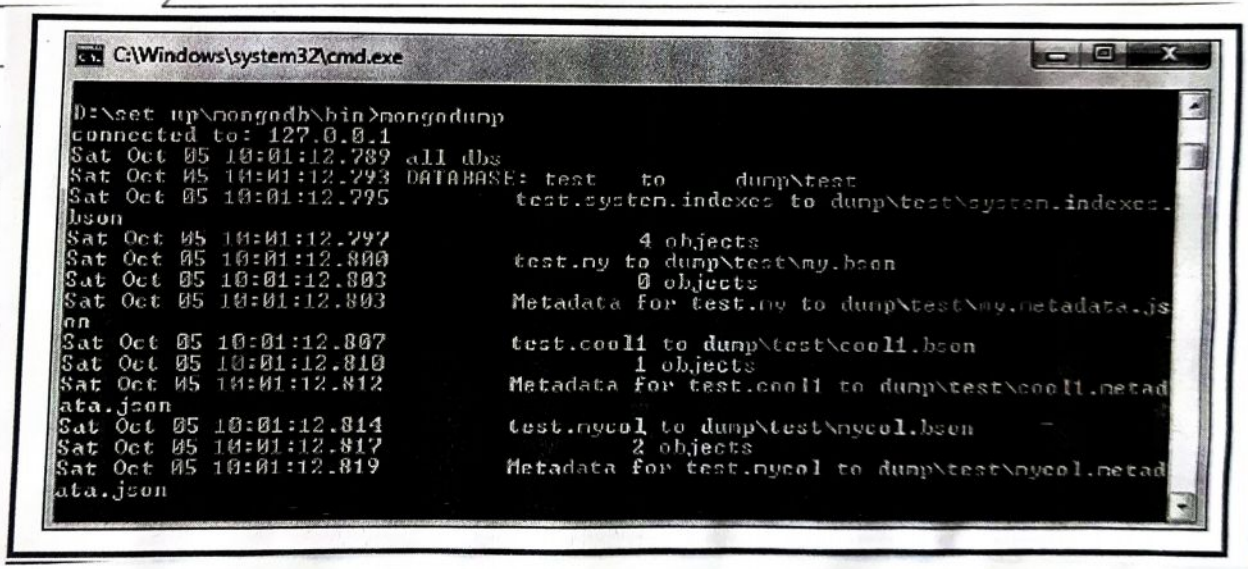
Dump MongoDB Data:-

To create backup of database in Mongodump command. This command will dump the entire data of your server into the dump directory There are many options available by which you can limit the amount of data or create backup of your remote server.

Syntax:-
> mongodump

Example.
Consider the mycol collection has the followin data.
>mongodump.

The command will connect to server running at 127.0.0.1 and port 27017 and back all data of the server to directory/bin/dump.



Syntax:-
① mongodump --host HOST_NAME -- port PORT_NUMBER
→ This command will backup all database of specified mongod instance.
Ex→ mongodump --host tutorialspoint.com --port 27017

② mongodump -dbpath DB_PATH -- out BACKUP_DIRECTORY
→ This command will backup only specified database.

at specified path.
→ mongodump --dbpath /data /db --out /data/ backup/

③ mongodump --collection COLLECTION --db DB_NAME
→ This command will backup only specified collection of specified database.
→ mongodump --collection mycol --db test

Restore data :-

To restore backup data MongoDB's mongorestore command is used. This command restores all of the data from the backup directory.

Syntax :-

```
>mongorestore.
```

MongoDB - Relationships -

Relationships in MongoDB represent how various documents are logically related to each other. Relationships can be modelded via Embeded and Referanced approches. Such relationships can be either 1:1, 1:N, N:1, or N:N

- Sample of user document

```
{
  "_id": ObjectId ("52ffc33cd8524f60001"),
  "name": "Tom Hanks"
  "contact": '987654321"
  "dob": "01-01-1991"
}
```

- Sample of address document.

```
{
  "_id": ObjectId("52ffc4a5d8524602"),
```

```
"building":- "24 A, Indina Apt",
"pincode" :123456,
  "city" : "Los Angeless"
  "state": "California".
}
```

Modeling Embedded Relationships:-
In the embedded approch, we will embed the address document inside the user document.

```
>db.users.insert ({
    {
     "_id":-ObjectId("52ffc33cd8524f4360001"),
     "contact" : "985674321",
     "dob": "01-01-1991"
     "name": "Tom Benzamin",
     "address" : [
        {
        "building" : "22A ,Indiana Apt",
         "pincode" : 123456,
         "city" : "Los Angels
         "state" : "california"
        },
        {
        "building":"170A, Acropolis Apt",
         "Pincode" : 456789,
         "city" : "Chicago",
         "state": "Illinois"
        }
     ]
    }
})
```

```
>db.users.findone({"name":"Tom Benzamin",
  {"address":1})
```

Note that in above query, db and users and the database and collection resp.

