1. Create DB:

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

```
db.createUser({
  user:"brad",
  pwd:"123",
  roles:["readWrite", "dbAdmin"]
          });
```

_id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows –
_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)

db.store.findById(_id);
db.customers.findOne({first_name: "ram"});

1.1. Find Specific Fields

```
db.customers.find({ title: 'Post One' }, {
  title: 1,
  author: 1
});
```

```

> show dbs
> use store
> db.createCollection('store');
> show collections
> db.store.insert({'interger':23});

> db.store.find().pretty();
> db.store.drop()

> db.store.insert({_id:1,name:'mukesh',country:'indian'});
> db.store.insertOne({_id:1,name:'mukesh',country:'indian'});

db.store.insertMany(
  [
    { _id: 20, devname: "John Wick", tools: "Visual Studio", born: 1948 },
    { _id: 21, devname: "Ganesh Roy", tools: "Net Beans", born: 1945 },
    { _id: 22, devname: "Deeksha Raul", tools: "Unity 3D", born: 1954 }
  ]);
```

1.2. Filtering:
It is also possible to filter your results by giving or adding some specific criteria in which you are interested to.

```
> db.store.find({name:'mukesh'});
> db.store.find({$and: [{name:'mukesh'},{country:'indian'}] });
> db.user.find({},{"first_name":1,_id:0}).limit(2);
> db.user.find().count();
> db.user.find().limit(4);
> db.customers.find({},{"first_name":1,_id:0}).limit(1).skip(1);

> db.store.find({name:{$in: ["mukesh"]}});
> db.store.find({"store.name": {$not: {$eq: 24}}}).pretty()
> db.store.find({"store.name": {$not: {$in: ["mukesh"]}}}}).pretty()
> db.store.find().pretty().limit(2)
> db.store.find().pretty().skip(1)
```

```
db.store.update({_id: 1},
            {
              $set: { name: ["Vocals", "Violin", "Octapad"] }
            })

> db.store.update({_id:2},{$set:{name:['mukesh'],salery:['1234567890']}});
```

```
db.store.remove({ name: "mukesh" })
db.store.remove({})
```

2.mongoDB_data_types:

MongoDB supports many datatypes.Some of them are-

String- String in MongoDB must be UTF - 8 valid.
Integer- Integer can be 32 bit or 64 bit depending upon your server.
Boolean
Double- This type is used to store floating point values.
Min/Max keys- This type is used to compare a value against the lowest and highest BSON elements.
Arrays- This type is used to store arrays or list or multiple values into one key.
Timestamp- ctimestamp.This can be handy for recording when a document has been modified or added.
Object- This datatype is used for embedded documents.
Null- This type is used to store a Null value.
Symbol- This datatype is used identically to a string;
however, it 's generally reserved for languages that use a specific symbol type.

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

Object ID- This datatype is used to store the document's ID.

Binary data- This datatype is used to store binary data.
Code- This datatype is used to store JavaScript code into the document.
Regular expression- This datatype is used to store regular expression.

3.joining the data from 2 table queries

MongoDB is not a relational database, but you can perform a left outer join by using the $lookup stage.

The $lookup stage lets you specify which collection you want to join with the current collection, and which fields that should match.

Consider you have a "`users`" collection and a "`comments`" collection:

```
> db.users.find()
{ "_id" : 1, "userId" : 1, "name" : "Al" }
{ "_id" : 2), "userId" : 2, "name" : "Betty" }
{ "_id" : 3, "userId" : 3, "name" : "Cameron" }

> db.comments.find()
{ "_id" : 1, "userId" : 1, "comment" : "Hi, I'm Al and I love comments." }
{ "_id" : 2, "userId" : 1, "comment" : "Hi, it's Al again. I really do love co
mments." }
{ "_id" : 3, "userId" : 2, "comment" : "I'm Betty. This is my first comment on
 this site." }
{ "_id" : 4, "userId" : 3, "comment" : "This is Cameron. I enjoyed reading you
r website." }
```

As you can see in our dataset there is a common field userId on both collections which allows us to match up each user with their comments.

