Project -11

“MongoDB”

SUBMITTED BY:

MANASWI M. PATIL

INDEX

1. INTRODUCTION

MongoDB is a popular, open-source, NoSQL database management system designed for scalability, flexibility, and performance. It stores data in a JSON-like format called BSON (Binary JSON). MongoDB is widely used in modern web applications. MongoDB is a document database and can be installed locally or hosted in the cloud.

JSON uses a simple and readable syntax, including key-value pairs enclosed in curly braces **{}**. Each key is followed by a colon **:** and its associated value. Multiple key-value pairs are separated by commas. It is used to Transmit data between server and a web application.

```
{
  title: "Post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"],
  date: Date()
}
```

Features of Mongodb:

1. **Document-Oriented Model:** MongoDB stores data in flexible, JSON-like BSON documents. Each document can have a unique structure, providing greater flexibility compared to traditional relational databases.
2. Schema-less Design: MongoDB does not require a predefined schema, allowing for dynamic and evolving data models.
3. **Scalability:** MongoDB supports horizontal scaling through sharding, allowing for the distribution of data across multiple servers. This ensures that the database can handle growing amounts of data and increased traffic.
4. **Indexing:** MongoDB supports the creation of indexes on any field, facilitating fast and efficient data retrieval. Indexing can significantly improve query performance.
5. **Query Language:** MongoDB provides a rich and expressive query language for interacting with the database. Queries can include filtering, sorting, and projection of specific fields.

MongoDB components:

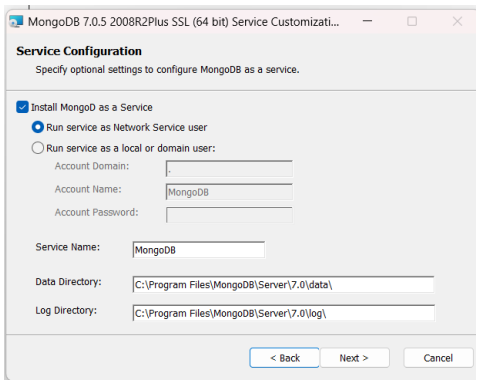
1. **MongoDB Server:** The core server component responsible for storing and managing data.

2. **MongoDB Storage Engine**: Responsible for managing the storage and retrieval of data on disk.
3. **Collections:** MongoDB organizes data into collections, which are analogous to tables in relational databases.
4. **Documents:** The basic unit of data in MongoDB. Documents are JSON-like BSON (Binary JSON) objects that store data in key-value pairs. They can have nested structures, arrays, and support a wide range of data types.
5. **Fields:** Each field in a document has a name (key) and a value.
6. **Indexes:** MongoDB supports the creation of indexes on fields within a collection. Indexes improve query performance by allowing the database to quickly locate and retrieve specific documents.
7. **Queries:** MongoDB provides a powerful and flexible query language for retrieving and manipulating data. Queries can include filtering, sorting, and projection of specific fields.
8. **Aggregation Framework:** A powerful and expressive framework for performing data transformations and computations within the database. Supports operations such as grouping, sorting, filtering, and projecting.
9. **Replica Sets:** A mechanism for providing high availability and fault tolerance by maintaining multiple copies (replicas) of data across different servers. Consists of a primary node and one or more secondary nodes.
10. **Sharding:** MongoDB's sharding feature allows horizontal scaling by distributing data across multiple servers or clusters.
11. **MongoDB Atlas:** MongoDB Atlas is the official cloud-based, fully managed MongoDB service provided by MongoDB, Inc.
12. **MongoDB Compass:** Compass allows users to view and analyse the structure of their data, build queries, and perform administrative tasks.

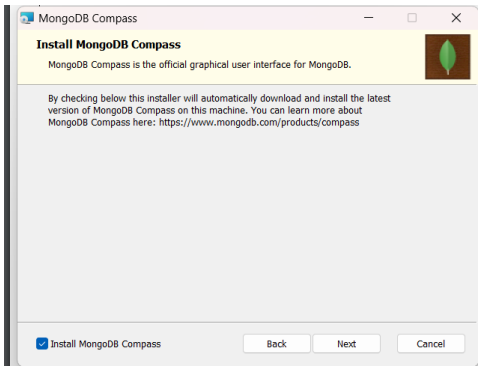
2. INSTALLATION

MongoDB database community editor:

- Download the MongoDB Community **.msi** installer from the following link: https://www.mongodb.com/try/download/community →Click on ‘Select Package’→Select Version →Platform →Package→Download→Run the installer.
- Select configuration→select type →next.

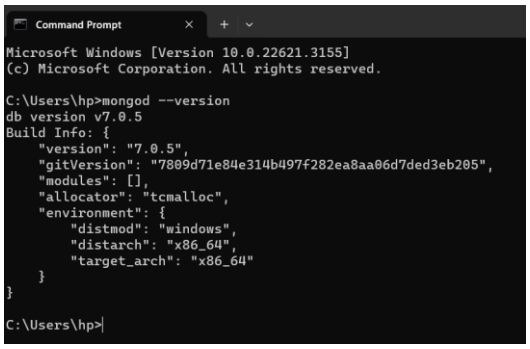


- **Install MongoDB compass**



Verify the version→ Go to c drive→program files→mongodb→server→ double click version 7.0 →bin→right click copy path→go to start button→type environment variable for system→environment variable→system variable→double click on path→new→paste the path you have copied from bin folder→OK.

