Introduction to MongoDB

Relevel
by Unacademy

Topic to be Covered

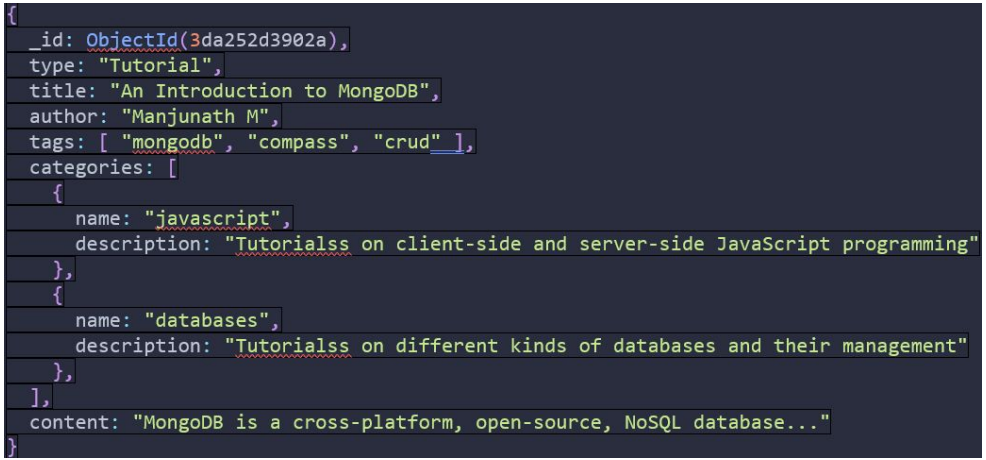
- Introduction to Mongo DB
- MongoDB Architecture's Key Components
- JSON and BSON
- Practical Use cases
- Installation of Mongo DB on Windows
- Database Operations
- Collection Operations
- CRUD Document operations on single and multiple docs
- _id/ObjectId
- Query and Projection Operators
- Example

Introduction to Mongo DB

- MongoDB is a free and open-source database that employs a document-oriented data model and a non-structured query language. It is currently is among the most powerful NoSQL systems and databases available.

- MongoDB is indeed a document-oriented database. This means that it stores its data in collections of JSON-like documents rather than tables and rows. These documents support embedded fields, allowing related data to be stored within them.

- Since MongoDB is a schema-less database, we don't have to define the number or type of columns before inserting our data.



As we've seen, the document contains a number of fields (type, title, and so on) that contain values. These values can include strings, numbers, arrays, arrays of sub-documents (such as the categories field), geo-coordinates, and other data.

The _id is designated to be used as a primary key. Its value must be distinct within the collection, immutable, and of any type other than an array.

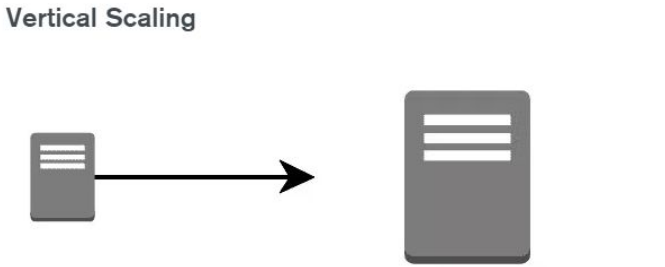
Scaling in MongoDB

As your application grows in size, each component of the application must scale in alongside with the size of your user base and data requirements. Traditionally, database scaling has been a major pain point for large applications or applications with high throughput, and options have been either limited in number or extremely costly to implement.

MongoDB, on the other hand, provides a comprehensive set of scaling options that are built into MongoDB Atlas, the company's database-as-a-service offering.

Now Let's look into some of different ways Mongo DB can scale

Vertical Scaling



Vertical scaling refers to increasing the processing power of a single server or cluster. Both relational and non-relational databases can scale up, but eventually, there will be a limit in terms of maximum processing power and throughput.

Horizontal Scaling

Horizontal scaling, also known as scale-out, refers to bringing on additional nodes to share the load.

This is difficult with relational databases due to the difficulty in spreading out related data across nodes.

With non-relational databases, this is made simpler since collections are self-contained and not coupled relationally. This allows them to be distributed across nodes more simply, as queries do not have to “join” them together across nodes.

Scaling can be achieved in Mongo DB with sharding and replica set

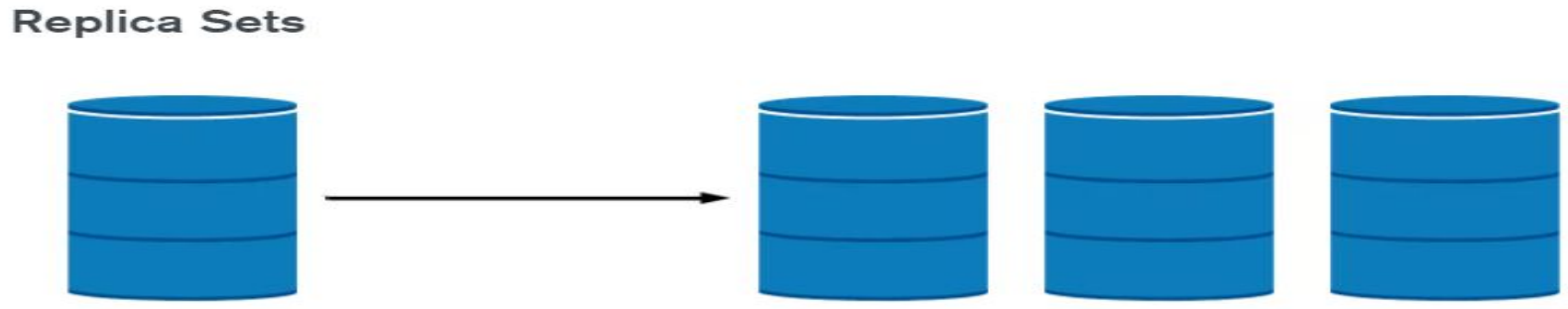
Sharding and ReplicaSet

Sharding

sharding is horizontal scaling by spreading data across multiple nodes. Each node contains a subset of the overall data. This is especially effective for increasing throughput for use cases that involve significant amounts of write operations, as each operation only affects one of the nodes and the partition of data it is managing.

