**What is MongoDB**?

MongoDB is a document-oriented NoSQL database used for high volume data storage. MongoDB is a database which came into light around the mid-2000s. It falls under the category of a NoSQL database.

**MongoDB Features**

1. Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.

2. The document structure is more in line with how developers construct their classes and objects in their respective programming languages. Developers will often say that their classes are not rows and columns but have a clear structure with key-value pairs.

3. As seen in the introduction with NoSQL databases, the rows (or documents as called in MongoDB) doesn’t need to have a schema defined beforehand. Instead, the fields can be created on the fly.

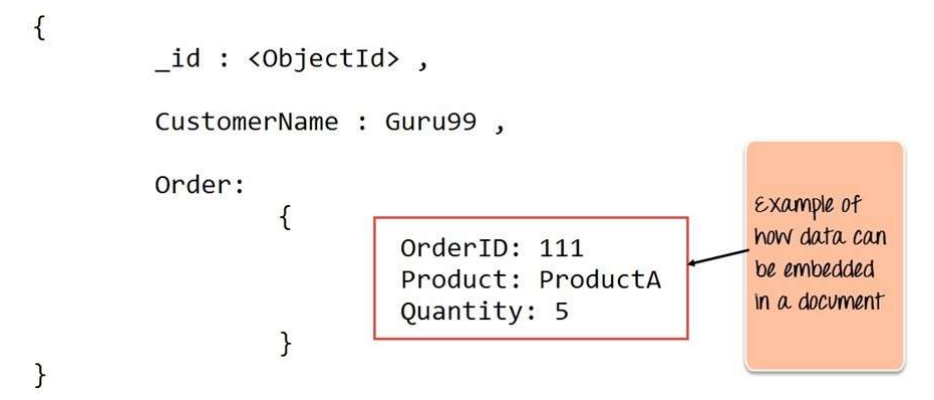
4. The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.

5. Scalability – The MongoDB environments are very scalable. Companies acrossthe world have defined clusters with some of them running 100+ nodes with around millions of documents within the database.

**MongoDB Example**

The below example shows how a document can be modeled in MongoDB.

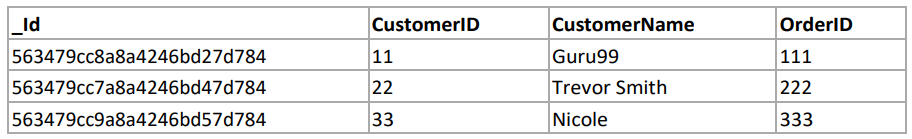
1. The \_id field is added by MongoDB to uniquely identify the document in the collection.

2. What you can note is that the Order Data (OrderID, Product, and Quantity ) which in RDBMS will normally be stored in a separate table, while in MongoDB it is actually stored as an embedded document in the collection itself. This is one of the key differences in how data is modeled in Mongo

**Key Components of MongoDB Architecture:-**

Below are a few of the common terms used in MongoDB

1. **\_id** – This is a field required in every MongoDB document. The \_id field represents a unique value in the MongoDB document. The \_id field is like the document’s primary key. If you create a new document without an \_id field, MongoDB will automatically create the field. So for example, if we see the example of the above customer table, Mongo DB will add a 24 digit unique identifier to each document in the collection.

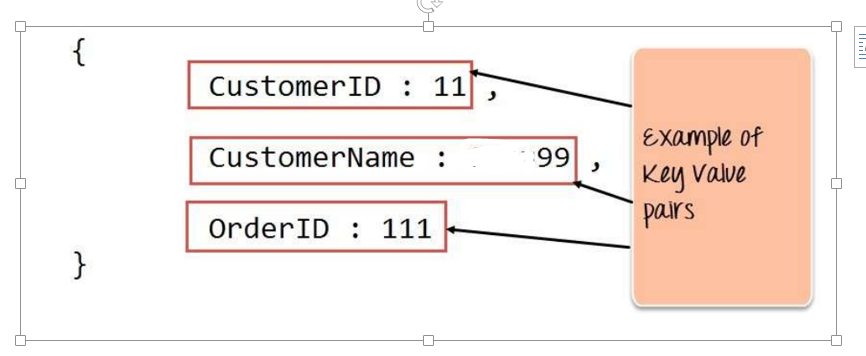


1. **Collection** – This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL. A collection exists within a single database. As seen from the introduction collections don’t enforce any sort of structure.
2. **Cursor** – This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results
3. **Database** – This is a container for collections like in RDMS wherein it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.