- Verify the version using command **“mongod – version”.**



“MongoDB”

Installing MongoDB Shell:

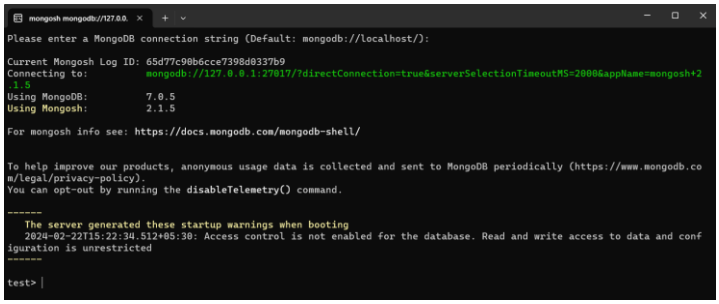
- Download the installer→click on this link→ https://www.mongodb.com/try/download/shell → Select Version → Platform → Package(zip) → Download.

Extract the zip file of mongoDB shell:

- Extract the zip file to the location where you have installed the MongoDB server→Open mongosh-2.1.3 folder→right click and extract all→ Go to the bin folder→ Copy path →Search for ‘Edit environment variable’ →System variable → Path→ Edit →Paste the bin folder path→OK →click on file from windows where you copiwd your install folder→open mongosh→click more option→Run anyway option.

Connect to localhost:

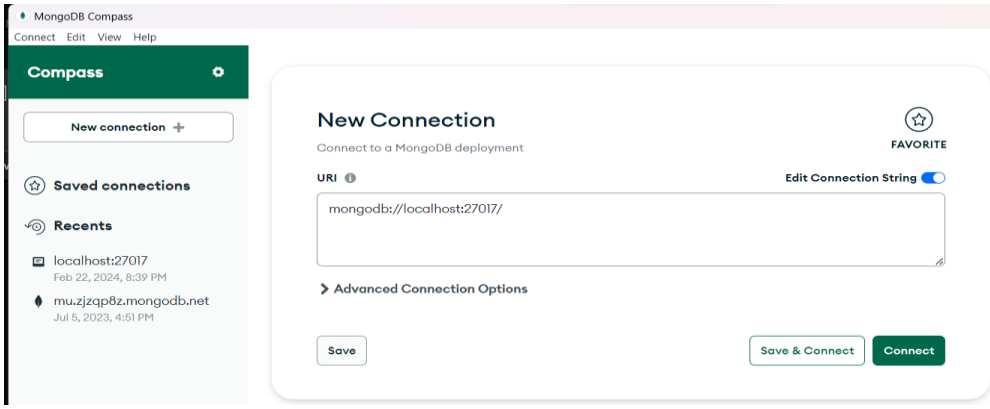
- Open cmd→type command as **“mongosh” or just enter.**



Installing MongoDB Compass:

- Go to→ https://www.mongodb.com/try/download/compass →Download exe file →Run the installer.

Connect to Localhost:



MongoDB Atlas: No need to install it.

- Go to → https://www.mongodb.com/cloud/atlas → and signup for free.

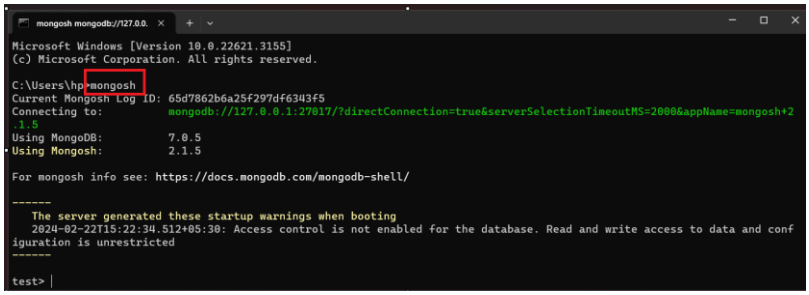
3. MONGODB CRUD OPERATIONS

MongoDB supports CRUD (Create, Read, Update, Delete) operations to interact with data in the database.

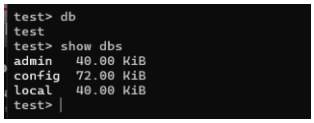
1. Create Database:

Open cmd → type “mongosh” to connect with MongoDB Server→You can use the “use” command to switch to an existing database or create a new

one.



- **Db** command show the connect database.
- **Show dbs** command shows the databases list which you have connected.



Notice that “db” is not listed. This is because the database is empty. An empty database is essentially non-existant.

- In MongoDB, a database is not actually created until it gets content. We need to create collections inside the database.

2. Create Collection:

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

```
db> db.createCollection("students")
{ ok: 1 }
```

This will create “student” collection.

Insert(): You can also create a collection during the insert process.

```
db> db.student.insertOne({"name":"manu"});
{
  acknowledged: true,
  insertedId: ObjectId('65d788cb6a25f297df6343f6')
}
```

Use **show collections** to see the list of collections.

```
db> show collections
student
students
db>
```

“MongoDB”

- **CREATE:** There are 2 methods to insert documents into a MongoDB database.

1. **insertOne():** Used to insert a single document.

```
db> db.student.insertOne({ id:1 ,name:"manu",address:"palghar"});
{
  acknowledged: true,
  insertedId: ObjectId('65d78a916a25f297df6343f7')
}
db>
```

2. **insertMany():** Used to insert multiple documents at once.

```
test> db.student.insertMany([{ id:2, name:"Manu", address:"Palghar"}, { id:3, name:"Hinali", address:"Mumbai"},{id:4, name:"Himali", address:"Boisar"}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65d8949e9f19091797532ed4'),
    '1': ObjectId('65d8949e9f19091797532ed5'),
    '2': ObjectId('65d8949e9f19091797532ed6')
  }
}
```

- **READ (Query Documents):**There are 2 methods to find and select data from a MongoDB collection:

  1. **find ():** Used to select data from a collection.

```
test> db.student.find();
[
  {
    _id: ObjectId('65d8943b9f19091797532ed2'),
    id: 2,
    name: 'Manu',
    address: 'Palghar'
  },
  {
    _id: ObjectId('65d8943b9f19091797532ed3'),
    id: 3,
    name: 'Hinali',
    address: 'Mumbai'
  },
  {
    _id: ObjectId('65d8949e9f19091797532ed4'),
    id: 2,
    name: 'Manu',
    address: 'Palghar'
  },
  {
    _id: ObjectId('65d8949e9f19091797532ed5'),
    id: 3,
    name: 'Hinali',
    address: 'Mumbai'
  },
  {
    _id: ObjectId('65d8949e9f19091797532ed6'),
    id: 4,
    name: 'Himali',
    address: 'Boisar'
  }
]
```

2. findOne(): Used to select only one document. If left empty, it will return the first document it finds.

```
test> db.student.findOne();
{
  _id: ObjectId('65d8943b9f19091797532ed2'),
  id: 2,
  name: 'Manu',
  address: 'Palghar'
}
```

3.use.count() to count the no. of documents.