ReplicaSet

Replica sets seem similar to sharding, but they differ in that the dataset is duplicated. Replication allows for high availability, redundancy/failover handling, and decreased bottlenecks on read operations. However, they can also introduce issues for applications with large amounts of write transactions, as each update must be propagated over to every replica set member.



Advantages

There are numerous advantages of MongoDB are:

1. **Open Source**: First of all and foremost, MongoDB is an open-source database with all of the features of open-source software such as licensing, customization, and so on.

1. **Horizontal scalability** MongoDB facilitates horizontal scaling with the help of database sharding. Since the data is structured horizontally, it becomes easy to spread it across different servers and access it in a simplified way. You can create clusters using real-time replications and shard high-volume data to sustain performance .

1. **Data replication**: MongoDB includes features that allow us to replicate data across multiple mirrored servers with minimal effort.

1. **Reliability**: With duplicated data on several servers, MongoDB has been shown to be trustworthy in protecting consumer data. It gives customers confidence that their data will be safe even if one server fails due to data replication on another server.

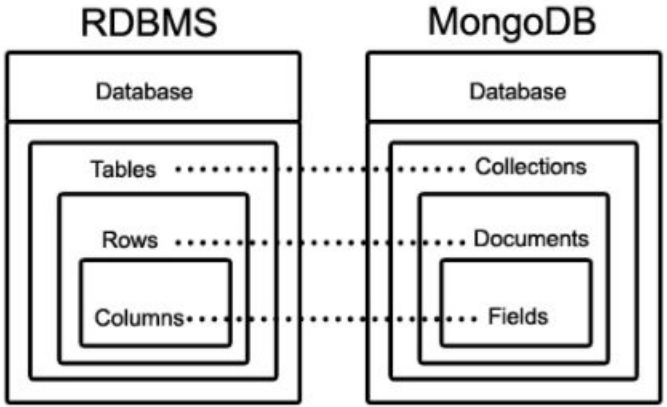
5. It offers a **dynamic schema design**, which allows documents in a collection to include different fields and structures. This is referred to as "Schema Free Design."

6. JSON-based document-oriented **NoSQL queries**, including data storage, are substantially faster than standard SQL queries.

Here's a table that summarizes the various terms:

SQL Server	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field
Index	Index

MongoDB Architecture's Key Components

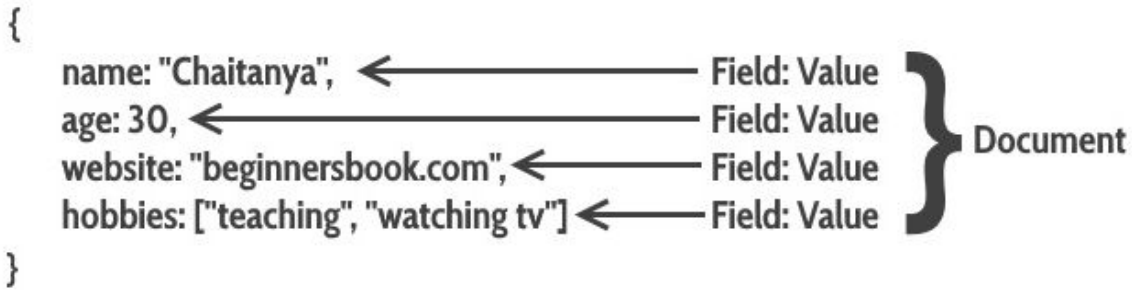


In Diagram, If you compare it with an RDBMS, documents correspond to the record (row) which have a unique identifier ID (primary key), fields/values are columns that have an associated key to them, and collections are the equivalent of a table.

- **Database:** In layman's terms, it is the physical container for data. Each database has its set of files on the file system, and several databases can exist on the same MongoDB server.
  It looks like this in Mongo Compass

- **Collection**: A collection is a group of database documents. A table is the RDBMS counterpart of a collection. A single database contains the entire collection. When it comes to collections, there are no schemas.
  Different documents within the collection may have different fields, but most documents within a collection are meant for the same purpose or to serve the same end objective.

- **Document:** A document is a collection of key-value pairs. Documents are linked to dynamic schemas. The advantage of dynamic schemas is that a document in a single collection does not have to have the same structure or fields.
  Furthermore, the common fields in a collection document might contain a variety of data types.



JSON and BSON

BSON

MongoDB employs records, which are made up of documents that include a data structure made up of field and value pairs. In MongoDB, the basic unit of data is the document. The documents resemble JavaScript Object Notation, however, they employ a version known as BSON.

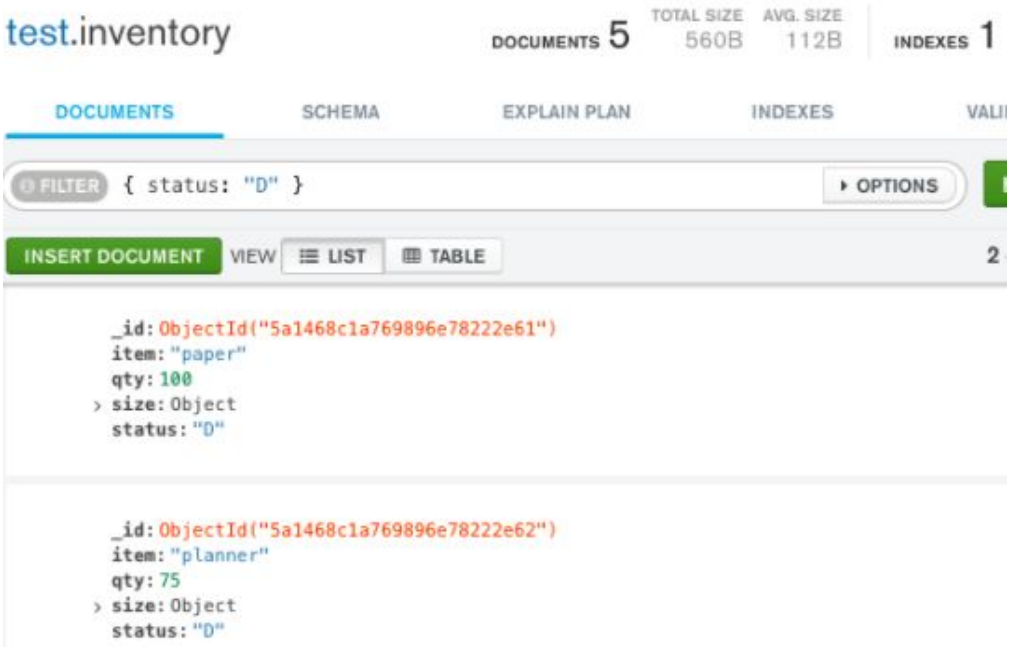
Binary JSON is abbreviated as BSON. It's a binary serialization protocol for JSON-like data.

