



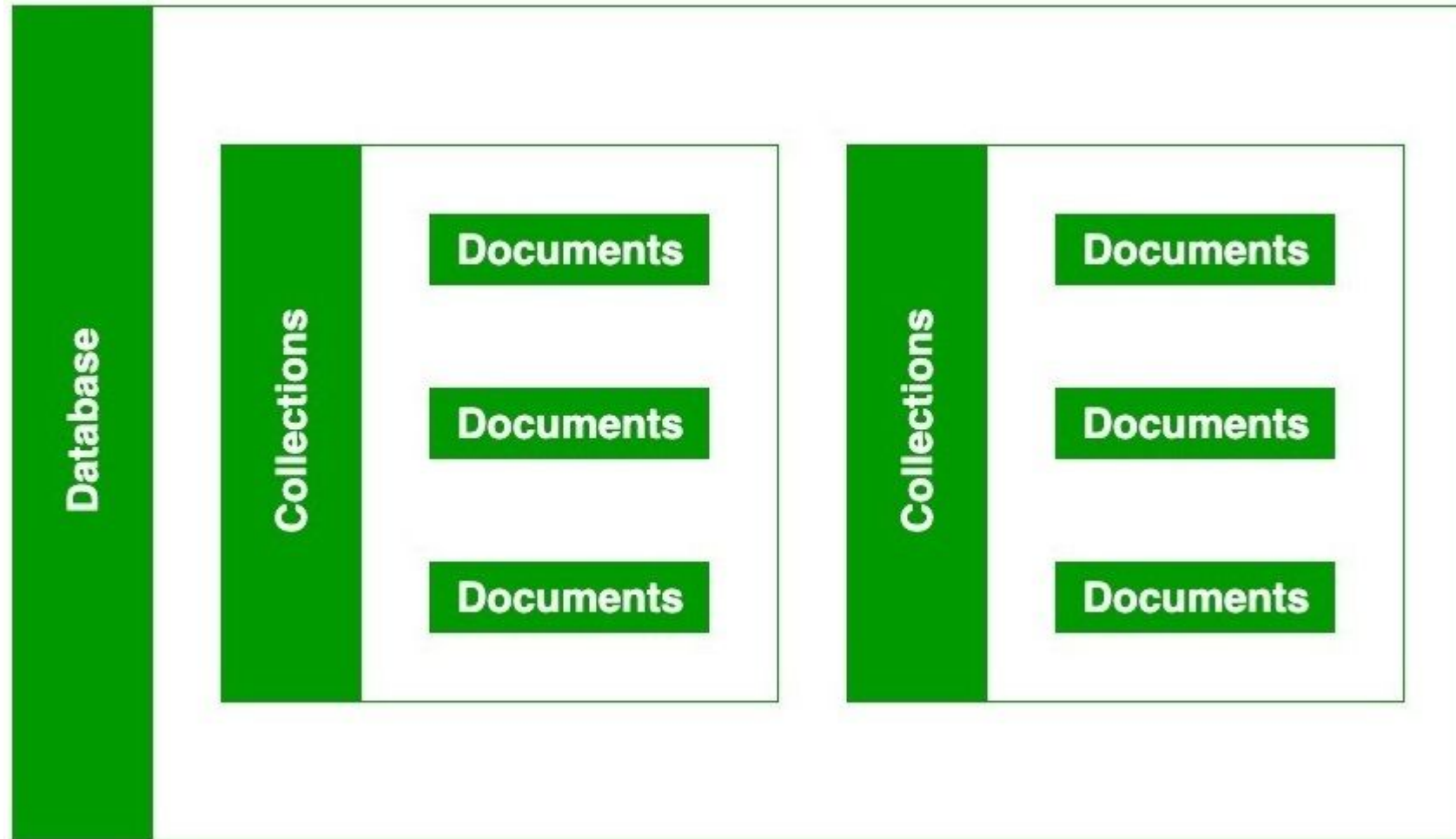
Explore | Expand | Enrich



<Codemithra />TM

- MongoDB is an open-source document-oriented database that is designed to store a large scale of data and also allows you to work with that data very efficiently.
- It is categorized under the NoSQL (Not only SQL) database because the storage and retrieval of data in the MongoDB are not in the form of tables.
- The MongoDB database is developed and managed by MongoDB.Inc under SSPL(Server Side Public License) and initially released in February 2009.
- It also provides official driver support for all the popular languages like C, C++, C#, and .Net, Go, Java, Node.js, Perl, PHP, Python, Motor, Ruby, Scala, Swift, Mongoid. So, that you can create an application using any of these languages.
- Nowadays there are so many companies that used MongoDB like Facebook, Nokia, eBay, Adobe, Google, etc. to store their large amount of data.