Use the aggregate() method with the $lookup stage.

```
db.users.aggregate([
  {
    $lookup:
      {
        from: "comments",
        localField: "userId",
        foreignField: "userId",
        as: "combined"
      }
 }
]).pretty()
```

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://127.0.0.1:27017/";
```

```
MongoClient.connect(url, (err, db) => {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.collection('orders').aggregate([
    { $lookup:
      {
        from: 'products',
        localField: 'product_id',
        foreignField: '_id',
        as: 'orderdetails'
      }
    }
  ]).toArray(function(err, res) {
    if (err) throw err;
    console.log(JSON.stringify(res));
    db.close();
  });
});
```

MongoDB can store lots and lots data. And work in a very performent way. Retrive data very fast.
Used in web and mobile applications.
Collection = Table
Data stored in collection as Documents(BSON). This Documents are Seamaless means we can store different data in same collection.
It's store embeded Documents(Document inside Document).
Good to use when there is no ton of inter connected relations Database, Collections, Document
data/db folder inside MongoDB

To insert data into MongoDB collection, you need to use MongoDB's insert() or save() method
We can store boolean, Numbers even files in collections. File store in files system.

3.1. Advantages of MongoDB over RDBMS:

Schema less − MongoDB is a document database in which one collection holds different documents.
Number of fields, content and size of the document can differ from one document to another.
Structure of a single object is clear.
No complex joins.
Ease of scale-out − MongoDB is easy to scale.
Conversion/mapping of application objects to database objects not needed.

3.2. Why Use MongoDB?

Document Oriented Storage − Data is stored in the form of JSON style documents.
Index on any attribute
Replication and high availability
Auto-Sharding
Rich queries
Fast in-place updates

3.3.Where to Use MongoDB?

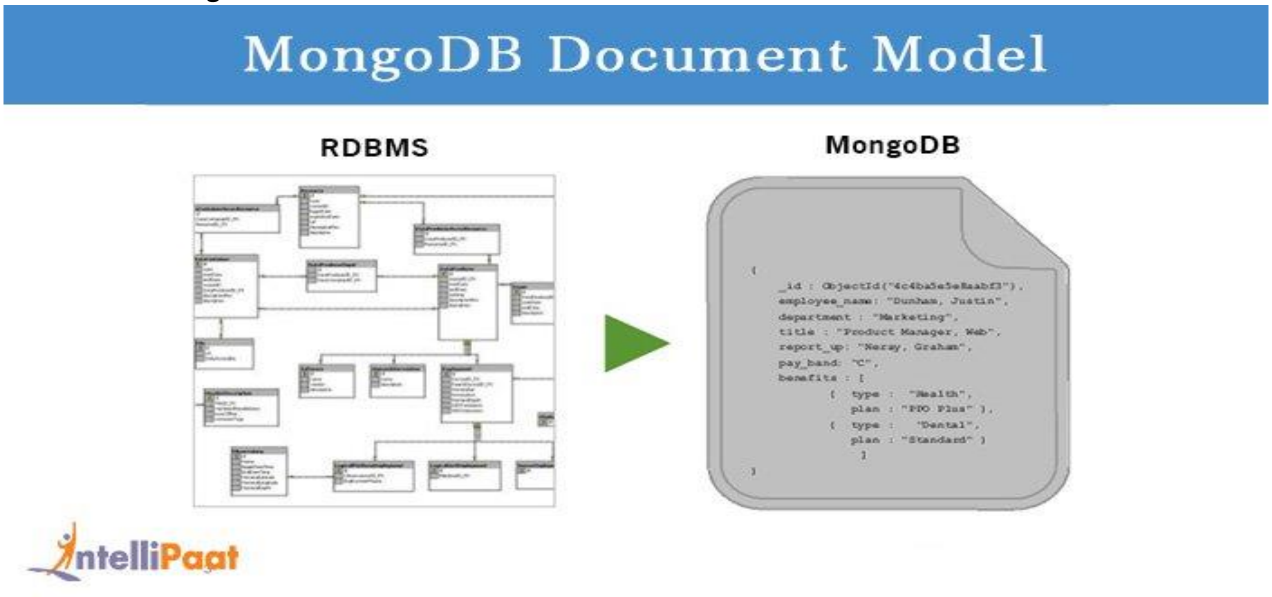
Big Data
Content Management and Delivery
Mobile and Social Infrastructure
User Data Management
Data Hub

4. Difference between DELETE, DROP and TRUNCATE:
TRUNCATE:
TRUNCATE SQL query removes all rows from a table, without logging the individual row deletions.
TRUNCATE is faster than the DELETE query.
TRUNCATE is executed using a table lock and the whole table is locked to remove all records.
TRUNCATE removes all rows from a table.
Minimal logging in the transaction log, so it is faster performance-wise.
Truncate uses less transaction space than the Delete statement.
Truncate cannot be used with indexed views.

DELETE :
DELETE is executed using a row lock, each row in the table is locked for deletion.
The DELETE command is used to remove rows from a table based on WHERE condition.
The delete can be used with indexed views.
Delete uses more transaction space than the Truncate statement.

DROP :
The DROP command removes a table from the database.
All the tables' rows, indexes, and privileges will also be removed.
No DML triggers will be fired.
The operation cannot be rolled back.
DROP and TRUNCATE are DDL commands, whereas DELETE is a DML command.
DELETE operations can be rolled back (undone), while DROP and TRUNCATE operations cannot be rolled

5. Data Modelling:



```
{
    _id : ObjectId("4c4ba5e5e8aabf3"),
    employee_name: "Dunham, Justin",
    department : "Marketing",
    title : "Product Manager, Web",
    report_up: "Neray, Graham",
    pay_band: "C",
    benefits : [
            {  type :  "Health",
               plan : "PPO Plus" },
            {  type :   "Dental",
               plan : "Standard" }
                ]
}
```

Data in MongoDB has a flexible schema.documents in the same collection. They do not need to have the same set of fields or structure, Common fields in a collection's documents may hold different types of data.

Data Model Design: MongoDB provides two types of data models: — Embedded data model and Normalized data model.
Based on the requirement, you can use either of the models while preparing your document.

In Embedded Data Model, you can have (embed) all the related data in a single document, it is also known as de-normalized data model.

For example, assume we are getting the details of employees in three different documents namely, Personal_details, Contact and, Address, you can embed all the three documents in a single one as shown below –

```
db.customers.insert([
  {
  _id: ,
  Emp_ID: "10025AE336"
  Personal_details:{
    First_Name: "Radhika",
    Last_Name: "Sharma",
    Date_Of_Birth: "1995-09-26"
  },
  Contact: {
    e-mail: "radhika_sharma.123@gmail.com",
    phone: "9848022338"
  },
  Address: {
    city: "Hyderabad",
    Area: "Madapur",
    State: "Telangana"
  }
]);
```

6. In Normalized Data Model, you can refer the sub documents in the original document, using references. For example,
you can re-write the above document in the normalized model as:

```
Employee:
{
  _id: <ObjectId101>,
  Emp_ID: "10025AE336"
}

Personal_details:
```

```
{
  _id: <ObjectId102>,
  empDocID: " ObjectId101",
  First_Name: "Radhika",
  Last_Name: "Sharma",
  Date_Of_Birth: "1995-09-26"
}

Contact:
{
  _id: <ObjectId103>,
  empDocID: " ObjectId101",
  e-mail: "radhika_sharma.123@gmail.com",
  phone: "9848022338"
}

Address:
{
  _id: <ObjectId104>,
  empDocID: " ObjectId101",
  city: "Hyderabad",
  Area: "Madapur",
  State: "Telangana"
}
```

7. Suppose a client needs a database design for his blog/website and see the differences between RDBMS and MongoDB schema design. Website has the following requirements.

Every post has the unique title, description and url.
Every post can have one or more tags.
Every post has the name of its publisher and total number of likes.
Every post has comments given by users along with their name, message, data-time and likes.
On each post, there can be zero or more comments.

In RDBMS schema, design for above requirements will have minimum three tables.
While in MongoDB schema, design will have one collection post and the following structure