BSON has various advantages over conventional JSON:

- Compact: In most circumstances, storing a BSON structure takes up less space than storing a JSON structure.

- Data Types: BSON includes data types that are not found in ordinary JSON, such as Date and BinData.

- One of the primary advantages of adopting BSON is that it is simple to traverse. BSON documents have additional metadata that allows for easy manipulation of a document's fields without having to read the entire content.


test.inventory
DOCUMENTS 5
TOTAL SIZE 560B
AVG. SIZE 112B
INDEXES 1
DOCUMENTS
SCHEMA
EXPLAIN PLAN
INDEXES
FILTER { status: "D" }
OPTIONS
INSERT DOCUMENT
VIEW
LIST
TABLE
_id: ObjectId("5a1468c1a769896e78222e61")
item: "paper"
qty: 100
size: Object
status: "D"
_id: ObjectId("5a1468c1a769896e78222e62")
item: "planner"
qty: 75
size: Object
status: "D"

Difference between JSON and BSON

JSON	BSON
JSON is an abbreviation for Javascript Object Notation.	BSON is an abbreviation for Binary Javascripts Object Notation.
The format of the file is standard.	The format of the file is Binary.
Language is organised independently and is used for server-browser communication by exchanging data.	A binary data transfer format including field titles, sort, and esteem. Typically, field titles and sorts are strings.
Indexes cannot be built.	Indexes can be built.
Slow	Fast
Uses less Space in comparison to BSON	Uses More Space as compared to JSON
It contains Basic data types	It gives extra data types

Practical Use Cases of Mongo DB

Some of the practical uses of mongo DB

1. **Content Management Systems**

   The method for modeling user comments on content like media and blog spots are introduced by “Storing Comments”.

   A model is proposed for designing a website content management system by “Metadata and Asset Management” in MongoDB.

1. **Product Data Management**

   For e-commerce websites and product data management and solutions, one can use MongoDB to store information because it has the flexible schema well suited for the job

   We can also manage a product catalog and learn the practices and methods for modeling from the Product Catalog document.

3. Operational Intelligence

MongoDB is beneficial for real-time analytics and operational intelligence use. One can learn “Storing Log Data” Document to know about the approaches and several ways to store and model machine-generated data with MongoDB.

They can also determine the Hierarchical Aggregation Method to store and process hierarchical reports from raw event data according to the minute, hour, or day.

Some Real-World Companies That Use MongoDB

1. **Aadhar**

   It is India's Unique Identification project, which has the biggest biometrics database in the world. Aadhar Project is using MongoDB as its database to store a massive amount of demographic and biometric data of more than 1.2 billion Indians.

1. **ShutterFly**

   Shutterfly is one of the most popular online photo sharing, and it is using MongoDB to manage and store more than 6 billion images, which has a transaction rate of up to 10,000 operations per second

1. **eBay**

   eBay is a multinational company that provides a platform for the customer to customer sales. It is currently running a large number of projects in MongoDB like merchandising categorization, cloud management, metadata storage, search suggestions.

4. **MetLife**

MetLife is a leading company in employee benefit programs, annuities, and insurance. There are more than 90 million customers in the Middle East, Europe, Aisa, Latin America, Japan, United States.MetLife is using MongoDB for its advanced customer service application called The Wall.This application provides a combined view of transactions, policy details, and other details of MetLife Customers

Installation of Mongo DB on Windows(20 min)

- Visit the official download center and get the version for your operating system. I've used Windows in this case.
- You'll go through a 'next after next' installation process after downloading MongoDB community server setup. After that, navigate to the C drive where you installed MongoDB. Navigate to Program Files and then to the MongoDB directory.

```
C: -> Program Files -> MongoDB -> Server -> 4.2(version) -> bin
```