5. **Document** - A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field **name and values**.

6. **Field** - A name-value pair in a document. A document haszero or more fields. Fields are analogous to columns in relational databases.

The following diagram shows an example of Fields with Key value pairs. So in the example below CustomerID and 11 is one of the key value pair’s defined in the document.



7. **JSON** – This is known as JavaScript Object Notation. This is a human readable, plain text format for expressing structured data. JSON is currently supported in many programming languages.

Just a quick note on the key difference between the \_id field and a normal collection field. The \_id field is used to uniquely identify the documents in a collection and is automatically added by MongoDB when the collection is created.

**Why Use MongoDB?**

Below are the few of the reasons as to why one should start using MongoDB

1. **Document-oriented** – Since MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very flexible and adaptable to real business world situation and requirements.

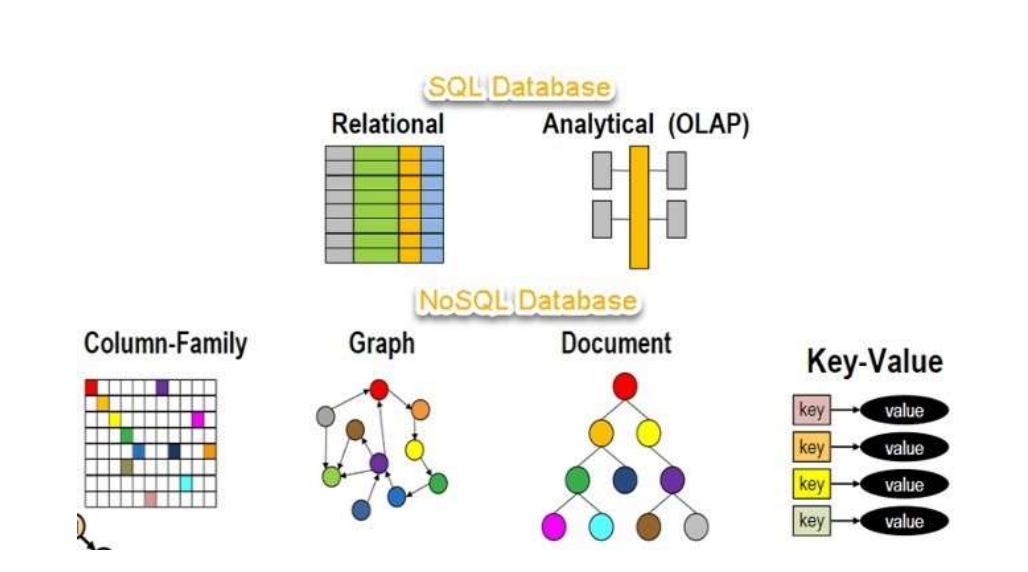
2.**Indexing** - Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.

3.**Replication** - MongoDB can provide high availability with replica sets. A replica set consists of two or more mongo DB instances. Each replica set member may act in the role of the primary or secondary replica at any time. The primary replica is the main server which interacts with the client and performs all the read/write operations. The Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.

