

UNIT: 1: INTRODUCTION TO NOSQL DATABASE

1. DEFINE : NOSQL

- NoSQL stands for “Not Only SQL”.
- NoSQL is a database that provides mechanism to store & retrieve data other than relational database.
- It is a class of DBMS that do not follow all of the rules of a RDBMS and cannot use traditional SQL to query data.
- It doesn't required a fixed schema and have simple API
- There are no needs to design a table to store data inside it.
- NoSQL database is used for distributed data stores with huge data storage needs
- NoSQL is used for Big data and real-time web apps.
- For example companies like Twitter, Face book, Google that collect terabytes of user data every single day.
- The primary objective of NoSQL database are:
 1. **Design simplicity**
 - Schema free so very easy to design & retrieve data
 2. **Horizontal scaling**
 - Support horizontal scaling so load of database is balanced.
 - Partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved & shared.
 3. **High availability**
 - Auto replication feature in NoSQL databases.
 - This makes it highly available because in case of any failure data replicates itself to the previous consistent state.
 4. **Replication**
 - It's a technique that copying of data to transferred to integrated with another location of database.
- Nosql database uses different data structure as compared to relational dataset.
- This are the list of some nosql database:
 1. Mongodb
 2. Redis
 3. Couch Db
 4. Cassandra
 5. Voldemort

6. Info grid
7. Hbase
8. Neo4j

2.SHORT NOTE: HISTORY OF NOSQL

- Relational database are invented in 1970s to that make possible to store data in the form of table which is combination or rows & columns.
- To communicate with relational database a language called SQL was introduced & Almost all relational database systems use SQL.
- SQL is more rigid to work with unstructured data.
- In the mid-1990s, the internet becomes more popular, and relational databases simply could not keep up with the flow of information demanded by users, as well as the larger variety of data types that occurred from this evolution.
- This led to the development of non-relational databases, often referred to as NoSQL

EVOLUTION OF NOSQL

- The acronym NoSQL was first used in **1998 by Carlo Strozzi** while naming his lightweight, open-source “relational” database that did not use SQL.
- Carlo Strozzi suggests that, because the current NoSQL movement "departs from the relational model altogether, it should therefore have been called more appropriately 'NoREL'-- referring to 'No Relational'.
- The name came up again in **2009** when **Eric Evans and Johan Oskarsson** used it to describe non-relational databases
- Johan Oskarsson, then a developer at Last.fm, reintroduced the term NoSQL in early 2009 when he organized an event to discuss "open source distributed, non relational databases".
- The name attempted to label the emergence of an increasing number of non-relational, distributed data stores, including open source clones of Google's Bitable/Map Reduce and Amazon's Dynamo.
- Based on **2014** revenue, the NoSQL market leaders are Mark Logic, MongoDB, and Datastax.
- Based on **2015** popularity rankings, the most popular NoSQL databases are MongoDB, Apache Cassandra, and Redis.

3.WHAT IS DIFFERENCE BETWEEN NOSQL & SQL DATABASE.

SQL	NOSQL
SQL stands for Structure Query Language	NOSQL stands for Not Only SQL
SQL is a query language that used to communicate with relational database	There is No declarative query language to communicate with database.
SQL databases are primarily called RDBMS or Relational Databases	NoSQL databases are primarily called as Non-relational or distributed database
SQL or relational database stores structure oriented data	NOSQL database deals with semi-structure & unstructured data
Traditional RDBMS uses SQL syntax and queries to analyze and get the data for further use.	NoSQL database system consists of various kinds of database technologies that developed in response to the demands presented for the development of the modern application.
SQL databases are table based databases	NoSQL databases can be document based, key-value pairs, graph databases
SQL databases have a predefined schema	NoSQL databases use dynamic schema for unstructured data.
SQL databases are vertically scalable	NoSQL databases are horizontally scalable
An ideal choice for the complex query intensive environment.	It is not good fit complex queries.
SQL databases are not suitable for hierarchical data storage.	More suitable for the hierarchical data store as it supports key-value pair method.
It was developed in the 1970s to deal with issues with flat file storage	Developed in the late 2000s to overcome issues and limitations of SQL databases.
Most of RDBMS systems are proprietary	Most of NOSQL systems are Open-source
It should be used when data validity is super important	Use when it's more important to have fast data than correct data
Examples: Oracle, Postgres, and MS-SQL.	Examples: MongoDB, Redis, Neo4j, Cassandra, Hbase.

4.WRITE DOWN ADVANTAGES & DISADVANTAGES OF NOSQL DATABASE.

CHARACTERISTICS OF NOSQL (ADVANTAGES):

1. Structured & unstructured data:

- NoSQL systems database can store structured data, semi structured data un structured data so that data can be managed easily.

2. Multi-format data:

- NoSQL systems store and retrieve data from many formats like key-value stores, graph databases, column-family (Bitable) stores, document stores, and even rows in tables.

3. Open source:

- NoSQL is an open source so its software's are easily available on internet.

4. Free of joins:

- NoSQL systems allow you to extract your data using simple interfaces without joins.

5. Schema-free:

- In NoSQL systems user do not need to define schema for data because it is schema free.

6. Works on many processors:

- NoSQL systems allow you to store your database on multiple processors and maintain high-speed performance.

7. Less cost:

- NoSQL database is less expensive for storage & transactions.

8. Linear scalability:

- Huge amount of data can store on NoSQL database.
- When you add more processors, you get a consistent increase in performance.

9. Innovative:

- NoSQL offers options to a single way of storing, retrieving, and manipulating data.

10. Highly distributable:

- NoSQL uses the powerful, efficient architecture instead of the expensive single architecture that make possible to distribute data very fast & efficiently.

11. Continues availability:

- A database server can stay online as well as offline.
- Users can also use server 24 * 7 so server may continues available according to needs of users

12. Non-relational:

- The information in NoSQL is stored in the form aggregate.

- A single record stores every information about the transaction, including the delivery address.

DISADVANTAGES:

- NoSQL is open-source database; so there is no reliable standard for NoSQL yet.
- GUI mode tools to access the database is not flexibly available in the market.
- Some database systems like MongoDB and CouchDB store data in JSON format Which means that documents are quite large
- Relational databases are a better choice in the field of Transaction Management than NoSQL.
- Backup is a great weak point for some NoSQL databases. There is no approach for the backup of data in a consistent manner.

5.WRITE DOWN EXTERNAL BENIFIT OF NOSQL.

1. Database administration:

- NoSQL database requires less administrative task because it has automatic management with data

2. Schema-free:

- In NoSQL systems user do not need to define schema for data because it is schema free.

3. Distributable:

- Nosql database can be distributed among multiple servers, so it can be easily available to multiple users.

4. Easy structure:

- Users can easily store the data in nosql database.
- Users can also retrieve the data easily from the server.

5. Object oriented:

- Nosql also supports object oriented programming.

6. Own access language:

- Nosql database has its own access language.
- There is no common language for all nosql database like SQL.

7. Automatic management:

- In nosql when user insert records it will automatically create the table in database as well as when you insert record a database will create unique id for new record automatically

8. Multiple data structure:

- Nosql database support different data structure like key-value, column list, parent-child structure etc.

6.WRITE DOWN TYPES OF NOSQL WITH ADVANTAGES

DISADVANTAGES & USE CASE:

- NOSQL provides different types of database technology.
- According to types of data storage there are four types of NoSQL database:
 - Key value type
 - Document type
 - Graph type
 - Wide-column stores type

Key value type:

- Key-value types are the simplest NoSQL type.
- In this type of database all data are stored with key & its associated value.
- It is also known as associative arrays, organized into rows.
- These databases store the data as a hash table with a unique key and a pointer to a particular item of data.
- The key-value stores are used whenever the data would be queried by precise parameters and needs to be retrieved really fast.
- It is designed in such a way to handle lots of data and heavy load.
- The simplest example of key value is :directory

Key	Value
Ami	(0281) 2571-662
Mansi	(0281) 2456-555
Priya	(0281) 2123-456
Toral	(0281) 2456-789

- This is a simple phone directory. The person's name is the key, and the phone number is the value.
- This type of database is very quick to query, due to its simplicity.
- E.g.: Riak, Voldemort, and Redis are the most well-known in this category.

Document value type:

- As like name document store database store data in document format.
- A document database is a type of non-relational database that is designed to store and query data as JSON-like documents.
- Document store NoSQL databases are similar to key-value databases in that there's a key and a value.
- In this type Data is stored as a value while its associated key is the unique identifier for that value.
- The difference is that, in a document database, the value contains structured or semi-structured data while key: value store
- This structured/semi-structured value is referred to as a document and can be in XML, JSON or BSON format.
- Document databases make it easier for developers to store and query data in a database by using the same document-model format they use in their application code.
- Document databases enable flexible indexing, powerful ad hoc queries, and analytics over collections of documents.
- For eg. Here is an XML document

```
<artist>
  <artist name>Iron Maiden</artist name>
  <albums>
    <album>
      <album name>The Book of Souls</album name>
      <date released>2015</date released>
      <genre>Hard Rock</genre>
    </album>
    <album>
      <album name>Powerslave</album name>
      <date released>1984</date released>
      <genre>Hard Rock</genre>
    </album>
  </albums>
</artist>
```

- And here's the same example, but this time written in JSON.

```
{
  '_id' : 1,
  'artistName' : { 'Iron Maiden' },
  'albums' : [
```

```
{
  'albumname' : 'The Book of Souls',
  'date released' : 2015,
  'genre' : 'Hard Rock'
},
{
  'albumname' : 'Powerslave',
  'date released' : 1984,
  'genre' : 'Hard Rock'
}
]
```

- Use cases: Document store databases are preferable for:
 - E-commerce platforms
 - Content management systems
 - Analytics platforms
 - Blogging platforms
 - Profile Management
 - catalogs

Examples

- MongoDB
- couchbase
- Elasticsearch.

7.EXPLAIN DATABASE AS A SERVICE.

DATABASE AS A SERVICE

- When a database stored on server then it is called service based database.
- This type of database uses MDF – multiple data format.
- Database as a service - DBaaS is a concept of cloud computing that allows you to store database on a cloud storage server.
- The DBaaS provides the equipment, software, and infrastructure needed for businesses to run their database on the DBaaS.
- It removes the worry about the key factors for managing a database like database administration, license, scalability, backup, security etc.
- DBaaS is a managed service offering assessment to a database along with the applications and their related data.

- The service provider responsible for installing, updating and ensuring for performance of the database
- DBaaS provide service on “pay as you go” model.
- It allowing the users to pay for the usage instead of a fixed cost paid for licenses and investment on purchasing and maintaining the physical machine.
- When u required more facility u have to pay for additional services.
- Database services take care of scalability and high availability of the database.
- The DBaaS model can also help reduce data and database redundancy and improve overall Quality of Service.
- Database configuration installation can be done by db itself.
- Some db itself provides its own DBaaS like oracle provides 11g and 12c as cloud services.
- Most db services offer web based console.
- Some db manager components provide many API for db services. For example create instance, create snapshot, monitoring, auto backup, security, reports, are the different services.
- DBaaS solutions can reduces the costing with cloud based db so very much affordable and flexibility for small businesses.
- Example:
 - Microsoft Azure SQL Database
 - MongoDB Atlas
 - SimpleDB
 - Amazon Relational Database Service - RDS
 - Google BigQuery
 - IBM Db2 on Cloud
 - Oracle Database 11g and 12c

ADVANTAGES:

- You can access data virtually from anywhere.
- You don't have to buy your own equipment or software licenses
- You don't have to hire database developers & other man power to maintain database
- You don't have to build a database system
- You don't pay the power bill for running all the servers
- Backup of data can share easily.
- A DBaaS often comes with uptime guarantees

- DBaaS teams are experienced, and know how to handle a variety of bugs and problems
- The DBaaS can usually devote more resources to their equipment, thus buying better servers and hardware than most small businesses can afford.

DISADVANTAGES:

- You don't have control of your own data.
- Reduced visibility of the backend
- You must have an internet connection to access data.
- If system goes down, you don't have access to your database.
- You don't have direct control to the servers that are running your database.
- Issue of cost-at-scale.

8.WHAT IS MONGODB?

- MongoDB is a document-oriented NoSQL database used for high volume data storage.
- MongoDB is an open source document based NoSQL database that can install on different platforms like Windows, Linux etc.
- mongoDB is a non relational database.
- It is a cross-platform, document oriented database that provides, high performance and easy scalability.
- MongoDB is a NoSQL type database that stores data in document format instead of having data in a relational type format it stores the data in documents.
- Relational database have a fixed data structure while mongodb is schema free.
- In MongoDB Data is stored in the form of JSON style documents.
- Main purpose to build MongoDB is :
 1. Scalability
 2. Performance
 3. High Availability
- MongoDB is suitable for following applications:
 1. Big Data
 2. Content Management and Delivery
 3. Mobile and Social Infrastructure
 4. User Data Management
 5. Data Hub
- MongoDB works on concept of collection and document.

1. Database
2. Collection
3. Document

FEATURES / ADVANTAGES OF MONGODB:

1. Document based database:

- MongoDB stores data in document format like JSON, BSON etc. so it is known as document storage database.

2. Schema-free:

- Unlike RDBS MongoDB do not contain any fixed schema.
- MongoDB contain different types of documents & there are no needs to define schema or structure.
- Each document contains different number of field size systems user does not need to define schema for data because it is schema free.

3. No Complex Join:

- MongoDB provides Simple query structure that make possible to access data easily.
- It's a non relational database so no needs to perform complex join operations to retrieve data.

4. Ad hoc queries:

- MongoDB supports searching by field, range queries, and regular expression searches.
- Queries can be made to return specific fields within documents.

5. Automatic primary key:

- MongoDB creates unique key for each document store inside it.
- When you store data it automatically gives unique key for each document which is known as _id.

6. Fast & alternative development:

- It has dynamic schema so it is fast for developer to build application against RDBMS.

7. Load balancing:

- MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances.
- MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

8. Scalability:

- A system that scales well will be able to maintain and increase its level of performance under larger operational demands.

- MongoDB supports horizontal scaling through Sharding, distributing data across several machines and facilitating high throughput operations with large sets of data.

9. Indexing:

- Indexes can be created to improve the performance of searches within MongoDB.
- Any field in a MongoDB document can be indexed.

10. Replication:

- MongoDB supports Master Slave replication.
- A master can perform Reads and Writes and a Slave copies data from the master and can only be used for reads or back up.

11. Duplication of Data:

- MongoDB can run over multiple servers.
- The data is duplicated to keep the system up and also keep its running condition in case of hardware failure.

9. WHAT IS DOCUMENT, DATABASE AND COLLECTION?

- **Database**

1. Database is a physical container for collections.
2. Each database gets its own set of files on the file system.
3. A single MongoDB server typically has multiple databases.

- **Collection**

1. Collection is a group of MongoDB documents.
2. It is the equivalent of an RDBMS table.
3. A collection exists within a single database.
4. Documents within a collection can have different fields.
5. All documents in a collection are of similar or related purpose.



- **Document –**

1. A record in a MongoDB collection is basically called a document.
2. The document, in turn, will consist of field name and values.

10. WRITE DOWN INSTALLATION STEPS OF MONGODB?

- First check the os architecture by placing following command in cmd
C:> wmic os get osarchitecture
- It will shows os architecture either 32bit or 64 bit.
- Download the mongodb msi installer package from www.mongodb.org
- Install mongodb with the installation wizard.
- Run `mongodb-win32-x86_64-2008plus-ssl-3.0.5-signed` in your download directory
- It will starts wizard .
- First screen shows welcome screen of mongodb click on **next button**
- Accept license agreement & click on **next button.**
- Next dialog box shows choose set type : select complete & **click on next.**
- Now click on **install button.**
- After the setup has been finished, you can check in you `c:\program files` a mongodb folder must have been created.

NOW SET UP MONGODB ENVIRONMENT:

- Mongodb requires a data directory to store all data. Mongodb's default data directory path is `\data\db`.
- To configure this go to your C: drive and create `data\db` folder ("db" folder inside "data" folder)
So Now full path of db folder will be - `C:\data\db`
- Now open cmd & type go to mongodb directory.
C:\users\stud cd C:\Program Files (x86)\mongodb\Server\3.0\bin
- Now run `mongod` method mongodb server has been started on port - 27017
- Now open another cmd prompt and type `mongo`.
- This shows that you have been connected with mongodb server.
- Now you can perform all mongodb commands.

11. HOW TO CREATE DATABASE IN MONGODB?

CREATE DATABASE:

use command:

- SYNTAX:

use <DATABASE_NAME>

- USE command is used to create a database in mongodb.
- The command will create a new database, if it doesn't exist otherwise it will return the existing database.

db command:

- **SYNTAX:**
db
- To check your currently selected database use the command db

show dbs:

- **SYNTAX:**
show dbs
- This command used to display all database created by user.

DELETE DATABASE:

- **SYNTAX:**
db.dropDatabase()
- To remove database from mongodb use db.dropDatabase().
- This will delete the selected database.
- before delete any database first use database with its name.
- If you have not selected any database, then it will delete default 'test' database.
- for example:

```
> use stud
> db.dropDatabase()
```

CREATE COLLECTION IN MONGODB:

- In MongoDB, the first basic step is to have a database and collection in place.
- The database is used to store all of the collections, and the collection in turn is used to store all of the documents.
- use insert() to create collection

insert():

- **syntax:**
db.collection_name.insert()

UNIT: 2 LEARNING MONGODB BY IMPLEMENTING WEB APPLICATION

1. WHAT IS QUERY? WHAT ARE THE TYPES OF QUERY DOCUMENT IN MONGODB?

- Query is a request which is used to retrieve data from the database.
- By using query we can instruct database engine to perform some specific task.
- Query make possible to interact with database.
- MongoDB use document database to store data.
- It provides many different types of queries to perform some specific operations.
- In MongoDB queries are defined by using methods.
- MongoDB provides following types of queries:

1. Key-value queries

- It is basically used to retrieve the documents according to specified keys/fields and its values
- For ex: Display documents where stream is mca
> **db.stud.find({stream:"mca"})**

2. Range queries

- It is used to retrieve the documents by specifying range of values.
- For ex: Display all documents where per between 60 to 80
> **db.stud.find({\$or:[{\$per: {\$gte:60}},{\$per:{\$lte:80}}]})**

3. Text search queries

- It is used to search text data from documents.
- It uses Boolean operators & return result in text argument

4. Aggregation queries

- It is used to produce aggregate result on group of key/ fields values rather than a single key
- You can use aggregate functions along with group key/fields
- For ex: Display stream wise highest percentage from documents
>**db.stud.aggregate**
([{\$group: {_id:stream}, highper {\$max:"\$per}}])

5. Map reduce queries

- Map reduce is function which is used to perform some complex queries.
- It allows complex data processing by first combine all the data then arrange it & reduce the data & display documents according to query specified by users

2. HOW TO CREATE DATABASE IN MONGODB?

OR

EXPLAIN DATABASE HANDLING COMMANDS.

CREATE DATABASE:

use command:

- **SYNTAX:**

use <DATABASE_NAME>

- USE command is used to create a database in MongoDB.
- The command will create a new database; if it doesn't exist otherwise it will return the existing database.
- for example:

```
> use student
```

DISPLAY DATABASE:

db command:

- **SYNTAX:**

db

- To check your currently selected database use the command db

show dbs:

- **SYNTAX:**

show dbs

- This command used to display all database created by user.

DELETE DATABASE:

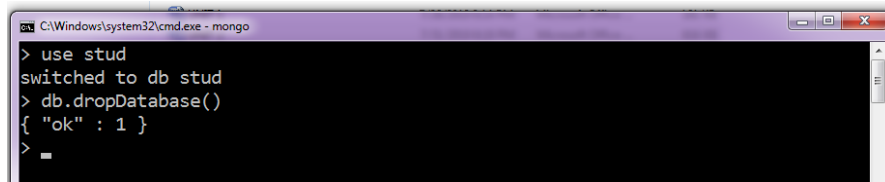
- **SYNTAX:**

db.dropDatabase()

- To remove database from MongoDB use `db.dropDatabase()`.
- This will delete the selected database.
- Before delete any database first use database with its name.
- If you have not selected any database, then it will delete default 'test' database.
- for example:

```
> use stud
> db.dropDatabase()
```

OUTPUT:



```
C:\Windows\system32\cmd.exe - mongo
> use stud
switched to db stud
> db.dropDatabase()
{ "ok" : 1 }
>
```

3. WRITE A NOTE ON COLLECTION HANDLING COMMANDS OF MONGODB.

CREATE COLLECTION IN MONGODB:

- In MongoDB, the first basic step is to have a database and collection in place.
- The database is used to store all of the collections, and the collection in turn is used to store all of the documents.
- collection can be created by using two ways:
 1. using `createCollection()` method
 2. using `insert()` to create collection

1. `db.createCollection()`:

- **syntax:**
`db.createCollection(name,options)`

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

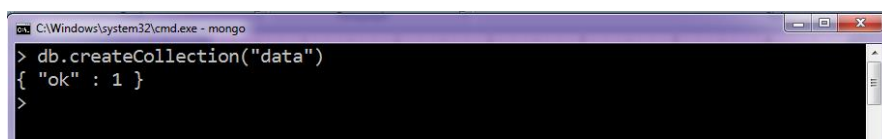
Field	Type	Description
Capped	Boolean	<ul style="list-style-type: none"> • (Optional) If true, enables a capped collection. • Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. • If you specify true, you need to specify size parameter also.
autoIndexId	Boolean	<ul style="list-style-type: none"> • (Optional) If true, automatically create index on _id field • Default value is false.
Size	Number	<ul style="list-style-type: none"> • (Optional) Specifies a maximum size in bytes for a capped collection. • If capped is true, then you need to specify this field also.
Max	Number	<ul style="list-style-type: none"> • (Optional) Specifies the maximum number of documents allowed in the capped collection.