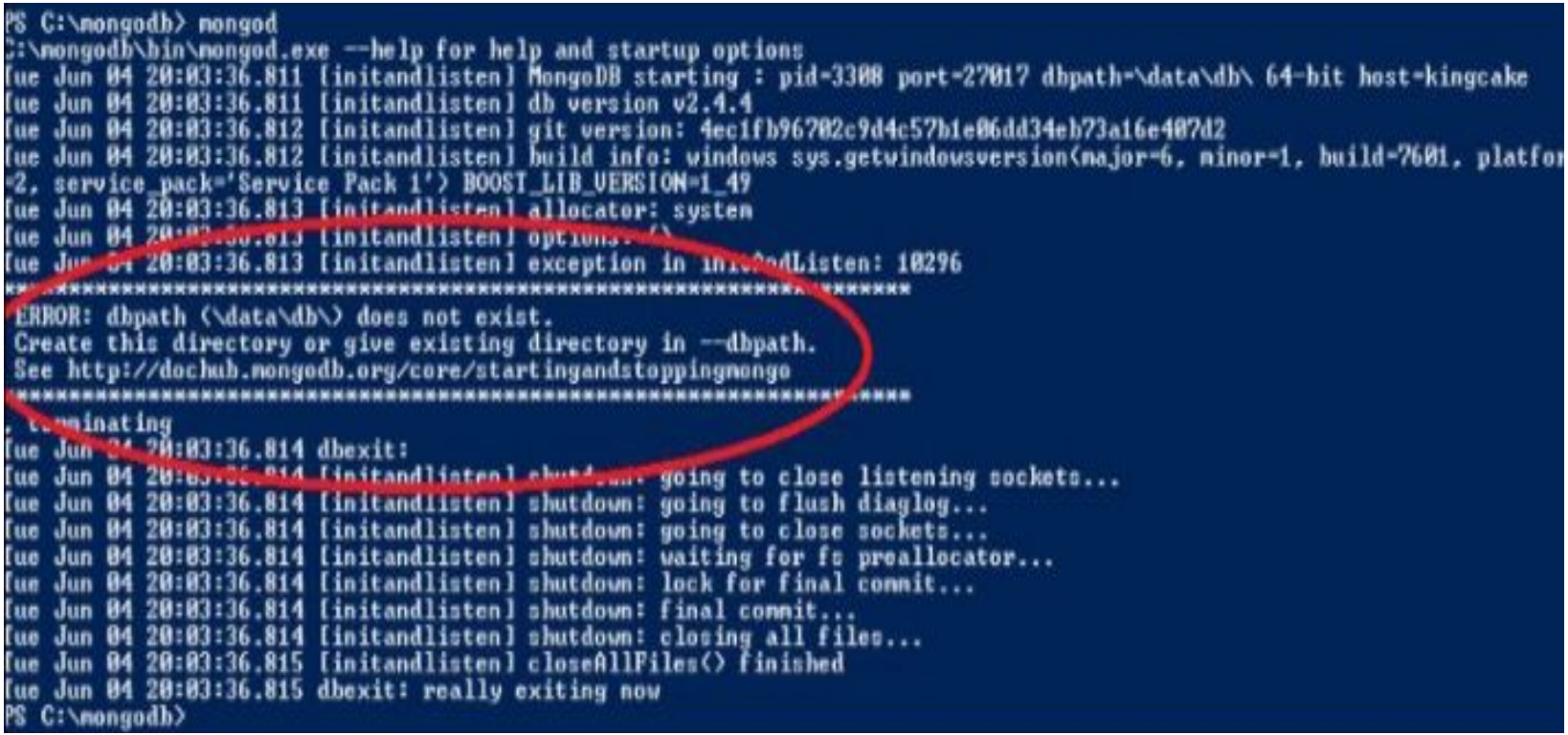
There are a few interesting executable files in the bin directory.
Mongod
Mongo

Mongod is an abbreviation for "Mongo Daemon." MongoDB's background process is mongod. Mongod's primary function is to manage all MongoDB server tasks. Accepting requests, responding to clients, and memory management are just a few examples

.**mongo** is a command-line shell that can communicate with clients (for example, system administrators and developers).These directories exist because MongoDB requires a folder to hold all data. MongoDB's default data directory path on the drive is /data/db.

These directories exist because MongoDB requires a folder to hold all data. MongoDB's default data directory path on the drive is /data/db.

If you run the MongoDB server without those directories, you will most likely encounter the following error:


PS C:\mongodb> mongod
C:\mongodb\bin\mongod.exe --help for help and startup options
Tue Jun 04 20:03:36.811 [initandlisten] MongoDB starting : pid=3308 port=27017 dbpath=\data\db\ 64-bit host=kingcake
Tue Jun 04 20:03:36.811 [initandlisten] db version v2.4.4
Tue Jun 04 20:03:36.812 [initandlisten] git version: 4ec1fb96702c9d4c57b1e06dd34eb73a16e407d2
Tue Jun 04 20:03:36.812 [initandlisten] build info: windows sys.getwindowsversion(major=6, minor=1, build=7601, platfor
=2, service_pack='Service Pack 1') BOOST_LIB_VERSION=1_49
Tue Jun 04 20:03:36.813 [initandlisten] allocator: system
Tue Jun 04 20:03:36.813 [initandlisten] options: {}
Tue Jun 04 20:03:36.813 [initandlisten] exception in initAndListen: 10296
ERROR: dbpath (\data\db\) does not exist.
Create this directory or give existing directory in --dbpath.
See http://dochub.mongodb.org/core/startingandstoppingmongo
Tue Jun 04 20:03:36.814 dbexit:
Tue Jun 04 20:03:36.814 [initandlisten] shutdown: going to close listening sockets...
Tue Jun 04 20:03:36.814 [initandlisten] shutdown: going to flush diaglog...
Tue Jun 04 20:03:36.814 [initandlisten] shutdown: going to close sockets...
Tue Jun 04 20:03:36.814 [initandlisten] shutdown: waiting for fs preallocator...
Tue Jun 04 20:03:36.814 [initandlisten] shutdown: lock for final commit...
Tue Jun 04 20:03:36.814 [initandlisten] shutdown: final commit...
Tue Jun 04 20:03:36.814 [initandlisten] shutdown: closing all files...
Tue Jun 04 20:03:36.815 [initandlisten] closeAllFiles() finished
Tue Jun 04 20:03:36.815 dbexit: really exiting now
PS C:\mongodb>

After you've created those two files, return to the bin folder in your MongoDB directory and launch your shell from within it. Execute this command:

```
mongod
```

Now MongoDB server is up and running

We require a mediator in order to work with this server. So, within the bin folder, start a new command window and execute the following command:

```
mongo
```

At the end, we'll see a 'connection accepted' message. That means our installation and configuration went fine!

Run

```
db
```

```
Navindu@Navindu MINGW64 ~/Desktop
$ mongo
MongoDB shell version v4.0.5
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("ea1b013a-7d7c-43a5-8e66-2e884a9a3105")
}
MongoDB server version: 4.0.5
db
test
```

Setting up Environment Variables

Go to Advanced System Settings

Click on Environment Variables ->

Click on Path(Under System Variables) and click on Edit

Simply paste the path to in bin folder and press OK!