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

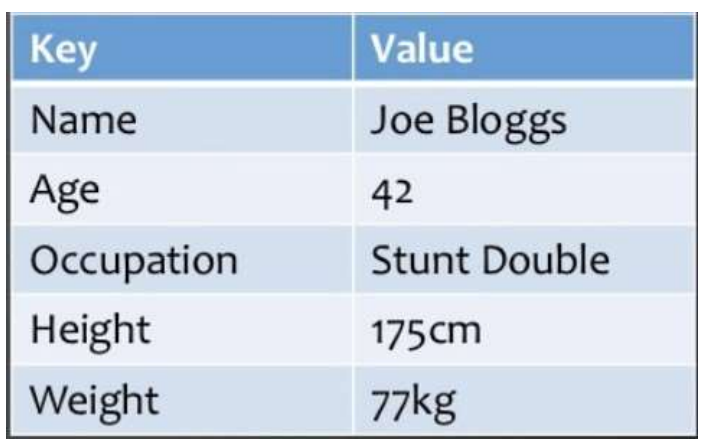
**What is NoSQL?**

NoSQL is a non-relational DMS, that does not require a **fixed schema**, avoids joins, and is easy to scale. NoSQL database is used for distributed data stores with humongous data storage needs. NoSQL is used for Big data and realtime web apps. For example, companies like Twitter, Facebook, Google that collect terabytes of user data every single day.

NoSQL database stands for “Not Only SQL” or “Not SQL.” Though a better term would NoREL NoSQL caught on. Carl Strozz introduced the NoSQL concept in 1998. Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi- structured, unstructured and polymorphic data

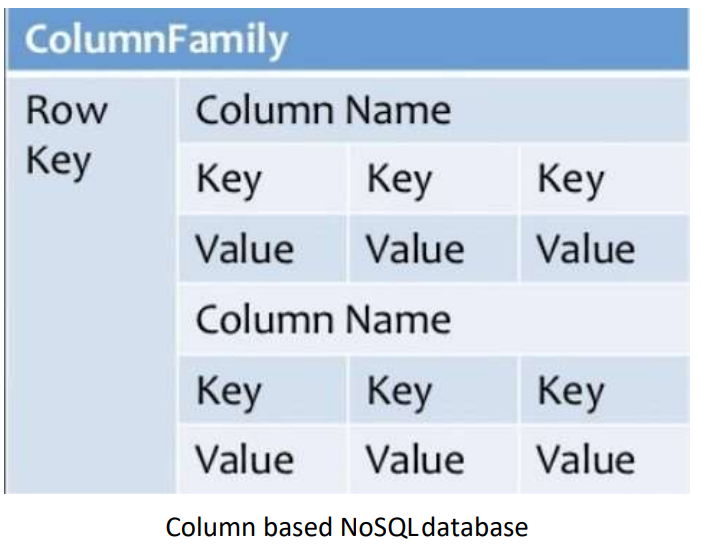
**Key Value Pair Based Data is stored in key/value pairs.**

It is designed in such a way to handle lots of data and heavy load. Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc. For example, a key-value pair may contain a key like “Website” associated with a value like “test”.



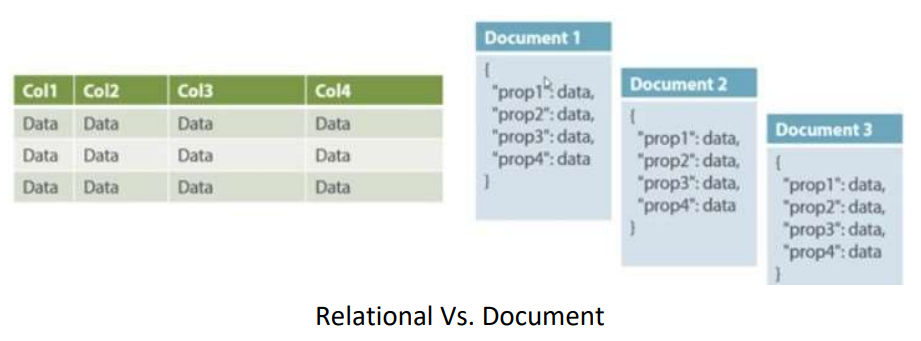
**Column-based**

Column-oriented databases work on columns and are based on BigTable paper by Google. Every column istreated separately. Values of single column databases are stored contiguously.