- There is no create database command in MongoDB. Actually, MongoDB do not provide any command to create database.
- MongoDB you don't need to create a database manually because MongoDB will create it automatically when you save the value into the defined collection at first time.
- You also don't need to mention what you want to create, it will be automatically created at the time you save the value into the defined collection.

Syntax:

use DATABASE_NAME

To check the currently selected database, use the command db: **>db**

To check the database list, use the command show dbs: **>show dbs**



- The dropDatabase command is used to drop a database. It also deletes the associated data files. It operates on the current database

- **Syntax:**

```
db.dropDatabase()
```



- In MongoDB, `db.createCollection(name, options)` is used to create collection.
- But usually you don't need to create collection. MongoDB creates collection automatically when you insert some documents.

- **Syntax:**

`db.createCollection(name, options)`

- **Name:** is a string type, specifies the name of the collection to be created
- **Options:** is a document type, specifies the memory size and indexing of the collection. It is an optional parameter.



| Field | Type | Description |
|-------------|---------|---|
| Capped | Boolean | (Optional) If it is set to 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 | (Optional) If it is set to true, automatically create index on ID field. Its default value is false. |
| Size | Number | (Optional) It specifies a maximum size in bytes for a capped collection. If capped is true, then you need to specify this field also. |
| Max | Number | (Optional) It specifies the maximum number of documents allowed in the capped collection. |

- In MongoDB, `db.collection.drop()` method is used to drop a collection from a database.
- It completely removes a collection from the database and does not leave any indexes associated with the dropped collections.
- The `db.collection.drop()` method does not take any argument and produce an error when it is called with an argument.
- **Syntax:**

```
sdb.COLLECTION_NAME.drop()
```



- The create or insert operations are used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database.
- You can perform, create operations using the following methods provided by the MongoDB:
db.collection.insertOne(): It is used to insert a single document in the collection.
- **db.collection.insertMany():** It is used to insert multiple documents in the collection.



Example

We are inserting details of a single student in the form of a document using the `db.collection.InsertOne()` method:

```
> use ethnus;
switched to db ethnus
> db.student.insertOne({
... name:"Arun",
... age:20,
... branch:"ISE",
... course:"Java AWS",
... mode:"offline",
... paid:true,
... amount:45000
... })
{
  "acknowledged" : true,
  "insertedId" : ObjectId("622af1459f263e3785d3559d")
}
```

Example

We are inserting details of the multiple students in the form of documents in the student collection using `db.collection.insertMany()` method.

```
> db.student.insertMany([  
... {  
... name : "Vijay",  
... age : 19,  
... branch : "CSE",  
... course : "Embedded System",  
... mode : "online",  
... paid : true,  
... amount : 25000  
... },
```

```
...  
... {  
... name : "Rahul",  
... age : 22,  
... branch : "Mechanical",  
... course : "Aptitude",  
... mode : "online",  
... paid : true,  
... amount : 25000  
... }  
...  
... ])
```



Read Operations

The Read operations are used to retrieve documents from the collection, or in other words, read operations are used to query a collection for a document.

You can perform read operation using the following method provided by the MongoDB:

db.collection.find(): It is used to retrieve documents from the collection.

To retrieve the details of the student we are using the following method:

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



- The update operations are used to update or modify the existing document in the collection. You can perform update operations using the following methods provided by the MongoDB:
- **db.collection.updateOne():** It is used to update a single document in the collection that satisfy the given criteria.
- **db.collection.updateMany():** It is used to update multiple documents in the collection that satisfy the given criteria.