4. **Querying Data:** To query, or filter, data we can include a query in our find() or findOne() methods.

```
test> db.student.find({name:"Manu"});
[
  {
    _id: ObjectId('65d8943b9f19091797532ed2'),
    id: 2,
    name: 'Manu',
    address: 'Palghar'
  },
```

5.Projection: Both find methods accept a second parameter called ‘projection’. This parameter is an object that describes which fields to include in the results. This parameter is optional. If omitted, all fields will be included in the results.

```
test> db.student.find({}, {id: 1, name:1});
[
  { _id: ObjectId('65d8943b9f19091797532ed2'), id: 2, name: 'Manu' },
  { _id: ObjectId('65d8943b9f19091797532ed3'), id: 3, name: 'Hinali' },
  { _id: ObjectId('65d8949e9f19091797532ed4'), id: 2, name: 'Manu' },
  { _id: ObjectId('65d8949e9f19091797532ed5'), id: 3, name: 'Hinali' },
  { _id: ObjectId('65d8949e9f19091797532ed6'), id: 4, name: 'Himali' }
]
```

6._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.

```
test> db.student.find({}, {_id: 0,id: 1, name:1});
[
  { id: 2, name: 'Manu' },
  { id: 3, name: 'Hinali' },
  { id: 2, name: 'Manu' },
  { id: 3, name: 'Hinali' },
  { id: 4, name: 'Himali' }
]
```

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.We will get an error if we try to specify both 0 and 1 in the same object.

```
test> db.student.find({}, {id: 1, name:0});
MongoServerError[Location31254]: Cannot do exclusion on field name in inclusion projection
```

UPDATE:

To update an existing document we can use the updateOne() or updateMany() methods.The first parameter is a query object to define which document or documents should beupdated. The second parameter is an object defining the updated data.

1. updateOne(): It will update the first document that is found matching the provided query.

Example: updating the "name" of id:2. To do this, we need to use the $set operator. Check the document again to see that the "name" has been updated.

```
test> db.student.updateOne({id:2},{$set:{name:"Diksha"}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

2. updateMany(): It will update all documents that match the provided query. Use {} to select all documents.

Example: Update ‘id’ on all documents by 1 using $inc (increment) operator.You will see that all the ids have been incremented by 1.

```
test> db.student.updateMany({}, {$inc: { id:1 }});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 5,
  upsertedCount: 0
}
```

“MongoDB”

```
test> db.student.find();
[
  {
    _id: ObjectId('65d8943b9f19091797532ed2'),
    id: 3,
    name: 'Diksha',
    address: 'Palghar'
  },
  {
    _id: ObjectId('65d8943b9f19091797532ed3'),
    id: 4,
    name: 'Hinali',
    address: 'Mumbai'
  },
  {
    _id: ObjectId('65d8949e9f19091797532ed4'),
    id: 3,
    name: 'Manu',
    address: 'Palghar'
  },
  {
    _id: ObjectId('65d8949e9f19091797532ed5'),
    id: 4,
    name: 'Hinali',
    address: 'Mumbai'
  },
  {
    _id: ObjectId('65d8949e9f19091797532ed6'),
    id: 5,
    name: 'Himali',
    address: 'Boisar'
  }
]
```

DELETE:

1. **deleteOne():** It will delete the first document that matches the query provided.

```
test> db.student.deleteOne({id:6})
{ acknowledged: true, deletedCount: 0 }
```

2. **deleteMany():** It will delete all documents that match the query provided.

```
test> db.student.deleteMany({ address:"Palghar"})
{ acknowledged: true, deletedCount: 2 }
```

Final document:

```
test> db.student.find();
[
  {
    _id: ObjectId('65d8943b9f19091797532ed3'),
    id: 4,
    name: 'Hinali',
    address: 'Mumbai'
  },
  {
    _id: ObjectId('65d8949e9f19091797532ed5'),
    id: 4,
    name: 'Hinali',
    address: 'Mumbai'
  },
  {
    _id: ObjectId('65d8949e9f19091797532ed6'),
    id: 5,
    name: 'Himali',
    address: 'Boisar'
  }
]
```

Drop Collection:

```
test> show collections;
student
test> db.createCollection('newcollection')
{ ok: 1 }
test> show collections;
newcollection
student
test> db.newcollection.drop()
true
test> show collections;
student
```

Drop Database:

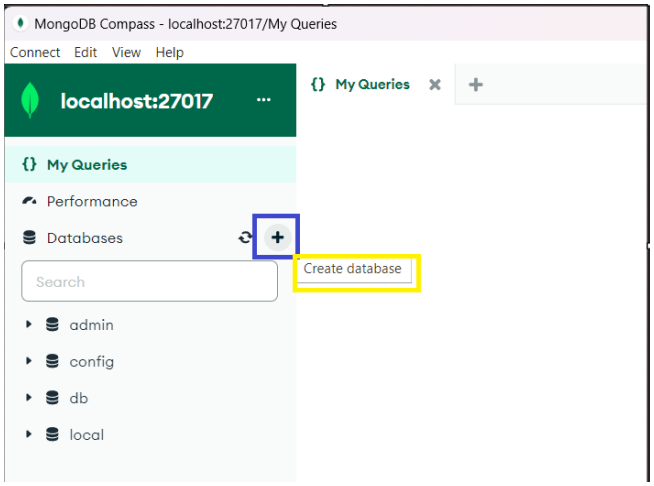
```
test> db.dropDatabase()
{ ok: 1, dropped: 'test' }
test> show dbs;
admin    40.00 KiB
config  108.00 KiB
db      120.00 KiB
local    40.00 KiB
test>
```

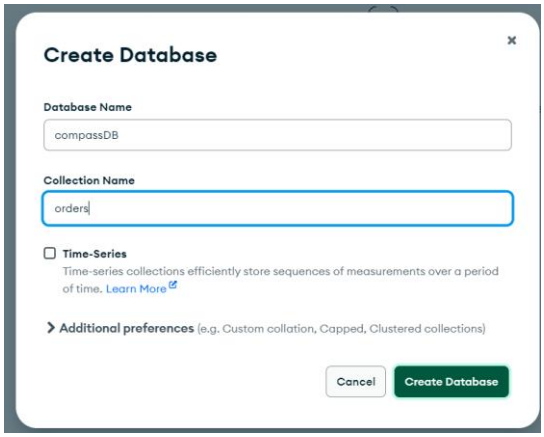
CRUD Operations in MongoDB Compass:

Open MongoDB Compass→ Connect to a local MongoDB server.

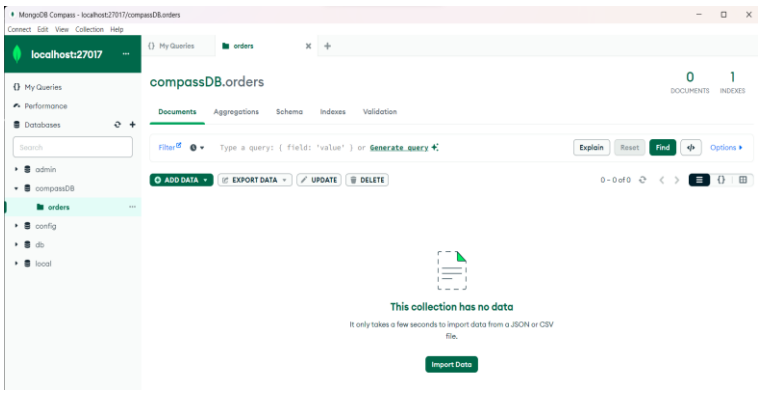
CREATE: Click on “Create database” button→ Add database name and collection name and if needed, configure additional options→Click on Create Database button.

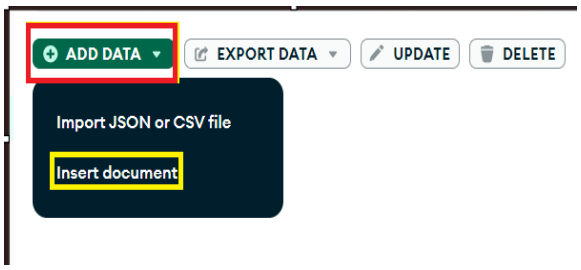
“MongoDB”



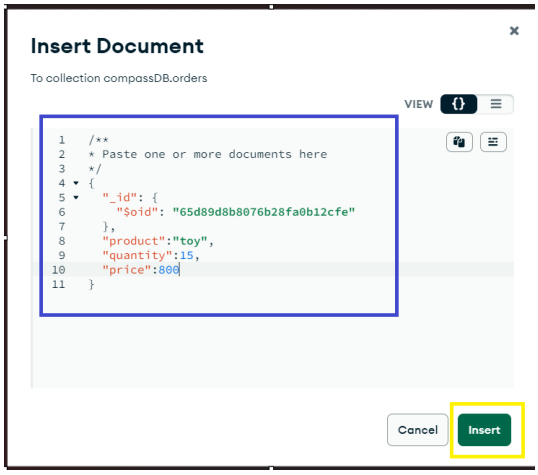


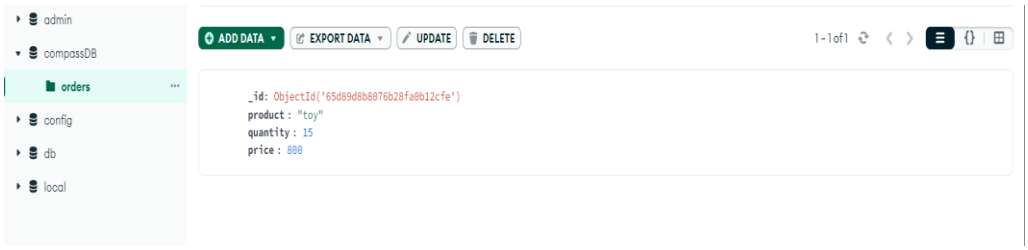
- Click on “ADD DATA”→Insert Document →Add items→ Insert.





“MongoDB”

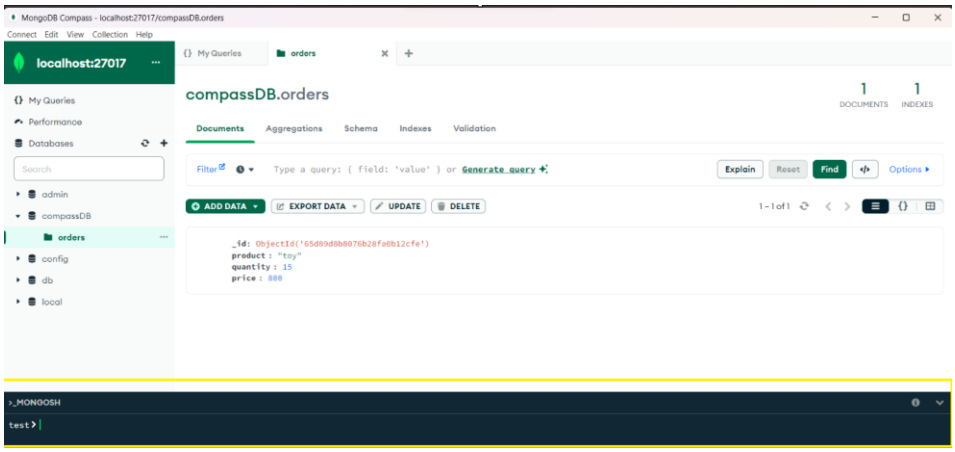




UPDATE: Click on Update to update the document.

DELETE: Click on Delete to delete the document.

- You can access the mongosh in the Compass at the bottom.



4. MongoDB Datatypes and Operators

Datatypes: String, Boolean, number, array, date, timestamp