```
{
  _id: POST_ID
  title: TITLE_OF_POST,
  description: POST_DESCRIPTION,
  by: POST_BY,
  url: URL_OF_POST,
  tags: [TAG1, TAG2, TAG3],
  likes: TOTAL_LIKES,
  comments: [
     {
        user:'COMMENT_BY',
        message: TEXT,
        dateCreated: DATE_TIME,
        like: LIKES
     },
     {
        user:'COMMENT_BY',
        message: TEXT,
        dateCreated: DATE_TIME,
        like: LIKES
     }
  ]
}
```

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

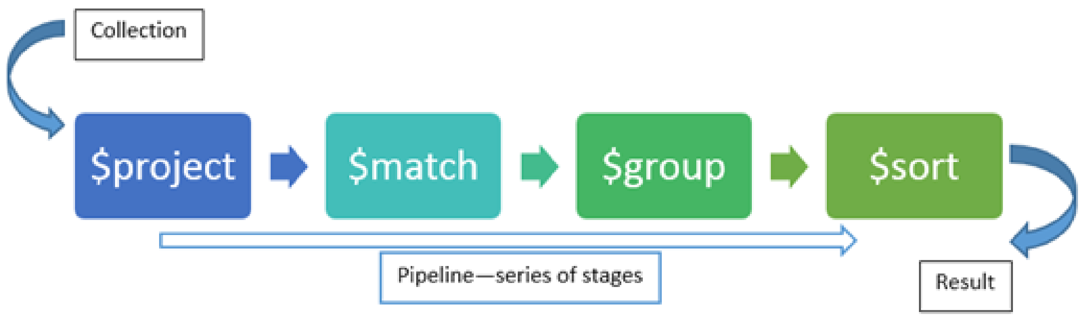
In SQL count(*) and with group by is an equivalent of mongodb aggregation.

if you want to display a list stating how many tutorials are written by each user, then you will use the following aggregate() method -

db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}]);

8.1. Pipeline Concept :

Aggregation Framework : The Aggregation Framework is a set of analytics tools within mongodb that allows you to run various reports or analysis on one or more mongodb collections.



Aggregation Pipeline –

- Take Input from a single collection.
- Pass the documents of the collection through one or more stages.
- Each stage perform different operations in the Pipeline.
- Each stage take as Input whatever the stage before produced as Output.

- The Input and Output for all stages are documents (stream of documents).
- At the end of Pipeline we get access to the output of the transformed and aggregated Output.

```
> db.user.find()
{ "_id" : 1, "name" : "GENWI", "founded_year" : 2010 }
{ "_id" : 2, "name" : "Needium", "founded_year" : 2010 }
{ "_id" : 3, "name" : "Ziippi", "founded_year" : 2011 }
{ "_id" : 4, "name" : "Pixelmatic", "founded_year" : 2011 }
{ "_id" : 5, "name" : "Clowdy", "founded_year" : 2013 }
```

Pipeline

```
db.user.aggregate([
    { $match : { founded_year : 2011 } },
    { $sort : {name:-1}  },
    { $project : { _id : 0, name : 1   } }
])
```

Or Pipeline

```
db.user.aggregate([
    { $match: { founded_year: { $gte: 2010 } } },
    { $group: {
        _id: "$founded_year",
        companies: { $push: "$name" }
        }},
    { $sort: { "_id": 1 } }
])
```

$project – Used to select some specific fields from a collection.

```
db.user.aggregate([    { $project : { _id : 0, name : 1   } }])
```

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

```
db.user.aggregate({ $match: { founded_year: { $gte: 2010 } }})
```

$group – This does the actual aggregation as discussed above.

```
db.user.aggregate({ $group :{_id:ObjectId("5ef64c903da2b374c85626a1"), count:{
$sum:1}}  })
```

$sort –

```
db.user.aggregate({ $sort : {name:-1}  })
```

$skip
$limit

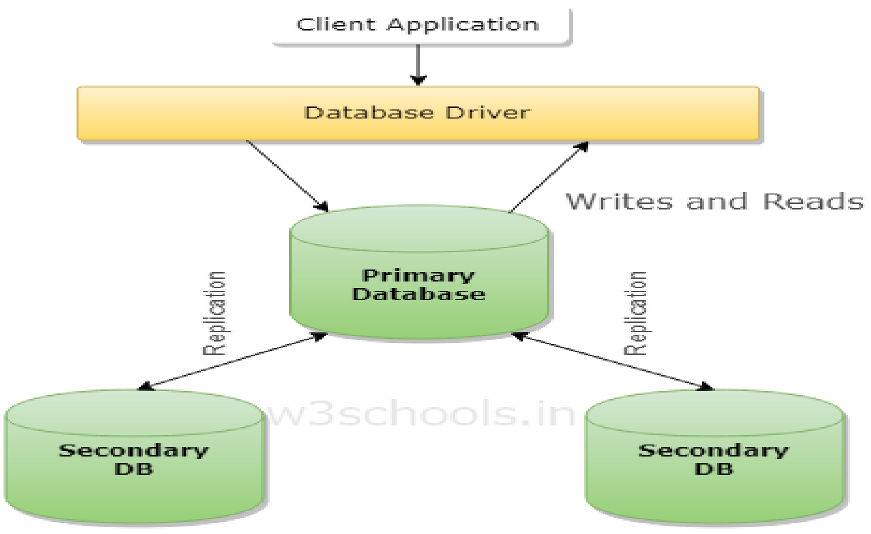
$unwind – What $unwind does is allow you to peel off a document for each element and returns that resulting document. To think of this in a classical approach, it would be the equivilent of "for each item in the tags array, return a document with only that item".