In my case, it's C:\Program Files\MongoDB\Server\4.2\bin

Database Operations

Open a command prompt and type Mongo in shell

You will see something like this

```
[mj@localhost ~]$ mongo
MongoDB shell version v4.2.2
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiS
Implicit session: session { "id" : UUID("08a624a0-b330-4233-b56b-1d5b1
MongoDB server version: 4.2.2
```

Show list of databases

Command: show dbs

```
> show dbs

admin           0.000GB
config          0.000GB
local           0.000GB
```

Create Database

Command: use DATABASE_NAME

```
> use exampledb
switched to db exampledb
```

This will create a database

```
>db.dropDatabase()
{ "dropped" : "exampledb", "ok" : 1 }
```

You can find all the scripts in below github page

https://github.com/lavanya23/Into-to-MongoDB/blob/main/Database_Operations

Create Database

Command: use DATABASE_NAME

```
> use exampledb
switched to db exampledb
```

This will create a database

```
>db.dropDatabase()
{ "dropped" : "exampledb", "ok" : 1 }
```

You can find all the scripts in below github page

https://github.com/lavanya23/Into-to-MongoDB/blob/main/Database_Operations

Collection Operations

Creating Collection

Using the **use** command, navigate to your freshly formed database.

There are two ways to create a collection. Let's see both.

One method is to input into the collection.

```
db.myCollection.insert({"name": "john", "age" : 22, "location": "colombo"})
```

This will result in the creation of your collection "myCollection" . Even if the collection does not exist, Then it will insert a document containing the user's name and age. These are called non-capped collections.

The second way is shown below:

- Creating a Non-Capped Collection

```
db.createCollection("myCollection")
```

- Creating a Capped Collection

```
db.createCollection("mySecondCollection", {capped : true, size : 2, max : 2})
```

- In this manner, we will create a collection without inserting any data. A "capped collection" has a maximum document count that prevents documents from overflowing.
- I've enabled capping in this example by setting its value to true. The size: 2 specifies a limit of two megabytes, and the max: 2 specifies a maximum of two documents. If you try to insert more than two documents into mySecondCollection and then use the find command, we will only see the most recently inserted documents.

Show list of Collections

We can use show collections to get the list of collections

```
> show collections
users
```

Dropping a Collection

remove method is used to remove collection

```
db.myCollection.remove({});
```

This is not equal to the drop() method The difference is that drop() is used to remove all of the documents within a collection, whereas remove() is used to delete all of the documents as well as the collection itself.

https://github.com/lavanya23/Into-to-MongoDB/blob/main/Collection_operations.txt

CRUD Document operations on single and multiple docs

Inserting Data

We can add data to a new collection or a previously created collection.

There are three methods of inserting data.

- insertOne() is used to insert a single document only.
- insertMany() is used to insert more than one document.
- insert() is used to insert documents as many as you want.

Below are some examples:

- insertOne()

The insert() method and the insertMany() method are very similar.

Also, you'll notice that we've added a new property called location to the document for John Doe. So, if you us
see that the location property is only attached to John Doe.

When it comes to NoSQL databases like MongoDB, this can be advantageous. It enables scalability.
https://github.com/lavanya23/Into-to-MongoDB/blob/main/Insert.txt

Below are some examples:

- insertOne()

```
db.myCollection.insertOne(
  {
    "name": "navindu",
    "age": 22
  }
)
```

- insertMany()

```
db.myCollection.insertMany([
  {
    "name": "navindu",
    "age": 22
  },
  {
    "name": "kavindu",
    "age": 20
  },

  {
    "name": "john doe",
    "age": 25,
    "location": "colombo"
  }
])
```

Update Data

An update operation modifies documents in a collection. MongoDB provides several methods for updating a document, similar to the create operation. For example:

```
db.collection.updateOne(<filter>, <update>, <options>)
db.collection.updateMany(<filter>, <update>, <options>).
```

The **filter** is a document that specifies the condition for updating the document. If you pass the method an empty document {()}, it will update all of the documents in the collection.

The **update** is a document that specifies which updates should be applied.

If you need to add a new field — say, registration — to all of the existing documents in a collection, do the following:

```
db.users.updateMany({}, {$set: { registration: "incomplete"}})
```

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
```

We pass an empty object as the first argument if we want to update all documents in the collection. The $set operator is an update operator that replaces the value of a field with the value specified. You can confirm that the new field was added () by using db.users.find().

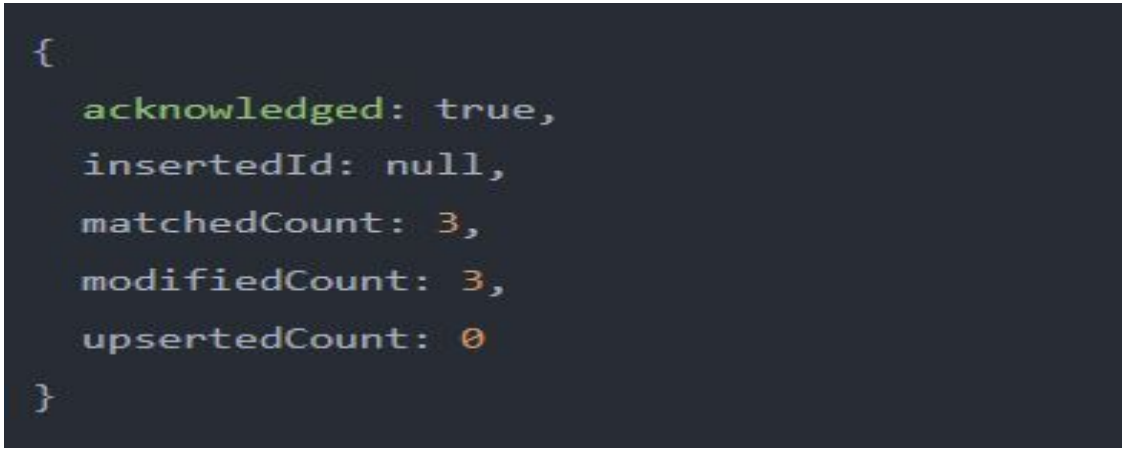
updateMany() takes a filter object as its first argument and updates the value of documents that meet certain criteria.For example, you could overwrite the value of registration to complete for all users over the age of 19. What you can do is as follows.

```
db.users.updateMany(

  {age:{ $gt: 19} },

  {$set: { registration:
"complete"}

})
```



We can use the **findOneUpdate** to update and return the document

db.collection.findOneAndUpdate() updates the collection's first matching document that matches the filter. If no documents match the filter, none are updated.

The following operation locates the first document containing the name R. Stiles and increases the score by 5.

The following operation locates the first document containing the name R. Stiles and increases the score by 5.