- for example:

EXAMPLE: 1

```
> use stud
> db.createCollection("data")
```

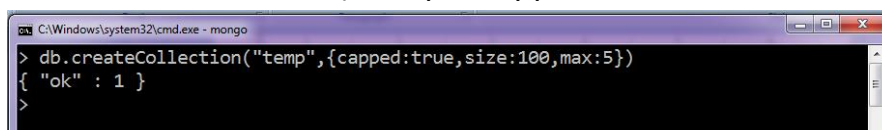
OUTPUT:



```
C:\Windows\system32\cmd.exe - mongo
> db.createCollection("data")
{ "ok" : 1 }
>
```

EXAMPLE: 2

```
> use stud
> db.createCollection("temp",{capped:true,size:100,max:5})
```



```
C:\Windows\system32\cmd.exe - mongo
> db.createCollection("temp",{capped:true,size:100,max:5})
{ "ok" : 1 }
>
```

- in above example
 1. you can add maximum 5 documents & when document size reach to 100 then MongoDB start overwrite document
- we can display collection by using **show Collections()**

2. insert():

- **syntax:**

db.collection_name.insert()

- **Insert () command** to insert documents into a collection.
- We can also use **db.collection.save()** to save document.

EXAMPLE:

```
> use student
> db.student.insert(
  {
    "rlno" : 1,
    "name": "meghna"
  })
```

DISPLAY ALL COLLECTION IN MONGODB:

- **syntax:**

show Collections

- MongoDB allows you to display all collection by using show collection method.
- This method list out all collection created in currently used document.

EXAMPLE:

```
> show collections
```

DELETE COLLECTION IN MONGODB:

- **syntax:**

db.collection_name.drop()

- MongoDB allows you to remove collection.
- To remove collection **drop()** will be used.
- After delete the collection method will return true, if the selected collection is not delete successfully then MongoDB return false.

- for example:

```
> show collections
> db.student.drop()
```

OUTPUT:



```
C:\Windows\system32\cmd.exe - mongo
> db.data.drop()
true
>
```

RENAME COLLECTION IN MONGODB:

- **syntax:**

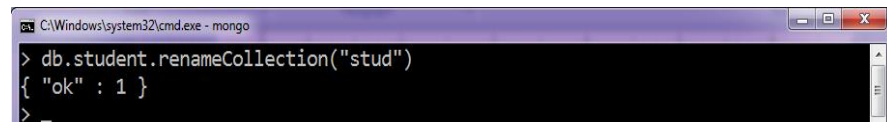
db.collection.renameCollection(target)

- MongoDB allows you to rename collection by using db.collection.renameCollection method.
- In syntax:
 1. Target specify the new name of collection.

EXAMPLE:

```
> db.student.renameCollection("stud")
```

OUTPUT:



```
C:\Windows\system32\cmd.exe - mongo
> db.student.renameCollection("stud")
{ "ok" : 1 }
>
```

COPY COLLECTION IN MONGODB:

- **syntax:**

db.collection.copyTo(target)

- MongoDB allows you to copy collection by using db.collection.copyTo().
- copyTo() returns the number of documents copied. If the copy fails, it throws an exception.
- In syntax:
 1. Target specify the new name of collection.

EXAMPLE:

```
> show collections
student
> db.stud.copyTo("student")
```

VIEW STORAGE SIZE OF COLLECTION IN MONGODB:

- **syntax:**
db.collection.storageSize()
- The total amount of storage allocated to this collection for document storage.

EXAMPLE:

```
> show collections
student
> db.student.storageSize()
```

Output:

32767

VIEW SIZE OF COLLECTION IN MONGODB:

- **syntax:**
db.collection.dataSize()
- MongoDB allows you to view the size of collection by using `db.collection.dataSize()`
- `dataSize()` return The size in bytes of the collection.

EXAMPLE:

```
> show collections
> db.student.dataSize()
```

Output:

336

4. HOW TO INSERT DOCUMENT IN MONGODB.

- MongoDB provides three ways to insert document.
 1. `insert()`
 2. `insertOne()`
 3. `insertMany()`

`insert()`:

- **syntax:**
db.collection_name.insert(<document>)
- **insert () command** to insert documents into a collection.
- We can also use **db.collection.save()** to save document.

EXAMPLE:

```
> use student
> db.student.insert
(
    {
        "rln0" : 1,
        "name": "meghna"
    }
)
```

OUTPUT:

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

- In above example :
 1. "insert statement" used to insert document into the collection.
 2. The second part of the statement is to add the Field name and the Field value, in other words, what is the document in the collection going to contain
- we can also add multiple values inside document using insert()
- To do this we have to create a JavaScript variable & then store values inside it.

EXAMPLE:

```
> use student
> var s=[
    {
        "rln0" : 1,
        "name": "meghna"
    },
    {
        "rln0" : 2,
        "name": "ekta"
    },
    {
        "rln0" : 3,
        "name": "mansi"
    }
];
> db.student.insert(s)
```

insertOne():

- **syntax:**

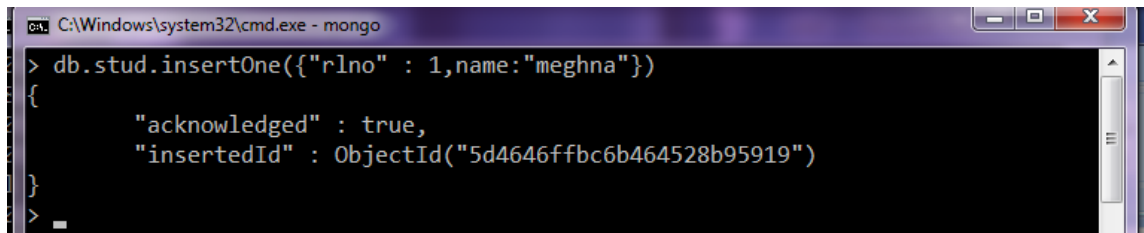
db.collection_name.insertOne(<document>)

- **insertOne () command** to insert a single documents into a collection.
- This method is available with MongoDB version 3.2 or newer version launch after 3.2
- After insert document in collection this method return unique ID of the document.
- This ID can be used to find document from collection.

EXAMPLE :

```
> use student
> db.stud.insertOne(
  {
    "rlno" : 1,
    "name": "meghna"
  })
```

OUTPUT:

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe - mongo". The prompt shows the command `> db.stud.insertOne({"rlno" : 1,name:"meghna"})` and its output: `{ "acknowledged" : true, "insertedId" : ObjectId("5d4646ffbc6b464528b95919") }`. The prompt is ready for the next command with `> _`.

insertMany():

- **syntax: db.collection_name.insertMany(<document1>,<document2>.....)**
- **insertMany () command** to insert a multiple documents into a collection.
- This method is available with MongoDB version 3.2 or newer version launch after 3.2
- After insert document in collection this method return unique ID of the each document.
- This ID can be used to find document from collection.

EXAMPLE:

```
> use student
> db.stud.insertMany(
```

```

{
  "rlno" : 1,
  "name": "meghna"
},
{
  "rlno":2,
  "name":"ekta"
})

```

OUTPUT:

```

{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5d43bf71bf1ab6306bca84a8"),
    ObjectId("5d43bf71bf1ab6306bca84a9")
  ]
}

```

save ()

Syntax: db.collection_name.save(<document>,writeConcern)

- MongoDB provides save() that allows you to insert document in collection

document	<ul style="list-style-type: none"> • Document that contain {key: value} to store in collection
writeConcern	<ul style="list-style-type: none"> • Optional • Boolean value that specify whether a message after inserting document will be displayed or not. • If false then omit write concern.

EXAMPLE:

```

> use student
> db.student.save
(
  {
    "rlno" : 10,
    "name": "meghna"
  }
)

```

OUTPUT:


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

5. EXPLAIN QUERING DOCUMENT IN MONGODB.

- MongoDB allows you to display data stored in collection.
- While performing a query operation, one can also use criteria's or conditions which can be used to retrieve specific data from the database?

DISPLAY ALL DOCUMENTS USING FIND()

find ()

- **syntax:** `db.collection_name.find(<criteria>)`
- find () allows you to display all document stored in your collection.
- By using this function we can display all documents of currently used database.
- We can specify criteria for retrieval documents with find().

EXAMPLE:

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

- Above example display all records/documents from stud collection.

DISPLAY ALL DOCUMENTS OF STUD COLLECTION USING FIND()

EXAMPLE:

```
>use college
>db.stud.find( {stream : "mscit" })
```

- Above example display all records/documents from stud collection who are in mscit stream.

DISPLAY ALL DOCUMENTS OF STUD IN READABLE FORMAT

- Two functions are used to display output in JSON format.
 1. pretty
 2. printJson

pretty ()

- **syntax:** `db.collection_name.find().pretty()`
- When we display output with find () the output we get is not in any format and less-readable.
- The pretty command improve the readability, we can format the output in json format.

EXAMPLE:

```
>use college  
>db.stud.find().pretty()
```

- Above example display all records/documents from stud collection in JSON format

```
> db.s.find().pretty()  
{  
  "_id" : ObjectId("5d3fd14d4bd0952689fd4b36"),  
  "rlno" : 1,  
  "name" : "ekta",  
  "streem" : "mscit"  
}  
{  
  "_id" : ObjectId("5d3fd14d4bd0952689fd4b37"),  
  "rlno" : 2,  
  "name" : "meghna",  
  "streem" : "mscit"  
}  
{  
  "_id" : ObjectId("5d3fd14d4bd0952689fd4b38"),  
  "rlno" : 3,  
  "name" : "nihar",  
  "streem" : "mscit"  
}
```

print JSON()

- **syntax:** `db.collection_name.find().forEach(printjson);`
- This function works same as pretty ().
- It is also used to display output in json format.

EXAMPLE:

```
>use college  
>db.stud.find().pretty().forEach(printjson)
```

- Above example display all records/documents from stud collection in JSON format

```
C:\Windows\system32\cmd.exe - mongo
> db.s.find().forEach(printjson);
{
  "_id" : ObjectId("5d3fd14d4bd0952689fd4b36"),
  "r_lno" : 1,
  "name" : "ekta",
  "stream" : "mscit"
}
{
  "_id" : ObjectId("5d3fd14d4bd0952689fd4b37"),
  "r_lno" : 2,
  "name" : "meghna",
  "stream" : "mscit"
}
{
  "_id" : ObjectId("5d3fd14d4bd0952689fd4b38"),
  "r_lno" : 3,
  "name" : "nihar",
  "stream" : "mscit"
}
```

- We can use various operators to retrieve documents from database by using find()

EXAMPLE:

1. DISPLAY ALL WHO'S PER MORE THAN 70

```
> db.stud.find ( {per : { $gt : 70 } }).pretty()
```

2. DISPLAY ALL WHERE STREAM = "MSCIT" OR STREAM ="MCA"

```
>db.stud.find ( { $or: [ { stream: { $eq: "MSCIT" } }, { stream { $eq: "MCA" } } ] })
```

6. HOW TO UPDATE DOCUMENT IN MONGODB.

update ():

- syntax:

db.collection_name.update (selection_criteria, updated_data)

- Update () is used to update existing data stored in document.
- We need to specify selection criteria for update a document & then update data using \$set keyword.

selection_criteria	<ul style="list-style-type: none">• Specify key: value of filed for updation
updated_data	<ul style="list-style-type: none">• Update data use \$set keyword• In this section you have to specify the field along with value

UPDATE A SINGLE DOCUMENT

- To update document:
 1. Issue the update command
 2. Choose the condition which you want to use to decide for update document.
 3. Use the **\$set** command to modify the Field Name
 4. Choose which Field Name you want to modify and enter the new value.

EXAMPLE:

```
db.stud.update({rolno:1},{$set :{streem:"mca"}})
```

OUTPUT:

```
> db.c.update({rlno:1},{$set: {nm:"megha"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

UPDATE MULTIPLE DOCUMENTS:

- We can update multiple documents in MongoDB by using update()
- To update multiple documents in MongoDB use **multi** option.
- Multi option contains Boolean value either true or false.
- By default multi option set to false; to do multiple update use multi : true
- To update multiple document
 1. Issue the update command
 2. Choose the condition which you want to use to decide for update document.
 3. Use the **\$set** command to modify the Field Name
 4. Choose which Field Name you want to modify and enter the new value.
 5. Specify **{multi:true}**

EXAMPLE:

```
>db.stud.update({streem:"mscit"},{$set :{streem:"mca"}},{multi:true})
```

OUTPUT:

```
WriteResult({ "nMatched" : 4, "nUpserted" : 0, "nModified" : 4 })
```

7. EXPLAIN UPDATE OPERATORS WITH EXAMPLE.

- There are several operators used with update command which is divided into two parts:
 1. Field update operator
 2. Array update operator

Field update operator

- Field update operators are basically used to update fields. Following are field update operators:
 1. \$set
 2. \$unset
 3. \$inc
 4. \$rename

\$set:

- **Syntax:** { \$set: { <field1>: <value1>, ... } }
- This operator used to set new value in a field/key.
- Parameters:

Name	Description
field1	<ul style="list-style-type: none"> • Specify name of the column or field
value1	<ul style="list-style-type: none"> • the new value for the field

EXAMPLE: set design is manager where empno is 1001

```
> db.emp.update({empno:1001},{ $set:design:"manager"}}
```

\$unset:

- **Syntax:** { \$unset: { <field1>: "", ... } }
- In MongoDB, the \$unset operator is used to delete a particular field.
- The \$unset has no effect when the field does not exist in the document.
- Parameters:

Name	Description
field1	<ul style="list-style-type: none"> • Specify name of the column or field

EXAMPLE: delete empno field form all documents

```
> db.emp.update({},{$unset:{empno:""}},{multi:"true"})
```

\$inc:

- **Syntax:** { \$inc: { <field1>: <amount1>, ... } }
- This operator used to increment values of field in MongoDB.
- Parameters:

Name	Description
field1	<ul style="list-style-type: none">• Specify name of the column or field
amount1	<ul style="list-style-type: none">• incremental value

EXAMPLE: increment salary of each employee by 1000 whose designation is clerk

```
> db.emp.update({design:"clerk"},{$inc: {sal:1000}},{multi:true})
```

\$rename:

- **syntax:** {\$rename: { <old_name1>: <new_name1>, <old_name2>: <new_name2>, ... } }
- The \$rename operator is used to update the name of a field.
- It is required to mention the new field name is different from the existing field name.

Name	Description
old_name1,old_name2	old name of the columns or fields
new_name1,new_name2	new name of the column or field

EXAMPLE: rename rln field as rollnumber of all documents

```
>db.stud.update({rlno:{ $gt:0 }},{$rename:{rlno:'roll_number'}},{multi:true})
```

ARRAY UPDATE OPERATOR

- Array update operators are basically used to either add, remove single or multiple values from array. Following are array update operators:
 1. \$addToSet
 2. \$pop
 3. \$pull
 4. \$pullAll
 5. \$push

\$addToSet:

- **Syntax:** `db.collection.update({ <field>: <value> }, { $addToSet: { <field>: <addition> } })`
- This operator used to add element in array

Name	Description
field1	<ul style="list-style-type: none">• Specify name of the column or field
value	<ul style="list-style-type: none">• The value, specified against a field for an expression, or condition, or matching criteria.
addition	<ul style="list-style-type: none">• The value of to be added for a field or column into an array if this value not exists.

EXAMPLE: documents contain 4 subject marks & rln0 5 contain 3 marks want to add marks in array

```
>db.stud.update({rln0:5},{ $addToSet: {marks:55}})
```

\$pop:

- **Syntax:** `db.collection.update({field: value }, { $pop: { field: [1 | -1] } })`
- \$pop operator removes the first or last element of an array.
- -1 is used to remove the **first** element of an array
- 1 is used to remove the **last** element.

EXAMPLE:

```
no:1,  
nm:"xyz",  
marks: [  
    40,50,60,70 ],  
sub: [ "c","cf","cs","net"]
```

Remove last element from marks:

```
>db.stud.update({no:1},{ $pop:{marks:1}})
```

\$pull:

- **syntax :** { \$pull: { <field1>: <value | condition>, <field2>: <value | condition>, ... } }
- The \$pull operator removes from an existing array all instances of a value or values that match a specified condition.
- **parameters:**

Name	Description
field, field1	name of the column or field to the document..
value,value1,value2,...	These are the values to be specified for the fields or columns.

EXAMPLE

```
>db.alpha.update({id:111},{ $pull:{text: { $in:["aa","e","h"]}}})
```

OUTPUT:

```
> db.alpha.find().pretty()
{
  "_id" : ObjectId("5d5a3ba728b2353adc8f2bbc"),
  "id" : 111,
  "text" : [
    "zee",
    "a",
    "b",
    "c",
    "d",
    "bb",
    "cc",
    "g"
  ]
}
```

\$pullAll:

- **syntax:**{ \$pullAll: { field1: [value1, value2, value3] } })
- The \$pullAll operator is used to remove multiple values specified with \$pullAll operator.
- It is similar to \$pull operator
- **Parameters:**

Name	Description
field, field1	name of the column or field to the document..
value,value1,value2,...	These are the values to be specified for the fields or columns.

EXAMPLE:

```
>db.alpha.update({id:111},{ $pullAll:{text: ["bb","g"]}})
```

OUTPUT:

```
> db.alpha.find().pretty()
{
  "_id" : ObjectId("5d5a3ba728b2353adc8f2bbc"),
  "id" : 111,
  "text" : [
    "zee",
    "a",
    "b",
    "c",
    "d",
    "cc"
  ]
}
```

\$push

- This operator used to insert element in array. Generally \$push insert element at the end of an array.
- by using \$push we can
 - push from the start of an array
 - push from the end of an array
 - push multiple elements at once
 - push at any location to an array

push element at the end of an array

EXAMPLE:

```
>db.alpha.insert({id:111,text:["a","b","d","e","g"]})
```

```
>db.alpha.insert({id:111},{ $push:{text:"h"}})
```

OUTPUT:

```
> db.alpha.find().pretty()
{
  "_id" : ObjectId("5d5a3ba728b2353adc8f2bbc"),
  "id" : 111,
  "text" : [
    "a",
    "b",
    "d",
    "e",
    "g",
    "h"
  ]
}
```

push element at specific location of an array

- To push an element to at specific position other than the end of the array, use the \$position modifier along with the \$each modifier.

EXAMPLE:

```
> db.alpha.update({id:111},{ $push:{text: { $each: ["c"],  
$position:2}}})
```

OUTPUT:

```
> db.alpha.find().pretty()  
{  
  "_id" : ObjectId("5d5a3ba728b2353adc8f2bbc"),  
  "id" : 111,  
  "text" : [  
    "a",  
    "b",  
    "c",  
    "d",  
    "e",  
    "g",  
    "h"  
  ]  
}
```

Push element at starting location of an array

- To push an element to a position other than the end of the array, use the \$position modifier along with the \$each modifier.
- in general subscript of an array begin with 0 so we have to specify negative position in 3.6 version

EXAMPLE:

```
> db.alpha.update({id:111},{ $push:{text: { $each: ["zee"], $position:0}}})
```

OUTPUT:

```
db.alpha.find().pretty()  
{  
  "_id" : ObjectId("5d5a3ba728b2353adc8f2bbc"),  
  "id" : 111,  
  "text" : [  
    "zee",  
    "a",  
    "b",  
    "c",  
    "d",  
    "e",  
    "g",  
    "h"  
  ]  
}
```

push multiple elements in an array

EXAMPLE:

```
> db.alpha.update({id:111},{ $push:{text: {$each: ["aa","bb","cc"], $position:5}}})
```

OUTPUT:

```
> db.alpha.find().pretty()
{
  "_id" : ObjectId("5d5a3ba728b2353adc8f2bbc"),
  "id" : 111,
  "text" : [
    "zee",
    "a",
    "b",
    "c",
    "d",
    "aa",
    "bb",
    "cc",
    "e",
    "g",
    "h"
  ]
}
```

8. HOW TO DELETE A DOCUMENT FROM COLLECTION IN MONGODB.

REMOVE DOCUMENTS

Remove ()

- **syntax:** `db.COLLECTION_NAME.remove(DELETION_CRITERIA, justone)`
- Remove () method is used to remove a document from the collection.
- By using this command we can delete a single document or all document stored in collection.
- Remove () method accepts two parameters.
- after performing deletion MongoDB display message along with a number that specify total no of document to be deleted

Deletion _criteria	<ul style="list-style-type: none">• Specify criteria for delete document.
justOne	<ul style="list-style-type: none">• it can Boolean value which is optional• if it is true then remove only one record from document

EXAMPLE:

1. DELETE ALL DOCUMENT FROM STUD COLLECTION

```
> use college  
> db.stud.remove({})
```

2. DELETE DOCUMENT FROM STUD COLLECTION WHO ARE IN MCA.

```
> use college  
> db.stud.remove({stream:"MCA"})
```

3. DELETE ONE DOCUMENT WHO IS IN MSCIT.

```
> use college  
> db.stud.remove({stream:"MSCIT"},{justOne : 1})
```

4. DELETE ALL DOCUMENT WHERE RLNO > 125

```
> use college  
> db.stud.remove({ rlno: { $gt: 125 }})
```

DELETE ONE DOCUMENTS

deleteOne()

- syntax:

db.COLLECTION_NAME.deleteOne(deletion_criteria,writeConcern)

Deletion _criteria	<ul style="list-style-type: none">• Specify criteria for delete document.
writeConcern	<ul style="list-style-type: none">• It an boolean value which is optional.• if it is false will not display default write concern

- MongoDB's **deleteOne()** method is used to remove a single document from the collection.
- by using this command we can delete a single document that match specific criteria
- example

EXAMPLE:

1. DELETE ONE DOCUMENT FROM STUD COLLECTION WHO ARE IN MCA.

```
> use college  
> db.stud.deleteOne({stream:"MCA"})
```

OUTPUT:

```
> db.stud.deleteOne({stream:"msc.it"})
{ "acknowledged" : true, "deletedCount" : 1 }
>
```

DELETE MULTIPLE DOCUMENTS

deleteMany ()

- syntax: `db.collection_name.deletemany(dellection_critteria,writeconcern)`

Deletion _criteria	<ul style="list-style-type: none"> • Specify criteria for delete document.
writeConcern	<ul style="list-style-type: none"> • Its Boolean value which is optional. • if it is false will not display default write concern

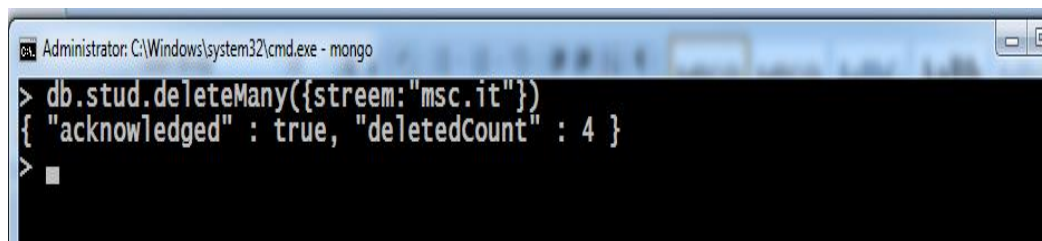
- MongoDB's **deleteMany ()** method is used to remove a more than one document from the collection.
- by using this command we can delete multiple documents that match specific criteria

EXAMPLE:

DELETE DOCUMENTS FROM STUD COLLECTION WHO ARE IN MCA.

