

# **Full Stack Module-V**

**Manual V8.3**

**ANUDIP FOUNDATION**

---



## ICONS AND THEIR MEANING



**HINTS:**  
*Get ready for helpful insites on difficult topics and questions.*



**STUDENTS:**  
*This icon symbolize important instreutions and guides for the students.*



**TEACHERS/TRAINERS:**  
*This icon symbolize important instreutions and guides for the trainers.*

Lesson No	Lesson Name	Practical Duration (Minutes)	Theory Duration (Minutes)	Page No
1	MongoDB – Overview	60	60	03
2	CRUD Operations	60	60	07
3	Basic Operations	60	60	12
4	Aggregations	60	60	15
5	Indexing	120	nil	19
6	Replication and Sharding	120	nil	22

**Total Duration:** \_\_\_\_Hours

**Lesson 01: MongoDB – Overview (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to learn about : <ul style="list-style-type: none"><li>• NOSQL, CRUD</li><li>• Aggregations</li><li>• Replications</li><li>• Sharding</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

**NoSQL**

NoSQL is an approach to database design that can accomodate a wide variety of data models, including key-value, document, columnar and graph formats.

NoSQL stands for "not only SQL," and is an alternative to traditional relational databases in which data is placed in tables and data schema is carefully designed before the database is built.

NoSQL databases are especially useful for working with large sets of distributed data.

The NoSQL database movement came about to address the shortcomings of relational databases and the demands of modern software development. MongoDB is the leading NoSQL database, with significant adoption among the Fortune 500 and Global 500.

**CRUD**

CRUD operations refer to the basic Insert, Read, Update and Delete operations.

**Aggregations**

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

The aggregate() Method is used for aggregation in MongoDB.

Syntax

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

**The Aggregation Pipeline**

The aggregation pipeline is a framework for data aggregation modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into aggregated results.

The following example uses the aggregation pipeline on the restaurant sample dataset to find a list of restaurants located in the Bronx, grouped by restaurant category:

```
var MongoClient = require('mongodb').MongoClient
, assert = require('assert');
var url = 'mongodb://localhost:27017/test';

MongoClient.connect(url, function(err, db) {
  assert.equal(null, err);
  simplePipeline(db, function() {
    db.close();
  });
});

var simplePipeline = function(db, callback) {
  var collection = db.collection( 'restaurants' );
  collection.aggregate(
    [ { '$match': { 'borough': "Bronx" } },
      { '$unwind': '$categories' },
      { '$group': { '_id': "$categories", 'Bronx restaurants': { '$sum': 1 } } }
    ],
    function(err, results) {
      assert.equal(err, null);

      console.log(results)
      callback(results);
    }
  );
}
```

Inside the aggregate method, there are three stages:

1. The first pipeline stage filters out all documents except those with 5 in the stars field.
2. The second stage unwinds the categories field, which is an array, and treats each item in the array as a separate document.
3. The third stage groups the documents by category and adds up the number of matching 5-star results.

## Replications

In a database management system, there happens to be a requirement for data synchronizing across different servers. This process of data synchronization is referred to as Replication. It definitely helps in the redundancy factor while increasing 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.

## Sharding

Sharding in MongoDB is the process of distributing data across multiple servers for storage. MongoDB uses sharding to manage massive data growth. With an increase in the data size, a single machine may not be able to store data or provide an acceptable read and write throughput.

MongoDB sharding supports horizontal scaling and thus is capable of distributing data across multiple machines. Sharding in MongoDB allows you to add more servers to your database to support data growth and automatically balances data and load across various servers. MongoDB sharding provides additional

write capacity by distributing the write load over a number of mongod instances. It splits the data set and distributes them across multiple databases, or shards.

### Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

**1. Create the document structure of a blog site, which is simply a comma separated key value pair.**

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by: 'Anudip Foundation',
  url: 'http://www.anudip.org',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100,
  comments: [
    {
      user:'user1',
      message: 'My first comment',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
    },
    {
      user:'user2',
      message: 'My second comments',
      dateCreated: new Date(2011,1,25,7,45),
      like: 5
    }
  ]
}
```

**2. Create the document structure of another blog site with different title, message, and description**

**Note:** After saving the code, click the created HTML file to view the output.

**Reviewing the chapter**

- NoSQL is an approach to database design that can accommodate a wide variety of data models, including key-value, document, columnar and graph formats.
- MongoDB is the leading NoSQL database, with significant adoption among the Fortune 500 and Global 500.
- CRUD operations refer to the basic Insert, Read, Update and Delete operations.
- Aggregations operations process data records and return computed results.
- The aggregation pipeline is a framework for data aggregation modeled on the concept of data processing pipelines.
- Replication protects a database from the loss of a single server.
- Sharding in MongoDB is the process of distributing data across multiple servers for storage.

### Testing your skills

1. NoSQL stands for \_\_\_\_\_  
a) Node only SQL, b) not only SQL, c) Node Online SQL, d) None of the above
2. CRUD operations refer to which of the following basic operations  
a) Read, b) Update, c) Delete, d) All of the above
3. Replication protects a database from the loss of \_\_\_\_\_server/s  
a) Single, b) Dual, c) Triple, d) none of the above
4. \_\_\_\_\_ is the process of distributing data across multiple servers for storage  
a) Aggregations, b) Replications, c) Sharding, d) None of the above
5. The Aggregate method has \_\_\_\_\_number of stages  
a) 3, b) 4, c) 5, d) 6

**Lesson 02: CRUD Operations (120 minutes)**

**Objective:** After completing this lesson you will be able to learn about :

- CRUD
- Upsert
- Query interface

**Materials Required:**

- Computer With Windows XP and above
- Stable Internet connection

**Self- Learning Duration:** 60 minutes

**Practical Duration:** 60 minutes

**Total Duration:** 120 minutes

CRUD operations refer to the basic Insert, Read, Update and Delete operations.

Let's focus on different steps used in performing CRUD (Create/Read/Update/Delete) operations in MongoDB.

**The CREATE operation**

*Inserting a document into a collection*

The command `db.collection.insert()` will perform an insert operation into a collection of a document.

Let us insert a document to a student collection. You must be connected to a database for doing any insert. It is done as follows:

```
db.student.insert({
  regNo: "3014",
  name: "Test Student",
  course: {
    courseName: "MCA",
    duration: "3 Years"
  },
  address: {
    city: "Bangalore",
    state: "KA",
    country: "India"
  }
})
```

```
ca C:\windows\system32\cmd.exe - mongo
> db.student.insert(
... {
...   regNo: "3014",
...   name: "Test Student",
...   course: {
...     courseName: "MCA",
...     duration: "3 Years"
...   },
...   address: {
...     city: "Bangalore",
...     state: "KA",
...     country: "India"
...   }
... }
... )
... )
WriteResult({ "nInserted" : 1 })
>
```

Look closely and observe that an entry has been made into the collection called student.

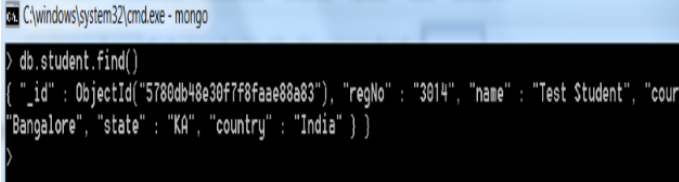
**The READ operation**



### *Querying a document from a collection*

The find() command will retrieve all the documents of the given collection. To retrieve (Select) the inserted document, run the below command:

```
db.collection_name.find()
```



```
C:\windows\system32\cmd.exe - mongo
```

```
> db.student.find()
{ "_id" : ObjectId("5780db48e30f7f8faae88a83"), "regNo" : "3014", "name" : "Test Student", "course" : "Bangalore", "state" : "KA", "country" : "India" }
```

Observe the above image closely to figure out that the record retrieved contains an attribute called \_id with some unique identifier value called ObjectId. This ObjectId acts as a document identifier.

If a record is to be retrieved based on some criteria, the find() method should be called passing parameters, then the record will be retrieved based on the attributes specified.