```
db> db.companyData.insertOne({name:"abc",isFunded:true,funding:12345678,employees:[{name:"gauri",age:23},{name:"sudksha"
,age:24}],date:new Date(),timeStamp:new Timestamp()})
{
  acknowledged: true,
  insertedId: ObjectId('65d795b16a25f297df6343f8')
}
db>
```

```
db> db.companyData.find()
[
  {
    _id: ObjectId('65d795b16a25f297df6343f8'),
    name: 'abc',
    isFunded: true,
    funding: 12345678,
    employees: [ { name: 'gauri', age: 23 }, { name: 'sudksha', age: 24 } ],
    date: ISODate('2024-02-22T18:42:57.290Z'),
    timeStamp: Timestamp({ t: 1708627377, i: 1 })
  }
]
db>
```

Query Operators: There are many query operators that can be used to compare and reference document fields.

1. Comparison:

a) $eq : Values are equal

b) $ne : Values are not equal

c) $gt : Value is greater than another value

d) $gte : Value is greater than or equal to another value

e) $lt :Value is less than another value

f) $lte :Value is less than or equal to another value

g) $in :Value is matched within an array

2. Logical:

a) **$and** Returns documents where both queries match

b) **$or** Returns documents where either query matches

c) **$nor** Returns documents where both queries fail to match

d) **$not** Returns documents where the query does not match

3. Evaluation:

$regex Allows the use of regular expressions when evaluating field values

$text Performs a text search

Update Operators:

1. Fields:

a) $inc Increments the field value

b) $set Sets the value of a field

“MongoDB”

```
test> db.student.updateOne({id:2},{$set:{name:"Diksha"}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
test> db.student.updateMany({}, {$inc: { id:1 }});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 5,
  upsertedCount: 0
}
```

2. Array:

a) $addToSet Adds distinct elements to an array

```
Atlas atlas-echpr9-shard-0 [primary] test> db.student.updateOne({ name:"samiksha"},{$addToSet: {role:"engineer"} });
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
Atlas atlas-echpr9-shard-0 [primary] test>
```

b) $pop Removes the first or last element of an array

```
Atlas atlas-echpr9-shard-0 [primary] test> db.student.updateOne({name:"Guddi"},{$pop:{Hobbies:1}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
```

c) $pull Removes all elements from an array that match the query

```
Atlas atlas-echpr9-shard-0 [primary] test> db.student.updateOne({name:"Guddi"},{$pull:{Hobbies:"dancing"}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
Atlas atlas-echpr9-shard-0 [primary] test>
```

d) $push Adds an element to an array

```
Atlas atlas-echpr9-shard-0 [primary] test> db.student.updateOne({Hobbies:"reading"},{$push:{Hobbies:"singing"}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
Atlas atlas-echpr9-shard-0 [primary] test>
```

5. MONGODB AGGREGATION PIPELINE

Aggregation operations allow you to group, sort, perform calculations, analyze data, and much more. Aggregation pipelines can have one or more "stages". The order of these stages is important. Each stage acts upon the results of the previous stage.

1. **$match:** This aggregation stage behaves like a find. It will filter documents that match the query provided.

Command: db.student.aggregate([{$match:{gender:"Male"}}])

2. **$group:** This aggregation stage groups documents by the unique _id expression provided.

Use {$push:”$$ROOT”} to retrieve all documents.

3. **$sort:** This aggregation stage groups sorts all documents in the specified sort order.

4. **$limit:** This aggregation stage limits the number of documents passed to the next stage.

5. **$project:** This aggregation stage passes only the specified fields along to the next aggregation stage.

INDEXING:

In MongoDB, indexing is a technique used to improve the performance of queries by allowing the database to locate and access documents more efficiently. Indexes are data structures that store a small amount of data about the documents in a collection, and they provide a quick way to look up and access the documents based on the values of one or more fields.

Types of Indexes:

1. Single Field Indexes: These are indexes created on a single field. They can significantly speed up queries that filter or sort based on that field.

2. Compound Indexes: These are indexes on multiple fields. Compound indexes can be beneficial for queries that filter or sort based on multiple criteria.

3. Text Indexes:

- Creating Single Field Index:

```
test> db.student.createIndex({age:1})
age_1
test> db.student.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { age: 1 }, name: 'age_1' }
]
```

- Creating Compound Indexes:

```
test> db.student.createIndex({age:1})
age_1
```

- Creating Text Index:

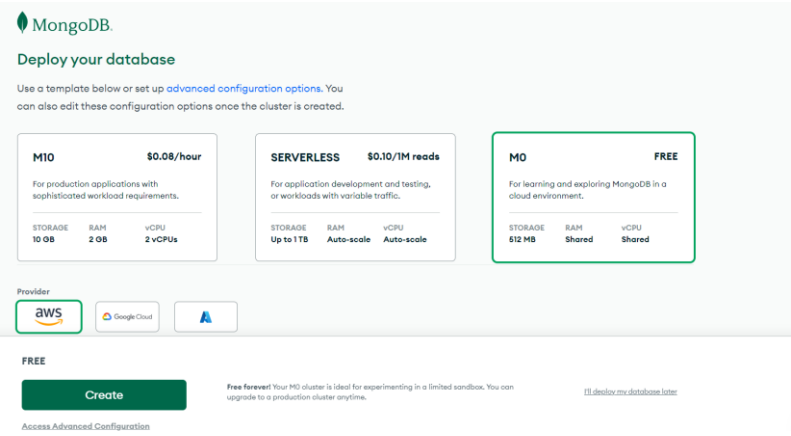
```
test> db.student.createIndex({name:"text"})
name_text
test> db.student.find({$text:{$search:"Himali"}})
```

- Drop Index:

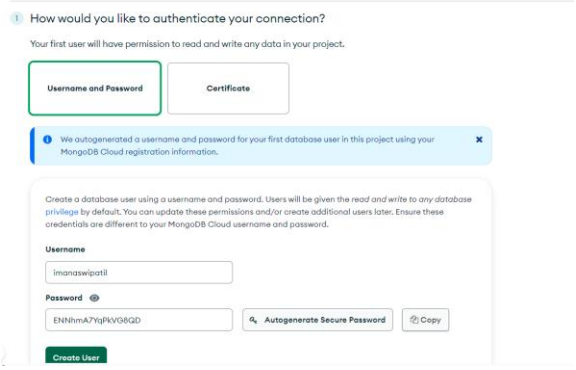
```
test> db.student.dropIndex("age_1_gender_1")
{ nIndexesWas: 4, ok: 1 }
```

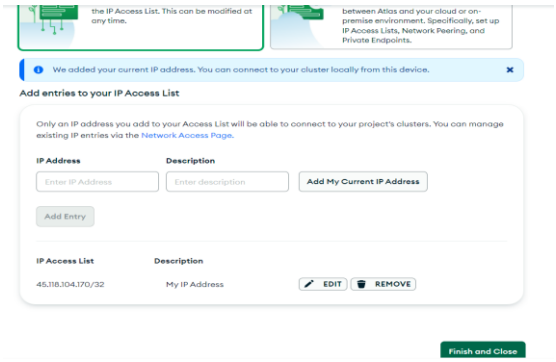
6. CLUSTER CREATION

- Go on → https://www.mongodb.com/cloud/atlas →Click on “Get Started Free”→Fill in
- Sign-Up Form →Click on Get Started→ Configure a Cluster: Click on Build Database →Choose Plan (FREE) →Choose→Cloud Provider, Region, and Cluster Name (Cluster0) → Create.

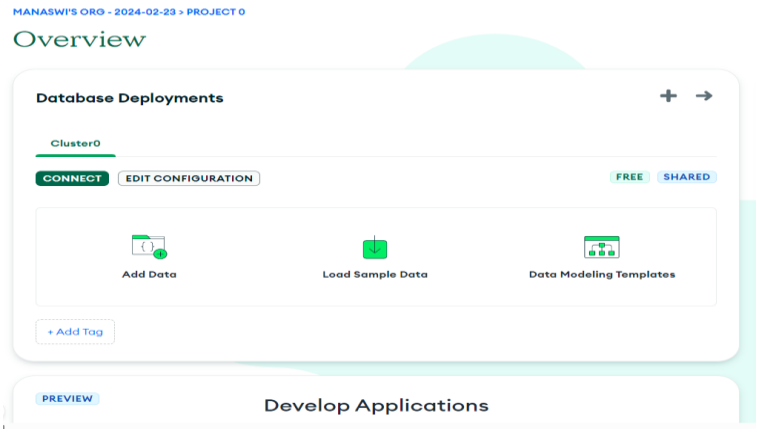


- Create User: Add username and password → Create User → Click on Add My Current IP Address→ Finish and Close.



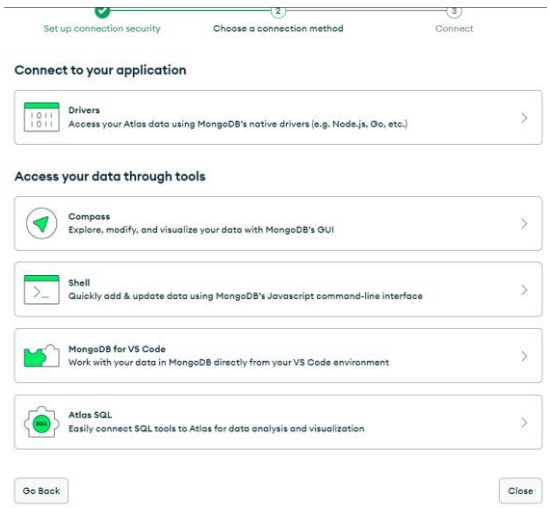


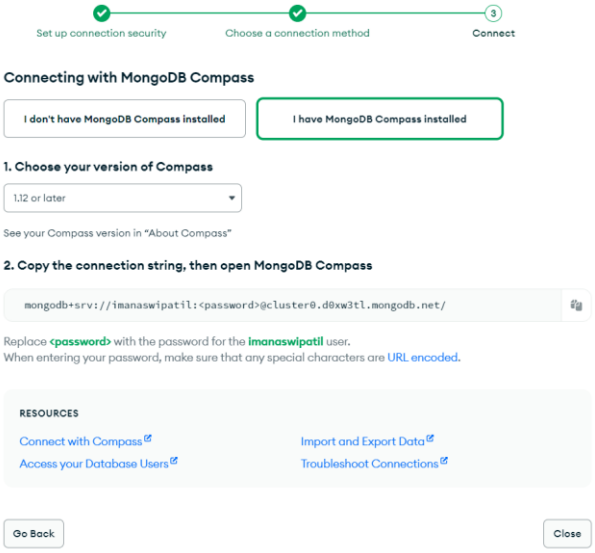
This is an overview: Click on Connect (You can load Sample Data).



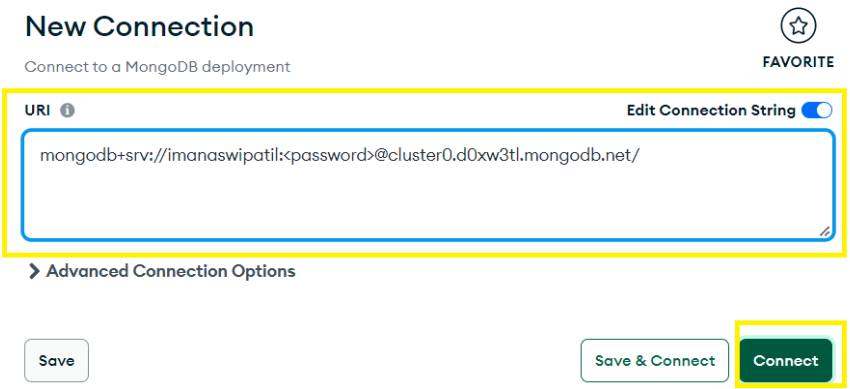
“MongoDB”

- You can connect MongoDB Atlas to following:
- Connect Cluster0 to Compass: Select Compass → Copy the Connection string.

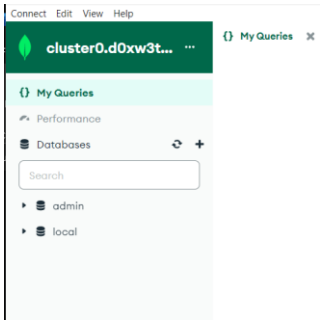




- Open MongoDB Compass: Paste that connection string and replace the <password>with User’s password.



Databases in Cluster0:



- **Connect Atlas to Shell:** Connect→ Select Shell →Copy Connection String →Paste in CMD → Add Password.

“MongoDB”

Connect to Cluster0

Set up connection security — Choose a connection method — 3 Connect

I don't have the MongoDB Shell installed | I have the MongoDB Shell installed

1. Select your mongo shell version

To check your Mongo shell version, run:
mongosh --version or mongo --version

mongosh (2.0 or later)

2. Run your connection string in your command line

Use this connection string in your application