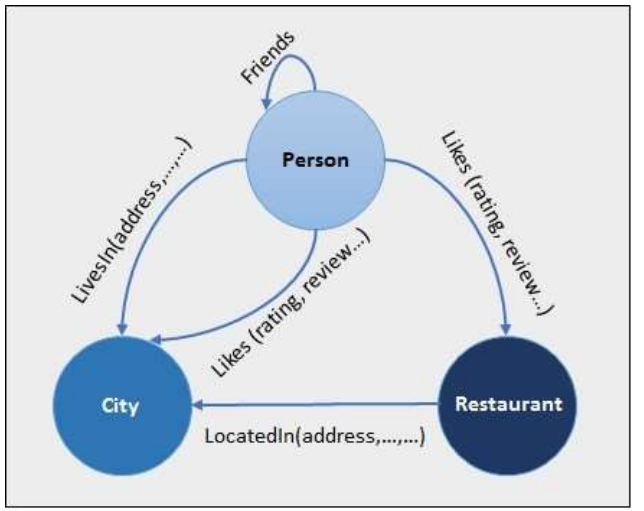
**Document-Oriented:**

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.

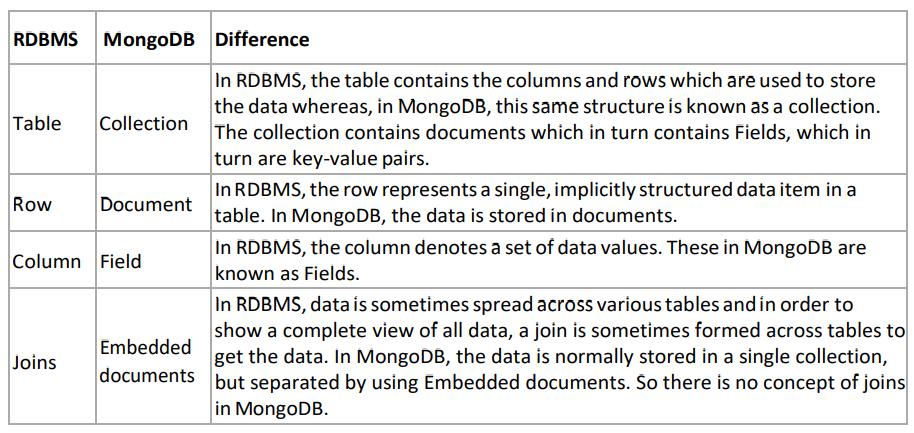


**Graph-Based :-**

A graph type database stores entities as well the relations amongst those entities. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge has a unique identifier.