```
db.collection_name.find({"fieldname":"value"})
```

For Example: Let us retrieve the record from the student collection where the attribute regNo is 3014 and the query for the same is as shown below:

```
db.students.find({"regNo":"3014"})
```

## **The UPDATE operation**

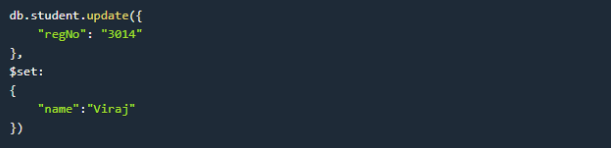
### *Updating a document in a collection*

In order to update specific field values of a collection in MongoDB, run the below query.

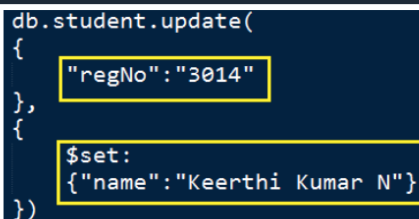
```
db.collection_name.update()
```

The update() method specified above will take the fieldname and the new value as argument to update a document.

Let us update the attribute name of the collection student for the document with regNo 3014.



```
db.student.update({
  "regNo": "3014"
},
$set:
{
  "name": "Viraj"
})
```



```
db.student.update(
{
  "regNo": "3014"
},
{
  $set:
  {"name": "Keerthi Kumar N"}
})
```

The Command Prompt will display the following result:

```
> db.student.update(
... {
...   "regNo": "3014"
... },
... {
...   $set:
...   { "name": "Keerthi Kumar N" }
... })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.find()
{ "_id" : ObjectId("5782060e30f7f8faee88a06"), "regNo" : "3014", "name" : "Keerthi Kumar N", "course" : { "courseName" : "MCA", "duration" : "3 Years" }, "address" : "Bangalore", "state" : "KA", "country" : "India" }
```

## The DELETE operation

*Removing an entry from the collection*

Let us now look into the deleting an entry from a collection.

In order to delete an entry from a collection, run the command as shown below:

```
db.collection_name.remove({"fieldname": "value"})
```

For Example: `db.student.remove({"regNo": "3014"})`

```
db.student.remove({"regNo" : "3014"})
> db.student.remove({"regNo" : "3014"})
WriteResult({ "nRemoved" : 1 })
> db.student.find()
>
```

Just observe that after running the `remove()` method, the entry has been deleted from the student collection.

## UPSERT operation

The UPSERT option is used with update method which creates a new document if the query does not retrieve any documents satisfying the criteria. The default value for this option is false. The UPSERT option does an insert based on the field and value pairs specified in the update parameter or field and value pairs specified in both query and update parameter.

This operation first searches for the document if not present then inserts the new document into the database.

```
> db.car.update(
... { name: "Qualis" },
... {
...   name: "Qualis",
...   speed: 50
... },
... { upsert: true }
... )
WriteResult({
  "nMatched" : 0,
  "nUpserted" : 1,
  "nModified" : 0,
  "_id" : ObjectId("548d3a955a5072e76925dc1c")
})
>
```

In the above example, the car with the name Qualis is checked for existence and if not, a document with car name "Qualis" and speed 50 is inserted into the database. The `nUpserted` with value "1" indicates a new document is inserted.

Note that to avoid inserting the same document more than once, create an index on the name field thereby ensuring that the document will be inserted only once for the UPSERT option on every update specified. If the UPSERT fails because of duplicate index key error, retrying results in the successful update operation.

## The Query Interface

In MongoDB, there are ways to query document. Have a look:

### The find() Method

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

The basic syntax of find() method is:

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

The find() method will display all the documents in a non-structured way.

### The pretty() Method

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

The basic syntax of pretty() method is:

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

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

## Practical Session

**Instructions: Create the code....using an editor....and save it with HTML extension.**

**1. Create a program where you need to insert a document to a student collection using the INSERT.**