```
> use college
> db.stud.deleteMany({stream:"msc.it"})
```

OUTPUT:



```
Administrator: C:\Windows\system32\cmd.exe - mongo
> db.stud.deleteMany({stream:"msc.it"})
{ "acknowledged" : true, "deletedCount" : 4 }
> █
```

9. EXPLAIN MONGODB DATATYPES.

- MongoDB stores data in BSON - Binary-encoded format of JSON.
- This data type is almost similar to java script type.
- MongoDB assigns each data type of an integer ID number from 1 to 255 when querying by type.

MongoDB Data types	ALIAS	Number	Description
Double	"double"	1	<ul style="list-style-type: none"> This type is used to store floating point values.
String	"string"	2	<ul style="list-style-type: none"> This is the most commonly used MongoDB data types, BSON strings are UTF-8
Object	"object"	3	<ul style="list-style-type: none"> Object data type stores embedded documents.
Array	"array"	4	<ul style="list-style-type: none"> It is used to store an array. This data type can store multiples of values and data types.
Binary data	"bindata"	5	<ul style="list-style-type: none"> These MongoDB data types store the binary data in it.
Undefined	"undefines"	6	<ul style="list-style-type: none"> This MongoDB data type stores the undefined values.
Object Id	"objected"	7	<ul style="list-style-type: none"> This data type in MongoDB stores the unique key Id of documents stored. There is an _id field in MongoDB for each document. The data which is stored in Id is in hexadecimal format. The size of ObjectId is 12 bytes
Boolean	"bool"	9	<ul style="list-style-type: none"> This type is used to store a boolean (true/ false) value.
Date	"date"	10	<ul style="list-style-type: none"> Date data type stores current date or time.
Null	"null"	11	<ul style="list-style-type: none"> This MongoDB data types stores a null value in it.
Regular Expression	"regex"	12	<ul style="list-style-type: none"> These MongoDB data types stores regular expressions in MongoDB. It maps directly to

			JavaScript RegExp.
JavaScript	"javascript"	13	<ul style="list-style-type: none"> These MongoDB data types store the JavaScript data without a scope.
Symbol	"symbol"	14	<ul style="list-style-type: none"> These MongoDB data types similar to the string data type.
JavaScript with scope	"javascript WithScope"	15	<ul style="list-style-type: none"> These MongoDB data types store JavaScript data with a scope.
Integer	"int"	16 and 18	<ul style="list-style-type: none"> It is used store numerical value. Integer can be 32 bit or 64 bit depending upon your server.
timestamp	"timestamp"	10	<ul style="list-style-type: none"> This data type is used to store a timestamp. It is useful when we modify our data to keep a record. This is 64-bit value data type.
Min key	"minkey"	255	<ul style="list-style-type: none"> Min key compares the value of the lowest BSON element. It's an internal data types.
Max key	"maxkey"	127	<ul style="list-style-type: none"> Max key compares the value against the highest BSON element. It's an internal data types.

10. EXPLAIN OPERATORS IN MONGODB.

- Operators are symbols that used to perform operations on documents.
- MongoDB provides various types of operators:

RELATIONAL	LOGICAL	ARRAY	ELEMENT QUERY
------------	---------	-------	---------------

COMPARISION OPERATOR	OPERATORS	OPERATORS	OPERATORS
\$eq	\$and	\$size	\$exists
\$gt	\$or	\$all	\$type
\$lt	\$not	\$elemMatch	
\$gte	\$nor		
\$lte			
\$ne			
\$in			
\$nin			

COMPARISION OPERATOR:

- This operators are used to compare the values & display document that match specified criteria

NAME	USAGE	SYNTAX
\$eq	Matches values that are equal to a specified value.	{ key : { \$eq :value } }
\$gt	Matches values that are greater than a specified value.	{ key : { \$gt :value } }
\$lt	Matches values that are less than a specified value.	{ key : { \$lt :value } }
\$gte	Matches values that are greater than or equal to a specified value.	{ key : { \$gte :value } }
\$lte	Matches values that are less than or equal to a specified value.	{ key : { \$lte :value } }
\$ne	Matches values that are not equal to a specified value.	{ key : { \$ne :value } }
\$in	Matches any of the values specified in array	{ key : { \$in :value } }
\$nin	Matches none of the values specified in an array.	{ key : { \$nin :value } }

EXAMPLE:

1. DISPLAY ALL WHERE STREAM = "MSCIT"

```
db.stud.find ( {stream : { $eq : "mscit" } })
```

2. DISPLAY ALL WHOSE PER MORE THAN 70

```
db.stud.find ( {per : { $gt : 70 } })
```

3. DISPLAY ALL WHO GET <=40 PER

```
db.stud.find ( {per : { $lte : 40 } })
```

4. DISPLAY WHERE PER IS 40,60,80,85

```
db.stud.find ( {per : { $in : [40,60,80,85] } })
```

LOGICAL OPERATOR:

- This operators are used to compare the values & display document that match specified criteria

NAME	USAGE	SYNTAX
\$and	returns all documents that match the conditions of both expression becomes true	{ \$and: [{ <exp1> }, { <exp2> }, ... , { <expN> }] }
\$not	Returns documents that do <i>not</i> match the query expression.	{ field: { \$not: { <operator-expression> } } }
\$or	returns all documents that match any one or all the conditions of both expression becomes true	{ \$or: [{ <exp1> }, { <exp2> }, ... , { <expN> }] }
\$nor	returns all documents that fail to match both clauses.	{ \$nor: [{ <exp1> }, { <expr2> }, ... { <expN> }] }

EXAMPLE:

1. DISPLAY ALL WHERE STREAM = "MSCIT" OR STREAM ="MCA"

```
db.stud.find ( { $or: [ { stream: { $eq: "MSCIT" } }, { stream { $eq: "MCA" } } ] })
```

2. DISPLAY ALL WHOSE STREAM IS MSCIT & ROLLNO>2

```
db.stud.find ( { $and: [ { stream: { $eq: "MSCIT" } }, { rln0 { $gt: 2 } } ] })
```

3. DISPLAY ALL WHO ARE NOT IN MCA

```
db.stud.find({stream:{ $not: { $eq:"mca" } } })
```

4. DISPLAY ALL WHERE STREAM = "MSCIT" OR STREAM ="MCA"

```
db.stud.find ( { $nor: [ { stream: { $eq: "MSCIT" } }, { stream { $eq: "MCA" } } ] } )
```

ELEMENT QUERY OPERATOR:

- There are two operators are used as element query operator:
 1. \$exists
 2. \$type

\$exists:

- **Syntax:** { field: { \$exists: <boolean> } }
- This operator matches the documents that contain the field, including documents where the field value is null.
- Pass either true or false as a Boolean value

EXAMPLE:

```
>db.stud.find({rlno :{$exists : true, $in: [2,5]}})
```

- Above example check that whether rollno field is available if available then it will shows document of rollno 2 & 5

\$type:

- **Syntax:** { field: { \$type: <BSON type> } }
- This operator returns the data type of value which is stored inside field.
- As we know than MongoDB stores unstructured data & data type are not predefined.
- Sometime it is necessary to know the type of value then \$type will be used.
- In syntax BSON type can be **double, int, decimal, long, string, array, date, time, null, object** etc.

EXAMPLE:

```
>db.stud.find({rlno :{$type : "string"}})
```

- Above example display all record in which we entered string value in rollno field.

ARRAY QUERY OPERATOR:

- There are two operators are used as element query operator:

1. \$all
2. \$elemMatch
3. \$size

\$all

- **Syntax:** { <field>: { \$all: [<value1> , <value2> ...] } }
- The \$all operator selects the documents where the value of a field is an array that contains all the specified elements.
- The elements might be any order.

EXAMPLE:

```
>use skills
> db.createCollection("sdata")
>db.sdata.insert(
  [{ "Name" : "Balaji", "skills" : [ "Dancing", "Cooking", "Singing" ] },
  { "Name" : "Ramesh", "skills" : [ "Cooking", "Singing" ] },
  { "Name" : "Suresh", "skills" : [ "Dancing", "Singing" ] }])
>db.s.find({skills:{$all:["Cooking","Dancing"]}})
```

- Above example will return only the documents which contains both dancing and cooking skills as a set ie Balaji.

OUTPUT

```
db.s1.find({skills:{$all:["Cooking","Dancing"]}}).pretty()

  "_id" : ObjectId("5d42597b35d0865ac7a64cfe"),
  "Name" : "Balaji",
  "skills" : [
    "Dancing",
    "Cooking",
    "Singing"
  ]
```

\$elemMatch

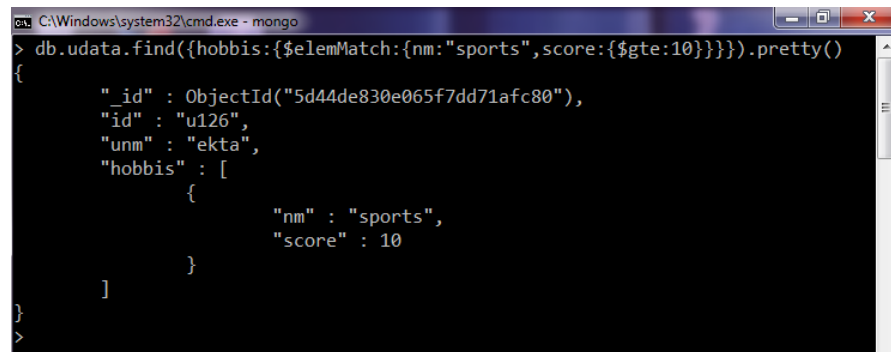
- **Syntax:** {<field>: {\$elemmatch: {<query1>, <query2>, ... } } }
- This operator matches documents that contain an array field with at least one element that matches all the specified query criteria.

EXAMPLE:

```
>use user
```

```
>db.odata.find({hobbis:{$elemMatch:{nm:"sports",score:{$gte:10}}}})
```

OUTPUT:

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe - mongo". The prompt shows a MongoDB query: > db.odata.find({hobbis:{\$elemMatch:{nm:"sports",score:{\$gte:10}}}}).pretty(). The output is a JSON document: { "_id" : ObjectId("5d44de830e065f7dd71afc80"), "id" : "u126", "unm" : "ekta", "hobbis" : [{ "nm" : "sports", "score" : 10 }] }.

```
> db.odata.find({hobbis:{$elemMatch:{nm:"sports",score:{$gte:10}}}}).pretty()
{
  "_id" : ObjectId("5d44de830e065f7dd71afc80"),
  "id" : "u126",
  "unm" : "ekta",
  "hobbis" : [
    {
      "nm" : "sports",
      "score" : 10
    }
  ]
}
```

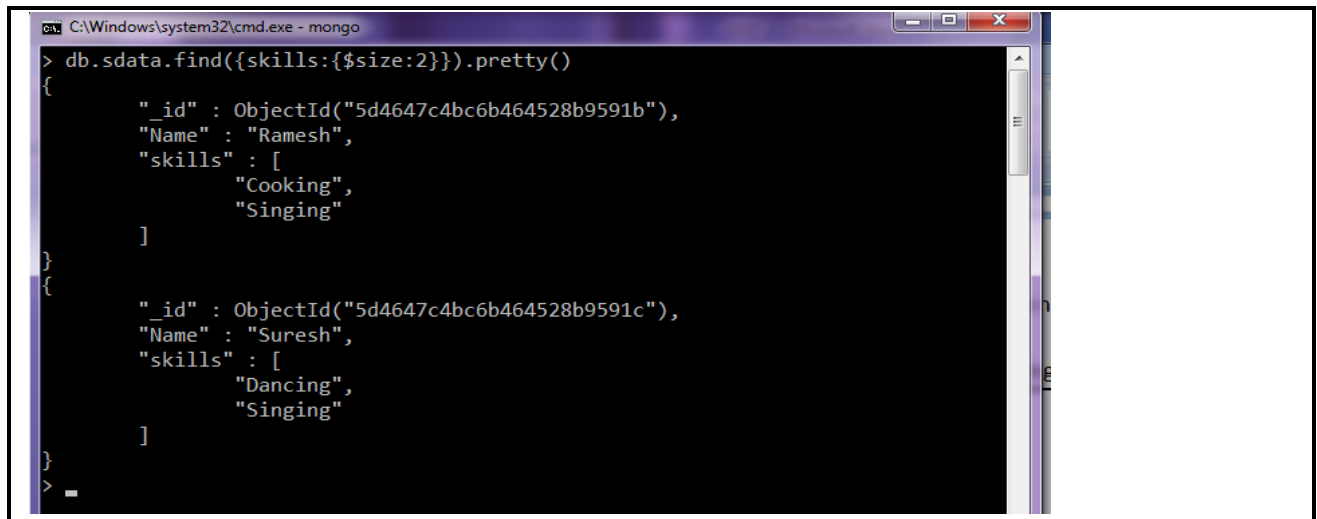
\$size

- **Syntax:** { <field>: { \$size: <elements> } }
- It's an array operator that Counts and returns the total the number of items in an array
- The \$size operator used to display the documents according to elements.

EXAMPLE:

```
>use skills
> db.createCollection("sdata")
>db.sdata.insert(
[ { "Name" : "Balaji", "skills" : [ "Dancing", "Cooking", "Singing" ] },
{ "Name" : "Ramesh", "skills" : [ "Cooking", "Singing" ] },
{ "Name" : "Suresh", "skills" : [ "Dancing", "Singing" ] }])
>db.s.find({skills:{$size:2}})
```

OUTPUT:

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe - mongo". The prompt shows a MongoDB query: `> db.sdata.find({skills:{$size:2}}).pretty()`. The output displays two JSON documents. The first document has an "_id" of "5d4647c4bc6b464528b9591b", a "Name" of "Ramesh", and "skills" of ["Cooking", "Singing"]. The second document has an "_id" of "5d4647c4bc6b464528b9591c", a "Name" of "Suresh", and "skills" of ["Dancing", "Singing"].

```
> db.sdata.find({skills:{$size:2}}).pretty()
{
  "_id" : ObjectId("5d4647c4bc6b464528b9591b"),
  "Name" : "Ramesh",
  "skills" : [
    "Cooking",
    "Singing"
  ]
}
{
  "_id" : ObjectId("5d4647c4bc6b464528b9591c"),
  "Name" : "Suresh",
  "skills" : [
    "Dancing",
    "Singing"
  ]
}
>
```

11. WHAT IS QUERY MODIFICATION? EXPLAIN COMMANDS

- In Mongo DB we got query modifiers such as the 'limit' and 'Orders' clause to provide more flexibility when executing queries.
- This modifier allows users to perform external functionality to retrieve data.
- MongoDB provides three modifiers:
 1. Limit()
 2. Sort()
 3. Skip()

Limit()

- **Syntax:**`db.collection_name.find().limit(number_of_documents)`
- Limit () provides the limitations of documents to be displayed in MongoDB.
- This method specifies that how many number of documents that you want to display in output.
- Using limit() method to limit the documents in the result
- It is used with find () command.
- In syntax
 - **Number of documents** specifies number of document to be display as an argument of limit ().

EXAMPLE:

```
>db.stud.find().limit(3).pretty()
>db.stud.find({rlno: {$gt:3}}).limit(2).pretty()
```

Skip ()

- **Syntax:**`db.collection_name.find().skip(number_of_documents)`
- Skip () methods provides a facility to skip the documents from total documents.
- This method skips the given number of documents in the Query result.
- You can also used it with limit()
 - **Number of documents** specify number of document to be skip

EXAMPLE:

```
>db.stud.find().skip(3).pretty()  
>db.stud.find({rln: {$gt:3}}).limit(5).skip(2).pretty()
```

- In first example skip first three documents form result & in second example display documents where roll number > 3 but first 2 documents skip by MongoDB

Sort ()

- **Syntax:**`db.collection_name.find().sort({key: value : [-1/1]})`
- Sort () methods used to arrange document either in ascending or descending order.
- When we want to display document that arrange in specific order than sort () will be used.
- This method contain 2 parameters
 - 1 → used 1 for ascending order
 - -1 → used -1 for descending order

EXAMPLE:

```
>db.stud.find().sort({name: 1}).pretty ()  
>db.stud.find. sort ({name: -1}).pretty ()
```

- In first example sort documents by name in ascending order while second example arrange document by name in descending order

12. WHAT IS INDEX? HOW TO CREATE INDEX IN MONGODB.

- Indexing is used to increase performance.
- When your database is too large & has lots of documents inside it; it is difficult for MongoDB to search document from it.
- To retrieve data MongoDB examine each document according to search criteria.

- These make speed of execution fall down.
- To improve execution of query index will be used.
- MongoDB provide two commands to create index
 - ensureIndex command
 - createIndex command

CREATE INDEX

ensureIndex():

- **syntax:** `db.collection_name.ensureIndex({key : 1 })`
- This command used to create index in MongoDB.
- Specify a fieldname on which you want to create index as a key.
- You can create index on one or field in MongoDB.
- Use 1 to arrange index field in ascending order & -1 for descending order.

EXAMPLE:

```
>db.m.ensureIndex({no:1})
```

OUTPUT:

```
> db.m.createIndex({no:1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

createIndex():

- **syntax:** `db.collection_name.createIndex({key : 1 })`
- This command used to create index in MongoDB.
- It works similar to ensureIndex command.
- You can create index on one or field in MongoDB.
- Use 1 to arrange index field in ascending order & -1 for descending order.

EXAMPLE:

```
>db.m.createIndex({no:1})
```

```
> db.m.createIndex({no:1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

- Both createIndex & ensureIndex will display same output.

- In output:
- **The numIndexesBefore:** 1 indicates the number of Field values which were there in the indexes before the command was run.
- **The numIndexesAfter:** 2 indicate the number of Field values which were there in the indexes after the command was run.
- **Here the "ok: 1"** output specifies that the operation was successful

SHOW / DISPLAY INDEXES

getIndexes():

- **syntax:** `db.collection_name.getIndexes()`
- This command used to display index generated by users in MongoDB.

EXAMPLE:

```
>db.stud.ensureIndex({per:1})
>db.stud.getIndexes()
```

REMOVE INDEXES

dropIndex():

- **syntax:** `db.collection_name.dropIndex({key:1})`
- This command used to remove index generated by users in MongoDB.

EXAMPLE:

```
>db.stud.dropIndex({per:1})
```

SHOW EXECUTION STATUS

Execute ()

- By using execute () method we can show execution states.
- To show how many documents are examine by MongoDB to find any document use executeStatus()

EXAMPLE:

- **Want to display all documents where per > 50 & see how many documents examine by MongoDB to perform this query.**


```
>db.stud.ensureIndex({per:{$gt:50}}). Explain ("executionStats")
```

13. HOW TO MANAGE RELATIONSHIP BETWEEN DOCUMENTS IN MONGODB.

- Relationship means connect documents logically.
- MongoDB does not provide a relationship concept as we define in RDBMS but it provide different model to establish relationship between documents.
- During designing of data model always consider the application usage of data such as queries, update and processing of data as well as structure of data itself.
- There are two tools in MongoDB that represents the relationship between data.
 1. Embedded Documents
 2. Referenced Documents

EMBEDDED DOCUMENTS:

- It is also known as De-Normalized data.
- This type of data model provides single document structure.
- In this, one collection will be embedded into another collection.
- It means that you can store related information in same document in the form of embedded documents.
- Main advantage of embedded document is that it allows application to retrieve and manipulate related data within a single query.
- So the execution of query becomes fast.
- Embedded documents model contain two types of relationship
 1. one to one relationship
 2. One to many relationships.

One to one relationship

- In this type of relational model one field contain more than one value in a single field.
- For example: There are two documents student & books .one collection will be embedded into another collection.

EXAMPLE:

```
{
  _Id: 123,
  Name: "apexa",
  Class : "mca",
  Book: [
    {
      book_id: 123,
      book_name:"php"
    }
  ]
}
```

One to many relationship

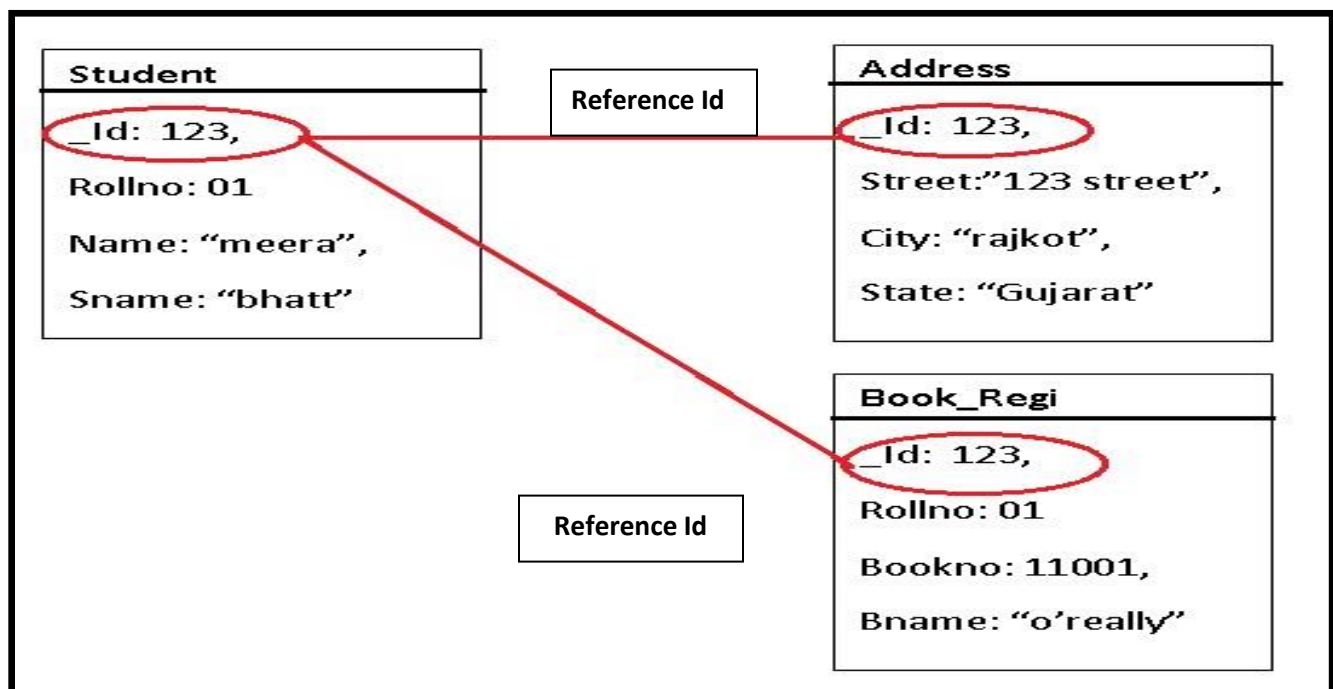
- In this type of relational model embedded documents allows users to store multiple documents within a single field/ key
- In this type of system one single document connected with multiple embedded documents so it is called one to many relationship.

EXAMPLE:

```
{
  _Id: 123,
  Name: "apexa",
  Class : "mca",
  Book: [
    { book_id: 123,
      book_name:"php"
    },
    { book_id: 124,
      book_name:"laravel"
    },
    { book_id: 125,
      book_name:"MongoDB"
    }
  ]
}
```

REFERENCES DOCUMENTS:

- It is also known as Normalized data model.
- In this approach, the documents are maintained separately.
- We can create relationship between data stored in different collections by using reference Id.
- MongoDB or application can use these references Id to retrieve the related data.
- Embedded documents contain duplicate data but references don't contain duplicate data.
- References implement only one relationship: **one To many relationship**



EXAMPLE:

```
{  
    OBJECT_Id: 123,  
    Rollno: 001  
    Name: "Meera",  
    Sname: "bhatt"  
}  
{  
    OBJECT_Id: 123,  
    Street: "123 Street",  
    City: "Rajkot",  
    State: "Gujarat"  
}  
{  
    OBJECT_Id: 123,  
    Rollno: 1,  
    Bookno: 11001,  
    Book name: "o'really"  
}
```

UNIT: 3 SESSION MANAGEMENT

1. EXPLAIN : HTTP SESSION

- Web server have short term memory as soon as they send response to client they forget who the client was.
- In many cases you need to maintain conversational state between client & server .
- So that Http session will be used.
- **http** – hyper text transfer protocol is a stateless protocol.
- HttpSession object can hold conversational state across multiple request from same client.
- HTTP session is a sequence of network request-response transaction.
- HTTP sessions consist of three stages:
 - The client establishes a TCP connection
 - The client sends its request, and waits for the answer.
 - The server processes the request, sending back its answer, providing a status code and appropriate data

Establishing a connection Section

- In client-server protocols, it is the client which establishes the connection.
- Opening a connection in HTTP means initiating a connection in the underlying transport layer TCP.
- With TCP the default port is 80.
- The URL of a page to fetch contains both the domain name, and the port number, though the port 80.

Sending a client request Section

- Once the connection is established, the user-agent can send the request (a user-agent is typically a web browser, but can be anything else, a crawler, for example).
- A client request consists of text directives divided into three blocks:
- The first line contains a request method followed by its parameters:

- the path of the document
- the HTTP protocol version
- The second line represent an HTTP header, giving the server information about what type of data is appropriate (e.g., what language, what MIME types)
- The third block is an optional data block, which may contain further data mainly used by the POST method.

Structure of a server response Section

- After the connected agent has sent its request, the web server processes it, and ultimately returns a response.
- Similar to a client request, a server response is formed of text directives, , though divided into three blocks:
 - The first line, the *status line*, consists of an acknowledgment of the HTTP version used.
 - Subsequent lines represent specific HTTP headers, giving the client information about the data sent (e.g. type, data size, compression algorithm used)
 - The final block is a data block, which contains the optional data

Response status codes Section

- HTTP response status codes indicate if a specific HTTP request has been successfully completed.
- Responses are grouped into five classes: informational responses, successful responses, redirects, client errors, and server errors.
 - 200: OK. The request has succeeded.
 - 301: Moved Permanently. This response code means that the URI of requested resource has been changed.
 - 404: Not Found. The server cannot find the requested resource.

2. EXPLAIN SESSION IN PHP

- In PHP session handler a special class is used for session handling.
- There are 7 method which are used for internal session handling like open, close, read, write, destroyed, GC and create_sid.
- Session.save-handler class will wrap the internal save handler and configure the directive which files by default.

- When a plain instance(object) of session handler is set as a save handler using `session_set_save_handler()` then it will wrap the current save handlers.

Methods:- 1.

1. Session handler `_createsession_sid`:- it return a new session Id
2. Session handler `_open()`:- it initialize a session .
3. Session handler `_close()`:-it close the session.
4. Session handler `_read()`:- it read the session data.
5. Session handler `_write()`:- it write the session data.
6. Session handler `_gc()`:- it clean up old session.
7. Session handler `_destroyed`:- it destroyed a session.

3. EXPLAIN: PHP NATIVE SESSION HANDLING:

- Php handle on session natively hear it configure the session on server (php.ini file) and not in the application.
- So if we have several website on one server then we only need to configure the server for the changes to be applied on all the websites.
- There is different libraries can be use for native session handling, hear you can also create a new library that starts a native php session and allows to store and retrieve different things from it.
- With the release of PHP4, session management was introduced as an extension to the PHP language.
- PHP provides several session-related functions, and developing applications that use PHP sessions is straightforward.
- The three important features of session management are mostly taken care of by the PHP scripting engine. How to use PHP sessions?, How sessions are started and ended ? and How session variables are used?.
- We list the PHP functions for building session-based web applications.
- Because not all browsers support cookies, and some users actively disable them.

- When a user first enters the session-based application by making a request to a page that starts a session, PHP generates a session ID and creates a file that stores the session-related variables.
- PHP sets a cookie to hold the session ID in the response the script generates.
- The browser then records the cookie and includes it in subsequent requests.
- Between the browser and the server when initial requests are made to a session-based application The out-of-the-box configuration of PHP session management uses disk-based files to store session variables.
 - Using files as the session store is enough for most applications in which the numbers of concurrent sessions are limited.
 - Starting a Session PHP provides a `session_start()` function that creates a new session and subsequently identifies and establishes an existing one.
 - The first time a PHP script calls `session_start()`, a session identifier is generated, and, by default, a Set-Cookie header field is included in the response.
 - The response sets up a session cookie in the browser with the name `PHPSESSID` and the value of the session identifier.
 - The PHP session management automatically includes the cookie without the need to call to the `setcookie()` or `header()` functions.
 - The session identifier (ID) is a random string of 32 hexadecimal digits, such as `fcc17f071bca9bf7f85ca281094390b4`.
 - As with other cookies, the value of the session ID is made available to PHP scripts in the `$HTTP_COOKIE_VARS` associative array and in the `$PHPSESSID` variable.
 - When a new session is started, PHP creates a session file. With the default configuration, session files are written in the `/tmp` directory using the session identifier, prefixed with `sess_`, for the filename.
 - However, if the identified session file can't be found, `session_start()` creates an empty session file.
 - Session Variables need to be registered with the `session_register()` function that's used in a session.
 - If a session has not been initialized, the `session_register()` function calls `session_start()` to open the session file.
 - Once registered, session variables are made persistent and are available to scripts that initialize the session.

- PHP tracks the values of session variables and saves their values to the session file: there is no need to explicitly save a session variable before a script ends.

4. EXPLAIN: SESSION MANAGER.

- Session Manager is a simple powerful extension that makes it quick and easy to save, update, remove, and restore sets of tabs in any browser.
- It create sessions for daily routines for time saving and easiness of work.
- In PHP session management will be performed by different functions.
- A session is a way to store information (in variables) to be used across multiple pages.
- Unlike a cookie, the information is not stored on the user's computer. Session variables remains until the user closes the browser.
- So; Session variables hold information about one single user, and are available to all pages in one application.
- If you need a permanent storage, you may want to store the data in a database.
- A session is started with the `session_start()` function.
- Session variables are set with the PHP global variable: `$_SESSION`. To remove all global session variables and destroy the session, use `session_unset()` and `session_destroy()` session handling mechanism of PHP to use a MongoDB database for managing sessions instead of using the filesystem.
- Before we proceed into implementation, we are going to briefly cover the basics, mainly the `session_set_save_handler()` function.
- The `session_set_save_handler()` function allows us to define our own functions for storing and retrieving session data.
- The function takes six arguments, each one being the name of a callback function.
- This is what the method signature looks like: `bool session_set_save_handler(callback $open, callback $close, callback $read, callback $write, callback $destroy, callback $gc)`
- Session Manager is a simple yet powerful extension that makes it quick and easy to save, update, remove, and restore sets of tabs.

- Common uses include: - Creating sessions for daily routines: pages you open in the morning, afternoon and evening.
- Amazon EC2(Amazon Elastic Compute Cloud) Simple Systems Manager (SSM) is an Amazon Web Services tool that allows an IT professional to automatically configure virtual servers in a cloud or in on-premises data center.
- AWS Systems Manager Parameter Store consists of standard and advanced parameters. Standard parameters are available at no additional charge.
- **IAM Role:-**
 - IAM authorizes to start a session for an EC2 instance (IAM policy).
 - The administrator uses the AWS Management Console or the terminal (AWS CLI and additional plugin required) to start a session via the Systems Manager.
 - The Session Manager sends audit logs to CloudWatch Logs or S3.
- **SSM Role:-**
 - AWS Systems Manager Session Manager.
 - Session Manager is a fully managed AWS Systems Manager capability that lets you manage your Amazon EC2 instances, on-premises instances, and virtual machines (VMs) through an interactive one-click browser-based shell or through the AWS CLI.
- **AWS Systems Manager Agent (SSM Agent):**
 - It is Amazon software that can be installed and configured on an EC2 instance, an on-premises server, or a virtual machine (VM).
 - SSM Agent makes it possible for Systems Manager to update, manage, and configure these resources

UNIT: 4 AGGREGATION QUERIES

14. EXPLAIN COUNT IN MONGODB?

- Count is method that used to count elements.
- In MongoDB there is various use of count.
- It can work with any other methods or individual also; By using count
 1. We can count total number or documents.
 2. We can count documents that match specific query criteria
 3. We can count documents than match multiple query criteria
 4. We can use count with find() to count documents that match query criteria
 5. We can count the document that you skip or limit during the condition
 6. We can used is as aggregation command

COUNT TOTAL NUMBER OF DOCUMENTS:

db.collection.count()

- **Syntax: db.collection_name.count()**
- The db.collection.count() method is used to return the count of documents.

EXAMPLE:

```
> db.stud.count()
```

COUNT TOTAL NUMBER OF DOCUMENTS THAT MATCH QUERY:

- **Syntax: db.collection_name.count(<query>)**
- The db.collection.count() method can also used to count records that match specified criteria.
- We need not to use find () to count some specific documents that match query.

EXAMPLE:

```
> db.stud.count({city:"Rajkot"})
```

COUNT WITH FIND() WITH SINGLE QUERY CONDITION

- **Syntax:** `db.collection_name.find(<query>).count()`
- The `db.collection.find().count()` method can also be used to count documents that match specified criteria.
- In syntax `<query>` is used to specify the selection criteria for counting documents.
- We can also produce the same output with `db.collection.count(<query>)`

EXAMPLE:

```
> db.stud.find({stream:"mba"}).count()
```

COUNT WITH FIND() WITH MULTIPLE QUERY CONDITION

- **Syntax:** `db.collection_name.find(LOP{<query1>,<query2>}).count()`
- The `db.collection.find().count()` method can also be used to count records that match specified criteria.
- When we used to count documents that match more than one condition then logical operators `$and`, `$or`, `$not` and `$nor` will be used.
- Count allows to count documents that contain more than one conditions/queries

EXAMPLE:

```
> db.stud.find({$and: [{stream:"mba"},{rlnr:{ $gt:2} }} ).count()
```

COUNT WITH SKIP() or LIMIT()

- `Skip ()` is used to skip number of documents & return remaining documents while `limit ()` displays specific number of documents.
- We need to specify how many documents are skipped in output by MongoDB as an **argument** of `skip()` and `limit()`
- By default `count()` ignores `skip()` & `limit ()` so when we use count with `skip()` or `limit()` it will ignore both methods
- To get number of documents after `skip()` we have to specify **'true'** parameter as an argument of `count()`

EXAMPLE:

```
> db.stud.find({stream:"mba"}).limit(3).count(true)
```

COUNT WITH AGGREGATION:

- We can use count with aggregation queries also.
- **Syntax:** `db.collection_name.aggregate({$group: {<key>, count:(criteria)}})`
- When we want to count documents of group of field.

EXAMPLE:

COUNT ALL DOCUMENT

```
> db.stud.aggregate([]).count()
```

```
> db.stud.aggregate([{$group:{_id:"$class",count:{$sum:1}}}]
```

OUTPUT:

```
> db.stud.aggregate([{$group:{_id:"$class",count:{$sum:1}}}]
{ "_id" : "mba", "count" : 3 }
{ "_id" : "mscit", "count" : 1 }
>
```

- We can also use javascript length method to count documents without using count()

EXAMPLE:

```
> db.stud.aggregate([]).toArray().length
```

15. EXPLAIN DISTINCT METHOD OF MONGODB.

- Distinct method is used to retrieve unique values from documents.
- This method is similar to distinct parameter of SQL that used with SELECT clause to retrieve unique records.
- To retrieve non repeated values form document we can use distinct method.
- We can also use distinct method to retrieve unique values from embedded documents also
- **Syntax:** `db.collection_name.distinct({field , query})`
- In syntax

Name	Description
field	<ul style="list-style-type: none">• Specify field name.
query	<ul style="list-style-type: none">• Specify criteria or condition for retrieve the unique values.

EXAMPLE:**1. RETRIEVE UNIQUE VALUES FROM CLASS FIELD**

```
> db.stud.distinct("class")
```

2. RETRIEVE UNIQUE VALUES FROM CLASS FIELD WHOSE ROLL NUMBER IS > 10

```
> db.stud.distinct("class", {rIno: {$gt:10}})
```

3. RETRIEVE UNIQUE VALUES FROM CLASS FIELD & ARRANGE IN ASCENDING ORDER

```
> db.stud.distinct("class").sort()
```

4. RETRIEVE UNIQUE VALUES FROM CLASS FIELD & ARRANGE IN DESCENDING ORDER

```
> db.stud.distinct("class").sort({class:-1})
```

5. RETRIEVE UNIQUE VALUES FROM EMBEDDED DOCUMENT

```
> db.stud.distinct("address.pincode")
```

16. WHAT AGGREGATION IN MONGODB?

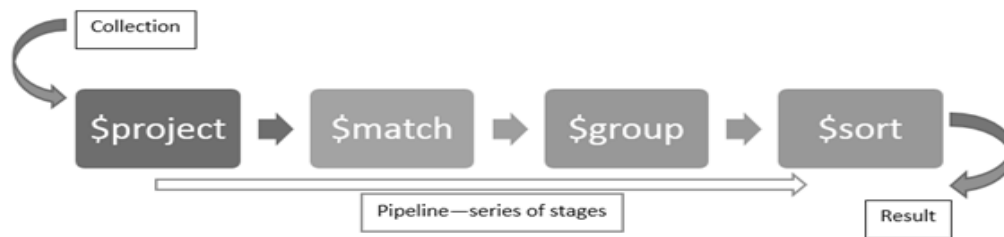
- Aggregation basically groups the data from multiple documents and returns one combined result.
- When we want to perform operation in group of field then aggregation will be use.
- It is similar to **group by clause** of SQL.
- Aggregation can be performed on single or more group of fields.
- MongoDB provides **Aggregate ()** to perform such operations which is aggregation pipelines.

Syntax: db.collection_name.aggregate(Aggregate_operation)

AGGREGATION PIPELINES:

- MongoDB's aggregation framework is based on the concept of data processing in a sequence.
- Aggregation pipeline is similar to the UNIX word pipelines.
- At the very first is the collection, the collection is sent through document by document.
- Then documents are piped through processing pipeline and they go through series of stages and then we get a result set.

STAGES OF AGGREGATION PIPELINE



- **Various stages in pipeline are:**
 1. \$project – select, reshape data
 2. \$match – filter data
 3. \$group – aggregate data
 4. \$sort – sorts data
 5. \$skip – skips data
 6. \$limit – limit data
 7. \$unwind – normalizes data
- Each stage always begin with \$
- Here is a sample document in which we are going to perform all operations

```
{
  "_id" : ObjectId("5d57f9e41a208742aa2943ad"),
  "empno" : 1002,
  "ename" : "riya",
  "gender" : "f",
  "qua" : "mca",
  "exp" : "1 yr",
  "deptno" : 20,
  "desig" : "operator",
  "sal" : 17000,
  "skill" : [
    "dancing",
    "sports",
    "travel"
  ]
}
```

\$project → first stage

- In aggregation pipeline the \$project stage is the first stage.
- This stage used to
 1. Include a key/field in output
 2. Exclude some key/field from document in output
 3. Insert computed fields/keys
 4. Rename key/fields in output that is called reshape
- There are also some simple functions that we can use on the key like \$toUpper, \$toLower, \$add, \$multiply etc.

INCLUDE/ EXCLUDE KEY IN OUTPUT:

- To include a key/Field from document we just specify key name with 1, & to exclude key specify 0 with key name,
- Remember that you can not use 0 & 1 both in same projection except `_id`

EXAMPLE:

1. Display `ename,deptno,sal` document but remove object `id`

```
>db.emp.aggregate([{$project: { _id:0, ename:1, deptno: 1, sal:1}}])
```

2. Display all fields from documents except `skills & exp`

```
>db.emp.aggregate([{$project: { exp:0, skills: 0}}])
```

- We can also get same output with `find()` by specifying fields/ keys

\$match → second stage

- It works exactly like **where clause** in SQL to filter out the records.
- When we want to match specific document in query then we can use `$match`.
- It is similar to the `find()` with criteria
- When we want to filter the results and only search for particular parts of the results set after we do the grouping.

EXAMPLE:

Display all documents where designation is operator

```
>db.emp.aggregate([{$project:{_id:0}},{$match:{design:"operator"}}])
```

\$group → third stage

- It is a third stage of aggregation pipeline.
- It is similar to group by option of SQL that we used to produce aggregate result.
- When we want to perform operation on group of field then group by will be used.
- To perform aggregation first group on field by using `$group`

Syntax:

```
db.collection_name.aggregate( {$group: {_id: <expression>, <field>:  
{accumulator : <expression>}}})
```

- In syntax accumulators specify functions that you want to produced aggregate result

EXAMPLE:

Display department wise highest salary from documents

```
>db.emp.aggregate([  
  {$project: {_id:0}},  
  {$group: {_id:"$deptno","maxsal": {$max: "$sal"}}}  
])
```

\$sort → fourth stage

- In aggregation pipeline the \$sort stage is the fourth stage.
- This stage is basically used to arrange data in either ascending order or descending order.
- To arrange output in ascending order specify 1 & use -1 to arrange output in descending order.

EXAMPLE:

Display department wise highest salary from documents & arrange in ascending order of department

```
>db.emp.aggregate([  
  {$project: {_id:0}},  
  {$group: {_id:"$deptno","maxsal":{$max: "sal"}}},  
  {$sort:{_id:1}}  
])
```

\$skip → fifth stage

- This stage is basically used to skip number of documents in output.
- \$skip shows specified number of documents by skipping documents

EXAMPLE:

Display department wise highest salary from documents & display only 3 documents & skip first 2 documents

```
>db.emp.aggregate([
  {$project: {_id:0}},
  {$group: {_id:"$deptno","maxsal":{$max: "sal"}}},
  {$sort:{_id:1}},
  {$skip:2}
])
```

\$limit → sixth stage

- This stage is basically used to either limit number of documents in output.
- \$limit shows specified number of documents

EXAMPLE:

Display department wise highest salary from documents & display only 3 documents

```
>db.emp.aggregate([
  {$project: {_id:0}},
  {$group: {_id:"$deptno","maxsal":{$max: "sal"}}},
  {$sort:{_id:1}},
  {$limit:3}
])
```

\$unwind → last stage

- This stage is basically used with \$project stage.
- This stage used to split down all array elements & show each value of document as a separate document.

EXAMPLE:

Unwind skill filed in several document

```
>db.emp.aggregate([
```

```
{ $project: { _id: 0, ename: 1, sal: 1, skill: 1 } },  
{ $unwind: "$skill" }  
})
```

OUTPUT:

```
"ename" : "riya", "sal" : 17000, "skill" : "dancing" }  
"ename" : "riya", "sal" : 17000, "skill" : "sports" }  
"ename" : "riya", "sal" : 17000, "skill" : "travel" }  
"ename" : "shiv", "sal" : 20000, "skill" : "dancing" }  
"ename" : "shiv", "sal" : 20000, "skill" : "reading" }  
"ename" : "riya", "sal" : 23000, "skill" : "chatting" }  
"ename" : "riya", "sal" : 23000, "skill" : "singing" }  
"ename" : "shivangi", "sal" : 50000, "skill" : "chatting" }  
"ename" : "shivangi", "sal" : 50000, "skill" : "singing" }  
"ename" : "priya", "sal" : 33000, "skill" : "chatting" }  
"ename" : "priya", "sal" : 33000, "skill" : "singing" }
```

17. EXPLAIN \$GROUP OR GROUPING IN MONGODB.

- Grouping is method that is used to find out aggregate result.
- When we want to perform operation on group of field then \$group will be used.
- It is similar to group by option of SQL.
- It is used to produced aggregate result based on \$group key

Syntax:

```
db.collection_name.aggregate( { $group: { _id: <expression>, <field>:  
{ accumulator : <expression> } } })
```

- In syntax accumulators specify functions that you want to produced aggregate result
- Here is a list of accumulators that we can used with \$group.

accumulator	Description
\$sum	Sum or count total number of documents in aggregation
\$avg	It calculates the average values from all the documents in the collection.
\$min	It is used to find out minimum values from all documents

\$max	It is used to find out maximum values from all documents
\$addToSet	It is used to Insert values to an array but no duplicates in the resulting document.
\$push	Insert values to an array in the resulting document.
\$first	Return the first document from the source document that match group field
\$last	Return the last document from the source document that match group field.

\$sum

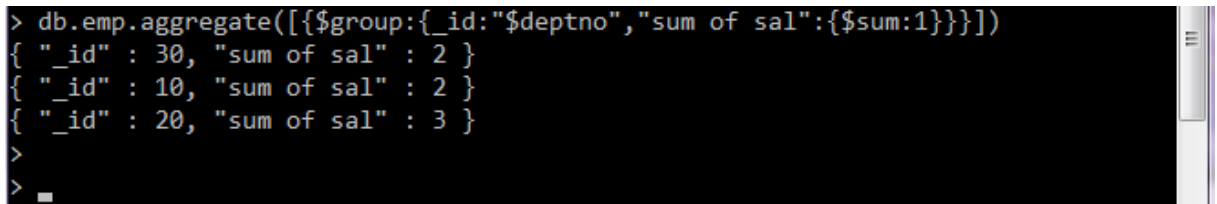
- Sum or count total number of documents in aggregation

EXAMPLE:

Display department wise salary (total number of salary) from documents

```
>db.emp.aggregate([
  {$group: {_id:"$deptno","sum of sal": {$sum: 1}}}
])
```

OUTPUT:

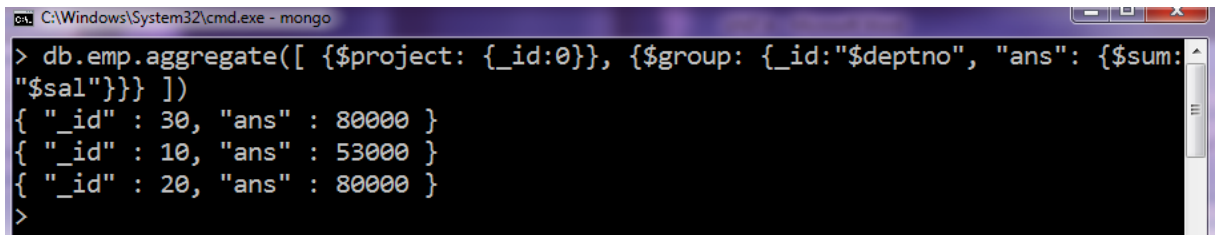


```
> db.emp.aggregate([{$group:{_id:"$deptno","sum of sal":{$sum:1}}}))
{ "_id" : 30, "sum of sal" : 2 }
{ "_id" : 10, "sum of sal" : 2 }
{ "_id" : 20, "sum of sal" : 3 }
>
>
```

Display department wise salary (sum of salary) from documents

```
>db.emp.aggregate([
  {$group: {_id:"$deptno","sum of sal": {$sum:"$sal"}}}
])
```

OUTPUT:



```
C:\Windows\System32\cmd.exe - mongo
> db.emp.aggregate([ {$project: {_id:0}}, {$group: {_id:"$deptno", "ans": {$sum:"$sal"}}} ])
{ "_id" : 30, "ans" : 80000 }
{ "_id" : 10, "ans" : 53000 }
{ "_id" : 20, "ans" : 80000 }
>
```

\$avg

- Used to calculate average

EXAMPLE:

Display department wise average of salary from documents

```
>db.emp.aggregate([
  {$group: {_id:"$deptno","average sal": {$avg:"$sal"}}}
])
```

OUTPUT:

```
> db.emp.aggregate([{$group:{_id:"$deptno","average sal": {$avg:"$sal"}}}])
{ "_id" : 30, "average sal" : 40000 }
{ "_id" : 10, "average sal" : 26500 }
{ "_id" : 20, "average sal" : 26666.666666666668 }
```

\$max

- Used to find out maximum value

EXAMPLE:

Display department wise maximum salary from documents

```
>db.emp.aggregate([
  {$group: {_id:"$deptno","departmentwise max sal": {$max:"$sal"}}}]])
```

OUTPUT:

```
> db.emp.aggregate([{$group:{_id:"$deptno","maximum department wise sal": {$max:"$sal"}}}])
{ "_id" : 30, "maximum department wise sal" : 50000 }
{ "_id" : 10, "maximum department wise sal" : 33000 }
{ "_id" : 20, "maximum department wise sal" : 40000 }
```

\$min

- Used to find out minimum value

EXAMPLE:

Display department wise maximum salary from documents

```
>db.emp.aggregate([
  {$group: {_id:"$deptno","departmentwise max sal": {$min:"$sal"}}}
])
```

OUTPUT:

```
{ "_id" : 30, "maximum department wise sal" : 30000 }
{ "_id" : 10, "maximum department wise sal" : 20000 }
{ "_id" : 20, "maximum department wise sal" : 17000 }
>
```

\$first

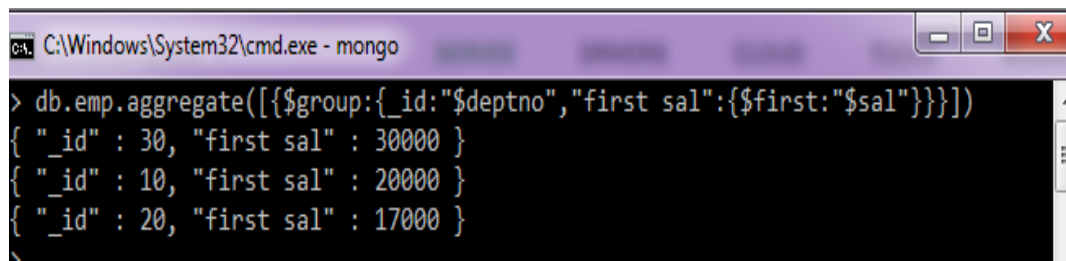
- Return the first document from the source document that match group field

EXAMPLE:

Display first document

```
>db.emp.aggregate([
  {$group: {_id:"$deptno","first sal": {$first:"$sal"}}}
])
```

OUTPUT:



```
C:\Windows\System32\cmd.exe - mongo
> db.emp.aggregate([{$group:{_id:"$deptno","first sal":{$first:"$sal"}}}])
{ "_id" : 30, "first sal" : 30000 }
{ "_id" : 10, "first sal" : 20000 }
{ "_id" : 20, "first sal" : 17000 }
```

\$last

- Return the last document from the source document that match group field

EXAMPLE:

Display first document

```
>db.emp.aggregate([
  {$group: {_id:"$deptno","first sal": {$last:"$sal"}}}
])
```

OUTPUT:

```
> db.emp.aggregate([{$group:{_id:"$deptno","first sal":{$last:"$sal"}}}])
{ "_id" : 30, "first sal" : 50000 }
{ "_id" : 10, "first sal" : 33000 }
{ "_id" : 20, "first sal" : 23000 }
>
```

18. HOW TO PERFORM JOIN OPERATION IN MONGODB?

- MongoDB allows you displaying documents that are stored in different collections by matching common field.
- It work similar to join operation that we performed in SQL.
- MongoDB provides an aggregation method called **\$lookup**.

\$lookup:

- **\$lookup** allows you to perform joins on collections in the same database.
- It works same as we perform left join in SQL.

Syntax:

```
{ $lookup:
  {
    from: "collection2",
    localField: "common field of parent table",
    foreignField: "common field of child table",
    as: "alias name for collection2"
  }
}
```

- In syntax:
 1. **From:** specify the name of 2nd collection from which you want to retrieve documents.
 2. **localField:** specify the common field (primary key field) name of collection1
 3. **foreignField:** specify the common field (foreign key field) name of collection 2
 4. **as:** specify alias name which is added in output to store document
- for example
 1. Here we have 2 collections product & order.

```
C:\Windows\System32\cmd.exe - mongo
> db.product.find().pretty()
{
  "_id" : ObjectId("5d6690a2a11077ccbd0e0206"),
  "proid" : 1,
  "pronm" : "pen",
  "qty" : 500,
  "price" : 15
}
{
  "_id" : ObjectId("5d6690a2a11077ccbd0e0207"),
  "proid" : 2,
  "pronm" : "pen",
  "qty" : 1000,
  "price" : 25
}
{
  "_id" : ObjectId("5d6690a2a11077ccbd0e0208"),
  "proid" : 3,
  "pronm" : "book",
  "qty" : 1500,
  "price" : 45
}

C:\Windows\System32\cmd.exe - mongo
> db.orders.find().pretty()
{
  "_id" : ObjectId("5d66911ba11077ccbd0e0209"),
  "proid" : 2,
  "qty" : 100,
  "comp_nm" : "abc"
}
{
  "_id" : ObjectId("5d66911ba11077ccbd0e020a"),
  "proid" : 3,
  "qty" : 450,
  "comp_nm" : "zee"
}
{
  "_id" : ObjectId("5d66911ba11077ccbd0e020b"),
  "proid" : 2,
  "qty" : 150,
  "comp_nm" : "xxx"
}
```

Example: 1 ➔ If we want to check the product with order detail

```
> db.product.aggregate([
{ $lookup:
  {
    from:"orders",
    localField:"proid",
    foreignField:"proid",
    as:"order_detail"
  }
}]).pretty()
```

OUTPUT:


```

{
  "_id" : ObjectId("5d6690a2a11077ccbd0e0206"),
  "proid" : 1,
  "pronm" : "pen",
  "qty" : 500,
  "price" : 15,
  "order_detail" : [ ]
}
{
  "_id" : ObjectId("5d6690a2a11077ccbd0e0207"),
  "proid" : 2,
  "pronm" : "pen",
  "qty" : 1000,
  "price" : 25,
  "order_detail" : [
    {
      "_id" : ObjectId("5d66911ba11077ccbd0e0209"),
      "proid" : 2,
      "qty" : 100,
      "comp_nm" : "abc"
    },
    {
      "_id" : ObjectId("5d66911ba11077ccbd0e020b"),
      "proid" : 2,
      "qty" : 150,
      "comp_nm" : "xxx"
    }
  ]
}
{
  "_id" : ObjectId("5d6690a2a11077ccbd0e0208"),
  "proid" : 3,
  "pronm" : "book",
  "qty" : 1500,
  "price" : 45,
  "order_detail" : [
    {
      "_id" : ObjectId("5d66911ba11077ccbd0e020a"),
      "proid" : 3,
      "qty" : 450,

```

Example: 2 → Display all orders of pen only.

```
> db.product.aggregate([{$match: {proid:2}},
```

```
{ $lookup:
```

```

{
  from:"orders",
  localField:"proid",
  foreignField:"proid",
  as:"order_detail"
}

```

```
]]) .pretty()
```

OUTPUT:

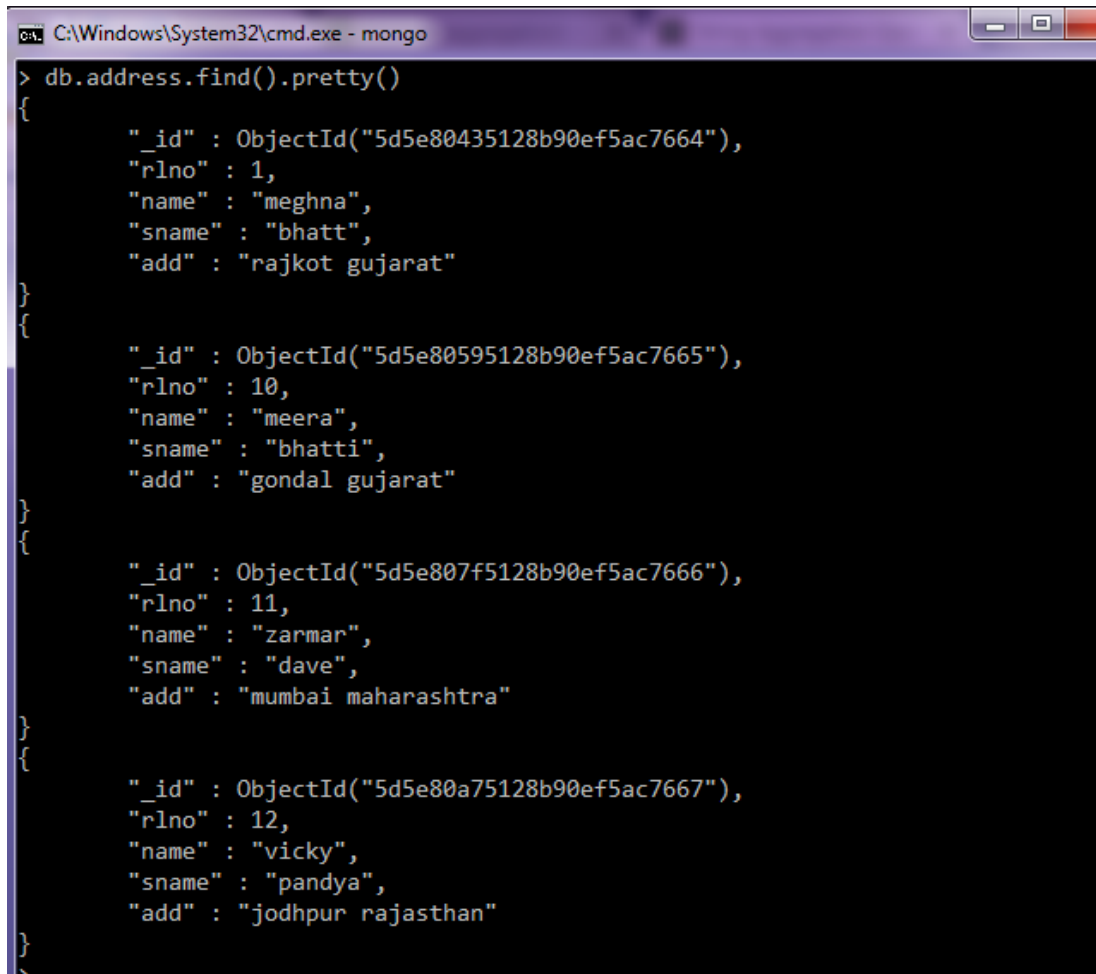
```

{
  "_id" : ObjectId("5d6690a2a11077ccbd0e0207"),
  "proid" : 2,
  "pronm" : "pen",
  "qty" : 1000,
  "price" : 25,
  "order_detail" : [
    {
      "_id" : ObjectId("5d66911ba11077ccbd0e0209"),
      "proid" : 2,
      "qty" : 100,
      "comp_nm" : "abc"
    },
    {
      "_id" : ObjectId("5d66911ba11077ccbd0e020b"),
      "proid" : 2,
      "qty" : 150,
      "comp_nm" : "xxx"
    }
  ]
}

```

19. EXPLAIN STRING AGGREGATION OPERATOR OF MONGODB?

- String expression are use to perform operation on textual data stored in field.
- We can use string expression with \$project stage of aggregation pipeline.
- Some string aggregation operators are:
 1. \$concat
 2. \$substr
 3. \$toUpper
 4. \$toLower
 5. \$strLenBytes
 6. \$strcasecmp
 7. \$split
- Here is a sample document of address.



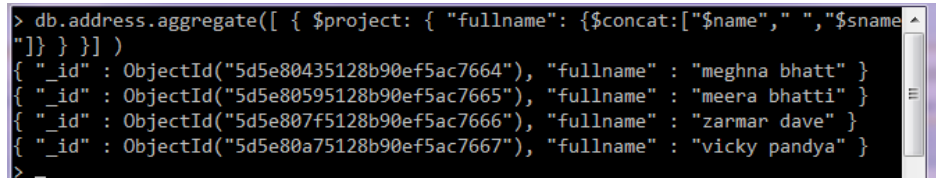
```
ca. C:\Windows\System32\cmd.exe - mongo
> db.address.find().pretty()
{
  "_id" : ObjectId("5d5e80435128b90ef5ac7664"),
  "rln" : 1,
  "name" : "meghna",
  "sname" : "bhatt",
  "add" : "rajkot gujarat"
}
{
  "_id" : ObjectId("5d5e80595128b90ef5ac7665"),
  "rln" : 10,
  "name" : "meera",
  "sname" : "bhatti",
  "add" : "gondal gujarat"
}
{
  "_id" : ObjectId("5d5e807f5128b90ef5ac7666"),
  "rln" : 11,
  "name" : "zarman",
  "sname" : "dave",
  "add" : "mumbai maharashtra"
}
{
  "_id" : ObjectId("5d5e80a75128b90ef5ac7667"),
  "rln" : 12,
  "name" : "vicky",
  "sname" : "pandya",
  "add" : "jodhpur rajasthan"
}
>
```

\$concat

- This operator is used to merge two or more string
- **Syntax:** { \$concat: [<expression1>, <expression2>, ...] }

EXAMPLE:

```
>db.address.aggregate([ {$project: { "full name": {$concat : ["$name"," " ,"$surname" ] }}}])
```



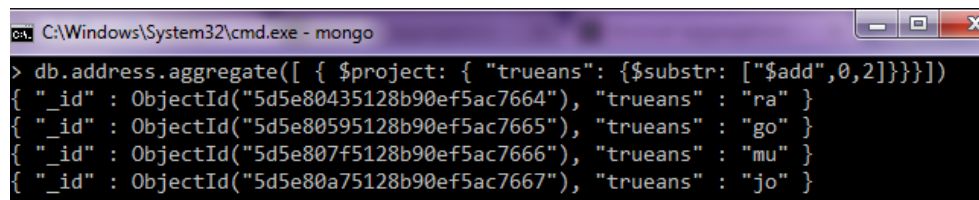
```
> db.address.aggregate([ { $project: { "fullname": {$concat:["$name"," ", "$surname" ] } } } ] )
{ "_id" : ObjectId("5d5e80435128b90ef5ac7664"), "fullname" : "meghna bhatt" }
{ "_id" : ObjectId("5d5e80595128b90ef5ac7665"), "fullname" : "meera bhatti" }
{ "_id" : ObjectId("5d5e807f5128b90ef5ac7666"), "fullname" : "zarmar dave" }
{ "_id" : ObjectId("5d5e80a75128b90ef5ac7667"), "fullname" : "vicky pandya" }
```

\$substr

- This operator is used to retrieve string from existing string.
- When we want to retrieve some specific part of string then \$substr will be used.
- **Syntax:** { \$substr: ["expression", <start>, <end>] }

EXAMPLE:

```
>db.address.aggregate([ {$project: { "substring": {$substr : ["$add",0,2] }}}])
```



```
ca. C:\Windows\System32\cmd.exe - mongo
> db.address.aggregate([ { $project: { "trueans": {$substr: ["$add",0,2]}}}])
{ "_id" : ObjectId("5d5e80435128b90ef5ac7664"), "trueans" : "ra" }
{ "_id" : ObjectId("5d5e80595128b90ef5ac7665"), "trueans" : "go" }
{ "_id" : ObjectId("5d5e807f5128b90ef5ac7666"), "trueans" : "mu" }
{ "_id" : ObjectId("5d5e80a75128b90ef5ac7667"), "trueans" : "jo" }
```

\$toUpper & \$toLower

- \$toUpper operator used to convert fields values in uppercase & \$lower is used to convert fields value in lower case
- **Syntax:** { \$toUpper | \$toLower: <"expression"> }

EXAMPLE:

```
>db.address.aggregate([ {$project: { "name":{$toUpper: "$name"},
"surname": {$toLower:"$surname" }}}])
```

OUTPUT:

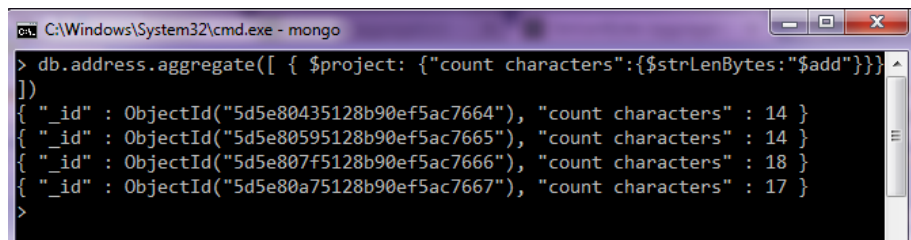
```
{ "_id" : ObjectId("5d5e80435128b90ef5ac7664"), "name" : "MEGHNA", "sname" : "bhatt" }
{ "_id" : ObjectId("5d5e80595128b90ef5ac7665"), "name" : "MEERA", "sname" : "bhatti" }
{ "_id" : ObjectId("5d5e807f5128b90ef5ac7666"), "name" : "ZARMAR", "sname" : "dave" }
{ "_id" : ObjectId("5d5e80a75128b90ef5ac7667"), "name" : "VICKY", "sname" : "pandya" }
>
```

\$strLenBytes

- This operator is used to find out length of string in bytes
- **Syntax:** { \$strLenBytes: <expression> }

EXAMPLE:

```
>db.address.aggregate([ {$project: { "count characters": {$strLenBytes :
"$add"}}}])
```



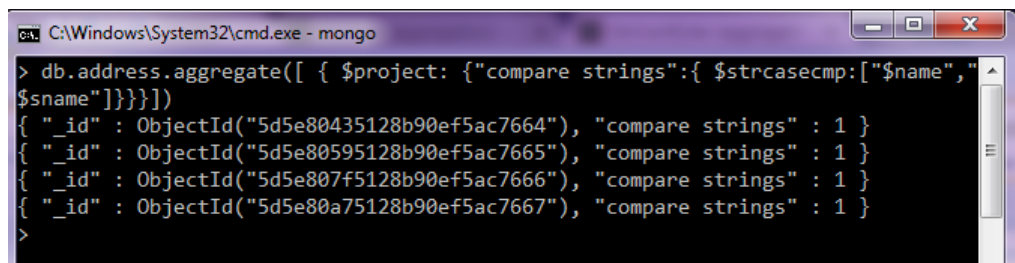
```
C:\Windows\System32\cmd.exe - mongo
> db.address.aggregate([ { $project: { "count characters":{$strLenBytes:"$add"}} }
])
{ "_id" : ObjectId("5d5e80435128b90ef5ac7664"), "count characters" : 14 }
{ "_id" : ObjectId("5d5e80595128b90ef5ac7665"), "count characters" : 14 }
{ "_id" : ObjectId("5d5e807f5128b90ef5ac7666"), "count characters" : 18 }
{ "_id" : ObjectId("5d5e80a75128b90ef5ac7667"), "count characters" : 17 }
>
```

\$strcasecmp

- This operator is used to compare two string & return a number either 1 , 0 or -1
 - if string1 > string2 then return 1
 - if string1 < string 2 then return -1
 - if string1 and string 2 are equal then return 0
- **Syntax:** { \$strcasecmp:[\$str1,\$str2] }

EXAMPLE:

```
>db.address.aggregate([ {$project: { "compare strings": {$strcasecmp:
["$name","$sname"]}}}])
```



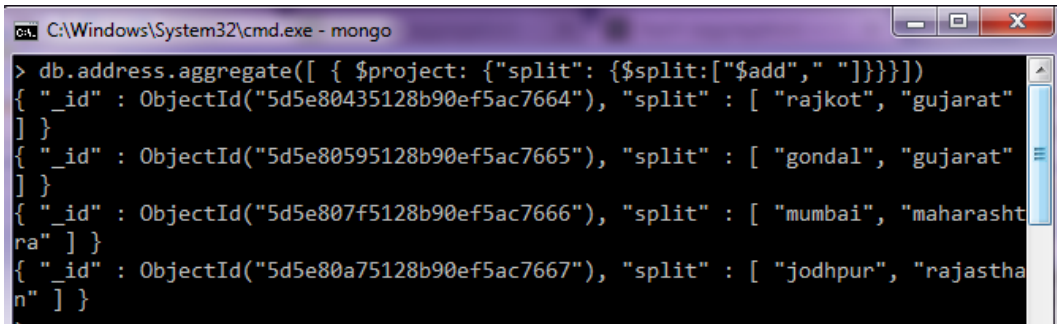
```
C:\Windows\System32\cmd.exe - mongo
> db.address.aggregate([ { $project: { "compare strings":{ $strcasecmp:["$name","
$sname"]}}}])
{ "_id" : ObjectId("5d5e80435128b90ef5ac7664"), "compare strings" : 1 }
{ "_id" : ObjectId("5d5e80595128b90ef5ac7665"), "compare strings" : 1 }
{ "_id" : ObjectId("5d5e807f5128b90ef5ac7666"), "compare strings" : 1 }
{ "_id" : ObjectId("5d5e80a75128b90ef5ac7667"), "compare strings" : 1 }
>
```

\$split

- This operator is used to split a string according to specified delimiter
- Delimiter is any character or symbol
- **Syntax:** { \$split: [<string expression>, <delimiter>] }

EXAMPLE:

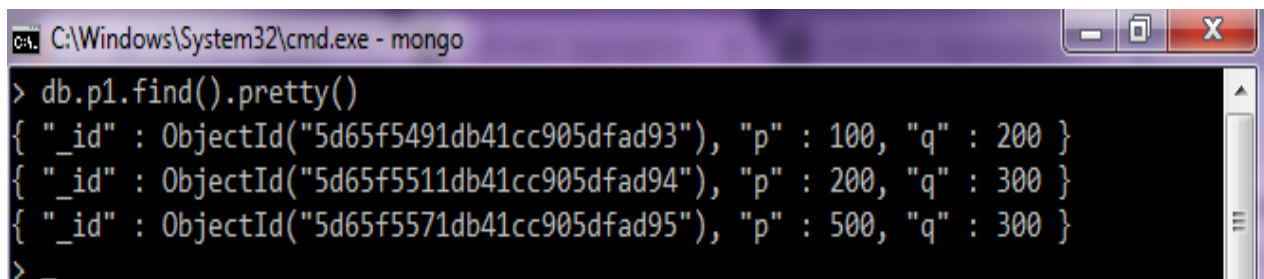
```
>db.address.aggregate([{$project:{ "split": {$split: ["$add"," "]}}}])
```



```
C:\Windows\System32\cmd.exe - mongo
> db.address.aggregate([{$project:{ "split": {$split: ["$add"," "]}}}])
{ "_id" : ObjectId("5d5e80435128b90ef5ac7664"), "split" : [ "rajkot", "gujarat" ] }
{ "_id" : ObjectId("5d5e80595128b90ef5ac7665"), "split" : [ "gondal", "gujarat" ] }
{ "_id" : ObjectId("5d5e807f5128b90ef5ac7666"), "split" : [ "mumbai", "maharashtra" ] }
{ "_id" : ObjectId("5d5e80a75128b90ef5ac7667"), "split" : [ "jodhpur", "rajasthan" ] }
```

20. EXPLAIN ARITHMETIC AGGREGATION OPERATOR OF MONGODB?

- Arithmetic expressions perform mathematic operations on numeric values
- So that we can use these operators with the fields whose datatype is of values is number
- We can use following arithmetic aggregation operators in MongoDB:
 - \$abs
 - \$sqrt
 - \$floor
 - \$ceil
 - \$trunc
 - \$add
 - \$subtract
 - \$multiply
 - \$divide
 - \$mod
 - \$pow
- Here is a sample document called p1 in which we are going to perform all arithmetic operations



```
C:\Windows\System32\cmd.exe - mongo
> db.p1.find().pretty()
{ "_id" : ObjectId("5d65f5491db41cc905dfad93"), "p" : 100, "q" : 200 }
{ "_id" : ObjectId("5d65f5511db41cc905dfad94"), "p" : 200, "q" : 300 }
{ "_id" : ObjectId("5d65f5571db41cc905dfad95"), "p" : 500, "q" : 300 }
```

\$abs

- This operator is used to convert negative number into positive.
- If the fields value is null then this function return null
- **Syntax:** { \$abs: <number> }

EXAMPLE:

```
>db.p1.aggregate([{$project: { "ans": {$abs: {$subtract: ["$p","$q"]}}}}])
```

OUTPUT:

```
> db.p1.aggregate([{$project: {"ans":{$abs: {$subtract: ["$p","$q"]}}}}])
{ "_id" : ObjectId("5d65f5491db41cc905dfad93"), "ans" : 100 }
{ "_id" : ObjectId("5d65f5511db41cc905dfad94"), "ans" : 100 }
{ "_id" : ObjectId("5d65f5571db41cc905dfad95"), "ans" : 200 }
```

\$sqrt

- This operator is finding out square root of positive number & return answer in double datatype.
- **Syntax:** { \$sqrt: <number> }

EXAMPLE:

```
>db.p1.aggregate([{$project: { "square root": {$sqrt: "$p"}}}}])
```

OUTPUT:

```
C:\Windows\System32\cmd.exe - mongo
> db.p1.aggregate([{$project: {"square root":{$sqrt:"$p"}}}}])
{ "_id" : ObjectId("5d65f5491db41cc905dfad93"), "square root" : 10 }
{ "_id" : ObjectId("5d65f5511db41cc905dfad94"), "square root" : 14.1421356237309
51 }
{ "_id" : ObjectId("5d65f5571db41cc905dfad95"), "square root" : 22.3606797749978
98 }
>
```

\$floor

- This operator Returns the lowest integer less than or equal to the specified number.
- **Syntax:** { \$floor: <number> }

EXAMPLE:

```
>db.p1.aggregate([{$project: { "floor": {$floor: "$code"}}}}])
```

OUTPUT:

```

C:\Windows\System32\cmd.exe - mongo
> db.p1.find().pretty()
{
  "_id" : ObjectId("5d65f5491db41cc905dfad93"),
  "p" : 100,
  "q" : 200,
  "code" : 5.5
}
{
  "_id" : ObjectId("5d65f5511db41cc905dfad94"),
  "p" : 200,
  "q" : 300,
  "code" : 11.25
}
{
  "_id" : ObjectId("5d65f5571db41cc905dfad95"),
  "p" : 500,
  "q" : 300,
  "code" : 9.8925
}
> db.p1.aggregate([{$project: {"floor": {$floor: "$code"}}}])
{ "_id" : ObjectId("5d65f5491db41cc905dfad93"), "floor" : 5 }
{ "_id" : ObjectId("5d65f5511db41cc905dfad94"), "floor" : 11 }
{ "_id" : ObjectId("5d65f5571db41cc905dfad95"), "floor" : 9 }

```

\$ceil

- This operator Returns the smallest integer greater than or equal to the specified number.
- Syntax: { \$ceil: <number> }

EXAMPLE:

```
>db.p1.aggregate([{$project: { "ceiling": {$ceil: "$code"}}}])
```

OUTPUT:

```

C:\Windows\System32\cmd.exe - mongo
> db.p1.find().pretty()
{
  "_id" : ObjectId("5d65f5491db41cc905dfad93"),
  "p" : 100,
  "q" : 200,
  "code" : 5.5
}
{
  "_id" : ObjectId("5d65f5511db41cc905dfad94"),
  "p" : 200,
  "q" : 300,
  "code" : 11.25
}
{
  "_id" : ObjectId("5d65f5571db41cc905dfad95"),
  "p" : 500,
  "q" : 300,
  "code" : 9.8925
}
> db.p1.aggregate([{$project: {"ceiling": {$ceil: "$code"}}}])
{ "_id" : ObjectId("5d65f5491db41cc905dfad93"), "ceiling" : 6 }
{ "_id" : ObjectId("5d65f5511db41cc905dfad94"), "ceiling" : 12 }
{ "_id" : ObjectId("5d65f5571db41cc905dfad95"), "ceiling" : 10 }
>

```

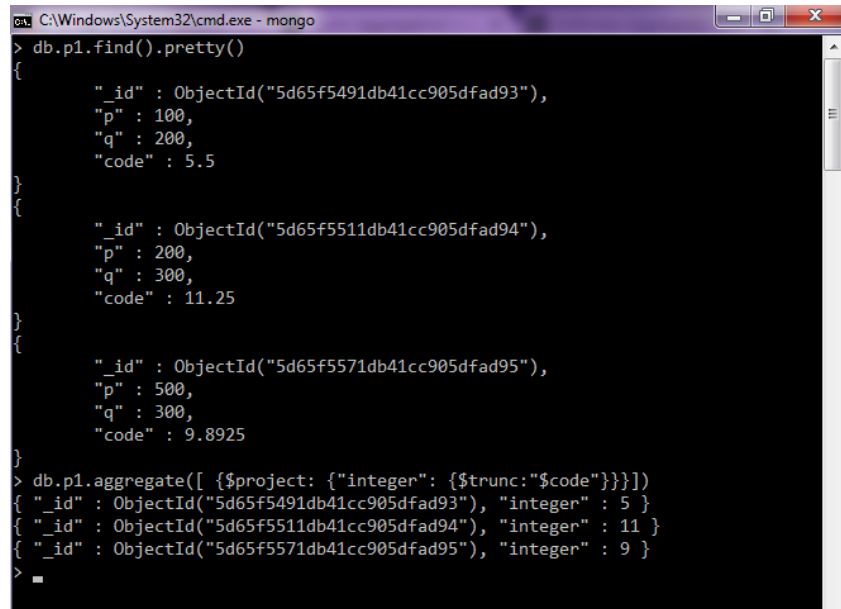
\$trunc

- This operator return integer portion from fractional numbers
- **Syntax:** { \$trunc: <number> }

EXAMPLE:

```
>db.p1.aggregate([{$project: { "integer": {$trunc: "$code"} } }])
```

OUTPUT:



```
C:\Windows\System32\cmd.exe - mongo
> db.p1.find().pretty()
{
  "_id" : ObjectId("5d65f5491db41cc905dfad93"),
  "p" : 100,
  "q" : 200,
  "code" : 5.5
}
{
  "_id" : ObjectId("5d65f5511db41cc905dfad94"),
  "p" : 200,
  "q" : 300,
  "code" : 11.25
}
{
  "_id" : ObjectId("5d65f5571db41cc905dfad95"),
  "p" : 500,
  "q" : 300,
  "code" : 9.8925
}
> db.p1.aggregate([{$project: { "integer": {$trunc: "$code"} } }])
{ "_id" : ObjectId("5d65f5491db41cc905dfad93"), "integer" : 5 }
{ "_id" : ObjectId("5d65f5511db41cc905dfad94"), "integer" : 11 }
{ "_id" : ObjectId("5d65f5571db41cc905dfad95"), "integer" : 9 }
>
```

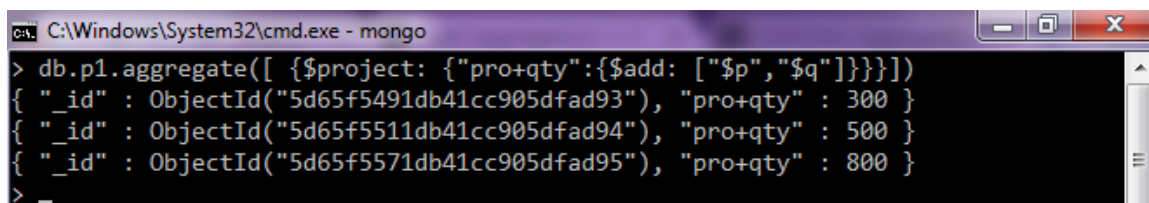
\$add

- This operator is used to add two numbers.
- **Syntax:** { \$add: [<number1>, <number2>] }

EXAMPLE:

```
>db.p1.aggregate([{$project: { "pro+qty": {$add: [ "$p", "$q" ] } } }])
```

OUTPUT:



```
C:\Windows\System32\cmd.exe - mongo
> db.p1.aggregate([{$project: { "pro+qty": {$add: [ "$p", "$q" ] } } }])
{ "_id" : ObjectId("5d65f5491db41cc905dfad93"), "pro+qty" : 300 }
{ "_id" : ObjectId("5d65f5511db41cc905dfad94"), "pro+qty" : 500 }
{ "_id" : ObjectId("5d65f5571db41cc905dfad95"), "pro+qty" : 800 }
>
```

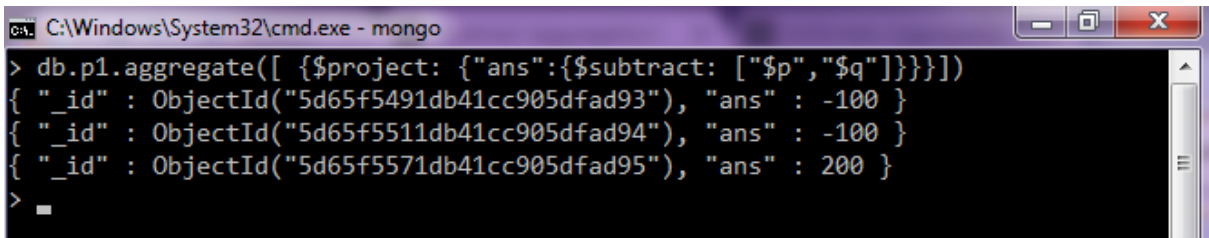
\$subtract

- This operator is used to subtract two numbers.
- **Syntax:** { \$subtract: [<number1>, <number2>] }

EXAMPLE:

```
>db.p1.aggregate([{$project: { "ans": {$subtract: ["$p","$q"]}}}]])
```

OUTPUT:



```
C:\Windows\System32\cmd.exe - mongo
> db.p1.aggregate([{$project: {"ans":{$subtract: ["$p","$q"]}}}]])
{ "_id" : ObjectId("5d65f5491db41cc905dfad93"), "ans" : -100 }
{ "_id" : ObjectId("5d65f5511db41cc905dfad94"), "ans" : -100 }
{ "_id" : ObjectId("5d65f5571db41cc905dfad95"), "ans" : 200 }
>
```

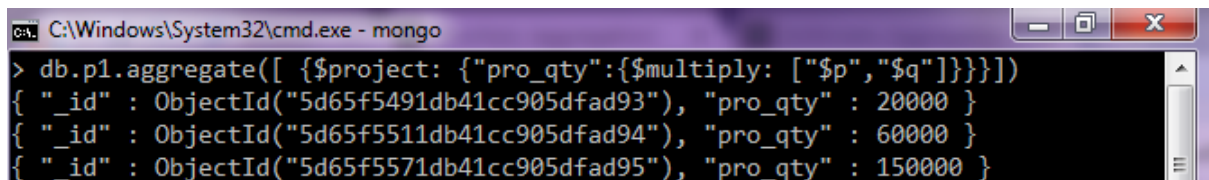
\$multiply

- This operator is used to multiply two numbers.
- Syntax: { \$multiply:[<number1>,<number2>]}

EXAMPLE:

```
>db.p1.aggregate([{$project: { "pro_qty": {$multiply: ["$p","$q"]}}}]])
```

OUTPUT:



```
C:\Windows\System32\cmd.exe - mongo
> db.p1.aggregate([{$project: {"pro_qty":{$multiply: ["$p","$q"]}}}]])
{ "_id" : ObjectId("5d65f5491db41cc905dfad93"), "pro_qty" : 20000 }
{ "_id" : ObjectId("5d65f5511db41cc905dfad94"), "pro_qty" : 60000 }
{ "_id" : ObjectId("5d65f5571db41cc905dfad95"), "pro_qty" : 150000 }
```

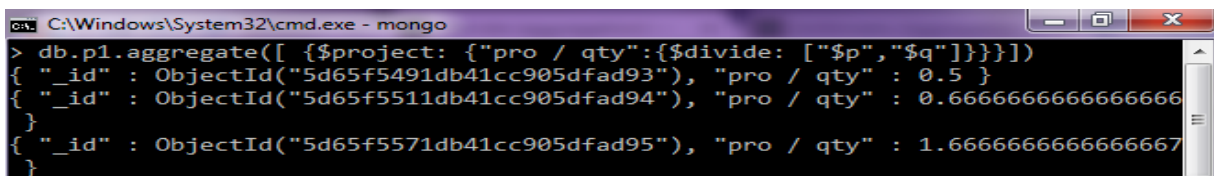
\$divide

- This operator is used to divide two numbers.
- Syntax: { \$divide:[<number1>,<number2>]}

EXAMPLE:

```
>db.p1.aggregate([{$project: { "pro/qty": {$divide: ["$p","$q"]}}}]])
```

OUTPUT:



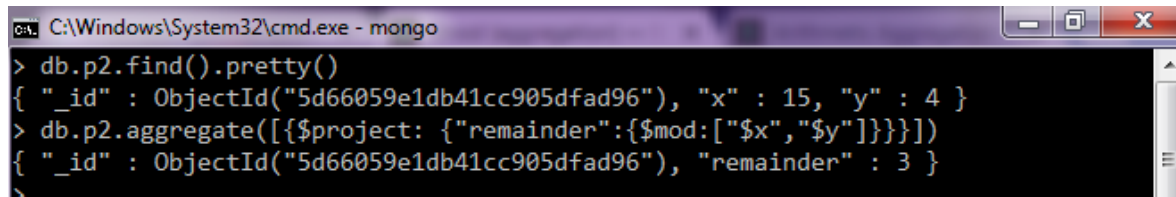
```
C:\Windows\System32\cmd.exe - mongo
> db.p1.aggregate([{$project: {"pro / qty":{$divide: ["$p","$q"]}}}]])
{ "_id" : ObjectId("5d65f5491db41cc905dfad93"), "pro / qty" : 0.5 }
{ "_id" : ObjectId("5d65f5511db41cc905dfad94"), "pro / qty" : 0.6666666666666666 }
{ "_id" : ObjectId("5d65f5571db41cc905dfad95"), "pro / qty" : 1.6666666666666667 }
```

\$mod

- This operator is used to divide two numbers & return reminder.
- Syntax: { \$mod:[<number1>,<number2>]}

EXAMPLE:

```
>db.p2.aggregate([{$project: { "remainder": {$mod: ["$x","$y"]}}}]])
```

OUTPUT:

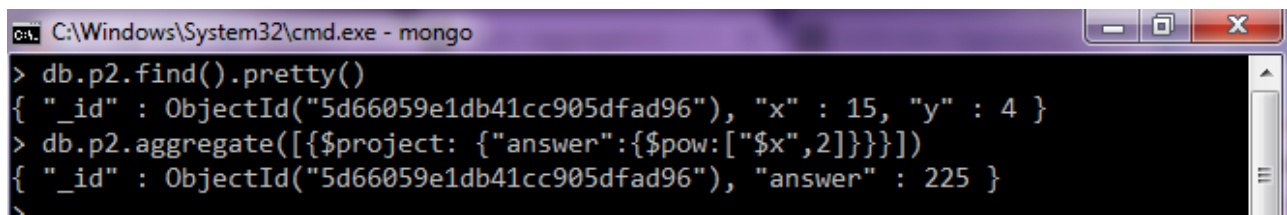
```
C:\Windows\System32\cmd.exe - mongo
> db.p2.find().pretty()
{ "_id" : ObjectId("5d66059e1db41cc905dfad96"), "x" : 15, "y" : 4 }
> db.p2.aggregate([{$project: {"remainder":{$mod:["$x","$y"]}}}]])
{ "_id" : ObjectId("5d66059e1db41cc905dfad96"), "remainder" : 3 }
```

\$pow

- This operator is used to return a number that specified a raise to number of expression.
- **Syntax:** { \$pow:[<expression>,<exponent>]}

EXAMPLE:

```
>db.p2.aggregate([{$project: { "answer": {$pow: ["$x",2]}}}]])
```

OUTPUT:

```
C:\Windows\System32\cmd.exe - mongo
> db.p2.find().pretty()
{ "_id" : ObjectId("5d66059e1db41cc905dfad96"), "x" : 15, "y" : 4 }
> db.p2.aggregate([{$project: {"answer":{$pow:["$x",2]}}}]])
{ "_id" : ObjectId("5d66059e1db41cc905dfad96"), "answer" : 225 }
```

21. EXPLAIN BACKUP AND RESTORE FACILITY OF MONGODB.

- Backup and Restore in MongoDB is an important part to handle a database.
- This facility allows you to secure your data can be removed or crashed or corrupted accidentally or by any mistake.
- Backup can be done in various formats.
- MongoDB provides various ways to backup and restore the database.
 - Backup by copying core data files.
 - Backup using mongodump tool.
 - MongoDB Cloud manager.

EXPORT OR BACKUP DATA:

- To create backup of database in MongoDB, we can use mongodump command.

- mongodump command reads data from a MongoDB database and creates high BSON files.
- You can also create backup file in different format.
- Using mongodump utility we can take back entire server, database, collection or a specific part of selected collections.

Mongodump:

- **Syntax: mongodump [options]**
- This command used to create backup or export database in mongodb.
- We can use following options with mongodump command.

--db	Specify database name.
--collection	Specify collection name.
--out	Specify location/ path where you want to store database.
-u	Specify user name.
-p	Specify password.

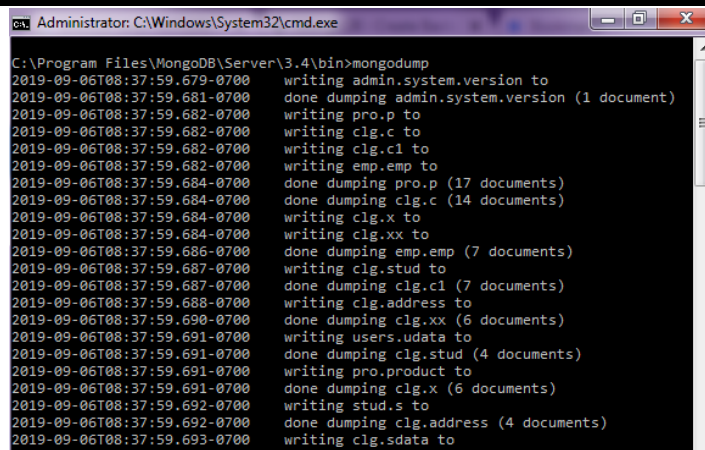
1. Create backup of entire mongodb databases

- **Steps:**
 - Open a command prompt and run “mongod” command.
 - Open another command prompt by run as an administrator and run “mongodump” command
 - It create backup of all database using default port 27017.
 - It will create a dump folder in current location & store all databases inside it.

EXAMPLE:

```
>mongodump
```

OUTPUT:



```
C:\Program Files\MongoDB\Server\3.4\bin>mongodump
2019-09-06T08:37:59.679-0700 writing admin.system.version to
2019-09-06T08:37:59.681-0700 done dumping admin.system.version (1 document)
2019-09-06T08:37:59.682-0700 writing pro.p to
2019-09-06T08:37:59.682-0700 writing clg.c to
2019-09-06T08:37:59.682-0700 writing clg.c1 to
2019-09-06T08:37:59.682-0700 writing emp.emp to
2019-09-06T08:37:59.684-0700 done dumping pro.p (17 documents)
2019-09-06T08:37:59.684-0700 done dumping clg.c (14 documents)
2019-09-06T08:37:59.684-0700 writing clg.x to
2019-09-06T08:37:59.684-0700 writing clg.xx to
2019-09-06T08:37:59.686-0700 done dumping emp.emp (7 documents)
2019-09-06T08:37:59.687-0700 writing clg.stud to
2019-09-06T08:37:59.687-0700 done dumping clg.c1 (7 documents)
2019-09-06T08:37:59.688-0700 writing clg.address to
2019-09-06T08:37:59.690-0700 done dumping clg.xx (6 documents)
2019-09-06T08:37:59.691-0700 writing users.udata to
2019-09-06T08:37:59.691-0700 done dumping clg.stud (4 documents)
2019-09-06T08:37:59.691-0700 writing pro.product to
2019-09-06T08:37:59.691-0700 done dumping clg.x (6 documents)
2019-09-06T08:37:59.692-0700 writing stud.s to
2019-09-06T08:37:59.692-0700 done dumping clg.address (4 documents)
2019-09-06T08:37:59.693-0700 writing clg.sdata to
```

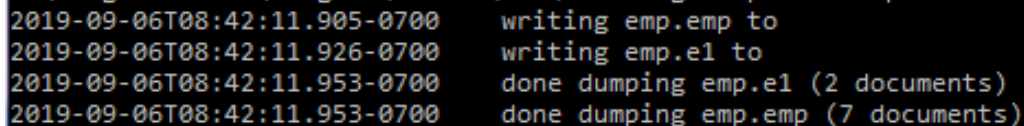
2. Create a backup of specific database

- When we want to create backup of some specific database we have to specify name of database along with mongodump command.
- You can also specify -- out argument to specify where to store backup data.

EXAMPLE:

```
>mongodump --db emp --out e:\meghna\
```

OUTPUT:



```
2019-09-06T08:42:11.905-0700 writing emp.emp to
2019-09-06T08:42:11.926-0700 writing emp.e1 to
2019-09-06T08:42:11.953-0700 done dumping emp.e1 (2 documents)
2019-09-06T08:42:11.953-0700 done dumping emp.emp (7 documents)
```

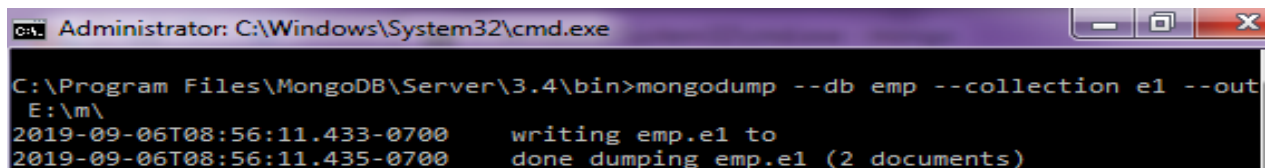
3. Create backup of specific collection.

- When we want to create backup of some specific collection of database we have to specify name of database along with collection name.

EXAMPLE:

```
>mongodump --db emp --collections e1 - -out e:\meghna\
```

OUTPUT:



```
C:\Program Files\MongoDB\Server\3.4\bin>mongodump --db emp --collection e1 --out
E:\m\
2019-09-06T08:56:11.433-0700 writing emp.e1 to
2019-09-06T08:56:11.435-0700 done dumping emp.e1 (2 documents)
```

Mongoexport:

- Syntax: mongoexport [options]

- We can use mongoexport tool use to take backup of some specific database , collection in json or csv formate.

-d	Specify database name.
-c	Specify collection name.
-o	Specify location/ path where you want to store database.
-f	Specify field name
--type	Specify type of export file json, csv etc

- We can also use mongoexport command to backup database in mongodb.

EXAMPLE:

1. EXPORT EMP DATABASE WITH ITS COLLECTION

```
>mongodump -d "emp" -c "e1" -o "e:\meghna\EMP.json"
```

OUTPUT:

```
C:\Program Files\MongoDB\Server\3.4\bin>mongoexport -d "emp" -c "e1" -o "E:\m\emp.json"
2019-09-06T08:59:44.940-0700    connected to: localhost
2019-09-06T08:59:45.017-0700    exported 2 records
```

2. EXPORT EMP DATABASE WITH ITS COLLECTION IN CSV FORMAT

```
>mongodump -d "emp" -c "emp" -f ename,gender,desig, deptno,sal -o "e:\meghna\EMP.json"
```

```
C:\Program Files\MongoDB\Server\3.4\bin>mongoexport -d "emp" -c "emp" -f ename,desig,gender,deptno,sal -o "e:\empdata.csv"
2019-09-06T09:06:16.811-0700    connected to: localhost
2019-09-06T09:06:16.865-0700    exported 7 records
```

IMPORT OR RESTORE DATA:

- The mongorestore command is used to restore the backup data.
- This command restores all the data from backup directory.
- Using mongorestore command we can restore either complete backup or a partition of backup.

Mongorestore:

- **Syntax:** mongorestore

- This command used to restore or import database in mongodb.

1. Restore entire mongodb databases

EXAMPLE:

```
>mongorestore
```

2. restore of specific database

- When we want to create backup of some specific database we have to specify name of database along with mongorestore command.

EXAMPLE:

```
>mongorestore -db emp e:\m\emp
```

Mongoimport:

- Syntax: mongoimport [options]

-d	Specify database name.
-c	Specify collection name.
-dir	Specify location/ path where you want to restore database.
--type	Specify json, csv etc

EXAMPLE:

```
>mongorestore -d emp -c emp -dir e:\m\empdata.csv
```

22. EXPLAIN MAP REDUCING.

- Map-Reduce are a data processing paradigm.
- Map-Reduce provide mechanism that focus on large volumes of data into useful aggregated results.
- MongoDB uses Map-Reduce command for map and reduce operations.
- Map-Reduce is used for processing large data sets.
- Map-Reduce command takes two functions as an input
 - Map function
 - Reduce function

Map function:

- It is also called mapper because it collects and arrange data.
- This function start collects data and build map.
- It required key and values with emit()
- This function groups data and stores value in an array

Reducer function:

- It is also called reducer because it eliminates extra data to generate aggregate results.
- Reduce function is used to process on data and generate aggregate result.

mapReduce()

- mapReduce() is used to perform map reduce task in mongodb.
- **Syntax:**

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

In syntax:

Map()	The map is a javascript function that maps a value with a key and emits a key-value pair.
Reduce()	The reduce is a javascript function that reduces or groups all the documents having the same key
Out	The out specifies the location of the map-reduce query result.
Query	The query specifies the optional selection criteria for selecting documents.
Sort	The sort specifies the optional sort criteria.
Limit	The limit specifies the optional maximum number of documents to be returned.

- **For example we have a collection called result**

```
C:\Windows\System32\cmd.exe - mongo
> db.result.find().pretty()
{
  "_id" : ObjectId("5d77bc61c1ca7e5153dd9876"),
  "name" : "ekta",
  "sub" : "sci",
  "mark" : 80
}
{
  "_id" : ObjectId("5d77bc61c1ca7e5153dd9877"),
  "name" : "ekta",
  "sub" : "maths",
  "mark" : 90
}
{
  "_id" : ObjectId("5d77bc61c1ca7e5153dd9878"),
  "name" : "avni",
  "sub" : "sci",
  "mark" : 80
}
{
  "_id" : ObjectId("5d77bc61c1ca7e5153dd9879"),
  "name" : "janki",
  "sub" : "maths",
  "mark" : 88
}
{
  "_id" : ObjectId("5d77bc61c1ca7e5153dd987a"),
  "name" : "raj",
  "sub" : "sci",
  "mark" : 44
}
{
  "_id" : ObjectId("5d77bc61c1ca7e5153dd987b"),
  "name" : "raj",
  "sub" : "maths",
  "mark" : 88
}
>
```

EXAMPLE:

CREATE MAP()

```
>Var mapdata=function(){emit(this.name,this.marks);}
```

CREATE REDUCE()

```
> var reducedata= function(name,marks) {return Array.sum(marks);}
```

USE MAPREDUCE()

```
>db.result.mapReduce( mapdata, reducedata, {out: "map_total"})
```

OUTPUT:

```
{
  "result" : "marks_tot",
  "timeMillis" : 412,
  "counts" : {
    "input" : 6,
    "emit" : 6,
    "reduce" : 2,
    "output" : 4
  },
  "ok" : 1
}
> map.marks_tot.find()
2019-09-11T18:22:15.735-0700 E QUERY [thread1] ReferenceError: map is not def
ined :
@(shell):1:1
> db.marks_tot.find()
{ "_id" : "avni", "value" : undefined }
{ "_id" : "ekta", "value" : NaN }
{ "_id" : "janki", "value" : undefined }
{ "_id" : "raj", "value" : NaN }
>
```


23. EXPLAIN EXTRACTING ANALYSIS OF DATA WITH MAP - REDUCE.

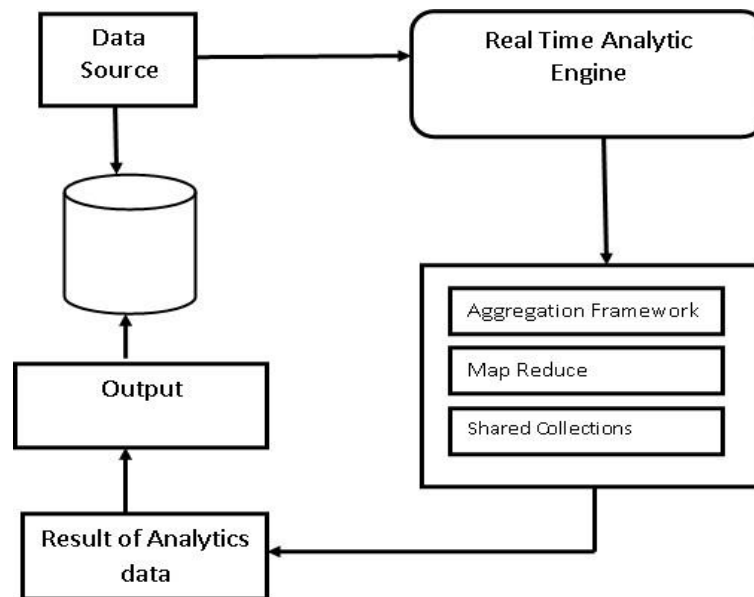
- Analyzing data is the process where we filter out result from external data & produce a proper result.
- Any business contain large amount of data then mongodb provides a solution that fulfill the needs of users.
- Monogdb provides an ideal operational database that provides high performance, storage & retrieval of large scale data.
- There are three categories to extract analyzing data
 - Using aggregation
 - Using Map-Reduce
 - using shared collections

Aggregation / aggregation pipeline

- The aggregation pipeline is a framework for data aggregation which is formed on the concept of data processing pipelines.
- Documents enter a multi-stage pipeline that transforms the documents into aggregated results.
- There are various stage of aggregation pipeline like \$project, \$match, \$group, \$sort , \$limit and \$unwind.

Using map reducing

- Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results.
- Map reduce performed by two functions:
 - Mapper
 - reducer
- In map reduce mongodb extract all from database using Map function & then merge the data which has same key.
- The map function emits key-value pairs.
- Then it reduce the external data by using reduce function.
- So map reduce provide result for only analysis and convert into aggregate result.



Solution call for analyzing data:

- High data volume
 - It contain lots of data source and each data from each datasource
- Dynamic query
 - Provide facility to extract data by providing query according to needs to users.
- Reduce delay time between the collections & queries:
 - It combines minimum time to process before event appears in report.
- Map reduce with annalistic data
 - Provide realtime approach so that output directly in the form of document. it run mongodb inside as local server

Shared collections

- Sharding is a method for distributing data across multiple machines.
- MongoDB uses sharding to support deployments with very large data sets and high throughput operations.
- Database systems with large data sets or high throughput applications can challenge the capacity of a single server.
- There are two methods for addressing system growth: vertical and horizontal scaling.
- MongoDB supports *horizontal scaling* through sharding.

- Starting in version 4.2, MongoDB not approve the map-reduce option to create a new sharded collection and the use of the sharded option for map-reduce.
- To output to a sharded collection, create the sharded collection first.
- MongoDB 4.2 also deprecates the replacement of an existing sharded collection.

24. EXPLAIN LOGGING.

- MongoDB provides a mechanism to send the data in logfile in a particular location.
- Any RDBMS maintain logfile to record the query processing data so that during system crash we can find out the reason.

--logpath

- By default mongod send its log to stdout directory.
- Most init scripts use the --logpath option to send logs to a file.
- If you have multiple MongoDB instances on a single machine then logs are stored in separate files.
- This will create the file if it does not exist, assuming you have write permissions to the directory.
- It will also overwrite the log file if it already exists, erasing any older log entries.
- If you'd like to keep old logs around, use the --logappend option in addition to --logpath is highly recommended.

--logLevel

- Mongod provides **db.setLogLevel()** to set loglevel in a single server system

Log level	Description
0	No logging
1	Logs for slow queries
2	Logs for all queries
3,4,5	According to different task maintain log

Example:

```
>db.setLogLevel(3)
```

- To set log level on multiple system **setParameter** command will be used.

Example:

```
>db.adminCommand({"setParameter":1, "logLevel":3})
```

- When we increase log level mongodb will print out almost all actions it takes, including contents of every request handled.

setProfilingLevel:

- By default, MongoDB logs information about queries that take longer than 100 ms to run.
- If 100 ms it too short or too long for your application, you can change the threshold with *setProfilingLevel*:

```
> db.setProfilingLevel(1, 500) // Only log queries that take longer than 500ms
```

OUTPUT:

```
{ "was" : 0, "slowms" : 100, "ok" : 1 }
```

```
> db.setProfilingLevel(0)
```

OUTPUT:

```
{ "was" : 1, "slowms" : 500, "ok" : 1 }
```

cron job

- cron job that rotates your log every day or week.
- If MongoDB was started with --logpath, sending the process a SIGUSR1 signal will make it rotate the log.
- logRotate command that used to perform this task.

EXAMPLE:

```
> db.adminCommand({"logRotate" : 1})
```

--logMessage

- MongoDB includes the severity level and the component for each log message when output to the console or a logfile.
- logMessage conatin following format:

```
<timestamp> <severity> <component> [<context>] <message>
```
- where

- timestamp is iso8601-local
- security is specify security levels

Log level	Description
F	Fatal
E	Error
W	Warning
I	Informational
D[1-5]	Debug have level between 1 to 5

- components specify functional categorization of the messages

25. EXPLAIN REAL TIME ANALYZING DATA IN MONGODB.

- MongoDB provides facility to analyze data at any place with fast real time & an less money.
- Most of companies use analytics on real time so analytics must be support simple and faster query retrieval.
- Companies use many applications for complex and long running analysis of data but these applications have slower response time and lower requirements.
- Real time analytics is hard for Relational databases that aren't capable of handling unstructured and semi-structured data.
- Batch processes are the right approach for some jobs. But in many cases, you need to analyze rapidly changing, multi-structured data in real time.
- MongoDB can incorporate any kind of data – any structure, any format, any source – no matter how often it changes. Your analytical engines can be real-time.
- MongoDB is built to scale out on commodity hardware, in your data center or in the cloud and without complex hardware or extra software.
- MongoDB can analyze data of any structure directly within the database, giving you results in real time, and without expensive data warehouse loads.
- Mongoddb is used for realtime analytics data in an area such as financial service, government services, high tech & retail services.
- **Financial services:**

- It includes area like risk analytics & reports, Reference Data Management, Market Data Management
- **Practical use case:**
 - citygroup Building a real-time event subscription service with MongoDB
 - As HSBC adopts technology to make things "simpler, better and faster" for customers, data is at the core of its innovation use mongodb.
- **Government:**
 - Many government service providers use MongoDB including City municipal services, National defense and intelligence agencies, Civilian agencies in healthcare, finance, energy, and more.
 - **Practical use case:**
 - US Department of Veterans Affairs
 - GOV.UK
- **High tech service:**
 - In the most competitive industry of all, tech innovators are using MongoDB to do things they could never do before. Faster. With Less Money.
 - **Practical use case:**
 - eBay Delivering all media metadata at 99.999% availability.
 - McAfee
 - Central data repository for global threat detection platform.

UNIT: 5 HANDLING FILES WITH GRIDFS

1. WHAT IS GRIDFS?

- GridFS is the file system which is used to storing and retrieving files.
- GridFS is a way through which we can store and retrieve large files such as audio files, video files, images, etc. in MongoDB.
- GridFS is a file system of MongoDB but the data of files are stored within MongoDB collections.
- MongoDB GridFS is used to store and retrieve files that go above the BSON document size limit of 16 MB.
- GridFS divides the file into small parts called as chunks.
- Each chunk is stored as a separate document.
- The default size for a chunk is 255kb for all chunks except the last one, which can be as large as necessary.
- MongoDB GridFS uses two collections to store files.
- One is used to store the file chunks and the second one to store file metadata.
- When we query GridFS for a file, the driver reassemble the chunks.
- With MongoDB GridFS, we can perform a range of queries on files stored.
- GridFS is also for storing files that you want to access without loading the entire file into memory.

GridFS Storage Fashion

- MongoDB, GridFS uses two collections to store files; One is used to store the file chunks and the second one to store file metadata.
- Collections are:
 - **File collection** → fs.files
 - **Chunks collection** → fs.chunks

The files collection

- This collection is used to store only metadata information of file.
- It represents information about file along with id, length, name & Metadata.
- Use `db.fs.files.find()` to access files collections.
- When you place `db.fs.files.find()` MongoDB will display below information.

```
{
  "_id" : <ObjectId>,
  "length" : <num>,
  "chunkSize" : <num>,
  "uploadDate" : <timestamp>,
  "md5" : <hash>,
  "filename" : <string>,
  "contentType" : <string>,
  "aliases" : <string array>,
  "metadata" : <any>,
}
```

files._id	It is a unique identifier for this document.
files.length	Size of document in bytes.
files.chunkSize	It specifies The size of each chunk in bytes. The default size is 255 kb.
files.uploadDate	It gives the date the document was first stored by GridFS.
files.md5	An MD5 hash of the complete file returned by the filemd5 command. It is of string type.
files.filename	It is optional. A human-readable name for the GridFS file.
files.contentType	It is optional. It specifies A valid MIME type for the GridFS file.
files.aliases	It is optional. It specifies An array of alias strings.
files.metadata	It is optional. Here, we store the additional information and metadata filed can be of any data.

The chunks collection

- Chunk is small part of file. Chunks collection stores all documents in small parts.
- Default size of each chunk is 255kb so it is possible that one file will divide into multiple chunks.
- db.fs.chunks.find() used to display chunks detail.
- When you place db.fs.chunks.find() it will show following output.