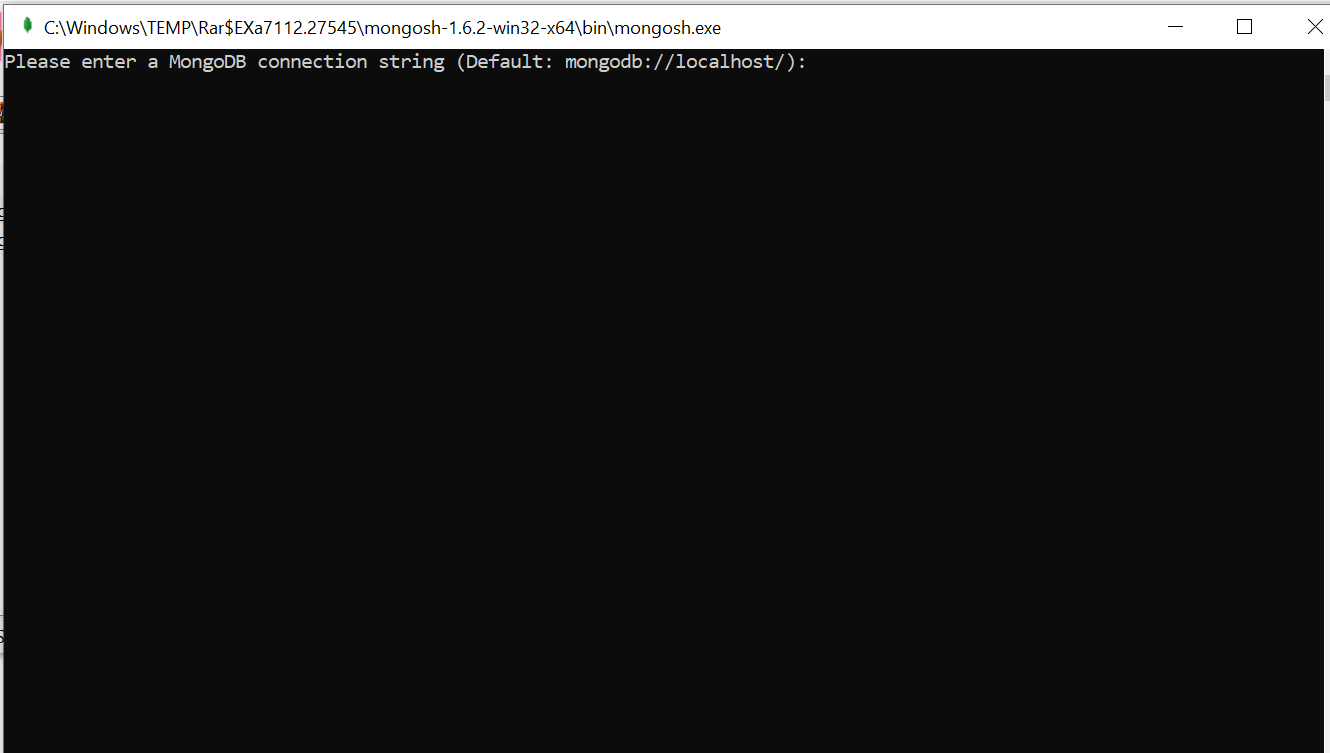
**Difference between MongoDB & RDBMS**



**To download the Mongo DB Shell, Click on below link:--**

**https://downloads.mongodb.com/compass/mongosh-1.6.2-win32-x64.zip**

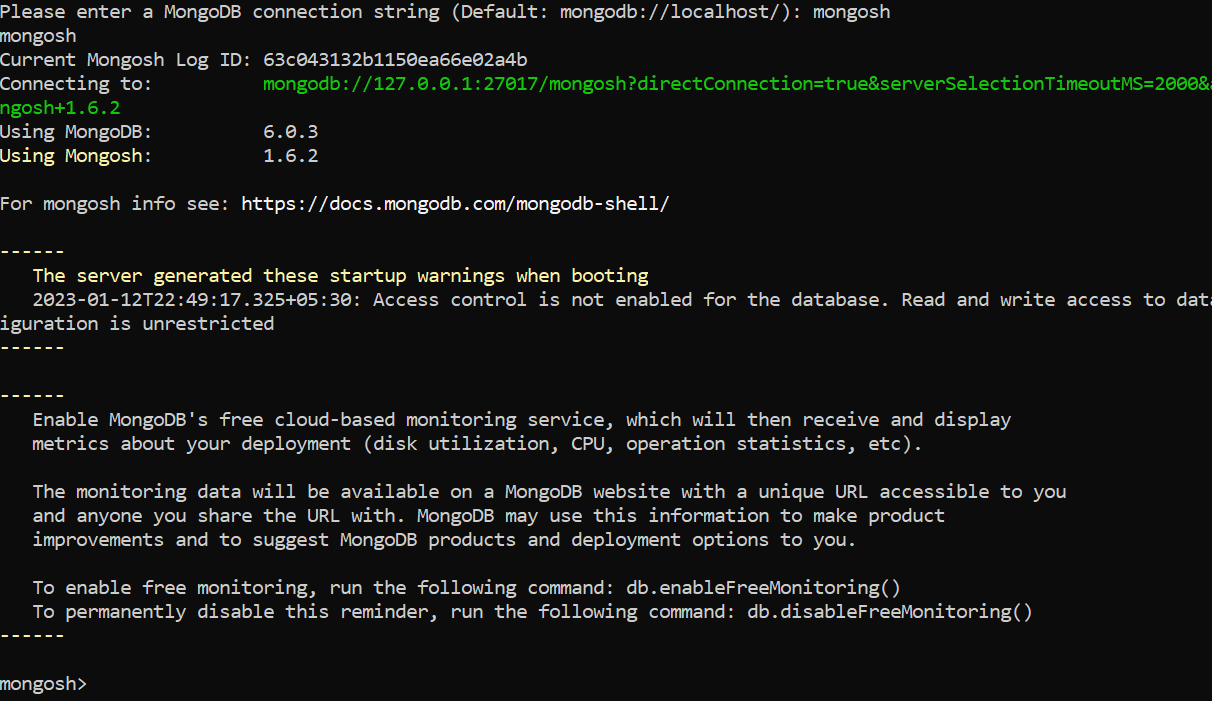
After downloading ZIP ,Open the **mongosh.exe file.(**



Once you the above screen it means you have successfully installed Mongo DB Shell.