Modelling Referenced Relationships:-
This is the approach of designing normalized relationship. In this approach, both the user and address documents will be maintained separatly but the user document will contain a field that will referance the address document id field.

```
{
"_id": ObjectId("52ffc33cd8543600001"},
"contact" : "987654321",
"dob" = "01-01-1991",
"name" : "Tom Benzamin",
"address_ids" : [
    ObjectId("52ffc4asd8542e000"),
    ObjectId("52ffc4asd8542e001"),
  ]
}
```

As shown above, the user document contains the array field address_ids which contains ObjectIds of corresponding address.

Database Referances:-

To implement a normalized database structure in MongoDB, we use the concept of Referanced Relationship.

DBRefs Vs Manual References:-

Where we would use DBRefs insted of manual referances, consider a database where we are storing different types of addresses. in different collections. Now, when a user collections document referance an address, it also needs to specify which collection to look into based on the address type. In such scenario where a document referances document from many collections, we should use DBRefs.

Using DBRefs:-

There are three fields in DBRefs.

① $ref:-

This field specifies the collection of the referenced document.

② $id:-

This field specifies the _id field of the referanced document.

③ $db:-

This is an optional field and contains the name of the database in which referanced document this.

Consider a sample user document having DBRef field address as shown in the code snipate.

```
{
  "_id": ObjectId("5340259 7d8524002"),
```

```
"address": {
"$ref": "address_home",
"$id": ObjectId("S324009e4d87 002"),
"$db" :"tutorial point"},
  "contact":"98765321"
   "dob" : "01-01-1991"
    "name": "Tom Benzamin".
```

The address DBRef field specifies that the referensed address document lies in address-home collection under tutorialpoint database and has an id of S34009e4d852782002.

The following code dynamically looks in the collection specified $ref parameter for a document with id as specified by $id parameter in DBref.

```
>var user = db.users.findone({"name": Tom
    Benzamin"})
>var dbRef = user.address
>db[dbRef.$ref].findone({"_id" : (dbRef.$id)})
```

The above code returns the following address document present in address-home collection

```
{
   "_id": ObjectId("S34 009e4d S2 002"),
   "building":"22A, Indiana Apt",
    "Pincode": 123456",
     "City": "Los Angeles",
     "State": "California"
   }
```

Covered Queries.

What is covered Queries?

As per the official MongoDB documentation, a covered query is in which.

1) All the field in the query are part of an index.
2) All the fields returned in the query are in the same index.

Since all fields present in the queries are part of an index. MongoDB matches the query conditions and returns the result using the same index without actually looking inside the document since indexes are present in RAM, fetching data from indexes is much faster as compared to fetching data by scanning document.

Using Covered Queries.

To test covered queries, consider the following document in users collection.

```
{
  "_id" : ObjectId ("5342597003"),
    "contact" : "9854321",
    "dob" : "01-01-1991",
    "gender" : "M"
    "name" : "Tom Benzamin".
    "user_name" : "tom benzamin".
}
```

We will first create a compound index for the users collection on the fields gender

user_name using query.

→
```
>db.users.createIndex({gender:1, user_name:1})
{
  "createdCollectionAutomatically": false,
  "numIndexesBefore": 1,
  "numIndexAfter": 2,
  "ok": 1
}
```

Now this index will cover following query.

```
>db.users.find({gender:"m"}, {user_name:1, _id:0})
{"user_name":"tombenzamin"}
```

That is to say that for the above query, MongoDB would not go looking into database documents Insted it would fetch the required data from indexed data which is very fast.

```
>db.users.find({gender:"m"}, {user_name:1})
{"_id":ObjectId("5340259785200 3"), "user_nam
    : "tombenzamin".}
```

- An Index cannot cover a query if.

① Any of the indexed field is an array.

② Any of the indexed field is a subdoc.

Diffrences.

Cassandra Vs MongoDB

	Cassandra	MongoDB.
①	Cassandra is high performance distributed database system	MongoDB is cross-platform document-oriented database system
②	Cassandra is written in Java	MongoDB is written in C++
③	Cassandra stores data in tabular form like SQL format	MongoDB stores data in JSON format.
④	Cassandra is got license by Apache	MongoDB is got license by AGPL and drives Apache.
⑤	Cassandra is mainly designed to handle large amounts of data across many commodity server	MongoDB is designed to deal with JSON-like document and across application easier and faster.
⑥	Cassandra provides high availability with no single point of failure.	MongoDB is easy to administer in the case of failure.