```
mongosh "mongodb+srv://cluster0.d0xw3tl.mongodb.net/" --apiVersion 1 --username imanaswipatil
```

You will be prompted for the password for the Database User, **imanaswipatil**. When entering your password, make sure all special characters are URL encoded.

RESOURCES

Add Data in the Shell | Access your Database Users | Troubleshoot Connections

```
mongosh mongodb+srv://<cl
Microsoft Windows [Version 10.0.22621.3155]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>mongosh "mongodb+srv://cluster0.d0xw3tl.mongodb.net/" --apiVersion 1 --username imanaswipatil
Enter password: ****************
Current Mongosh Log ID: 65d8b8fb75e7ff41b6548d5d
Connecting to:          mongodb+srv://<credentials>@cluster0.d0xw3tl.mongodb.net/?appName=mongosh+2.1.5
Using MongoDB:          6.0.13 (API Version 1)
Using Mongosh:          2.1.5

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

Atlas atlas-echpr9-shard-0 [primary] test> |
```

- **Connect Atlas to VS Code:** Open VS Code →go on view→Open Command Palette →search MongoDB:Connect →Click Connect with Connection String →Copy the Connection→String→Paste it in Command Palette (Replace <password>→with user’s password.

Set up connection security — Choose a connection method — 3 Connect

Connecting with MongoDB for VS Code

1. Install MongoDB for VS Code.

In VS Code, open "Extensions" in the left navigation and search for "MongoDB for VS Code." Select the extension and click install.

2. In VS Code, open the Command Palette.

Click on "View" and open "Command Palette."
Search "MongoDB: Connect" on the Command Palette and click on "Connect with Connection String."

3. Connect to your MongoDB deployment.

Paste your connection string into the Command Palette.

```
mongodb+srv://imanaswipatil:<password>@cluster0.d0xw3tl.mongodb.net/
```

Replace **<password>** with the password for the **imanaswipatil** user.
When entering your password, make sure all special characters are URL encoded.

4. Click “Create New Playground” in MongoDB for VS Code to get started.

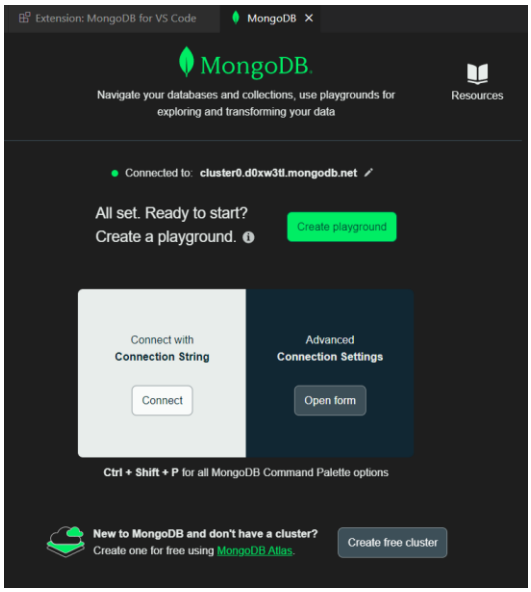
Learn more about Playgrounds

RESOURCES

Connect to MongoDB through VSCode | Explore your data with playgrounds | Access your Database Users | Troubleshoot Connections

“MongoDB”


>mongodb+srv://imanaswipatil:ENNhmA7qPkVG8QD@cluster0.d0xw3tl.mongodb.net/


Extension: MongoDB for VS Code
MongoDB
MongoDB.
Navigate your databases and collections, use playgrounds for exploring and transforming your data
Resources
Connected to: cluster0.d0xw3tl.mongodb.net
All set. Ready to start?
Create a playground.
Create playground
Connect with
Connection String
Connect
Advanced
Connection Settings
Open form
Ctrl + Shift + P for all MongoDB Command Palette options
New to MongoDB and don't have a cluster?
Create one for free using MongoDB Atlas.
Create free cluster

7. MONGODB DRIVERS

A MongoDB driver is a software component that enables an application to interact with a MongoDB database.

Following is the current officially supported drivers:

C C++ C#

C	C++	C#	Go
java	Node.js	php	python
ruby	rust	scala	swift

- **Node.js Driver:**

To use MongoDB with Node.js, you will need to install the mongodb package in your Node.js project.

- Use npm install mongodb command in your project terminal.
- Create an index.js file in your project directory.
- Connection String.
- Go to MongoDB Atlas →Connect→Copy Connection StringReplace the <password>your MongoDB Atlas password.