```
db.student.insert({  
  regNo: "3014",  
  name: "Test Student",  
  course: {  
    courseName: "MCA",  
    duration: "3 Years"  
  },  
  address: {
```

```
city: "Bangalore",  
state: "KA",  
country: "India"  
}  
})
```

## 2. Create a program that deletes an entry from a collection

**Note:** After saving the code, click the created HTML file to view the output

### Reviewing the chapter

- CRUD operations refer to the basic Insert, Read, Update and Delete operations.
- The UPSERT option is used with update method which creates a new document if the query does not retrieve any documents satisfying the criteria.

### Testing your skills

1. Inserting a document into a collection is \_\_\_\_\_ operation

a) Read, b) Create, c) Update, d) Delete

2. Querying a document from a collection is \_\_\_\_\_ operation

a) Read, b) Create, c) Update, d) Delete

3. \_\_\_\_\_ creates a new document if the query does not retrieve any documents satisfying the criteria

a) Update, b) Insert, c) Upsert, d) None of the above

4. To display the results in a formatted way, \_\_\_\_\_ method can be used

a) Read, b) Pretty, c) Find, d) None of the above

5. \_\_\_\_\_ method returns only one document

a) find, b) findSingle, c) readOne, d) findOne

**Lesson 03: Basic Operations (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to learn about : <ul style="list-style-type: none"><li>• Basic Operations With Mongo Shell</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

MongoDB shell is the best tool out there to discover MongoDB features and manage every single aspect of your server deployments, instances, databases, collections and documents. It is based on JavaScript language for executing command and queries. Please do not worry if you have little or no knowledge of JavaScript: you will be able to mostly understand every example effortlessly as there is a common pattern to follow.

With JSON being a format to manage document, it is also used to specify commands and queries as well as to return their results. Such unification brings a lot of benefits because JSON is inherently simple, human-friendly and easy to understand.

MongoDB has many internal and experimental commands. Their usage is limited to very specific scenarios you may never encounter (or their behavior might be unstable).

**Basic operations**

`db.collection.insertMany()` can insert multiple documents into a collection. Pass an array of documents to the method.

`insertMany()` returns a document that includes the newly inserted documents `_id` field values.

Use `db.collection.insertOne()` to insert a single document.

**Query Documents***Select All Documents*

To select all documents in the collection, pass an empty document as the query filter document to the `db.collection.find()` method:

```
db.inventory.find( {} )
```

To query for documents that match specific equality conditions, pass the find() method a query filter document with the <field>: <value> of the desired documents. The following example selects from the inventory collection all documents where the status equals "D":

```
db.inventory.find( { status: "D" } )
```

#### *Match an Embedded Document*

Equality matches on the whole embedded document require an exact match of the specified <value> document, including the field order. For example, the following query selects all documents where the field size equals the document { h: 14, w: 21, uom: "cm" }:

```
db.inventory.find( { size: { h: 14, w: 21, uom: "cm" } } )
```

#### *Match a Field in an Embedded Document*

The following example selects all documents where the field uom nested in the size field equals the string value "in":

```
db.inventory.find( { "size.uom": "in" } )
```

#### *Match an Element in an Array*

The following example queries for all documents where tags is an array that contains the string "red" as one of its elements:

```
db.inventory.find( { tags: "red" } )
```

#### *Match an Array Exactly*

The following example queries for all documents where the field tags value is an array with exactly two elements, "red" and "blank", in the specified order:

```
db.inventory.find( { tags: ["red", "blank"] } )
```

### **Practical Session**

**Instructions: Create the code....using an editor....and save it with HTML extension.**

#### **1. Create a program to insert data into MongoDB collection**