Key Points of MongoDB.

① MongoDB is well suited for Bigdata and social Infrastructure.

② MongoDB is provide replication, High availability and Auto-sharding.

③ MongoDB is used by companies like Foursquore Intuit, Shutterfly etc.

MongoDB - Analyzing Queries

Analyzing queries is very important aspect of masuring how effective the database and indexing design is.

Using $explain :-

The $explain operator provide information on the query. Indexes used in a query and other statistic. It is very useful when analyzing how well your indexes are optimized.

```
>db.users.createIndex({gender:1, user_name:1})
 {
      "numIndexesBefore":2,
      "numIndexesAfter":2,
       "note":"all Indexes already exists",
       "ok":1
 }

>db.users.find({gender:"M"},{user_name":1,
   _id:0}).explain().
```

The above explain() query returns the following analyzed result.


```
{
  "queryPlanner": {
      "plannerVersion": 1,
      "namespace": "mydb.users",
      "indexFilterSet": false,
      "parsedQuery": {
            "gender": {
                  "$eq": "m"
            }
      },
      "queryHash": "13403 7D3C",
      "planCacheKey": "DEAA17C",
      "winningPlan": {
            "stage": "PROJECTION_COVERED",
            "transformBy": {
                  "user_name": 1,
                  "_id": 0
            },
      "inputStage": {
            "stage": "IXSCAN",
            "keyPattern": {
                  "gender": 1,
                  "user_name": 1,
            },
      "indexName": "gender_1_user_name_1",
      "isMultikey": false,
      "multikeypaths": {
            "gender": [ ]
            "user_name": [ ]
```

```
        "isMultikey": false,
        "multikeyPaths": {
                "gender": [],
                "user_name": []
            },
        "isUnique": false,
        "isSparse": false,
        "isPartial": false,
        "indexVersion": 2,
        "direction": "forward",
        "indexBounds": {
                "gender": [
                    "[\m,\m]"
                ], ] } } },
        "rejectedPlans": []
    },
    "ServerInfo": {
        "host": "Krishna"
        "Port": 27017
        "Version": "4.2.1",
        "gitversion": "ed ..... 317e"
        },
        "ok": 1
    }
```


Using $hint.

The $hint operator forces the query optimizer to use the specified index to run a query. This is particullarly useful when you want to test performance of a query with diffrent Indexes.

```
>db.users.find({gender:"m"},{user_name:1,
  _id})hint({gender:1, user_name:1})
{"user_name":"tombenzamin"}
```

To analyze the above query using $explain.

```
>db.users.find({gender:"m",{user_name:1}).
  hint({gender:1, user_name:1}).explain()
```

MongoDB - ObjectId

An objectId is 12 byte BSON type having the following structure.

1. The first 4 bytes representing the seconds since the unix epoch.
2. The next 3 bytes are machine identifier.
3. The next 2 byte consists of process id.
4. The last 3 bytes are random counter value.

MongoDB uses ObjectIds as the default value of _id field of each document, which is generated while the creation of any document.

Creating New ObjectId.

To generate a new ObjectId use following code.

```
>newObjectId=ObjectId()
```

The above statement returned the following uniqully generated id -

ObjectId ("S34 9b4 2781d08c 09f3")

Insted of MongoDB generating the ObjectId, you can also provide a 12 byte id.

```
> myObjectId = ObjectId ("S349b42781d08c9f3")
```

Creating Timestamp of a Document :-

Since the _id ObjectId by default store 4 byte timestamp, in most cases you do not need to store the creation time of any document. You can fetch the creation time of a document using getTimestamp method.

```
>ObjectId ("S349b4ddd2781989of4").getTimestamp()
```

This will return the creation time of this document in ISO date format.

```
ISO date ("2014-04-12T21:49:17Z")
```

Converting ObjectId to String:-

You may need the value of ObjectId in string format. To convert ObjectId in string use the following code.

```
>newObjectId.str.
```

The above code will return the string format of the Guid.

S43b4dd781c0989f3.

Map Reduce :-

Map-reduce is data processing paradigm for condensing large volume of data into useful aggregated result. MongoDB command for map-reduce operations. MapReduce is generally used for processing large data sets.

MapReduce Command.

```
>db.collection.mapReduce(
 function(){emit(key, value);},
 function(key, values)
          (return reduceFunction),{
      out: collection,
      query: document,
      sort: document,
      limit: number
    }
 )
```

In the above syntax.