Example

In this example we are updating the detail of one student by setting the age:

```
>db.student.updateOne({name: "Gnanamurthy"},{$set:{age: 24}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```



Delete Operations

- The delete operation are used to delete or remove the documents from a collection.
- You can perform delete operations using the following methods provided by the MongoDB:
- **db.collection.deleteOne():**It is used to delete a single document from the collection that satisfy the given criteria.
- **db.collection.deleteMany():**It is used to delete multiple documents from the collection that satisfy the given criteria.



Example

We are deleting the document from the student collection using **db.collection.deleteOne()** method:

First we retrieve all the elements of the student collection using **db.collection.find().pretty()** method:

```
> db.student.find().pretty()
{
  "_id" : ObjectId("622af1459f263e3785d3559d"),
  "name" : "Arun",
  "age" : 24,
  "branch" : "ISE",
  "course" : "Java AWS",
  "mode" : "offline",
  "paid" : true,
  "amount" : 25000
}
```

Then we use the deleteOne() method:

```
> db.student.deleteOne({name: "Gnanamurthy"})
{ "acknowledged" : true, "deletedCount" : 1 }
```

We are deleting the document from the student collection using **db.collection.deleteOne()** method:

First we retrieve all the elements of the student collection using **db.collection.find().pretty()** method:

```
> db.student.find().pretty()
{
  "_id" : ObjectId("622af1459f263e3785d3559d"),
  "name" : "Arun",
  "age" : 24,
  "branch" : "ISE",
  "course" : "Java AWS",
  "mode" : "offline",
  "paid" : true,
  "amount" : 25000
}
```


- **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.



MongoDB insert document

The insert() Method

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

Syntax

The basic syntax of insert() command is as follows –

```
>db.COLLECTION_NAME.insert(document)
```

```
> db.codemithra.insert({  
... _id : ObjectId("507f191e810c19729de860ea"),  
... title:"MongoDb Overview",  
... description:"MongoDb is a nosql database",  
... by:"ethnus code mithra",  
... url: "http://www.codemithra.com",  
... tags: ['mongo','database','NoSQL'],  
... likes:100  
... })
```



insertOne() method

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

Syntax

The basic syntax of insert() command is as follows –

```
>db.COLLECTION_NAME.insertOne(document)
```

First create a collection called empDetails-

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

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

```
> db.empDetails.insertOne(
```

```
... {
```

```
... First_Name: "Pooja",
```

```
... Last_Name: "Somasundar",
```

```
... Date_Of_Birth: "1987-12-02",
```

```
... e_mail: "psomasundar123@gmail.com",
```

```
... phone: "9448048461"
```

```
... })
```

```
{ "acknowledged" : true,
```

```
  "insertedId" : ObjectId("622ee89598f8c1bb739a294a")
```

```
}
```

insertMany() method

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

```
> db.empDetails.insertMany(  
... [  
... {  
... First_Name: "Radhika",  
... Last_Name: "Sharma",  
... Date_Of_Birth: "1995-09-26",  
... e_mail: "radhika_sharma.123@gmail.com",  
... phone: "9000012345"  
... },  
... {  
... First_Name: "Rachel",  
... Last_Name: "Christopher",  
... Date_Of_Birth: "1990-02-16",  
... e_mail: "Rachel_Christopher.123@gmail.com",  
... phone: "9000054321"  
... },  
... {
```



insertMany() method

```
... First_Name: "Fathima",  
... Last_Name: "Sheik",  
... Date_Of_Birth: "1990-02-16",  
... e_mail: "Fathima_Sheik.123@gmail.com",  
... phone: "9000054321"  
... }  
... ]  
... )
```



The find() Method

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

Syntax

The basic syntax of find() method is as follows –

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

Create a database and a collection:

```
> use sampleDB
```

```
switched to db sampleDB
```

-

```
db.createCollection("mycol")
```



Query Document

And inserted 3 documents using the insert method-

```
> db.mycol.insert([
... {
... title: "MongoDB Overview",
... description: "MongoDB is no SQL database",
... by: "codemithra",
... url: "http://www.codemithra.com",
... tags: ["mongodb", "database", "NoSQL"],
... likes: 100
... },
... {
... title: "NoSQL Database",
... description: "NoSQL database doesn't have tables",
... by: "code mithra",
... url: "http://www.codemithra.com",
... tags: ["mongodb", "database", "NoSQL"],
... likes: 20,
... comments: [
```

Query Document

