**What is Mongodb ?**

* MongoDB is a document-oriented NoSQL database which is used to store huge data as documents. It has collection just like tables in relational databases. It has no schema. We can use JSON object to store data here but behind the scenes mongodb server stores this json into binary format.

**What is mongod?**

* It is a ececutable file, used to start the mongodb server locally

**What is mongo/mongosh?**

* It is a mongodb shell, used to connect to mongodb to execute our queries.

We can specify the location where we want to save our data in local. But it should have data and logs folder inside it. Then start the server like the following:

*mongod --dbpath /path/data --logpath /path/logs/mongo.log*

In Windows there is an option to start mongodb as a service so it will be running all the time in background.

**How do I start/stop MongoDB from running in the background?**

* One liner to start or stop mongodb service using command line.

1. To start the service use: *NET START MONGODB.*

2. To stop the service use: *NET STOP MONGODB.*

**Command to show all the database**: *show dbs*

**Create or use a dtabase**: *use <db\_name>*

**To use a collection and store one data**: *db.products.insertOne({name:"Abhishek Ghosh",age:24})* it will create a document in products collection. After inserting one document it will give one id and acknowledgement. We can also insert nested documents.

**To show all the datas in products collection use this command**: *db.products.find()*

**To show it in a json structure:** *db.products.find().pretty()*

By default mongodb adds an unique id which is of type ObjectId to every document and we can search items with that and also mongodb create one default index with this \_id by default. We can also add our \_id like the following

*db.products.insertOne({\_id:"abhishek-test-0001",name:"Abhishek Ghosh"})*

**To search any document using \_id**: *db.products.find({\_id:ObjectId('62a6ff6edb132197c5e887a0')})*

Mongodb uses BSON instead of JSON to store data.

CRUD Operations

Create operations:

* insertOne(data, options) -> for inserting one item
* insertMany(data, options) -> for inserting multiple items

Read operations:

* find(filter, options) -> find all the data based on the filter
* findOne(filter, options) -> find the first matching element based on the filter

Update operations:

* updateOne(filter, data, options) -> to update one document
* updateMany(filter, data, options) -> for updating multiple documents
* replaceOne(filter, data, options) -> for replacing the entire document

Delete operations:

* deleteOne(filter, options) -> delete only the first item with matching filter
* deleteMany(filter, options) -> delete all items matching with the filter

**Delete the first element with name with “Abhishek Ghosh”** -> *db.products.deleteOne({name:"Abhishek Ghosh"})*

**Update the age to 24 where name is “Abhishek Pal”** -> *db.products.updateOne({name:"Abhishek Pal"},{$set:{age:24}})*

**Add a field height to all the documents** -> *db.products.updateMany({},{$set:{height:"Unknown"}})*

**{} this means all the documents**

**Insert two items at a time ->**

*db.products.insertMany(*

*... [{name:"Nasim Molla",*

*... age:25},*

*... {name:"Sayan Mandal",*

*... age: 24}])*

**Find all the students whose age is greater than 24** -> *db.products.find({age:{$gt:24}})*

**Print all the names for the student whose age is greater than 24 (no \_id)** -> *db.products.find({age:{$gt:24}},{"name":1,\_id:0})*

**If we use update without $set then the document will be replaced with the data we have provided**.( Rather use replace than update for full replacement)

> **db.products.insertOne({})**

{"acknowledged" : true,"insertedId" : ObjectId("62a7faec7866653913689afd")}

> **db.products.update({\_id:ObjectId("62a7faec7866653913689afd")},{name:"Anirban Ghosh",age:23})**

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> **db.products.find({\_id:ObjectId("62a7faec7866653913689afd")})**

{ "\_id" : ObjectId("62a7faec7866653913689afd"), "name" : "Anirban Ghosh", "age" : 23 }

**What is cursor?**

* When we find anything with shell rather than giving everything in one shot it gives us the cursor of 20 elements and to move to the next 20 we have to enter “it”. To see it we can use toArray method on the cursor which will exhaust the cursor and make one array with all the elements and show that.
* Cursor will fetch only the needed element.
* findOne will not give us cursor object as it will only give us one element.
* db.products.find().toArray()
* db.products.find().forEach((doc)=>{printjson(doc)})

**What is projection?**

* Rather than show all the fields of a document we can choose whatever we want to show.
* It will also helps us to reduce the bandwidth usage as server will not send all the elements.
* To get all the student with age is 24 : *db.products.find({age:24},{"name":1})*
* By default \_id is set to 1 so if we want to remove is as well we have to use this type of query. *db.products.find({age:24},{"name":1,\_id:0})*

**One Document can maximum hold 100 level of nesting**

**Maximum size of document can be 16 mb**

**Set status object for age greater than 24** -> *db.products.updateMany({age:{$gt:24}},{$set:{status:{married:false,single:false}}} )*

I**f we have a list of strings then like hobbies then we can search like this (It will find the first document that has a list of hobbies containing “Drama**” -> *db.products.findOne({hobbies:”Drama”})*

**We can we run a query in nested object** -> *db.products.findOne({“status.single”: false})*