- map is a javascript function that maps a value with key and emits a key-value pair.
- reduce is javascript function that reduce or groups all the documents having the same key
- Out specifies the location of the map-reduce query result.
- query specifies the optional selection criteria for selecting document.

- Sort : Specifies the optional sort criteria.
- limit specifies the optional maximum number of documents to be returned.

Using MapReduce.
Consider the following document structure storing user posts. The document structure user-name of the user

```
{
  "post_text": "tutorial is an awsome",
  "user_name" : "mark",
  "Status": "active"
}
```

Now we will use mapReduce function, on our posts collection to select all the active posts, group them on the basis of user-name.

```
> db.posts.mapReduce (
   function() {emit (this.user_id , 1);},
   function (key, value) {return Array.sum(values)},
    {
      query : {status : "active"},
      out : "Post_total"
    }
  )
```

The above mapReduce query outputs the following result.

```
{ "result" : "post_total",
  "timeMillis" : 9,
  "Counts" : {
      "input" : 4,
      "emit" : 4,
      "reduce" : 2,
      "output" : 2,
  },
  "ok" : 1
}
```

CouchDB vs MongoDB.

Comparision Feature	CouchDB	MongoDB
Data Model	It follows the document oriented model and data is presented in JSON format	It follows the document oriented model but data is presented in BSON format.
Interface	CouchDB uses HTTP / Rest based interface. It is very Intutive and very well designed.	MongoDB uses binary protocol and custom protocol over TCP/IP
Object Storage	In CouchDB, database contains documents	In MongoDB, database contain collection and collection contain documents.

4.	Query Method	CouchDB, database contains document method.	MongoDB follows Map/Reduce (JS) creating collection + object-based query lang.
5.	Replication	CouchDB supports master-master replication with custom conflict resolution functions	MongoDB support master-slave replication.
6.	Concurrency	It follows MVCC	Update in-place
7.	Preference	CouchDB favours availability.	MongoDB favors consistency.
8.	Performance Consistency	In CouchDB is safer than MongoDB	In MongoDB, database contains collection and contain document.
9.	Preferences	CouchDB favours availability	MongoDB favors consistency.
10.	Performance Consistency	In CouchDB is safer than MongoDB	In MongoDB, database contains collection and collection doc
11.	Consistency	CouchDB is eventually consistent	MongoDB is strongly consistent.

MongoDB - Regular Expression.

Regular Expressions are frequently used in all languages to search for pattern or word in any string. MongoDB also provides functionality of regular Expression for string pattern matching using the $regex operator. MongoDB uses PCRE as regular expression language.

Unlike text search, we do not need to do any configuration or command to use regular exp.

Assume we have inserted a document in a database named post.

```
>db.post.insert(
{
 "Post_text": "enjoy the mongodb articles on
   tutorial points",
 "tags" :[
      "mongodb", "tutorial point"
      ]
   }
 WriteResult({"nInserted":1})
```

Using Regex Expression :-

The following regex query searches for all the posts. containing string tutorialspoint in it.

```
>db.posts.find({post_text:{$regex:"tutorials-
   point"}}).pretty()
{
    "_id" : ObjectId("5dd7ce28f1dd483e710"),
```

```
"Post_text":"enjoy the mongodb articles on
  tutorialspoint",
"tags":[
        "mongodb","tutorialspoint"
  ]
}
{
  "_id":ObjectId("5dd7d111f1dd458e7103fe2"),
  "post_text":"enjoy the mongodb articles on
      tutorialspoint",
  "tags":["mongodb","tutorialspoint"]
  }
>
```

The same query can also be written as

```
> db.posts.find({post_text:/tutorialspoint/})
```

Using regex Expression with case Insensitive.
To make the search case insensitive, we use the $options parameter with value $i, The following command will look for strings having word tutorialspoint. irrespective of smaller or capital case.

```
> db.posts.find({text:{$regex:"tutorrialspoint",
  $options:"$i"}})
```

One of the result returned from this query the following document which contains the word tutorialspoint in diffrent cases.

Using regex for Array Elements.

We can also use the concept of regex on array field. This is particularly very important when we implement the functionality of tags so if you want to search for all the posts having tags beginning from the word tutorialpoint

```
>db.posts.find({$regex:"tutorialspoint"})
```

Optimizing Regular Expression Queries-

① If the document fields are indexed, the query will use of indexed values to match the regular expression. This makes the search very fast as compared to the regular expression scanning the whole collection.

② If the regular expression is prefix expression all the meant to start with certain string character.

Capped Collection.