```
... {  
... user:"user1",  
... message: "My first comment",  
... dateCreated: new Date(2013,11,10,2,35),  
... like: 0  
... }  
... ]  
... }  
... ])
```

Following method retrieves all the documents in a collection-

- `db.mycol.find()`



Query Document

The pretty() Method

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

Syntax

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

Example:

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

The findOne() method

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

Syntax:

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

Example:

```
> db.mycol.findOne({title: "MongoDB Overview"})
```



What is Aggregation?

- In MongoDB, aggregation operations process the data records/documents and return computed results.
- It collects values from various documents and groups them together and then performs different types of operations on that grouped data like sum, average, minimum, maximum, etc to return a computed result.
- It is similar to the aggregate function of SQL.
- MongoDB provides three ways to perform aggregation:
 - Aggregation pipeline
 - Map-reduce function
 - Single-purpose aggregation



Aggregate() method

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

Syntax

Basic syntax of aggregate() method is as follows –

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```



Aggregation Expressions

- **\$sum**: Sums up the defined value from all documents in the collection.
Eg: `db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])`
- **\$avg**: Calculates the average of all given values from all documents in the collection. Eg: `db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])`
- **\$min**: Gets the minimum of the corresponding values from all documents in the collection. Eg: `db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])`
- **\$max**: Gets the maximum of the corresponding values from all documents in the collection. Eg: `db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])`
- **\$push**: Inserts the value to an array in the resulting document. Eg: `db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push : "$url"}}}])`
- **\$addToSet**: Inserts the value to an array in the resulting document but does not create duplicates. Eg: `db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])`

- **\$first**: Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “\$sort”-stage.
- `db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])`
- **\$last**: Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “\$sort”-stage.
- `db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])`

- MongoDB uses indexing in order to make the query processing more efficient.
- If there is no indexing, then the MongoDB must scan every document in the collection and retrieve only those documents that match the query.
- Indexes are special data structures that stores some information related to the documents such that it becomes easy for MongoDB to find the right data file.
- The indexes are order by the value of the field specified in the index.



Create an Index

MongoDB provides a method called `createIndex()` that allows user to create an index.

Syntax –

```
db.COLLECTION_NAME.createIndex({KEY:1})
```

The key determines the field on the basis of which you want to create an index and 1 (or -1) determines the order in which these indexes will be arranged(ascending or descending).



Example

```
> db.empDetails.createIndex({"phone":9000054321})
{
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "createdCollectionAutomatically" : false,
  "ok" : 1
}
```

The createIndex() method also has a number of optional parameters.

These include:

- background (Boolean)
- unique (Boolean)
- name (string)
- sparse (Boolean)



- In order to drop an index, MongoDB provides the `dropIndex()` method.

Syntax –

- `db.NAME_OF_COLLECTION.dropIndex({KEY:1})`
- The `dropIndex()` methods can only delete one index at a time.
- In order to delete (or drop) multiple indexes from the collection, MongoDB provides the `dropIndexes()` method that takes multiple indexes as its parameters.

Syntax –

- `db.NAME_OF_COLLECTION.dropIndexes({KEY1:1, KEY2, 1})`



Example

```
> db.empDetails.dropIndex({"phone":9000054321})  
{ "nIndexesWas" : 3, "ok" : 1 }  
  