Please use below command:--

-------🡪mongosh



Create Database using mongosh

After connecting to your database using mongosh, you can see which database you are using by typing db in your terminal.

Show dbs—it will show the all databases.

## Change or Create a Database

You can change or create a new database by typing use then the name of the database.

Use scooden

**Remember:** In MongoDB, a database is not actually created until it gets content!

## Create Collection using mongosh

## You can create a collection using the createCollection() database method.

db.createCollection("posts")

{ ok: 1 }

Atlas atlas-8iy36m-shard-0 [primary] blog>

You can also create a collection during the insert process.

db.posts.insertOne(object)

**Remember:** In MongoDB, a collection is not actually created until it gets content!

# MongoDB mongosh Insert

## Insert Documents

There are 2 methods to insert documents into a MongoDB database.

### **insertOne()**

To insert a single document, use the insertOne() method.

This method inserts a single object into the database.

db.posts.insertOne({

title: "Post Title 1",

body: "Body of post.",

category: "News",

likes: 1,

tags: ["news", "events"],

date: Date()

})

### **insertMany()**

To insert multiple documents at once, use the insertMany() method.

This method inserts an array of objects into the database.

db.posts.insertMany([

{

title: "Post Title 2",

body: "Body of post.",

category: "Event",

likes: 2,

tags: ["news", "events"],

date: Date()

},

{

title: "Post Title 3",

body: "Body of post.",

category: "Technology",

likes: 3,

tags: ["news", "events"],

date: Date()

},

{

title: "Post Title 4",

body: "Body of post.",

category: "Event",

likes: 4,

tags: ["news", "events"],

date: Date()

}

])

# **MongoDB mongosh Find:-**

There are 2 methods to find and select data from a MongoDB collection, find() and findOne().

### **find()**

To select data from a collection in MongoDB, we can use the find() method.

This method accepts a query object. If left empty, all documents will be returned.

db.posts.find()

### **findOne()**

To select only one document, we can use the findOne() method.

This method accepts a query object. If left empty, it will return the first document it finds.

**Note:** This method only returns the first match it finds.

db.posts.findOne()

## Projection:- Use to select necessary data rather than selecting whole data.

**Note:** This parameter is optional

How to display only title and date fields in the results.

db.posts.find({}, {title: 1, date: 1})

Notice that the \_id field is also included. This field is always included unless specifically excluded.

We use a 1 to include a field and 0 to exclude a field.

db.posts.find({}, {\_id: 0, title: 1, date: 1})

**Note:** You cannot use both 0 and 1 in the same object. The only exception is the \_id field. You should either specify the fields you would like to include or the fields you would like to exclude.

if we try to specify both 0 and 1 in the same object.

db.posts.find({}, {title: 1, date: 0})

# **MongoDB mongosh Update:**To update an existing document we can use the updateOne() or updateMany() methods.

## updateOne()

The updateOne() method will update the first document that is found matching the provided query.

Let's see what the "like" count for the post with the title of "Post Title 1":

db.posts.find( { title: "Post Title 1" } )

Now let's update the "likes" on this post to 2. To do this, we need to use the $set operator.

db.posts.updateOne( { title: "Post Title 1" }, { $set: { likes: 2 } } )

Check the document again and you'll see that the "like" have been updated.

db.posts.find( { title: "Post Title 1" } )

## Insert if not found

If you would like to insert the document if it is not found, you can use the upsert option.

db.posts.updateOne(

{ title: "Post Title 5" },

{

$set:

{

title: "Post Title 5",

body: "Body of post.",

category: "Event",

likes: 5,

tags: ["news", "events"],

date: Date()

}

},

{ upsert: true }

)

## updateMany()

The updateMany() method will update all documents that match the provided query.

Update likes on all documents by 1. For this we will use the $inc (increment) operator:

db.posts.updateMany({}, { $inc: { likes: 1 } })

Now check the likes in all of the documents and you will see that they have all been incremented by 1.