Capped collections are fixed-size circular collection that follow the insertion order to support high performance for create, read, and delete operations. By circular, it means that when fixed allocation to collection is exausted It will start deleting the oldest document in the collection without providing any explicit commands

Creating Capped Collection:-

To create a capped collection, we use the

normal. create collection command but with capped option as true and specifying the maximum size of collection in bytes.

```
>db.createCollection("cappedLogCollection",
    {capped:true, size:1000})
```

In addition to collection size we can also limit the number of document in the collection using the max parameter.

```
>db.createCollection("cappedlogCollection", {
  capped:true, size:1000, max:1000}
```

If you want to check whether a collection is capped or not use following command.

```
>db.cappedLogCollection.isCapped()
```

If there is an existing collection which you are planning to convert to capped. you can do it with.

```
>db.runCommand({"ConvertToCapped":"Ports",
    size:1000})
```

Querying Capped Collection:-

By default, a find query on a capped collection will display insertion order. But if you want the documents to be retrived in reverse order use the sort command as shown

```
>db.cappedLogCollection.find().sort({$natural:-1})
```

Regarding capped collection,

① We cannot delete documents from a capped collection.

② There are no default indexes present in a capped collection, not even on _id field.

③ While inserting a new document, MongoDB does not have to actually look for a place to accomodate new document on the disk. It can blindly insert the new document at tail of the collection. This makes insert operations in capped collection very fast.

④ Simillarly, while reading documents MongoDB returns the documents in the same order as present on disk. This makes the read operation very fast.

Mongo DB - Auto Increment Sequence.

Mongo DB does not have out-of-the-box auto-increment functionality, like SQL databases, By default, it uses 12-byte ObjectId for the _id field as the primary key to uniqely indentify the documents. However they may be scenarioes where we may want the _id field to have some auto-incremented value other than Object Id.

Since this is not a default feature in MongoDB, we will programmatically achieve this functionality by using a counters collection as suggested by MongoDB collection.

Using Counter Collection :-

Consider the following products document. We want _id field to be an auto-incremented-integer sequence starting from 1,2,3,4 upto n

```
{
  "_id":1;
  "Product_name":"Apple iphone",
  "category":"mobiles"
}
```

For this, create a counters collection, which will keep track to last sequence value for all the sequense field.

```
>db.createCollection("counters")
```

Now we will insert the following document in the counters collection with projectid as its key.

```
>db.counters.insert({
      "_id":"productid",
      "Sequence_value":0
  })
  WriteResult({"nInserted":1})
?
```

The field sequence_value keeps track of the last value of the sequence

```
>db.counters.insert({_id:"productid", Sequence_value:0})
```

Creating Javascript Function:-

Now, we will create function getNextSequence-Value which will take the sequence name as its input, increment the sequence number by 1. and return the updated sequence number. In our case the sequence name is productid.

```
>function getNextSequenceValue(sequeneName)
 {
    var sequenceDocument = db.counters.
                                findAndModify({
  query : {_id : sequenceName }, update :
  {$inc: {sequence_value: 1 }},
      new: true
   });
  return sequenceDocument.sequence_value;
  }
```

Using the Javascript value. Function.

We will now use the function getNextSequence Value will creating a new document and assigning the returned sequence value as document's _id. field.

Insert 2 sample document using following code.

```
>db.products.insert({
  "_id": getNextSequenceValue("productid"),
    "product_name" : "Apple iphone",
    "category" : "mobiles"
  })
 >db.products.insert({
```

```
"_id" : getNextSequenceValue("productid"),
 "product_name" : "Samsung S3",
 "category" : "mobiles"
3)
```

We have used the getNextSequenceValue fun to set value for the _id field.

```
>db.products.find()
```

Redis Vs MongoDB.

	Redis	MongoDB.
Introduction	Redis is in memory data structure store used as database, cache and message broker	MongoDB is one of the most popular NoSQL database which follows the document store structure.
Primary database model	Redis follow key-value store model	MongoDB follows document store model
Official website	redis.io	www.mongodb.com.
Technical Documentation	You can get tech documentation of Redis on redis.io	You can get tech documentation of MongoDB on docs.mongodb.com