```
db.user.aggregate(
  { $project : {
      author : 1 ,
      title : 1 ,
      tags : 1
  }},
  { $unwind : "$tags" }
);
```

o/p:

```
{ "_id" : 1, "title" : "this is my title", "author" : "bob", "tags" : "fun" }
{ "_id" : 2, "title" : "this is my title", "author" : "bob", "tags" : "good" }
{ "_id" : 3, "title" : "this is my title", "author" : "bob", "tags" : "fun" }
```

9.Replication:



The Replication in MongoDB

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

Why Replication?
To keep your data safe
High (24*7) availability of data
Disaster recovery
No downtime for maintenance (like backups, index rebuilds, compaction)
Read scaling (extra copies to read from)
Replica set is transparent to the application

How Replication Works in MongoDB :
MongoDB achieves replication by the use of replica set. A replica set is a group of mongod instances that host the same data set. In a replica, one node is primary node that receives all write operations. All other instances, such as secondaries, apply operations from the primary so that they have the same data set. Replica set can have only one primary node.

Replica set is a group of two or more nodes (generally minimum 3 nodes are required).

In a replica set, one node is primary node and remaining nodes are secondary.

All data replicates from primary to secondary node.

At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.

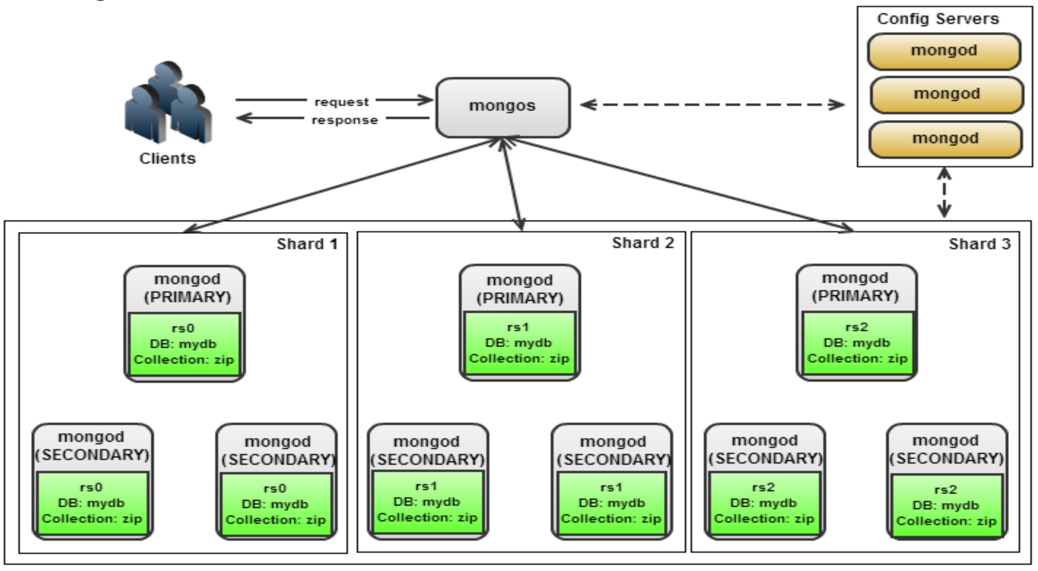
After the recovery of failed node, it again join the replica set and works as a secondary node.

Replica Set Features :

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

Set Up a Replica Set :
In this tutorial, we will convert standalone MongoDB instance to a replica set. To convert to replica set, following are the steps –

10.Sharding :

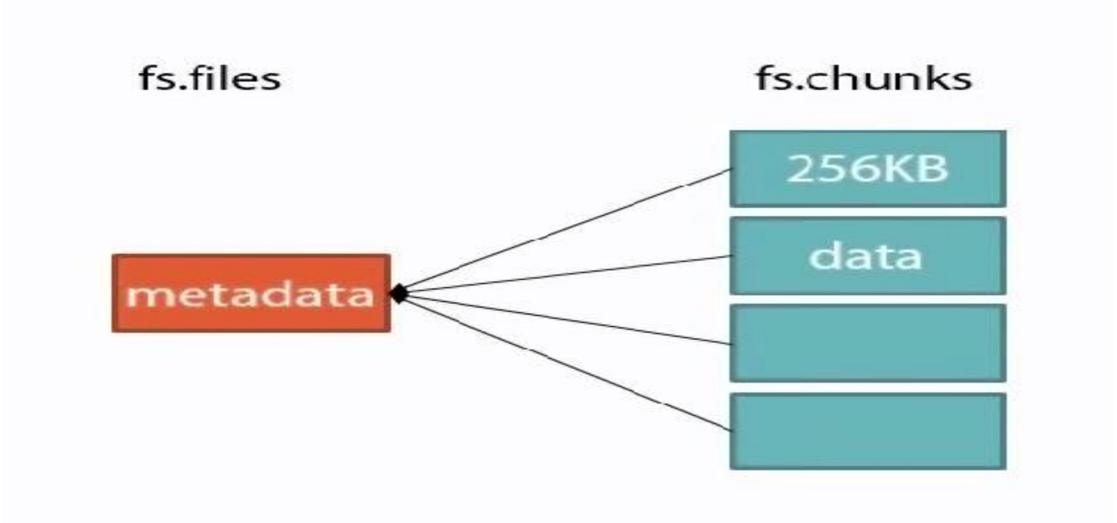

Config Servers
mongod
mongod
mongod
Clients
request
response
mongos
Shard 1
mongod (PRIMARY)
rs0 DB: mydb Collection: zip
mongod (SECONDARY)
rs0 DB: mydb Collection: zip
mongod (SECONDARY)
rs0 DB: mydb Collection: zip
Shard 2
mongod (PRIMARY)
rs1 DB: mydb Collection: zip
mongod (SECONDARY)
rs1 DB: mydb Collection: zip
mongod (SECONDARY)
rs1 DB: mydb Collection: zip
Shard 3
mongod (PRIMARY)
rs2 DB: mydb Collection: zip
mongod (SECONDARY)
rs2 DB: mydb Collection: zip
mongod (SECONDARY)
rs2 DB: mydb Collection: zip

Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth. As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves the problem with horizontal scaling. With sharding, you add more machines to support data growth and the demands of read and write operations.

Why Sharding?

- In replication, all writes go to master node
- Latency sensitive queries still go to master
- Single replica set has limitation of 12 nodes
- Memory can't be large enough when active dataset is big
- Local disk is not big enough
- Vertical scaling is too expensive

11.MongoDB - GridFS :



GridFS is the MongoDB specification for storing and retrieving large files such as images, audio files, video files, etc. It is kind of a file system to store files but its data is stored within MongoDB collections. GridFS has the capability to store files even greater than its document size limit of 16MB.

GridFS divides a file into chunks and stores each chunk of data in a separate document, each of maximum size 255k.

GridFS by default uses two collections **fs.files** and **fs.chunks** to store the file's metadata and the chunks. Each chunk is identified by its unique _id ObjectId field. The fs.files serves as a parent document. The files_id field in the fs.chunks document links the chunk to its parent.

Adding Files to GridFS :
Open your command prompt, navigate to the mongofiles.exe in the bin folder of MongoDB installation folder and type the following code (after put song.mp3 in same place) –