> db.empDetails.dropIndexes({"phone":9000012345})  
{ "nIndexesWas" : 2, "ok" : 1 }
```

Different types of Indexes

MongoDB provides different types of indexes that are used according to the data type or queries. The indexes supported by MongoDB is are as follows:

- 1.Single field Index
- 2.Compound Index
- 3.Multikey Index
- 4.Geospatial Indexes
- 5.Text Index
- 6.Hash Index
- 7.Wildcard Index



Replication

- 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?

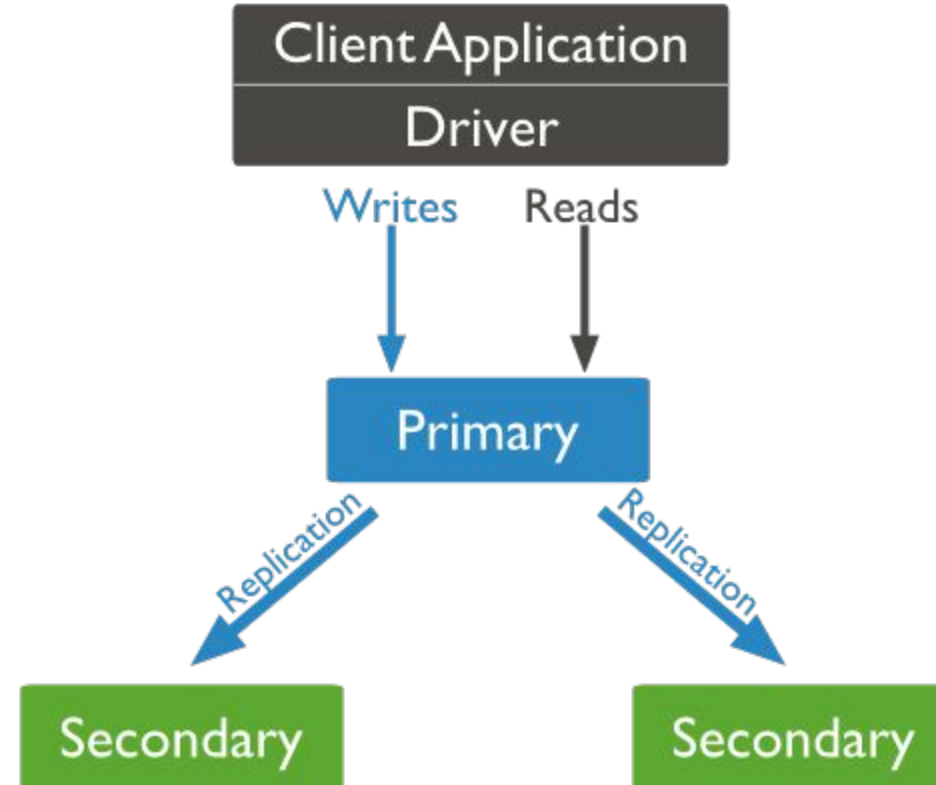
- To keep your data safe
- High (24*7) availability of data
- Disaster recovery
- No downtime for maintenance (like backups, index rebuilds, compaction)
- Read scaling (extra copies to read from)
- Replica set is transparent to the application

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.



Diagram of Replica Set



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

We will convert standalone MongoDB instance to a replica set. To convert to replica set, following are the steps –

Shutdown already running MongoDB server.

Start the MongoDB server by specifying --replSet option. Following is the basic syntax of --replSet –

```
mongod --port "PORT" --dbpath "YOUR_DB_DATA_PATH" --replSet "REPLICA_SET_INSTANCE_NAME."
```

Example


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

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

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

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

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



To add members to replica set, start mongod instances on multiple machines. Now start a mongo client and issue a command `rs.add()`.

Syntax

The basic syntax of `rs.add()` command is as follows –

```
>rs.add(HOST_NAME:PORT)
```

Example

Suppose your mongod instance name is `mongod1.net` and it is running on port 27017. To add this instance to replica set, issue the command `rs.add()` in Mongo client.