```
>db.mycol.insert({
```

```
_id: ObjectId(7df78ad8902c),  
title: 'MongoDB Overview',  
description: 'MongoDB is no sql database',  
by: 'tutorials point',  
url: 'http://www.tutorialspoint.com',  
tags: ['mongodb', 'database', 'NoSQL'],  
likes: 100  
})
```

## 2. Create a program using the find() method to display documents in a non-structured way.

**Note:** After saving the code, click the created HTML file to view the output.

### Reviewing the chapter

In this chapter, we learnt about MongoDB shell and the basic operations associated with the same.

### Testing your skills

1. \_\_\_\_\_ returns a document that includes the newly inserted documents \_id field values  
a) insertAll(), b) insertMany(), c) insertNew(), d) None of the above
2. To query for documents that match specific equality conditions, \_\_\_\_\_ method is used  
a) query, b) search, c) equal , d) find
3. \_\_\_\_\_ method is used to insert a single document  
a) insert(), b) insertSingle(), c) insertNew(), d) insertOne()
4. MongoDB shell can be used to manage  
a) server deployments, b) instances, c) databases, d) All of the above

## Lesson 04: Aggregations (120 minutes)

<b>Objective:</b> After completing this lesson you will be able to learn about : <ul style="list-style-type: none"> <li>Basics</li> <li>Types</li> <li>Methods</li> </ul>	<b>Materials Required:</b> <ul style="list-style-type: none"> <li>Computer With Windows XP and above</li> <li>Stable Internet connection</li> </ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

Aggregation in MongoDB is nothing but an operation used to process the data that returns the computed results. Aggregation basically groups the data from multiple documents and operates in many ways on those grouped data in order to return one combined result.

In sql count(\*) and with group by is an equivalent of MongoDB aggregation.

Aggregate function groups the records in a collection, and can be used to provide total number(sum), average, minimum, maximum etc out of the group selected.

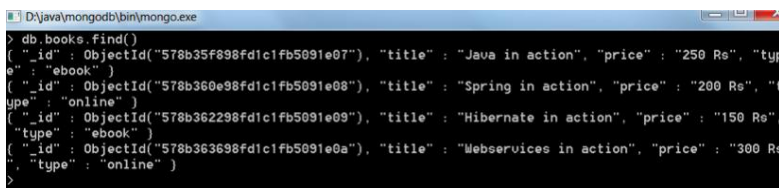
In order to perform the aggregate function in MongoDB, aggregate () is the function to be used.

The syntax for aggregation is as follows:

*db.collection\_name.aggregate(aggregate\_operation)*

### A sample example to understand the use of Aggregate Function in MongoDB

Let us consider a collection named books with the data as shown below:



```

> db.books.find()
{ "_id" : ObjectId("578b35f898fd1cfb5091e07"), "title" : "Java in action", "price" : "250 Rs", "type" : "ebook" }
{ "_id" : ObjectId("578b360e98fd1cfb5091e08"), "title" : "Spring in action", "price" : "200 Rs", "type" : "online" }
{ "_id" : ObjectId("578b362298fd1cfb5091e09"), "title" : "Hibernate in action", "price" : "150 Rs", "type" : "ebook" }
{ "_id" : ObjectId("578b363698fd1cfb5091e0a"), "title" : "Webservices in action", "price" : "300 Rs", "type" : "online" }
  
```

Now, from the above collection, let us use the aggregate function to group the books which are of the type ebook and online. Following command will be used:

*db.books.aggregate([{\$group : {\_id: "\$type", category: {\$sum : 1}}}]])*

The output:



The above aggregate function will give result, which says there are 2 records of the type ebook and 2 records of the type online. So the above aggregation command, has grouped our collection data, based on their type. Have a look:

```
D:\java\mongodb\bin\mongo.exe
> db.books.aggregate([{$group : {_id : "$type", category : {$sum : 1}}]})
{ "_id" : "ebook", "category" : 2 }
{ "_id" : "online", "category" : 2 }
>
```

### Different expressions of Aggregate function

- **\$sum** Summates the defined values from all the documents in a collection
- **\$avg** Calculates the average values from all the documents in a collection
- **\$min** Return the minimum of all values of documents in a collection
- **\$max** Return the maximum of all values of documents in a collection
- **\$addToSet** Inserts values to an array but no duplicates in the resulting document
- **\$push** Inserts values to an array in the resulting document
- **\$first** Returns the first document from the source document
- **\$last** Returns the last document from the source document

### Different types and methods of aggregates in MongoDB

In MongoDB, three types of aggregation are available as shown below:

1. Aggregation Pipeline
2. Map Reduce
3. Single Purpose Aggregation

#### Aggregation Pipeline

Aggregation Framework in MongoDB is developed on the concept of data processing pipelines. In this pipeline, a set of various functions are applied on a document which is entered in the pipeline to aggregate the final result.

Basically, two operations are performed on any document within the pipeline. First, the records are filtered just like how queries are performed and in the second phase, the transformation of the document happens to change its type for output purpose.

On the other hand, pipeline operations are also used for sorting, grouping, merging & aggregation of arrays and arrays of the document. Somehow pipelines can also be used to summarize the content or to calculate the average and concatenation of record.

#### Map Reduce

MongoDB also provides the Map Reduce feature for aggregation purposes. Generally, there are two phases of Map Reduce. In the first phase, each document is processed and emits common and redundant part of the document to pass a unique record for the next phase.

In the second phase, all the unique parts get together and aggregate to produce a single result. Map Reduce also provide sorting, filtering, and document modification.

### Single Purpose Aggregation

In the single purpose aggregation, only one filter is applied to calculate the result. In simple words, if we have to aggregate a whole collection based on one filter, then we have to use single-purpose aggregation operations.

### Practical Session

**Instructions: Create the code....using an editor....and save it with HTML extension.**

#### 1. Create a program that starts an aggregation pipeline while displaying the name and address of a person