```
{ _id: 6305, name : "A. MacDyver", "assignment" : 5, "points" : 24 },
{ _id: 6308, name : "B. Batlock", "assignment" : 3, "points" : 22 },
{ _id: 6312, name : "M. Tagnum", "assignment" : 5, "points" : 30 },
{ _id: 6319, name : "R. Stiles", "assignment" : 2, "points" : 12 },
{ _id: 6322, name : "A. MacDyver", "assignment" : 2, "points" : 14 },
{ _id: 6234, name : "R. Stiles", "assignment" : 1, "points" : 10 }
```

```
db.grades.findOneAndUpdate(
   { "name" : "R. Stiles" },
   { $inc: { "points" : 5 } }
)
```

{ _id: 6319, name: "R. Stiles", "assignment" : 2, "points" : 17}

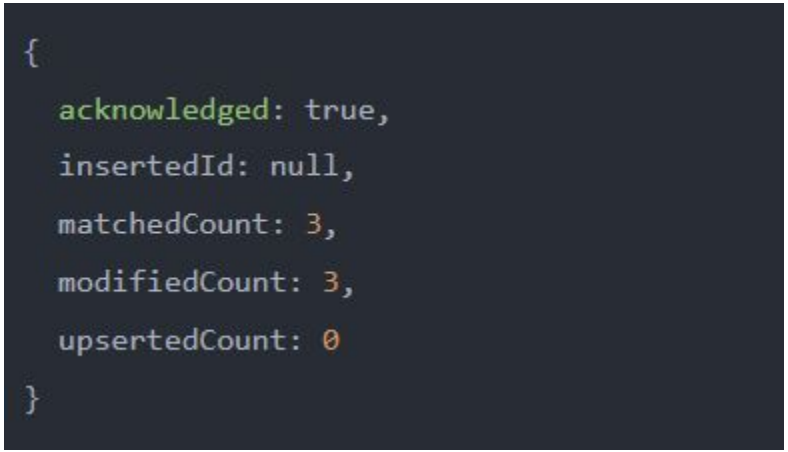
We can do the following to update a single user's registration information:

```
db.users.updateOne(

 {email:
"tom12@example.com" },

 {$set: { registration:
"complete"}

}
```



https://github.com/lavanya23/Into-to-MongoDB/blob/main/Update.txt

Update Operators

Fields

Name	Description
`$currentDate`	Sets the value of a field to current date, either as a Date or a Timestamp.
`$inc`	Increments the value of the field by the specified amount.
`$min`	Only updates the field if the specified value is less than the existing field value.
`$max`	Only updates the field if the specified value is greater than the existing field value.
`$mul`	Multiplies the value of the field by the specified amount.
`$rename`	Renames a field.
`$set`	Sets the value of a field in a document.
`$setOnInsert`	Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.
`$unset`	Removes the specified field from a document.

Update Operators on Array

Array

Operators

Name	Description
`$`	Acts as a placeholder to update the first element that matches the query condition.
`$[]`	Acts as a placeholder to update all elements in an array for the documents that match the query condition.
`$[<identifier>]`	Acts as a placeholder to update all elements that match the `arrayFilters` condition for the documents that match the query condition.
`$addToSet`	Adds elements to an array only if they do not already exist in the set.
`$pop`	Removes the first or last item of an array.
`$pull`	Removes all array elements that match a specified query.
`$push`	Adds an item to an array.
`$pullAll`	Removes all matching values from an array.

Operators on Modifiers and Bitwise

Modifiers

Name	Description
`$each`	Modifies the `$push` and `$addToSet` operators to append multiple items for array updates.
`$position`	Modifies the `$push` operator to specify the position in the array to add elements.
`$slice`	Modifies the `$push` operator to limit the size of updated arrays.
`$sort`	Modifies the `$push` operator to reorder documents stored in an array.

Bitwise

Name	Description
`$bit`	Performs bitwise `AND`, `OR`, and `XOR` updates of integer values.

Delete Data

A delete operation is used to remove a document from a collection. To delete a single document, use the db.collection.deleteOne(filter>, options>) method, and to delete multiple documents, use the db.collection.deleteMany(filter>, options>) method.

```
db.collection.deleteOne({_id: 1})
```

To delete documents based on certain criteria, we can use the filter operators that we used for the read and update operation:

```
db.users.updateOne(
{email: "tom@example.com" },
{$set: { status: "dormant"}
})
```

{ "acknowledged" : true, "matchedCount" : 4, "modifiedCount" : 4 }

```
> db.users.deleteMany( { status: { $in: [ "dormant",
"inactive" ] } } )


{ "acknowledged" : true, "deletedCount" : 1 }
```

https://github.com/lavanya23/Into-to-MongoDB/blob/main/Delete.txt

Querying Data

A read operation is used to retrieve a single document or a set of documents from a collection. The read operation has the following syntax:

```
> db.collection.find(query, projection)
```

Please find below command to retrieve all user documents:

```
> db.users.find().pretty()
{
  "_id" : ObjectId("5e25bb58ba0cf16476aa56ff"),
  "name" : "Tom",
  "age" : 15,
  "email" : "tom@example.com"
}
{
```


"_id" : ObjectId("5e25bb58ba0cf16476aa5700"),
"name" : "Bob",
"age" : 35,
"email" : "bob@example.com"
}
{
"_id" : ObjectId("5e25bb58ba0cf16476aa5701"),
"name" : "Kate",
"age" : 27,
"email" : "kate@example.com"
}
{
"_id" : ObjectId("5e25bb58ba0cf16476aa5702"),
"name" : "Katherine",
"age" : 65,
"email" : "katherine@example.com"
}

In a SQL database, this corresponds to the SELECT * FROM USERS query.

There are many cursor methods, including the pretty method. We can chain these methods together to modify both our query and the documents returned by the query.

Count.count() returns the collection's total number of items in document.

We must filter queries in order to return a subset of the collection, such as finding all users who are under the age of 30. You can change the query as follows:

```
> db.users.find({ age: { $lt: 30 } })

{ "_id" : ObjectId("5e25bb58ba0cf16476aa56ff"), "name" : "Tom", "age" : 15, "email" :
"tom@example.com" }

{ "_id" : ObjectId("5e25bb58ba0cf16476aa5701"), "name" : "Kate", "age" : 27, "email" :
"kate@example.com" }
```

In this case, $lt is a query filter operator that selects documents with age fields that are less than 30. There are numerous comparison and logical query filters to choose from.

Using a regex, we can replicate SQL-like queries in Mongo. SELECT * FROM users WHERE name LIKE 'Kat percent ', for example, translates to db.users.find(name: /Kat.*/).

https://github.com/lavanya23/Into-to-MongoDB/blob/main/Querying.txt

Assume you want to show people who are under the age of 25 and live in Colombo.