```
>rs.add("mongod1.net:27017")
```

```
>
```

You can add mongod instance to replica set only when you are connected to primary node. To check whether you are connected to primary or not, issue the command `db.isMaster()` in mongo client.



- 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.

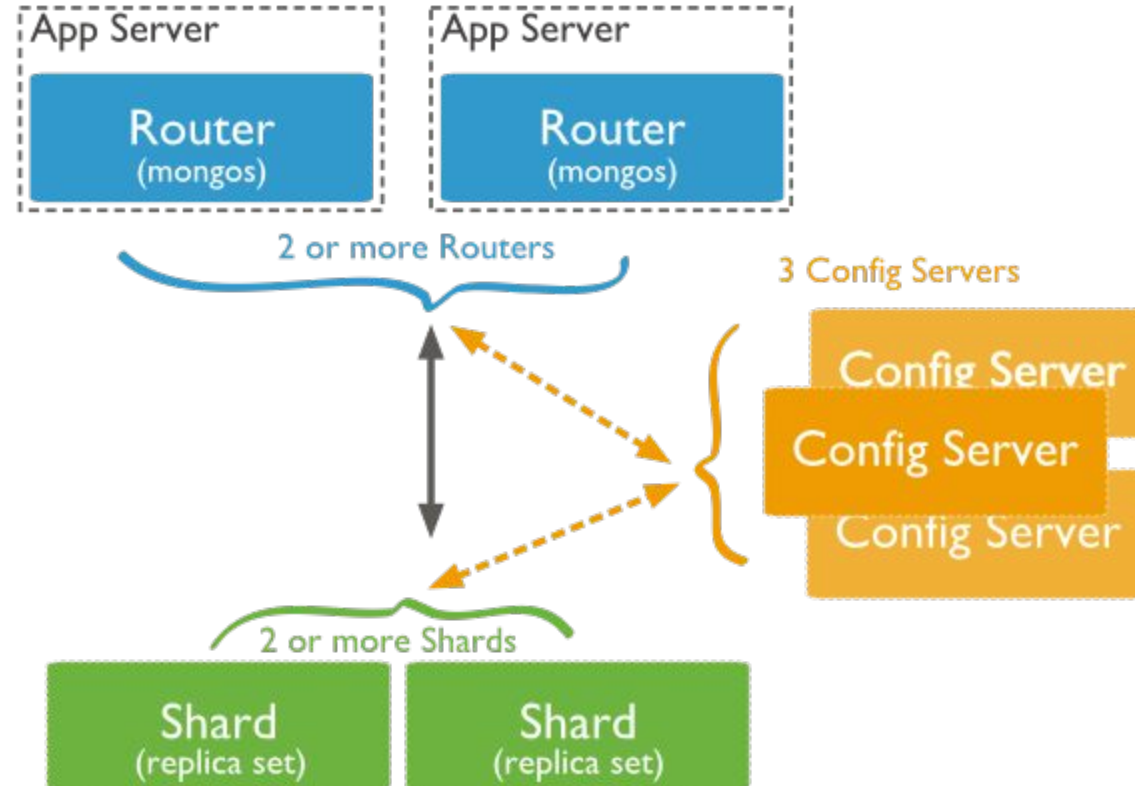


Why Sharding?

- 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



Sharding in MongoDB



In the following diagram, there are three main components –

Shards – Shards are used to store data.

They provide high availability and data consistency.

In production environment, each shard is a separate replica set.

Config Servers – Config servers store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards. The query router uses this metadata to target operations to specific shards. In production environment, sharded clusters have exactly 3 config servers.

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



Sharding Types

- There are four main kinds:
- ranged/dynamic sharding
- algorithmic/hashed sharding
- entity/relationship-based sharding
- geography-based sharding.

- By simply hashing a primary key value, most distributed databases randomly spray data across a cluster of nodes. This imposes performance penalties when data is queried across nodes and adds application complexity when data needs to be localized to a specific region.
- By exposing multiple sharding policies, MongoDB offers you a better approach.
- Data can be distributed according to query patterns or data placement requirements, giving you much higher scalability across a diverse set of workloads:
- **Ranged Sharding:** Documents are partitioned across shards according to the shard key value.
- Documents with shard key values close to one another are likely to be co-located on the same shard.
- This approach is well suited for applications that need to optimize range based queries, such as co-locating data for all customers in a specific region on a specific shard.
- **Hashed Sharding:** Documents are distributed according to an MD5 hash of the shard key value.
- This approach guarantees a uniform distribution of writes across shards, which is often optimal for ingesting streams of time-series and event data.
- **Zoned Sharding:** Provides the ability for developers to define specific rules governing data placement in a sharded cluster.



- 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.

The find() Method

MongoDB's find() method, explained in MongoDB Query Document accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB, when you execute find() method, then it displays all fields of a document. To 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.

Syntax

The basic syntax of find() method with projection is as follows –

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



Example

Consider the collection mycol has the following data –

```
{_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"},  
{_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"},  
{_id : ObjectId("507f191e810c19729de860e3"), title: "Ethnus courses overview"}
```

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

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

```
{"title":"MongoDB Overview"}
```

```
{"title":"NoSQL Overview"}
```

```
{"title":" Ethnus courses overview "}
```

```
>
```

- The Limit() Method
- To limit the records in MongoDB, you need to use limit() method. The method accepts one number type argument, which is the number of documents that you want to be displayed.
- Syntax
- The basic syntax of limit() method is as follows –
- `>db.COLLECTION_NAME.find().limit(NUMBER)`

Example

Consider the collection myycol has the following data.