```
{  
  "id": "1",  
  "firstName": "Jane",  
  "lastName": "Doe",  
  "phoneNumber": "555-555-1212",  
  "city": "Beverly Hills",  
  "state": "CA",  
  "zip": 90210  
  "email": "Jane.Doe@compose.io"  
}
```

#### 2. Create a program that starts an aggregation pipeline while displaying the name and address of you

**Note:** After saving the code, click the created HTML file to view the output.

### Reviewing the chapter

- Aggregation in MongoDB is nothing but an operation used to process the data that returns the computed results.
- Aggregation basically groups the data from multiple documents and operates in many ways on those grouped data in order to return one combined result.
- Aggregation Framework in MongoDB is developed on the concept of data processing pipelines.

**Testing your skills**

1. Summates the defined values from all the documents in a collection  
a) sum, b) avg, c) all, d) Both (a) and (b)
  
2. Inserts values to an array but no duplicates in the resulting document  
a) insert, b) push, c) first, d) None of the above
  
3. How many types of aggregations available in MongoDB  
a) 2, b) 4, c) 3, d) none of the above
  
4. Pipeline operations are also used for which of the following:  
a) aggregate pipeline, b) map reduce, c) single purpose aggregation, d) all of the above
  
5. if we have to aggregate a whole collection based on one filter, it is called  
a) aggregate pipeline, b) map reduce, c) single purpose aggregation, d) none of the above

**Lesson 05: Indexing (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to learn about : <ul style="list-style-type: none"><li>• Introduction</li><li>• Types</li><li>• Properties</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 120 minutes	<b>Practical Duration:</b> nil
<b>Total Duration:</b> 120 minutes	

**Introduction**

Typically, indexes are data structures that can store collection's data set in a form that is easy to traverse. Queries are efficiently executed with the help of indexes in MongoDB.

Indexes help MongoDB find documents that match the query criteria without performing a collection scan. If a query has an appropriate index, MongoDB uses the index and limits the number of documents it examines.

Indexes store field values in the order of the value. The order in which the index entries are made support operations, such as equality matches and range-based queries. MongoDB sorts and returns the results by using the sequential order of the indexes. The indexes of MongoDB are similar to the indexes in any other databases. MongoDB defines the indexes at the collection level for use in any field or subfield.

**Types**

MongoDB supports the following index types for querying.

- **Default \_id:** Each MongoDB collection contains an index on the default \_id (Read as underscore id) field. If no value is specified for \_id, the language driver or the mongod (read as mongo D) creates a \_id field and provides an ObjectId (read as Object ID) value.
- **Single Field:** For a single-field index and sort operation, the sort order of the index keys do not matter. MongoDB can traverse the indexes either in the ascending or descending order.
- **Compound Index:** For multiple fields, MongoDB supports user-defined indexes, such as compound indexes. The sequential order of fields in a compound index is significant in MongoDB.

- **Multikey Index:** To index array data, MongoDB uses multikey indexes. When indexing a field with an array value, MongoDB makes separate index entries for each array element.
- **Geospatial Index:** To query geospatial data, MongoDB uses two types of indexes—2d indexes (read as two D indexes) and 2d sphere (read as two D sphere) indexes.
- **Text Indexes:** These indexes in MongoDB searches data string in a collection.
- **Hashed Indexes:** MongoDB supports hash-based sharding and provides hashed indexes. These indexes the hashes of the field value.

## Properties

Following are the index properties of MongoDB.

### *Unique Indexes*

The unique property of MongoDB indexes ensures that duplicate values for the indexed field are rejected. In addition, the unique indexes can be interchanged functionally with other MongoDB indexes.

### *Sparse Indexes*

This property ensures that queries search document entries having an indexed field. Documents without indexed fields are skipped during a query. Sparse index and the unique index can be combined to reject documents with duplicate field values and ignore documents without indexed keys.

### *Total time to Live or TTL Indexes*

These are special indexes in MongoDB used to automatically delete documents from a collection after a specified duration of time. This is ideal for deleting information, such as machine-generated data, event logs, and session data that needs to be in the database for a shorter duration.

## Reviewing the chapter

- Indexes are data structures that can store collection's data set in a form that is easy to traverse. Queries are efficiently executed with the help of indexes in MongoDB.

- Indexes store field values in the order of the value.

### Testing your skills

1. Which of the following are types of indexes

a) Multikey, b) compound, c) text, d) all of the above

2. These indexes in MongoDB searches data string in a collection

a) Text, b) Geospatial, c) Default\_id, d) none of the above

3. Which index is used for indexing an array

a) Multikey, b) compound, b) Geospatial, c) Default\_id

4. This property ensures that queries search document entries having an indexed field

a) Unique, b) Sparse, c) TTL, d) none of the above

5) The full form of TTL is\_\_\_\_\_

a) Testing To Launch, b) Total Testing Limitations, c) Total Time to Live, d) none of the above