```
db.myCollection.find({$and:[{age : {$lt : 25}}, {location: "colombo"}]});
```

Mongo DB _id/ObjectId(10 min)

Every document in the collection has an “_id” field that is used to uniquely identify the document in a particular collection it acts as the primary key for the documents in the collection. “_id” field can be used in any format and the default format is ObjectId of the document.

An ObjectID is a 12-byte Field Of BSON type

- The first 4 bytes representing the Unix Timestamp of the document
- The next 3 bytes are the machine Id on which the MongoDB server is running.
- The next 2 bytes are of process id
- The last Field is 3 bytes used for increment the objectid.

```
ObjectId(<hexadecimal>)
```

ObjectId format:-ObjectId takes a single parameter, which is an optional Hexadecimal ObjectId in String. We can give the document our own ObjectId, but it must be unique.

ObjectId Methods:

str: Returns the ObjectId's hexadecimal string format.
ObjectId.getTimestamp(): This method returns the object's timestamp as a Date.
ObjectId.valueOf(): This function returns the hexadecimal format of a String Literal.
ObjectId.toString(): This function returns the ObjectId in String format in javascript.

ObjectId Creation

```
newObjectId = ObjectId()
```

Output:

```
ObjectId("5f92cbf10cf217478ba93561")
```

```
> use gfg
switched to db gfg
> newobjectId=ObjectId()
ObjectId("5f92cbf10cf217478ba93561")
>
```

Timestamp

```
var id =new ObjectId();
id.getTimestamp()
```

Output:

```
ISODate("2020-10-23T12:32:42Z")
```

```
> use gfg
switched to db gfg
> var id=new ObjectId()
> id.getTimestamp()
ISODate("2020-10-23T12:32:42Z")
>
```

Converting ObjectId to string

```
new ObjectId().str
```

Output:

```
5f92cdce0cf217478ba93563
```

```
> use gfg
switched to db gfg
> new ObjectId().str
5f92cdce0cf217478ba93563
>
```

Query and Projection Operators

The comparison, logical, element, evaluation, Geospatial, array, bitwise, and comment operators are all available in the MongoDB query operator.

MongoDB Comparison Operators

1. **$eq**

   The equality condition is specified by $eq. It finds documents in which the value of a field equals the specified value.

Syntax:

```
{ <field> : { $eq: <value> } }
```

Example

```
db.books.find ( { price: { $eq: 300 } } )
```

The preceding example searches the books collection for all documents where the price filed equals 300.

1. **$gt**

The $gt selects a document whose field value is greater than the specified value.

Syntax:

```
{ field: { $gt: value } }
```

Example:

```
db.books.find ( { price: { $gt: 200 } } )
```

$gte

The $gte selects documents whose field value is greater than or equal to a given value.

Syntax:

```
{ field: { $gte: value } }
```

Example:

```
db.books.find ( { price: { $gte: 250 } } )
```

$in

The $in operator selects documents where a field's value equals any value in the specified array.

Syntax:

```
{ filed: { $in: [ <value1>, <value2>, .....] } }
```

Example:

```
db.books.find( { price: { $in: [100, 200] } } )
```

$lt

The $lt operator selects documents whose field value is less than the specified value.

Syntax:

```
{ field: { $lt: value } }
```

Example

```
db.books.find ( { price: { $lt: 20 } } )
```

$lte

The $lte operator selects documents whose field value is less than or equal to a given value.

Syntax:

```
{ field: { $lte: value } }
```

Example

```
db.books.find ( { price: { $lte: 250 } } )
```

$ne

The $ne operator selects documents in which the field value does not equal the specified value.

Syntax:

```
{ <field>: { $ne: <value> } }
```

Example:

```
db.books.find ( { price: { $ne: 500 } } )
```

$nin

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

Syntax:

```
{ field : { $nin: [ <value1>, <value2>, .... ] } }
```

Example:

```
db.books.find ( { price: { $nin: [ 50, 150, 200 ] } } )
```

MongoDB Logical Operator

1. $and

On an array, the $and operator performs a logical AND operation. The array should contain one or more expressions, and it selects documents that satisfy all of the expressions in the array.

Syntax:

```
{ $and: [ { <exp1> }, { <exp2> }, ....]}
```

Example

```
db.books.find ( { $and: [ { price: { $ne: 500 } }, { price: { $exists: true } } ] } )
```

$not

The $nor operator acts as a logical NOR on an array of one or more query expressions, selecting documents that fail all of the query expressions in the array.

Syntax:

```
{ $nor: [ { <expression1> } , { <expresion2> } , ..... ] }
```

Example:

```
db.books.find ( { $nor: [ { price: 200 }, { sale: true } ] } )
```

$or

It operates as a logical OR operation on an array of two or more expressions, selecting documents that satisfy at least one of the expressions.

Syntax:

```
{ $or: [ { <exp_1> }, { <exp_2> }, ... , { <exp_n> } ] }
```

Example:

```
db.books.find ( { $or: [ { quantity: { $lt: 200 } }, { price: 500 } ] } )
```

MongoDB Element Operator

1. $exists

When the Boolean is true, the exists operator matches the documents that contain the field. It also matches documents with null field values.

Syntax:

```
{ field: { $exists: <boolean> } }
```

Example:

```
db.books.find ( { qty: { $exists: true, $nin: [ 5, 15 ] } } )
```

$type

The type operator selects documents whose field value is an instance of the specified BSON type.

Syntax:

```
{ field: { $type: <BSON type> } }
```

Example:

```
db.books.find ( { "bookid" : { $type : 2 } } );
```

MongoDb Evaluation Operators

Evaluation

Name	Description
`$expr`	Allows use of aggregation expressions within the query language.
`$jsonSchema`	Validate documents against the given JSON Schema.
`$mod`	Performs a modulo operation on the value of a field and selects documents with a specified result.
`$regex`	Selects documents where values match a specified regular expression.
`$text`	Performs text search.
`$where`	Matches documents that satisfy a JavaScript expression.

$expr

Syntax:

```
{ $expr: { <expression> } }
```

Example:

```
db.store.find( { $expr: {$gt: [ "$product" , "price" ] } } )
```

$jsonSchema

Syntax:

```
{ $jsonSchema: <JSON schema object> }
```

$mod

Syntax:

```
{ field: { $mod: [ divisor, remainder ] } }
```

Example:

```
db.books.find ( { qty: { $mod: [ 200, 0] } } )
```

$regex

Syntax:

```
{ <field>: /pattern/<options> }
```

$text

Syntax:

```
{
  $text:
   {
     $search: <string>,
     $language: <string>,
     $caseSensitive: <boolean>,
     $diacriticSensitive: <boolean>
   }
}
```

Example:

```
db.books.find( { $text: { $search: "Othelo" } } )
```

$where

```
db.books.find( { $where: function() {
  return (hex_md5(this.name)== "9b53e667f30cd329dca1ec9e6a8")
} } );
```

GeoSpatial operators

Geospatial

Name	Description
`$geoIntersects`	Selects geometries that intersect with a GeoJSON geometry. The 2dsphere index supports `$geoIntersects`.
`$geoWithin`	Selects geometries within a bounding GeoJSON geometry. The 2dsphere and 2d indexes support `$geoWithin`.
`$near`	Returns geospatial objects in proximity to a point. Requires a geospatial index. The 2dsphere and 2d indexes support `$near`.
`$nearSphere`	Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The 2dsphere and 2d indexes support `$nearSphere`.

$geoIntersects

```
{
  <location field>: {
    $geoIntersects: {
      $geometry: {
        type: "<object type>" ,
        coordinates: [ <coordinates> ]
      }
    }
  }
}
```

Example:

```
          db.places.find(
{
  loc: {
    $geoIntersects: {
      $geometry: {
        type: "Triangle" ,
        coordinates: [
          [ [ 0, 0 ], [ 3, 6 ], [ 6, 1 ] ]
        ]
      }
    }
  }
}
```

$geoWithin

Syntax:

```
        {
<location field>: {
  $geoWithin: {
    $geometry: {
      type: <"Triangle" or "Rectangle"> ,
      coordinates: [ <coordinates> ]
    }
  }
}
```

$near

Syntax:

```
    {
<location field>: {
 $near: {
  $geometry: {
    type: "Point" ,
    coordinates: [ <longitude> , <latitude> ]
  },
  $maxDistance: <distance in meters>,
  $minDistance: <distance in meters>
 }
}
```

Example:

```
db.places.find(
{
  location:
    { $near :
      {
        $geometry: { type: "Point",  coordinates: [ -73.9667, 40.78 ] },
        $minDistance: 1000,
        $maxDistance: 5000
      }
    }
}
```

$nearSphere

Syntax:

```
{
  $nearSphere: [ <x>, <y> ],
  $minDistance: <distance in radians>,
  $maxDistance: <distance in radians>
}
```

Example:

```
db.legacyPlaces.find(
  { location : { $nearSphere : [ -73.9667, 40.78 ], $maxDistance: 0.10 } }
)
```

MCQs

1. What exactly is MongoDB?

A. An Access database.
B. A SQL database.
C. A file system.
D. A NoSQL database.

Answer D

2. Which of the following is not a NoSQL database ?

A. SQL
B. MONGODB
C. CASSANDRA

Answer A

3. Command to create database in mongodb?

A. create databaseName
B. use databaseName
C. new databaseName
D. None of the above

Answer B

4. Examples of NoSQL Database Type

A. MONGODB
B. JSON
C. CASSANDRA
D. A & C

Answer D

5. How to start Mongo DB Server in window OS?

A. mongod
B. mongo
C. start-mongo
D. start-mongo.sh

Answer A

6. How many bits involve in _id?

A. 9
B. 10
C. 11
D. 12

Answer D

7. MongoDB supports which of the following formats?

A. XML
B. BSON
C. SQL
D. All

Answer B

8. What is the name of MongoDB's interactive shell?

A. mongo
B. mongodb
C. dbmong
D. None

Answer A

9. Which of the following will provide information about the sName "Lavk"?

A. db.Students.find({sName: "Lavk"},{"sName":1, "GPA":1, "_id":0})
B. db.Students.find({sName: "Lavk"},{"sName":1, "GPA":0})
C. db.Students.find({sName: Lavk})
D. db.Students.find({sName: "Lavk"})

Answer D

10. What is the MongoDB equivalent of the following SQL query? SELECT * FROM posts WHERE author matches " percent rose% "

A. db.posts.find({ author: /rose/ })
B. db.posts.find({ author: {$like: /rose/} })
C. db.posts.find({ $like: {author: /rose/} })
D. db.posts.find({ author: /^rose^/ })

Answer A

Practice Problems

- Create a database with name “example_db” and collection “shibs” and insert object like this

```
{name:'USS
EnterpriseD',operator:'Starfleet',type:'Explorer',class:'Galaxy',crew:750,codes:[10,11,12]} .

{name:'USS Prometheus',operator:'Starfleet',class:'Prometheus',crew:4,codes:[1,14,17]}

{name:'USS Defiant',operator:'Starfleet',class:'Defiant',crew:50,codes:[10,17,19]}

{name:'IKS Buruk',operator:' Klingon Empire',class:'Warship',crew:40,codes:[100,110,120]}

({name:'IKS Somraw',operator:' Klingon Empire',class:'Raptor',crew:50,codes:[101,111,120]}

{name:'Scimitar',operator:'Romulan Star
Empire',type:'Warbird',class:'Warbird',crew:25,codes:[201,211,220]}

{name:'Narada',operator:'Romulan Star
Empire',type:'Warbird',class:'Warbird',crew:65,codes:[251,251,220]}
```

THANK YOU