Sharding

Sharding is a technique used for distributing data across multiple servers.

MongoDB supports Horizontal scaling by sharding.

Mongo leverages Sharding for splitting up of a large collection among multiple servers.

* Ex:- ets and high throughput operations through this.
* Ex:- all aadhar card data will not be stored in same table , based on sharding key data will be in multiple nodes , like 1st 1cr data in a mongo cluster and 2nd crores data in another mongo cluster like 100 crores mongo data would have stored in 100 diff clusters
* **Capped collections**.

**Like max 2mb** – we can define the table size , if there are too many data , older will be deleted, it means at any time , table will hold max 2mb data

##### Replication

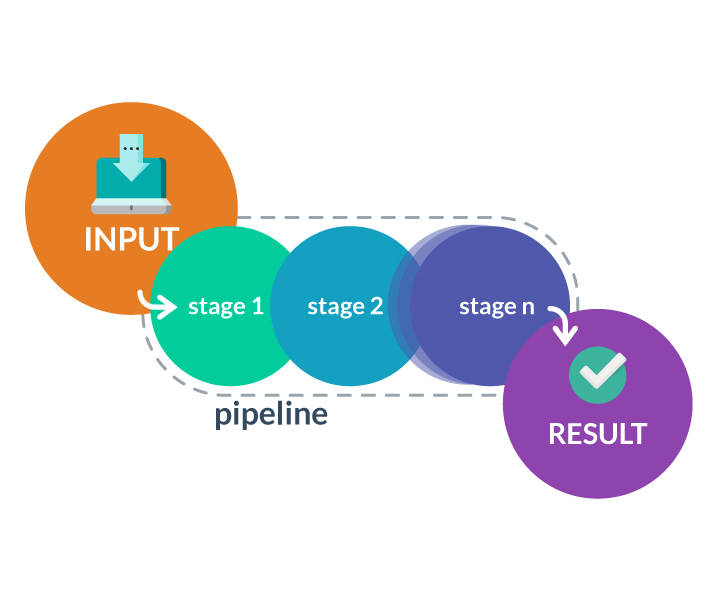
MongoDB uses **replica sets** for high availability.

Replica sets contain two or more copies of the data. Each replica set may act as a primary or secondary replica set. By default, read and write operations are performed on the primary replica.

The secondary replica will maintain a copy of primary data.

## Aggregation

In MongoDB, aggregation process records and return computed results.



**Aggregation** operations are very important for SQL and NO SQL databases. In MongoDB, collections of documents are taken as inputs and return aggregated results in the form of documents, cursor or a collection.

**There are three ways to perform aggregation in MongoDB.**

* Aggregation Pipeline
* Map-Reduce
* Single Purpose Aggregation Operations.

Aggregation can be categorized as :

Pipeline Aggregation: Documents are piped through processing pipeline and executes in different stages and transforms the documents into a final aggregated result.

Map-Reduce: It splits a larger problem into smaller chunks and sends to different machines for processing. It comprises two phasesWhen all the machines operations are finished, all the chunks outputs will be combined together to generate a final solution: reduce and map.

* **Map**: Documents are processed and emit one or more objects for each input document.
* **Reduce**: Combines the output generated by map operation.
* **Finalize**: It is optional in nature. It can be used to make a final modification to results.

Single Purpose

These operations will aggregate documents from a single collection.

About mongo db

A document is a set of ***key-value*** pairs that support dynamic schema. A document is similar to Row in RDBMS. In Relational databases, schemas should be defined before we add any data whereas MongoDB allows the insertion of data without a predefined schema.

Dynamic schema implies that the documents stored in the database can have different fields, with different types for each field.

Ex:- The first document consists of fields "city" and "Date of joining" that are not present in the second document. The second document consists of fields "Occupation", "DOB" and "Salary" that are not present in the first one.

db.employee.insert({Name: 'Abc12',

City: 'Delhi', DOJ : '12/01/1980'})

db.employee.insert({Name: 'Abc123',

Occupation: 'Software Engineer',

DOB : '29/07/1988',Salary:'250000'})

A collection consists of a group of MongoDB documents. It is similar to RDBMS table. Documents inside a collection can have same or different fields.

Indexes

In MongoDB, indexes support **fast and efficient execution of queries**.Generally we create indexes on most used column.

In the absence of indexes, MongoDB will scan every document in a collection to select those documents that match the query statement. If an index exists, the number of documents to inspect can be restricted.

How to store records having more than 16 mb

##### Architecture - GridFs

**GridFs** helps us to store data that has a size of more than 16MB.

When a document exceeds the size of 16MB, MongoDB uses **GridFS** specification. It splits a larger record into small ***chunks*** and stores the chunks in documents with a maximum size of 255KB.