```
{
  "_id" : <ObjectId>,
  "files_id" : <ObjectId>,
  "n" : <num>,
  "data" : <binary>
}
```

chunks._id	Unique Objected of the chunk
chunks.files_id	We can specify the _id of the parent document in the files collection
chunks.n	Sequence number of the chunk. The numbering starts from 0
chunks.data	The chunk's payload as a BSON Binary type.

ADVANTAGES GRIDFS:

- Allows you to storing large number of files.
- Support many types of format in storing files.
- Provides metadata information of every file

DISADVANTAGES OF GRIDFS:

- Not allowed to update the content of entire file.
- GridFS does not provide a way to do an atomic update of a file.
- The file serving performance will be slower than natively serving the file from your web server and file system.

2. EXPLAIN MONGOFILES COMMAND.

- mongoFiles is a command which is used to store, delete , update & display files stored in GridFS objects.
- Using this command GridFS can store any type of file.
- This command serves as an interface between objects stored in your file system and GridFS.

SYNTAX:

mongofiles <options> <commands> <filename>

- **Options:**

- Specify option to connect with mongodb database. You can use following options:

--help	It provides help of mongofiles command
--port	It Provides a port number of server.
--host	It provides host number.By default mongofiles attempts to connect to a MongoDB process running on the localhost port number 27017.
--version	It provides the mongofiles release number.
--username	It provides Specify username for authentication
--password	It provides Specify username for authentication
--local	It Specifies the local filesystem name of a file for put and get operations.
--replace	Alters the behavior of mongofiles put to replace existing GridFS objects with the specified local file
--db or -d	Specify name of database

- **Commands:**

- Specify command to perform operations. We can use commands like:

put <filename>	Copy the specified file from the local file system into GridFS storage.
get <filename>	Copy the specified file from GridFS storage to the local file system.
List	Lists the files in the GridFS store.
delete <filename>	Delete the specified file from GridFS storage.
search <sting>	Lists the files in the GridFS store with names that match any portion of <string>

- **Filename:** specify the name of file.

Example:

1. Run mongofiles utilites & Store a large file in a database

```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Meghna>mongofiles -d stud put F:\MEGHNA_SUB_MATERIAL\MONGODB\1.mp4
2021-02-08T21:15:39.544+0530 Failed: error connecting to db server: no reachable servers

C:\Users\Meghna>mongofiles -d stud put F:\MEGHNA_SUB_MATERIAL\MONGODB\1.mp4
2021-02-08T21:15:52.271+0530 connected to: localhost
added file: F:\MEGHNA_SUB_MATERIAL\MONGODB\1.mp4

C:\Users\Meghna>
```

2. Run mongod shell Use database & show collections

```
Administrator: C:\Windows\system32\cmd.exe - mongo
> show dbs
emp      0.078GB
gridfs   0.078GB
local    0.078GB
stud     0.078GB
>
> use stud
switched to db stud
> show collections
a
ans
ans1
fs.chunks
fs.files
s1
s2
stud
system.indexes
total
> =
```

3. View metadata by placing db.fs.find().pretty()

```
Administrator: C:\Windows\system32\cmd.exe - mongo
> use stud
switched to db stud
> show collections
a
ans
ans1
fs.chunks
fs.files
s1
s2
stud
system.indexes
total
> db.fs.files.find().pretty()
{
  "_id" : ObjectId("60215cb0db37b616680b7bd0"),
  "chunkSize" : 261120,
  "uploadDate" : ISODate("2021-02-08T15:45:57.275Z"),
  "length" : 28822230,
  "md5" : "e8a3c3e0811a5ba2a6b1bc1cf12b699b",
  "filename" : "F:\\MEGHNA_SUB_MATERIAL\\MONGODB\\1.mp4"
}
```

4. View chunks by placing db.chunks.find()

5. Using count method you can view chunks

```
rvnwWhNGcIzC1C1Va0vdPN3KM13jrZy11k1wVgbKooUteZJIsaP71SVEidOZYur4P6x7q586x7QYNP/va
TZ3d8K3Uzu0dc108qNR9Na6PVY+TUVZTm5Sk/40Tm9D1Ko6P8XgTKeu1HmNFR3dqbcZVimDo6LAmoRfc
eNjxC4YQp2p72/5QwtpRqi3SGdzMUewYXsDmxficzd7T9bEccC/mb7EbbwprjEY8pPd/TqdIvvYXSLIO
UyM402xDokj/KhpMQuhHueLnBnn/GCX06/I4QCSJXQpPspjN1MvVr45MIRJB0gLUdG4on7rmM2cVQ0jZ
DBu8YQ5xCQGGk8S35p5xD8or3ZyOn2Ua51tMmdJn3PW1WRbyxyOF6yu6ua0VsR5E+oc1PVNo87hBeQ4x
rp8Sw0c7cSABWhite26rv1CsseFxyItpY6oKrER8A89ubk1uICTiZau4m17PRIh6A8soSRu1Lfx6M8YA1
9hHjbjVjAQPF+8DfwMYoFsXmhrG9yt5oTjMwoXocVV6mSG911L+UzODsN9YEgvj++Rrw/XDttjh71oHK
Bj+3Yf07QW0cGkNqohDAVR/rP2WfQJjar+Pv/x0u4YCKokQd6d1mRGEi/moVf8N6nUWGo1u3B+/FsELX
TK50nE+xHEBRGF8pSFAMM9p2FK/1GYn0m8tkDGC8ivkvKe1pb6Hbm1V1LfEaqQ47t+gJepWnBqKsfFU
2nNtDJCKSONLU09u+qBe40kuBzrXnJGvQir3XPfIWluuCWrtzM8zbtodMn2Q+GsLuTgvU/URPuQduGti
SVoj1T0lrRkuI8Uhm734XJZuryWJRGbEEJai21lorJZp0RMXm1hWHPq+z1RqjLNYDPjFyRv0qbfs0PAa
YmcXnesjQ3HtGalwkMKRiVVx7n+hB")
}
Type "it" for more
> db.fs.chunks.find().count();
111
```

3. HOW TO STORING, SERVING & READING FILE WITH GRIDFS?

- Mongofiles command is used to perform different operation with grideFS like storing and retrieving files from local system to grideFS.
- Mongofiles is a separate command so you have to perform the mongofiles command by opening command prompt.
- To use GridFS we have to open separate commander window & execute **mongofiles** command.
- Mongofiles include following commands.

LIST OUT ALL FILES AVAILABLE IN GRIDFS

list

- **SYNTAX: mongofiles -d <dbname> list**
- It will display the file list of mongo connected to local host.
- It also display the size of file.

EXAMPLE:

```
mongofiles -d STUD list
```

STORE FILE IN GRIDFS

put

- **SYNTAX: mongofiles -d <dbname> put <filename>**
- If you want to add or store the file into gridefs then put command is used.
- You can store different types of files like text files, image file, mp3, etc...

- To store any file in gridfs you have to store file in /bin directory .
- Then we can use put command to store file in GRIDFS.

EXAMPLE:

```
mongofiles -d STUD put myfile.txt
```

- When store file in gridfs it will create to files inside collection: db.fs.files & db.fs.chunks
- Use find() to view the content of these files.
 - a. db.fs.files
 - It will display id, chunks size, upload data, length, md5 and filename will display as a output.
 - b. db.fs.chunks
 - It display the output like _id, file_id, and data.

GET FILE FROM GRIDFS**get**

- **SYNTAX:** `mongofiles -d <dbname> get <filename>`
- The get command is used to retrieve the file from grideFS to the local system
- This command can't display content of file directly

EXAMPLE:

```
mongofiles -d STUD get myfile.txt
```

DELETE FILE STORED IN GRIDFS**delete**

- **SYNTAX:** `mongofiles -d <dbname> delete <filename>`
- If you want to delete a file of grideFS then you have to use the delete command in the mongofiles command prompt.

EXAMPLE:

```
mongofiles -d STUD delete myfile.txt
```

SEARCH STRING FROM FILE STORED IN GRIDFS

search

- **SYNTAX:** `mongofiles -d <dbname> search <string>`
- Lists the files in the GridFS store with names that match any portion of <string>.

EXAMPLE:

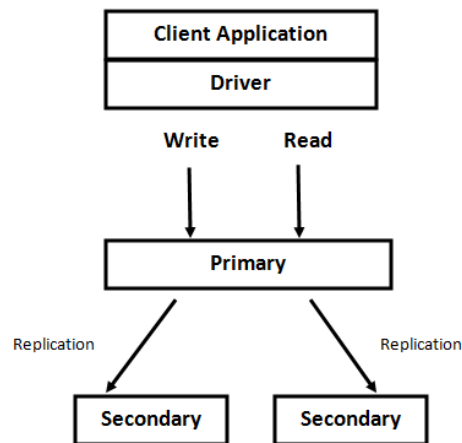
```
mongofiles -d STUD search xyz
```

4. WRITE A DETAIL NOTE ON REPLICATION IN MONGODB.

- Replication is a process or method to synchronize the data across multiple servers.
- The replication process always protects a MongoDB database from the loss of a server due to hardware failure or any other reason.
- In mongodb replication is performed by using **replicaset**.
- In MongoDB, a replica set contains two or more MongoDB server.
- In this group of MongoDB servers, one server is known as a Primary Server and others are known as Secondary servers.
- Every secondary server always keeps copies of the primary's data.
- So, if any time the primary server goes down, then the new primary server is selected from the existing secondary server and process goes on.

Replica set

- Mongodb perform replication using replica set.
- A replica set contains two types of mongodb instances.
- **Primary Instance:**
 - The primary instance receives all write operations.
- **Secondary Instance:**
 - The secondary instance applies operations from the primary so that they have the same data set.
- In a replica set only one primary instance is allowed and all other instances are secondary instances



- Above figure shows 3 members replica set that contain 1 primary instance and 2 secondary instances.
- When a primary instance receives a write operation from a user then it updates its oplog (operation log).
- The oplog is a special kind of capped collection for storing all the operations that modify the data of the database.
- MongoDB first applies the operation on the primary instance then records the operation in the primary's operation log (oplog).
- Now the secondary instance copies the operations and applies them.
- All secondary replica sets contain a copy of primary instance's oplog.

Arbiter Instance:

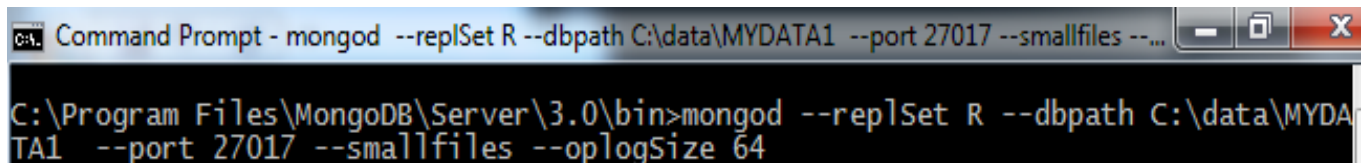
- Sometimes, due to automatic failover or maintenance, the secondary node can't receive the data from primary node.
- If for more than 10 seconds the primary instance doesn't communicate with the secondary the replica set attempts to select a secondary member to become a new primary.
- This task is done by using arbiter

ADVANTAGES:

- We can keep the data safe.
- Ensure the high availability of data
- We can take care of disaster recovery
- No downtimes required for maintenance
- Replica set is always transparent to the application.

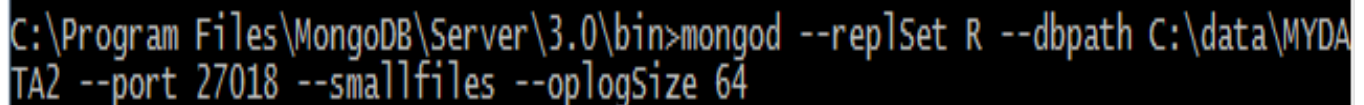
5. HOW TO DEPLOY REPLICATION IN MONGODB.

- Perform following steps to deploy replication.
- In bellow example we create 1 primary & 2 secondary server.
- **step :1**
 - Create 3 directory in data directory : i created mydata1, mydata2, mydata3
- **step :2**
 - Start mongod instance in new cmd & perform following command:
 - This command create replicaset r (r is name of my replicaset)



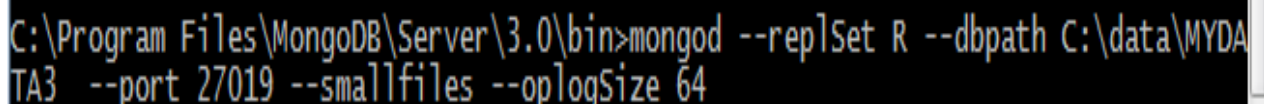
```
Command Prompt - mongod --replSet R --dbpath C:\data\MYDATA1 --port 27017 --smallfiles --...
C:\Program Files\MongoDB\Server\3.0\bin>mongod --replSet R --dbpath C:\data\MYDATA1 --port 27017 --smallfiles --oplogSize 64
```

- **step :3**
 - start mongod instance in new cmd & perform following command to start secondary server 1:



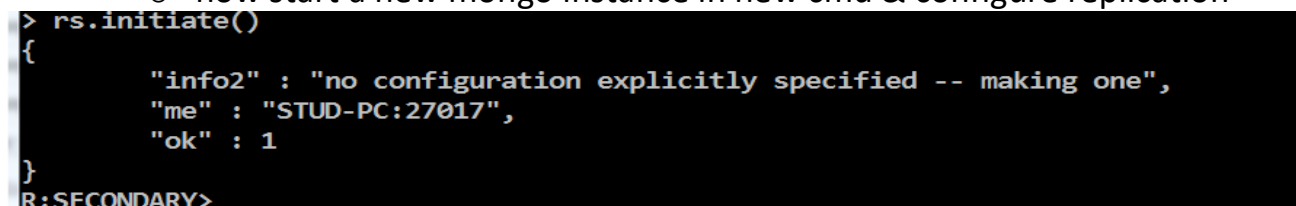
```
C:\Program Files\MongoDB\Server\3.0\bin>mongod --replSet R --dbpath C:\data\MYDATA2 --port 27018 --smallfiles --oplogSize 64
```

- **step :4**
 - start mongod instance in new cmd & perform following command to start secondary server 2 & same for secondary server 3:



```
C:\Program Files\MongoDB\Server\3.0\bin>mongod --replSet R --dbpath C:\data\MYDATA3 --port 27019 --smallfiles --oplogSize 64
```

- **step :5**
 - now start a new mongo instance in new cmd & configure replication



```
> rs.initiate()
{
  "info2" : "no configuration explicitly specified -- making one",
  "me" : "STUD-PC:27017",
  "ok" : 1
}
R:SECONDARY>
```

- **step :6**
 - Open mongo from port 27017 & set configuration
>mongo --port 27017

- Now we add the remaining two mongod instances in the replica set using the “rs.add()” method

>rs.add(“MEGHNA-PC:27018”);

>rs.add(“MEGHNA-PC:27019”)

- To view status use rs.status()

- **step :7**

- now create a new database & insert record which can be access from secondary port

```
R:PRIMARY> rs.initiate(config)
{
  "info" : "try querying local.system.replset to see current configuration",
  "ok" : 0,
  "errmsg" : "already initialized",
  "code" : 23
}
R:PRIMARY> use product
switched to db product
R:PRIMARY> db.pro.insert({id:"123",name:"pen"});
WriteResult({ "nInserted" : 1 })
R:PRIMARY> db.pro.find().pretty()
{
  "_id" : ObjectId("5dad8cb2dff96036d198f1f4"),
  "id" : "123",
  "name" : "pen"
}
R:PRIMARY> ■
```

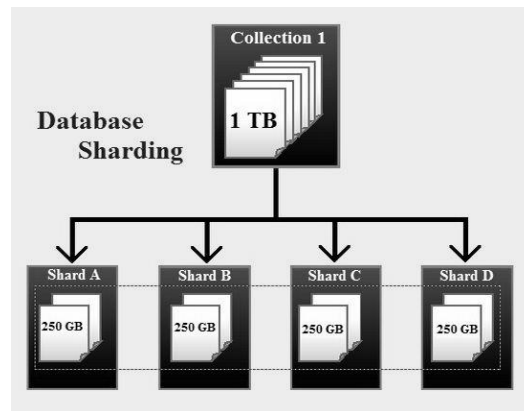
- **step :8**

- Now connect with port 27018
- Place **rs.slaveok ()**
- & then you can see replica copy of your database

6. WHAT IS SHARDING? HOW TO PERFORM SHARDING.

- Sharding is the process of storing data records across multiple machines.
- As the size of the data increases, a single machine can't store all the data and is unable to provide an acceptable read and write throughput.
- To overcome this issue, database systems have two basic approaches:
 - Vertical scaling
 - Horizontal scaling
- MongoDB perform horizontal scaling by using sharding
- Shard divides the data set and distributes the data over multiple servers.
- These multiple server is known as shards.
- Each shard is an independent database and collectively, the shards make up a single logical database.

- Sharding distributes data over multiple shards so it reduces the number of operations for each shard.
- For example:
 - If a database contains 1TB of data and we have 4 shards then each shard will contain approximate 250GB of data.
 - If we have 100 shards then each shard contains approximately 10GB data.



- A sharded cluster contains 3 components. The following describes these components.
- **Shards:**
 - Shards are used to store the data.
 - Each shard contains a replica set. Shards provide high data availability and data consistency.
- **Config Servers:**
 - Config servers are used to store the metadata of clusters.
 - A sharded cluster has exactly 3 config servers.
 - These 3 config servers contain the mapping of the cluster's data stored in shards.
 - Config servers help the query router to select the desired shards to do the operations.
- **Query Routers:**
 - Query routers are the Mongo's instance that interfaces with the client applications and does the operations of the appropriate shards.

- When a query comes to the query router then the query router first uses the metadata (config server) and selects the single or multiple shards and performs the desired operations to shards and then returns the results to the client
- MongoDB always tries to balance the data distribution, for this MongoDB uses the following two approaches.
 - Splitting
 - Balancing

Splitting:

- Splitting is a background process that restricts the chunks from growing too large.
- When the size of a chunk crosses the value of a specified chunk size, MongoDB splits the chunks into half.
- In the split process MongoDB does not modify the shards or migrate any data. The splits provide sufficient meta-data change.

Balancing:

- Balancing is a background process.
- Due to an uneven distribution of sharded collections, the query router runs a balancer process.
- The balancer process migrates chunks from the shard containing a large number of chunks to the shard that contains the least numbers of chunks.

7. WRITE A NOTE ON OPTIMIZATION IN MONGODB.

- Optimization is an act or process to make something is fully perfect.
- It is a simple method to perform a design or a system as effective as possible.
- There are many factors which affects the database performance and responsiveness.
- Optimization is used to analyze the db its performance, indexing, query structure, data model, application design, operational factor, architecture, system configuration etc.
- Optimization is used with mongo db including different types of category like.
 - Analyzing
 - To check mongo db performance
 - Check performance of current operation

- Optimize query performance
- Optimize different design notes

Analyzing mongodb

- You can developed and operate an application with mongo db then some time you may need analyze the performance of application and its database.
- In some cases the number of connections between application & database can have an ability to handle it.
- You can use serverStatus command that provides overview of database state by **db.serverStatus()** command.
- You can also view other information connections.
 - **Connections.current**
 - It specify total number of current clients that connect to database instance
 - **Connections.available**
 - It specifies total number of unused connections available for new clients.

Check mongodb performance & check performance of current operation

- There are different techniques for evaluate the performance of mongo db with different operations.
- Mongo db provide db profiler that show characteristics of each operation of database.
- Profiler use to locate different queries for write operation those are running.
- Profiler provides two different methods which are as under.
 - **db.currentOp ()**
 - This method is used for display the record of current operation running on mongodb instance.
 - **db.explain ()**
 - This method returns query plan for following operations like remove(), distinct(), count(), group(), find(), aggregate() etc.

Optimize query performance

- The optimization is very use full for faster processing of multiple queries or a query on multiple fields.
- It can allow creating the index for searching operations.

- By using the index structure the query execution will be fast and the index contents very small space in memory.

Syntax

Db. <collname>.createindex ({field name});

Example

db.mscit.createindex ({"name":1});

Optimize different design notes

- During the design notes in optimization technique include different types of data related functionality.
- It includes 3 types of functionality which are as under.
- **Dynamic schema**
 - mongodb provides dynamic schema types db and data show there is no need to define the fix structure.
- **Case sensitive string**
 - mongo db screen are case sensitive show STUD & stud is different as well as data is type sensitive.
 - For example


```

{"rno":1} / /number
{"rno":"1"} / /string
          
```
- **BSON document size limit**
 - Size of BSON document is set to 16 MB per document.
 - If you need to store large document which is greater than 16 MB than you can use GridFS
- **Default update one document**
 - By default when we perform update command it will update only one document.
 - If you want to update multiple documents then use multi property along with update

8. EXPLAIN DATABASE ADMINISTRATION WITH MONGODB.

- Administration include ongoing & maintenance of MongoDB instance & its deployment.

- Administration is used to keep track & control on database & its resources.
- The database administration provides the operation and maintains of MongoDB instance.
 - Database administration provide configuration operation and deployment process and many different task which are as under Administration, Deployment, Performance & Monitoring , Optimization ,Replication etc

Administration:-

- Every database management system support the administration of database.
- It means that hear all the data and different use are managed property.
- The administration task includes:
 - Deployment of the system
 - Starting & stopping database
 - User creation & assign privileges.
 - Backup & recovery.
 - Logging
 - Replication

- **Deployment:-**

- It is a one type of installation or implementation process where user allows to write different script code for your data and database.
- To deploy MongoDB you need to choose right hardware & appropriate sever technology.
- In deployment hardware we need to focus on processor , Ram & disk devices.

- **Starting MongoDB server:-**

- Hear you have to start the server first with mongod.exe file.
- There are different command line option used with mongod commands:
- **--dbpath**
 - Each mongo instance required one directory called data directory to store data.
 - If you are working with more than one instance you required separate mongo data directory.
 - When MongoDB start up, it creates a mongod.lock file in data directory
- **--port**

- Specify the port number for server to listen request. The default port of MongoDB is 27017.
 - To run multiple server from single machine you need to choose port
- **--fork**
 - If you start mongo first time , it take few minutes to allocate database file.
 - The parent process will not return to forking until pre-allocation is done.
 - You must have to use --logpath if you used --fork
 - Used with linux command
- **--directoryperdb**
 - This puts each database in its own directory.
 - It allows you to mount different database on different disks.
- **--config**
 - Use this option if you are using file based configuration.
 - To perform file based configuration you need to create config file along with following options:
Port=
Fork=true
Logpath= /data/log/MongoDB.log
Logappend=true
- **Stopping MongoDB server:-**
 - If you want to stop MongoDB server then you have to use the shutdown command called **db.shutdownserver()**
- **Create user & assign privileges**
 - Use createuser command to create user in MongoDB.
 - **Syntax:**

```
createUser ({
  User: <username>,
  Password:<password>,
  Roles: [ { role: <role>,db: <databasename>}]
})
```

Example

```
Db.createUser ({user: "meghna" , passwrod: "m123", roles:[  
{role:"read",db:"emp"}]})
```

Privileges / role:	Details
Dbadmin	<ul style="list-style-type: none">Provides role to perform administration task such as index, gatherstatistics, schema etc
Dbo	<ul style="list-style-type: none">User can perform any administrative action on database.This role includes readwrite, dbadmin and useradmin role
Useradmin	<ul style="list-style-type: none">User can have ability to create & modify user roles in current database.User can allows other user to grant privileges of database
Useradminanydatabase	<ul style="list-style-type: none">By using this role you can become admin of any database
Read	<ul style="list-style-type: none">Read content
Readwrite	<ul style="list-style-type: none">Read & write content

- **Backup & recovery**
- Backup can create a copy of MongoDB data.
- Backup can be performed by using three methods:
 - Backup copying datafile
 - Using mongodump
 - Monogodb cloud manager
- **Logging**
 - By default mongod sends its logs to stdout
 - Most of init script use --logpath option to send log to logfile.
 - if you have a single machine that contain multiple MongoDB instance then we need to create separate log files.
 - log files must have read access.
 - you can also specify log levels using setParameter() command.
- **Replication**

- MongoDB provides a facility to synchronizing data across multiple server which is known as replication.
- A replica set is a group of mongod instances that maintain the same data set.
- The replication process always protects a MongoDB database from the loss of a server due to hardware failure or any other reason.
- Another purpose of replication is the possibility of load balancing.
- **sharding**
 - Sharding splits large data sets into small data sets across multiple mongod instances.
 - Sometimes the data within mongod will be so large, that queries against such big data sets can cause a lot of CPU utilization on the server.
 - So mongod provide facility to divide data into small parts & share across sever by providing facility of sharding.