```
{_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"},
```

```
{_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"},
```

```
{_id : ObjectId("507f191e810c19729de860e3"), title: "Tutorials Point Overview"}
```



Limit() method example

Following example will display only two documents while querying the document.

```
>db.mycol.find({},{"title":1,_id:0}).limit(2)
```

```
{"title":"MongoDB Overview"}
```

```
{"title":"NoSQL Overview"}
```

```
>
```

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



MongoDB Skip() Method

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

Syntax

The basic syntax of skip() method is as follows –

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

Example

Following example will display only the second document.

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

```
{"title":"NoSQL Overview"}
```

```
>
```



Sorting records

The sort() Method

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

Syntax

The basic syntax of sort() method is as follows –

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

Example

Consider the collection mycol has the following data.

```
{_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"}  
{_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"}  
{_id : ObjectId("507f191e810c19729de860e3"), title: "Tutorials Point Overview"}
```

Following example will display the documents sorted by title in the descending order.

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

Dump MongoDB Data

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

Syntax

The basic syntax of mongodump command is as follows –

```
>mongodump
```

Example

Start your mongod server. Assuming that your mongod server is running on the localhost and port 27017, open a command prompt and go to the bin directory of your mongodb instance and type the command mongodump

```
>mongodump
```

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



Following is a list of available options that can be used with the mongodump command.

| Syntax | Description | Example |
|--|---|---|
| mongodump --host HOST_NAME --port PORT_NUMBER | This commmand will backup all databases of specified mongod instance. | mongodump --host tutorialspoint.com --port 27017 |
| mongodump --dbpath DB_PATH --out BACKUP_DIRECTORY | This command will backup only specified database at specified path. | mongodump --dbpath /data/db/ --out /data/backup/ |
| mongodump --collection COLLECTION --db DB_NAME | This command will backup only specified collection of specified database. | mongodump --collection mycol --db test |

Restore() method

Restore data

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

Syntax

The basic syntax of mongorestore command is –

```
>mongorestore
```



- When you are preparing a MongoDB deployment, you should try to understand how your application is going to hold up in production. It's a good idea to develop a consistent, repeatable approach to managing your deployment environment so that you can minimize any surprises once you're in production.
- The best approach incorporates prototyping your set up, conducting load testing, monitoring key metrics, and using that information to scale your set up.



To monitor your deployment, MongoDB provides some of the following commands –

`mongostat`

This command checks the status of all running mongod instances and return counters of database operations. These counters include inserts, queries, updates, deletes, and cursors.

To run the command, start your mongod instance. In another command prompt, go to bin directory of your mongodb installation and type `mongostat`.

```
D:\set up\mongodb\bin>mongostat
```



mongotop

This command tracks and reports the read and write activity of MongoDB instance on a collection basis. By default, mongotop returns information in each second, which you can change it accordingly. You should check that this read and write activity matches your application intention, and you're not firing too many writes to the database at a time, reading too frequently from a disk, or are exceeding your working set size.

To run the command, start your mongod instance. In another command prompt, go to bin directory of your mongodb installation and type mongotop.

```
D:\set up\mongodb\bin>mongotop
```



An ObjectId is a 12-byte BSON type having the following structure –

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

MongoDB uses ObjectIds as the default value of `_id` field of each document, which is generated while the creation of any document. The complex combination of ObjectId makes all the `_id` fields unique.



ObjectId

Creating New ObjectId

To generate a new ObjectId use the following code –

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

The above statement returned the following uniquely generated id –

```
ObjectId("5349b4ddd2781d08c09890f3")
```

Instead of MongoDB generating the ObjectId, you can also provide a 12-byte id –

```
>myObjectId = ObjectId("5349b4ddd2781d08c09890f4")
```



ObjectID

Converting ObjectID to String

In some cases, you may need the value of ObjectID in a string format. To convert the ObjectID in string, use the following code –

```
>newObjectId.str
```

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

```
5349b4ddd2781d08c09890f3
```



Atomic operations

The recommended approach to maintain atomicity would be to keep all the related information, which is frequently updated together in a single document using **embedded documents**. This would make sure that all the updates for a single document are atomic.

Example:

In this document, we have embedded the information of the customer who buys the product in the product bought by field. Now, whenever a new customer buys the product, we will first check if the product is still available using product available field. If available, we will reduce the value of product available field as well as insert the new customer's embedded document in the product bought by field. We will use findAndModify command for this functionality because it searches and updates the document in the same go.