**Lesson 06: Replication and Sharding (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to learn about : <ul style="list-style-type: none"><li>• Purpose</li><li>• Properties</li><li>• Mechanics</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 120 minutes	<b>Practical Duration:</b> nil
<b>Total Duration:</b> 120 minutes	

**Replication – Definition and purpose**

Replication refers to a database setup in which several copies of the same dataset are hosted on separate machines. The main reason to have replication is redundancy. If a single database host machine goes down, recovery is quick since one of the other machines hosting a replica of the same database can take over. A quick fail-over to a secondary machine minimizes downtime, and keeping an active copy of the database acts as a backup to minimize loss of data.

**How replication is handled in MongoDB?**

In MongoDB, replication is handled through something called a Replica Set: a group of MongoDB server instances that are all maintaining the same data set. One of those instances will be elected as the primary, and will receive all write operations from client applications. The primary records all changes in its operation log, and the secondaries asynchronously apply those changes to their own copies of the database. If the machine hosting the primary goes down, a new primary is elected from the remaining instances.

**Procedures to create a replica set**

The step-by-step procedures to create a replica set are listed below:

- Install the MongoDB binaries to each additional machine that will be hosting the database.
- Modify the database configuration file to use a given Replica Set name, and copy that configuration file onto each database machine.
- Start the MongoDB service on each of the new machines, and restart the service on the original one.
- Run a command in the MongoDB shell to initiate the Replica Set.
- Update the Deadline Repository's database settings to recognize the Replica Set. This can be done with one call to Deadline Command.

## Sharding - Definition and purpose

For larger render farms, scaling becomes a key performance issue. Having a large number of clients performing high-throughput operations can really test the limits of a single database instance. Sharding is a strategy that can mitigate this by distributing the database data across multiple machines. It is essentially a way to perform load balancing by routing operations to different database servers.

To use sharding in MongoDB we must configure a Shard Cluster.

## Procedures to create a Shard Cluster

The step-by-step procedures to create a Shard Cluster are listed below:

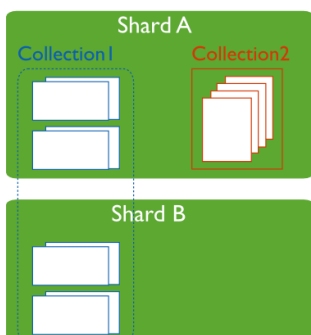
- Create a Replica Set for the Config Servers, as well as a Replica Set for each shard.
- Run at least one instance of the mongos application, connecting it to the Config Server replica set.
- Add the shards to the Shard Cluster through the MongoDB console.
- Enable sharding on the database.
- Configure how each collection in the database will be sharded (more on this below).
- Update the Deadline Repository settings to connect to the router (mongos) servers.

## Components of a Shard Cluster

- Shards: Each shard contains a portion of the complete database dataset. Each shard is a replica set, meaning that you can think of each shard as being a group of machines that each maintain their own copy of this portion of data.
- Config Servers: The config servers contain settings and metadata about the shard cluster. Since this information is crucial to the operation of the cluster, the config servers are also deployed as a replica set.
- Routers: Routers are instances of a MongoDB service called mongos that acts as the interface between clients and the sharded cluster. The routers are responsible for forwarding database operations to the correct shard.

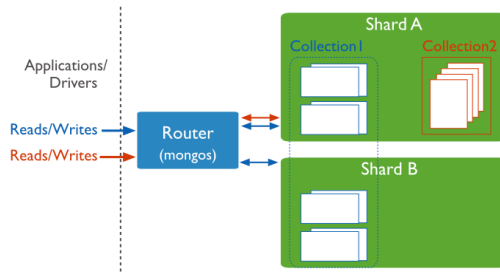
## Understanding the mechanics behind Sharded and Non-Sharded Data

Sharding operates on the collection level. You can shard multiple collections within a database or have multiple databases with sharding enabled. However, in production deployments, some databases and collections will use sharding, while other databases and collections will only reside on a single shard.





Regardless of the data architecture of your sharded cluster, ensure that all queries and operations use the mongos router to access the data cluster. Use the mongos even for operations that do not impact the sharded data.



Do keep a note of the fact that as you configure sharding, you will use the `enableSharding` command to enable sharding for a database. This simply makes it possible to use the `shardCollection` command on a collection within that database.

### Reviewing the chapter

- Replication refers to a database setup in which several copies of the same dataset are hosted on separate machines.
- In MongoDB, replication is handled through something called a Replica Set: a group of MongoDB server instances that are all maintaining the same data set.
- Sharding is essentially a way to perform load balancing by routing operations to different database servers.

### Testing your skills

1. Which of the following are components of shard cluster

a) Routers, b) modifiers, c) promise, d) detector

2. \_\_\_\_\_ contain settings and metadata about the shard cluster

a) Shards, b) ErrorHandler, c) config servers, d) None of the above

3. \_\_\_\_\_ are instances of mongos that acts as the interface between clients and the sharded cluster

a) Shards, b) routers, c) Config servers, d) None of the above

4. \_\_\_\_\_ way to perform load balancing by routing operations to different database servers

a) Sharding, b) Replication, c) Routing, d) None of the above

5. Sharding operates on the \_\_\_\_\_ level

a) Operational, b) Multiple, c) Reaction, d) Collection