Mongo commands

Show switch toa db

The command to check databases in MongoDB Server:

show dbs

# to switch to a database type use command

Use db1

Different date methods are Date(), newDate(), and ISODate().

var datevalue = Date()

sample insertion

db.product.insert(

{Product:"orange", quantity:5

,datevalue});

db.customer.insert ({name:"abc",age:ageint,salary:salint});

### inserting timestamp

var t = new Timestamp()

db.stock.insert({name :"orange",qnty:500,stamp:t});

### inserting array data types

var myProducts =

[

{

"Productid" : 1,

"ProductName" : "Pen"

},

{

"Productid" : 2,

"ProductName" : "Pencil"

},

{

"Productid" : 3,

"ProductName" : "Book"

},

];

db.prod.insert(myProducts);

### Inserting a document

##### Insert vs insert many

You can use insert() or save() method to Insert a document or documents into a collection. From version 3.2, you can use insertmany() to insert multiple documents.

***Syntax:***

db.Collection.insert(document)

db.topic.insert({title: 'MongoDB', desc: 'MongoDB is doc oreinted db'})

db.Collection.insertmany(document1,document2...)

db.topic.insert({

title: 'MongoDB',

desc: 'MongoDB is document store',

tags: ['MongoDB','NoSQL database'],

})

##another example

db.custorders.insert({

customer\_ID:'A2',

Desc:'Orange',

SalesAmount:5000,

Status:'Active'

}

)

#Inserting arrays another example

db.bookshop.insert([

{

title: 'Mongodb',

desc:'Mongodb is a nosql database',

creator:'Sydney team',

tags:['mongodb','database','NosqlDB'],

user:500

}

])

### insertMany

try {

db.school.insertMany( [

{ "\_id" :"1", "Fullname" :"Mridhula", "subject" : "Science", "score" : 680 },

{ "\_id" : "2", "Fullname" : "Mridhula", "subject" : "Mathematics", "score" : 980 }

]);

Array insertion

db.bookshop.insert([{title:'MongoDb',desc:'Its a no sql databse', creator :'sydney team',tags:['mongodb','database'],user:500}])

db.Products.save({ \_id: 3, Fruitname: "WaterMelon", Retailer: [ {ProviderName: "Hypercity", ProviderKey: "1445345"} ] });

### update example

db.project.update(

{"Job No":"2"},

{$set :{"Level": "Intermediate"}}

);

## Mongo commands

### Between

Always if many queries are there use array to hold multiple conditions

db.applications.find( {"Reviews":{ $gte:100000, $lt:150000 },"Type":"Paid"})

### Count

Db.purchase\_orders.count({product:”tooth\_brush”});

db.applications.count({"Content Rating":"Teen"});

### distinct

it will return the unique values based on that column

db.customers.distinct("Customer\_ID")

Only distinct records of Customer ID get displayed.

Find () Data from the database:

We can use find() method to display documents in a collection.

* find() returns output as cursor. MongoDB server return queries in batches. Batch size will not surpass the maximum Bson document size.

**Printing compactly in single line**

db.school.find({});

db.totalscores.find();

u can use this instead if pretty;pretty will print in multiple lines

**Syntax:**

db.CollectionName.find()

db.applications.find({Size:"100M"});

db.Player.find( { score: 10340} )

db.Player.find( { score: { $gt: 10000 } } )

db.school.find( { score: { $gt: 670 }, subject: "sports" } )

db.school.find( { score: { $gt: 670 } } ) ;

with 2 conditions

db.hdfstomongo.find(

{

PW\_WAGE\_LEVEL :'Level I',

"PREVAILING\_WAGE":{ $gte:80000, $lt:90000 }

}

);

db.applications.find( {"Reviews":{ $gte:100000, $lt:150000 },"Type":"Paid"})

db.project.find({"Estimated Time": "More than 6 month"});

Now you have to find out details of students whose subjects is sports and score greater than 670

db.school.find( { score: { $gt: 670 }, subject: "sports" } )

***Example:***

db.topic.find()

#### Finiding in array elements

db.Products.save({

 \_id: 5,

 Fruitname: "Plum",

 Retailer: [

  {ProviderName: "DMART", ProviderKey: "13443"},

  {ProviderName: "KMART", ProviderKey: "768"}

 ]

});

`db.Products.find({ "Retailer.ProviderKey": "1232",  "Retailer.ProviderName": "FutureRetail" }).explain("executionStats");`

### Limit

If u want to find only ½ documents then use this limit functions

db.project.find({}).sort({"Posted":1}).limit(1);

in above we have sorted by field posted and we want only 1 record. So used limit

### pretty- Displaying in a formatted way

We can use pretty() to display records in a formatted way.