# **MongoDB mongosh Delete:-**We can delete documents by using the methods deleteOne() or deleteMany().

These methods accept a query object. The matching documents will be deleted.

## deleteOne()

The deleteOne() method will delete the first document that matches the query provided.

db.posts.deleteOne({ title: "Post Title 5" })

## deleteMany()

The deleteMany() method will delete all documents that match the query provided.

db.posts.deleteMany({ category: "Technology" })

## MongoDB Query Operators:--

## There are many query operators that can be used to compare and reference document fields

### **Comparison**

The following operators can be used in queries to compare values:

* $eq: The $eq specifies the equality condition. It matches documents where the value of a field equals the specified value.

db.posts.find ( { category: { $eq: 'Event'} } )

### **$gt**

The $gt chooses a document where the value of the field is greater than the specified value. Value is greater than another value

**Syntax:**

db.posts.find ( { likes: { $gt: 2 } } )

### **$gte**

The $gte choose the documents where the field value is greater than or equal to a specified value.  Value is greater than or equal to another value

**Syntax:**

db.posts.find ( { likes: { $gt: 2 } } )

### **$in**

The $in operator choose the documents where the value of a field equals any value in the specified array. Value is matched within an array

**Syntax:**

db.posts.find( { likes: { $in: [3, 4,5,6] } } )

### **$lt**

The $lt operator chooses the documents where the value of the field is less than the specified value.

**Syntax:**

db.posts.find ( { likes: { $lt: 3 } } )

### **$lte**

The $lte operator chooses the documents where the field value is less than or equal to a specified value.

**Syntax:**

db.posts.find ( { likes: { $lt: 3 } } )

### **$ne**

The $ne operator chooses the documents where the field value is not equal to the specified value.

**Syntax:**

db.posts.find ( { likes: { $ne: 4 } } )

### **$nin**

The $nin operator chooses the documents where the field value is not in the specified array or does not exist.

**Syntax:**

db.posts.find ( { likes: { $nin: [ 3,4 ] } } )

### **Logical**

The following operators can logically compare multiple queries.

### **$and**

The $and operator works as a logical AND operation on an array. The array should be of one or more expressions and chooses the documents that satisfy all the expressions in the array.

**Syntax:**

db.posts.find ( { $and: [ { likes: { $ne: 6 } }, { likes: { $gt: **2** } } ] } )

### **$not**

The $not operator works as a logical NOT on the specified expression and chooses the documents that are not related to the expression.

**Syntax:**

db.posts.find ( { likes: { $not: {$ne: 2 } } } )

db.posts.find ( { likes: { $not: { $eq: 2 } } } )

### **$nor**

The $nor operator works as logical NOR on an array of one or more query expression and chooses the documents that fail all the query expression in the array.

**Syntax:**

db.posts.find ( { $nor: [ { likes: 5 }, { likes: 6 } ] } )

db.posts.find ( { $nor: [ { likes: 2 }, { likes: 6 } ] } )

### **$or**

It works as a logical OR operation on an array of two or more expressions and chooses documents that meet the expectation at least one of the expressions.

**Syntax:**

db.posts.find ( { $or: [ { likes: { $eq: 6 } }, { likes: { $eq: 2 } } ] } )

### **$regex**

It provides regular expression abilities for pattern matching strings in queries. The MongoDB uses regular expressions that are compatible with Perl.

**Syntax:**

db.posts.find({category: {$regex : "Event"}}).pretty()

db.posts.find({category: {$regex : "^T"}}).pretty()

### **$text**

The $text operator searches a text on the content of the field, indexed with a text index.

**Syntax:**

db.posts.createIndex({category:"text", "attributes.attribute\_value": "text" })

db.posts.find( { $text: { $search: 'Event' } }, { category: 1 })

* db.posts.dropIndex("category\_text\_attributes.attribute\_value\_text")



[www.linkedin.com/company/77769084/](https://www.linkedin.com/company/77769084/)

[www.instagram.com/scodeenglobal\_india/](https://www.instagram.com/scodeenglobal_india/)