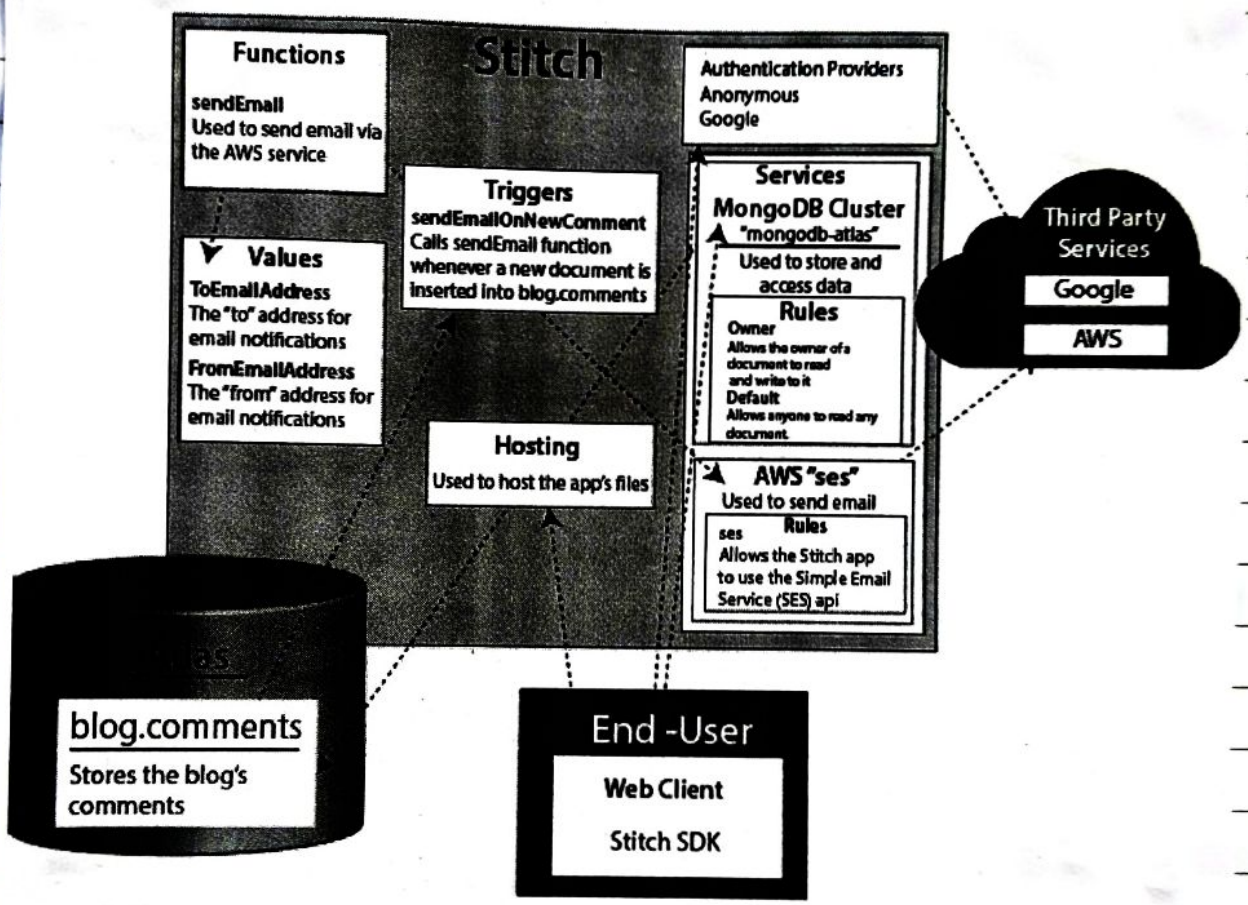
Cloud based	No	No
Server Operating System	BSD, Linux, OS X Windows	Linux, OS X, Solaris Windows.
Data Scheme	Schema-free	Schema-free
Secondary Index	No	Yes.
SQL	No	No
Server-side Scripts	Lua	JavaScript.
Foreign keys	No	No

MongoDB Cloud.

Mongodb Stitch

MongoDB provides a serverless platform to build an application quickly without setting up server infrastructures. MongoDB Stitch is designed as an upgrated version of MongoDB Atlas. It automatically integrates the connection to our database. Stitch illuminates the development and implementation process. It achived

it by neglating the requirement of building and deploying our backend. MongoDB stitch is available as a backend service that allows us to configure data-authentication, data access rules and services easily.



MongoDB stitch provides an upgradable infrastructure design to handle the request. It also coordinate the service and database interactions. We don't need to spend time and resource on tasks such as configuring our servers.

MongoDB Atlas :-

MongoDB Atlas is a cloud service by MongoDB. It is built for developers who'd rather spend time building apps them managing database The service is available on AWS, Azure and GCP.

It is worldwide cloud database service for modern application that give best-in-class automation and proven practice gurantee availability, scalability and compliance with the foremost demanding data security and privacy standards.

⑥ Advantage of MongoDB

1] Global clusters for world-class application :-
Using MongoDB Atlas, we are free to choose the cloud partner and ecosystem that fit our business stratergy.

2] Secure for Sensitive data -
It offers built-in security controls for all our data. It enables enterprise-grade features to integrate with our existing security protocol and compliance standard.