```
mongofiles.exe -d gridfs put song.mp3
```

gridfs is collection name and song.mp3 is file name.

To see the file's document in database, you can use find query
db.fs.files.find()

We can also see all the chunks present in fs.chunks collection related to the stored file with the following code, using the document id returned in the previous query

```
> db.fs.chunks.find({files_id:ObjectId("5ef6621233e86dcb54461e54")})
> db.fs.chunks.find({},{_id:0,data:0})
```

Data is stored in json like syntax

BSON(Binary JSON) Data Formate:

```
db.customers([{
  name:'Max',
  age:29,
  address:{
      city:'Munich'
  },
  hobbies:[
      {Name:'Cooking'},
      {name:'Sports'}
  ]
}
]);
```

12. Chaining
db.customers.find().limit(2).sort({ title: 1 }).pretty()

Foreach

```
 db.customers.find().forEach(doc=>{

  print('name',doc.First_Name)
  });
```

13. Rename Field

db.posts.update({ title: 'Post Two' },{ $rename: { likes: 'views' }});

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

```
db.customers.save(
  {"_id" : ObjectId(5ee08dc1e206f48220a3b08c), "first_name":"Tutorials Point N
ew Topic","by":"mongodb"}
);
```

14.1. By default, MongoDB will update only a single document. To update multiple documents, you need to set a *parameter* ***multi: true.***

```
 db.customers.update({first_name:"mukesh"},
   {$set:{first_name:"Mongodb"}},{multi:true})
```

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

Following example updates the age and email values of the document with name 'Radhika'.