```
>db.createCollection("products")
{ "ok" : 1 }
> db.productDetails.insert(
  {
    "_id":1,
    "product_name": "Samsung S3",
    "category": "mobiles",
    "product_total": 5,
    "product_available": 3,
    "product_bought_by": [
      {
        "customer": "john",
        "date": "7-Jan-2014"
      },
      {
        "customer": "mark",
        "date": "8-Jan-2014"
      }
    ]
  }
)
WriteResult({ "nInserted" : 1 })
>
```


Atomic operations

```
>db.products.findAndModify({
  query:{_id:2,product_available:{$gt:0}},
  update:{
    $inc:{product_available:-1},

    $push:{product_bought_by:{customer:"rob",date:"9-Jan-2014"}}
  }
})
```

- Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB uses mapReduce command for map-reduce operations. MapReduce is generally used for processing large data sets.

MapReduce Command

Following is the syntax of the basic mapReduce command –

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

- The map-reduce function first queries the collection, then maps the result documents to emit key-value pairs, which is then reduced based on the keys that have multiple values.
- In the syntax –
 - map is a javascript function that maps a value with a key and emits a key-value pair
 - reduce is a javascript function that reduces or groups all the documents having the same key
 - out specifies the location of the map-reduce query result
 - query specifies the optional selection criteria for selecting documents
 - sort specifies the optional sort criteria
 - limit specifies the optional maximum number of documents to be returned



Using MapReduce

Consider the following document structure storing user posts. The document stores user_name of the user and the status of post.

```
{  
  "post_text": "tutorialspoint is an awesome website for tutorials",  
  "user_name": "mark",  
  "status": "active"  
}
```

Now, we will use a mapReduce function on our posts collection to select all the active posts, group them on the basis of user_name and then count the number of posts by each user using the following code –

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

The above mapReduce query outputs the following result –

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

The result shows that a total of 4 documents matched the query (status:"active"), the map function emitted 4 documents with key-value pairs and finally the reduce function grouped mapped documents having the same keys into 2.



- Starting from version 2.4, MongoDB started supporting text indexes to search inside string content. The Text Search uses stemming techniques to look for specified words in the string fields by dropping stemming stop words like a, an, the, etc
- Enabling Text Search
 - Initially, Text Search was an experimental feature but starting from version 2.6, the configuration is enabled by default.



Creating Text Index

Consider the following document under posts collection containing the post text and its tags –

```
> db.posts.insert({
  "post_text": "enjoy the mongodb classes on CLAP",
  "tags": ["mongodb", "tutorialspoint"]
},
{
  "post_text" : "writing tutorials on mongodb",
  "tags" : [ "mongodb", "tutorial" ]
})
WriteResult({ "nInserted" : 1 })
```



Using Text Index

Now that we have created the text index on `post_text` field, we will search for all the posts having the word `tutorialspoint` in their text.

```
> db.posts.find({$text:{$search:"tutorialspoint"}}).pretty()  
{  
  "_id" : ObjectId("5dd7ce28f1dd4583e7103fe0"),  
  "post_text" : "enjoy the mongodb classes on CLAP",  
  "tags" : [  
    "mongodb",  
    "tutorialspoint"  
  ]  
}
```



Text Search

Deleting Text Index

To delete an existing text index, first find the name of index using the following query –

```
>db.posts.getIndexes()  
[  
  {  
    "v" : 2,  
    "key" : {  
      "_id" : 1  
    },  
    "name" : "_id_",  
    "ns" : "mydb.posts"  
  },  
  {  
    "v" : 2,  
    "key" : {  
      "fts" : "text",  
      "ftsx" : 1  
    },  
    "name" : "post_text_text",  
    "ns" : "mydb.posts",  
    "weights" : {  
      "post_text" : 1  
    },  
    "default_language" : "english",  
    "language_override" : "language",  
    "textIndexVersion" : 3  
  }  
]  
>
```