3] Designed for developer productivity :-
MongoDB Atlas moves faster with general tools to work with our data and a platform of services that makes it easy to build secure and extend application that run in MongoDB

- Reliable for mission-critical workload :-
  It is build with distributed fault tolerance and automated data recovery.

- Built for optional performance :-
  It makes it easy to scale our database

- Reliable for mission-critical workload :-
  It is built with distributed fault taularance and automated data recovery.

- Build for optimal performance :-
  It makes it easy to scale our databases in any directions. We can get more out of our existing resources with performance optimization tools and real-time visibility into database metrics.

- Managed for operational efficiency.
  It comes with build in operational best practices. So we can focus deliverying buisness value and accelarating development insted of managing database.

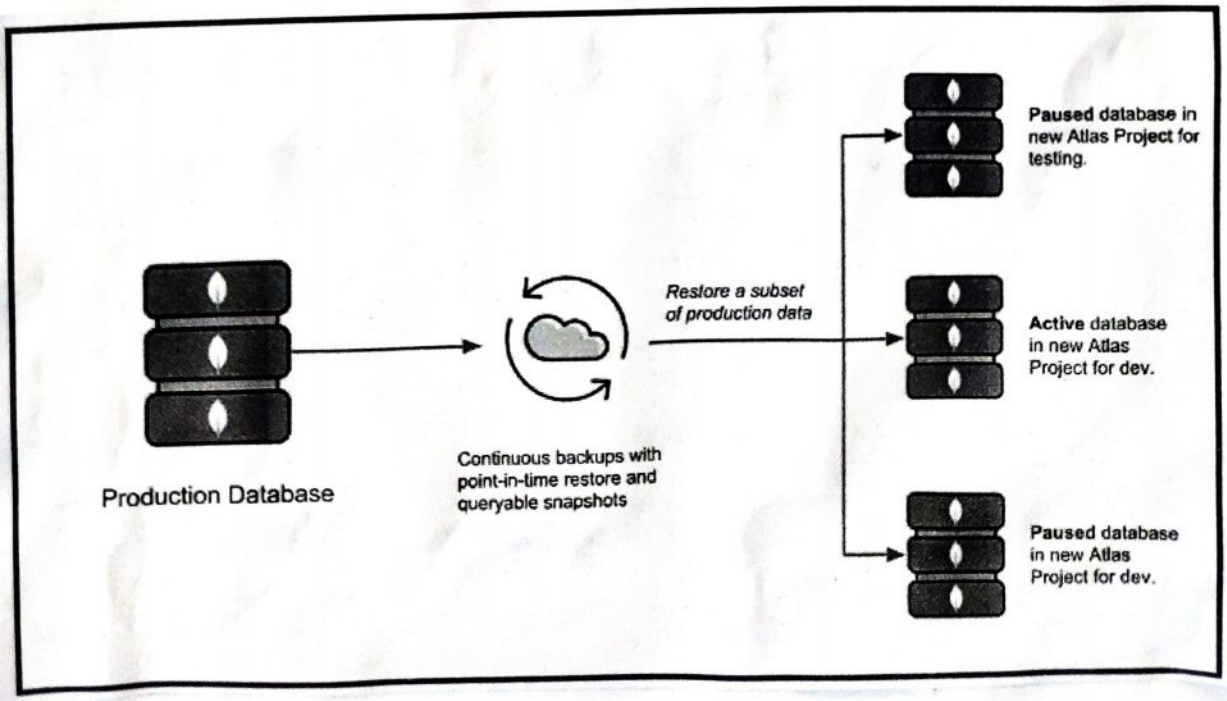
MongoDB Cloud Manager :-
The mongodb cloud manager is used to our manage our infrastructure by automating, monitoring and backups.

Automation :-
MongoDB nodes and clusters will be configured and maintained with the help of automation on each MongoDB host

Monitering :-
It provides real time reporting, alerting and visualization on key database and hardware indicators.

Backup :-
The scheduled snapshot and point-in-time recovery of our MongoDB shareded cluster and replica set are offered by the backup facility of the cloud manager.

How Backup Works.
The cloud manager takes a snapshot of the data that we have specified to take Backup when we activate backup for the MongoDB deployment. First, create an invisible manager of the replica set and performs the initial syns of the deployment data and Then the backup tails all the replica set onlog the Backup continuously.



Restoration of Data :-

We can restore data from a complete scheduled or a selected point between picture.