```
db.customers.updateOne(
  {first_name: 'ram'},
  { $set: { Age: '30',e_mail: 'radhika_newemail@gmail.com'}}
);
```

14.3. MongoDB updateOne() method: This methods updates a single document which matches the given filter.

```
db.customers.updateOne(
  {first_name: 'mukesh'},
  { $set: { Age: '30',e_mail: 'radhika_newemail@gmail.com'}}
);
```

14.4. MongoDB updateMany() :
The customers() method updates all the documents that matches the given filter.

```
db.customers.updateMany({Age:{ $gt: "25" }},
{ $set: { Age: '00'}}
);
```

14.5.Replace documents

```
   db.customers.update({first_name:"John"},{$set:{gender:"femail"}});
```

15.increments numeric value is first

```
db.customers.update({first_name:"John"},{$set:{age:45}});
db.customers.update({first_name:"John"},{$inc:{age:5}});
```

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

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection.

```
  > db.store.remove({})
```

16.2. If there are multiple records and you want to delete only the first record, then set justOne parameter in remove() method.

```
> db.store.remove({_id:1},{justOne:true});
```

17.Update documents

```
> db.store.update({_id:1},{$set:{name:['python','javascript']}})
```

17.1.Update something is not in collections

```
> db.store.update({_id:1},{$set:{title:'oops'}})
```

18.Queries

```
db.customers.remove({first_name:"Mery"});
db.customers.find({name:"kitchen"});
db.customers.find({$or:[{first_name:"Mukesh"},{first_name:"Troy}]});
```

18.1.Find under the age of 40:

```
db.customers.find({age:{$lt:40}});
db.customers.deleteOne({first_name:"sherif"});
db.customers.deleteMany({first_name:"sherif"});
```

18.2.

```
db.customers.find({ age: { $gt: 40 }});           // gt or gte
db.customers.find({ age: { $lt: 40 }});           // lt or lte
db.customers.find().sort({ field: 1 });   // ascending order is indicated by 1
```

```
18.3. > db.store.aggregate([{$match:{_id:5}},{$match:{'Scores':{$gt:90}}}]);
```

19. projection :
In MongoDB, projection means selecting only the necessary data rather than selecting whole of the data of a document.
If a document has 5 fields and you need to show only 3, then select only 3 fields from them.
when you execute find() method, then it displays all fields of a document. To limit this, you need to set a list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the fields.

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

Capped collections restrict updates to the documents if the update results in increased document size. Since capped collections store documents in the order of the disk storage, it ensures that the document size does not increase the size allocated on the disk. Capped collections are best for storing log information, cache data, or any other high volume data.

db.createCollection("student", { capped : true, autoIndexID : true, size : 5242880, max : 5000 });

This will create a collection named student, with maximum size of 5 megabytes and maximum of 5000 documents.

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

- Indexes improve MongoDB query excution
- Without index whole collextion must be scanned (COLLSCAN)
- Index stores sorted field values
- If appropriate index exists, MongoDB performs only index scan (IXSCAN)

Indexes are special data structures, that store a small portion of the data set in an easy-to-traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

db.indexing.find();
```
{
  "_id" : ObjectId("5ef6f997c5897e0cab0063c1"),
  "title" : "Mistborn",
  "year" : 2006,
  "author" : {
          "firstname" : "Brandon",
          "lastname" : "Sanderson"
  }
}
```

21.1.Create Index in collection: To create an index you need to use **ensureIndex()** method.
We can select any field to create index from collection.
```
db.indexing.ensureIndex({year:1})
```

In ensureIndex() method you can pass multiple fields, to create index on multiple fields.
```
db.indexing.ensureIndex({first_name:1, last_name:-1});
```

The getIndexes() method method returns the description of all the indexes in the collection.

```
db.indexing.getIndexes()
```

The dropIndex() method:

```
db.indexing.dropIndex({year:1})
```