```
JS index.js ×
C: > Users > hp > JS index.js > run

const uri = "mongodb+srv://imanaswipatil:ENNhmA7YqPkVG8QD@cluster0.d0xw3tl.mongodb.net/";
const client = new MongoClient(uri);

async function run() {
    try {
        await client.connect();
        const db = client.db('myAltasDB');
        const collection = db.collection('student');
        const first = await collection.findOne();
        console.log(first);
    }finally{
        await client.close();
    }
}
run().catch(console.error);
```

- Run the file in the terminal

```
PS C:\Users\hp> node index.js
null
```

Its null because we haven’t create database.

- To find all documents:

```
const first = await collection.find({}).toArray();
```

- **Schema Validation:** Schema validation rules can be created in order to ensure that all documents a collection share a similar structure.

“MongoDB”

```
test> use mydb
switched to db mydb
mydb> db.createCollection("posts", {

...   validator: {
...     $jsonSchema: {
...       bsonType: "object",
...       required: [ "title", "body" ],
...       properties: {
...         title: {
...           bsonType: "string",
...           description: "Title of post - Required."
...         },        body: {
...           bsonType: "string",
...           description: "Body of post - Required."
...         },
...         category: {
...           bsonType: "string",
...           description: "Category of post - Optional."
...         },
...         likes: {
...           bsonType: "int",
...           description: "Post like count. Must be an integer - Optional."
...         },
...         tags: {
...           bsonType: ["string"],
...           description: "Must be an array of strings - Optional."
...         },
...         date: {
...           bsonType: "date",
...           description: "Must be a date - Optional."
...         }
...       }
...     }
...   }
... })
{ ok: 1 }
```

- **Document Validation failed:**

```
mydb> db.posts.insertOne({title:"mytitle-1",description:"Hello"})
Uncaught:
MongoServerError: Document failed validation
Additional information: {
  failingDocumentId: ObjectId('65c60c3136c89bb7d88f9433'),
  details: {
    operatorName: '$jsonSchema',
    schemaRulesNotSatisfied: [
      {
        operatorName: 'required',
        specifiedAs: { required: [ 'title', 'body' ] },
        missingProperties: [ 'body' ]
      }
    ]
  }
}
```

- **Document Insertion Successful:**

```
mydb> db.posts.insertOne({title:"mytitle-1",description:"Hello",body:"This is my first post"})
{
  acknowledged: true,
  insertedId: ObjectId('65c60c1036c89bb7d88f9432')
}
mydb> db.posts.find()
[
  {
    _id: ObjectId('65c60c1036c89bb7d88f9432'),
    title: 'mytitle-1',
    description: 'Hello',
    body: 'This is my first post'
  }
]
mydb> |
```

- **How to create Schema using mongoose:**

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a higher-level, schema-based abstraction over the MongoDB driver, making it easier to interact with MongoDB databases using JavaScript or TypeScript.

- Install mongoose: npm install mongoose
- Run the MongoDB Server.

“MongoDB”

- Define the Schema (Create userModel.js)

```
> Users > hp > JS userModel.js > [⊗] userSchema > createdAt > default
// user.model.js
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

//  Define the Schema
const userSchema = new Schema({
    firstName: {
        type: String,
        required: true
    },
    lastName: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true,
        unique: true
    },
    age: {
        type: Number,
        min: 0
    },
    createdAt: {
        type: Date,
        default: Date.now
    },
});

// Create a Model
const User = mongoose.model('User', userSchema);

// Export the Model
module.exports = User;
```

- Use the Schema in Your Application (Create app.js)

```
index.js ●  JS userModel.js  JS app.js ×
> Users > hp > JS app.js > then() callback > [⊗] newUser
// app.js or wherever you set up your application
const mongoose = require('mongoose');
const User = require('./userModel');

mongoose.connect("mongodb://0.0.0.0:27017/employees")
    .then(() => {
        console.log('Connected to MongoDB');

        // Now you can use the User model for CRUD operations
        // For example, create a new user
        const newUser = new User({
            firstName: 'Manaswi',
            lastName: 'Patil',
            email: 'manaswipatil@example.com',
            age: 23
        });

        newUser.save()
            .then(user => {
                console.log('User created:', user);
            })
            .catch(error => {
                console.error('Error creating user:', error);
            });
    })
    .catch(error => {
        console.error('MongoDB connection error:', error);
    });
```

- Run the app – node app.js

```
PS C:\Users\hp> node app.js
Connected to MongoDB
User created: {
  firstName: 'Manaswi',
  lastName: 'Patil',
  email: 'manaswi@example.com',
  age: 23,
  _id: new ObjectId('65d8e519791b9b2f8311c129'),
  createdAt: 2024-02-23T18:34:01.758Z,
  __v: 0
}
```