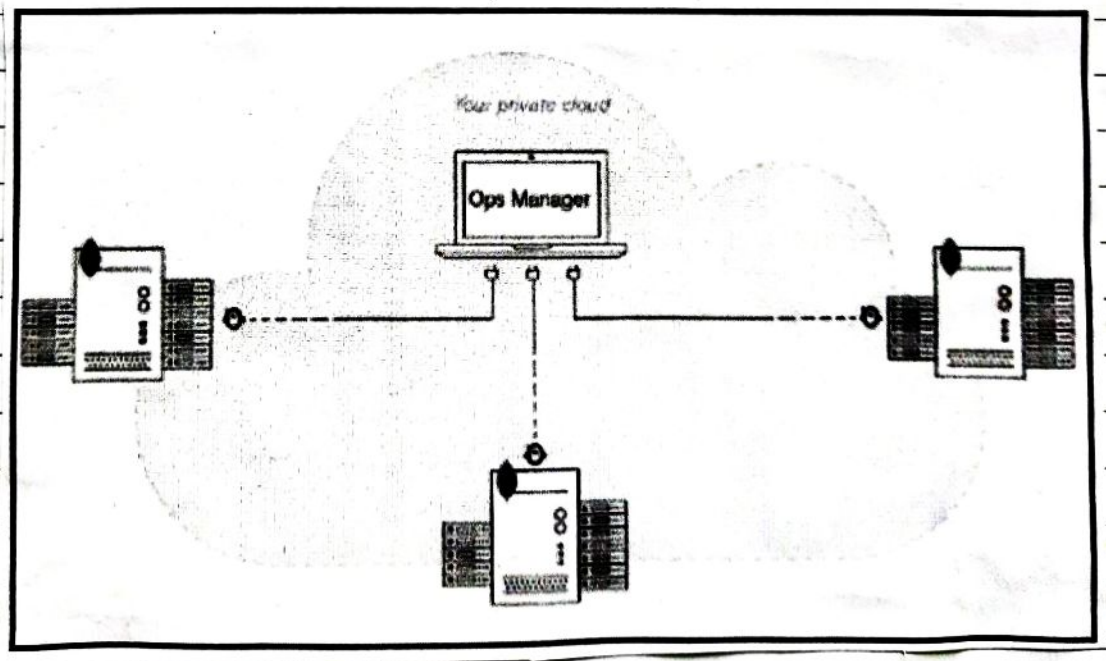
① We can restore from checkpoints between snapshot for sharded cluster.

② We can restore from selected points in time for replica set.

Cloud manager reads directly from selected points in time for replica sets.

MongoDB OPS manager :-

Using the ops manager, we can moniter, automate and backup our MongoDB infrastructure



Automation :-

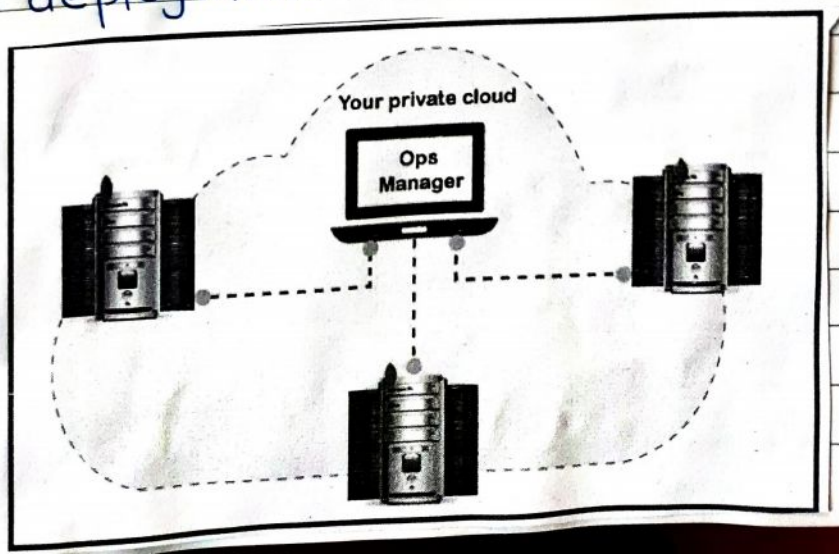
We can configure and maintain the nodes and cluster of MongoDB using the OPS Manager

Monitering :-

We can report, visualize and gets alert in real time using MongoDB, the OPs manager monit.
oring feature.

Backup :-

The scheduled snapshot and point-in-time recovery of our MongoDB sharded cluster and Replica sets are offered by the backup facility of the manager.

Server Pool of OPs manager :-

Using the Ops Manager Server pool, we can get administrative privillage pool of provisioned server that already have Automation installed maintained by the administrative of OPS manager we can request server from server pool to host the deployment when we want to create a new deployment.