***Syntax:***

db.collection.find().pretty()

### limit- to return only certain number of records

db.collection.find().limit(number)

db.collection.findOne({})

methods

db.collectionname.remove()

db.collectionname.deleteMany()

db.topic.deleteOne({title : 'MongoDB'})

 updateOne(),updateMany(), and replaceOne()

switch to a database

Show dbs

use <Database Name>

### **show / List all collections**:

You use show collection to list all Collection in a database.

show collections

### Export-- the collection data

mongoexport-d <database>

-c <Collection name >

-o <Output file name >.json

***Example:***

mongoexport -d customer -c order -o student.json

### save / insert

db.Products.save({

 \_id: 1,

 Fruitname: "Mango",

 Retailer: [

  {ProviderName: "FutureRetail", ProviderKey: "1232"},

  {ProviderName: "More", ProviderKey: "1423"}

 ]

});

### import CSV into mongo

mongoimport -d<datatbasename>

-c <collectionname>

--file<filename>

***Run the mongoimport only in seperate terminal -- once u type "mongo" and clicked enter u can not run "mongoimport" command***

***because this is a seprate tool. so run this command on seperate terminal***

***Example:***

***The below example is perfectly working fine***

***mongoimport -d hadoop -c hdfstomongo --type csv --file dataset.csv --headerline***

**mongoimport -d sports -c olympic --type csv --file olympix\_data.csv --headerline**

**mongoimport -d mongo -c applications --type csv --file googleplaystore.csv --headerline**

**mongoimport -d mongo -c applications --type csv --file googleplaystore.csv –headerline**

***mongoimport -d mongo -c mongomini --type csv --file h1b\_dataset.csv --headerline***



mongoimport -d customers -c orders --file student.json

### Slice

db.book.find( {}, { comments: { $slice: [ -100, 5 ] } } )

* beginning with the last 100 items, returns 5 comments

### explain

this will explain the plan – means performance of the query..

db.collectionname.find().explain()

db.books.find({year:1936}).explain()

db.school.find( { score: { $gt: 670 } }).explain("executionStats")

### current Running operations

db.currentOp()--This method returns all in-progress operations on the database.

db.killOp()-you can terminate certain operation with the following command

### Creating Indexes

To see if any index is already created

`db.Products.find({ "Fruitname": "Plum" }).explain();`

If there are no indexes u will get collscan

"winningPlan" : { "stage" : "COLLSCAN"}

If index is present we will get as below

winningPlan" : {

                        "stage" : "FETCH",

                        "inputStage" : {

                                "stage" : "IXSCAN",

db.school.getIndexes(); // to fetch the current Indexes

db.school.createIndex( { score:1} );

#### Create Compound Indexes Create Compound Indexes on 2 columns

for the fields 'Score' and 'Subjects':

db.school.createIndex( { score: 1, subject: 1 } );

**Create Compound Indexes by changing the order Change the order of the Index:**

db.school.createIndex( { subject: 1 ,score: 1,} )

index order is also more important, while creating the compound indexes In Compound Indexes, the order of the fields affect the performance of the query.

In this example query, the Compound Index {subject: 1, score: 1} is more efficient than the compound index {score: 1, subject: 1}. Because as u created the index on sub

#### Creating indexes on array type fields

db.Products.save({

 \_id: 5,

 Fruitname: "Plum",

 Retailer: [

  {ProviderName: "DMART", ProviderKey: "13443"},

  {ProviderName: "KMART", ProviderKey: "768"}

 ]

});

Create index on array fields

=============================

`db.Products.createIndex({ "Retailer.ProviderKey": 1,  "Retailer.ProviderName": 1 }); `

Diagnostic tools

* mongotop
* mongostat
* mongosniff
* mongoperf

### mongostat:

Utility provides a quick summary of a currently running instance and returns the counters of database operation. Counters consist of inserts, updates, deletes, cursors, and queries. It helps to troubleshoot performance issues. Example:

mongostat

### mongotop:

It helps to track the amount of time a MongoDB instance spends on writing and reading data. [mongotop](https://www.mongodb.com/docs/database-tools/mongotop/#mongodb-binary-bin.mongotop) provides statistics on a per-collection level. By default, [mongotop](https://www.mongodb.com/docs/database-tools/mongotop/" \l "mongodb-binary-bin.mongotop" \t "_self) returns values every second.

**Example:** To return every 30 seconds

mongotop 30

### mongoperf**:**

**It is a tool for quickly testing disk I/O performance. It accepts configuration options in the form of a file that holds a JSON document.**

### mongosniff**:**

**It helps to investigate MongoDB database activity. From version 3.4, it is replaced with mongoreplay. You can reproduce and investigate issues by recording and replaying the operations that trigger an issue.**

mongo:

**Powerful interface for interactive JavaScript shell. It helps system administrators, developers to test queries and operations directly with the database.**

mongod:

**Primary process for MongoDB. This will manage data requests, control data access, and performs other background management operations.**

**Miscellaneous topics**

MongoDB supports different storage engines:

* MMAPv1
* WiredTiger
* In-Memory Storage Engine