Project – 8

MongoDB

Submitted By:

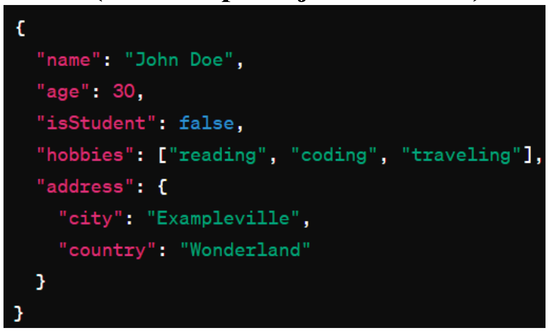
Nishigandha Patil

INDEX

INTRODUCTION TO MONGODB

MongoDB is a popular, open-source NoSQL database that provides a flexible and scalable way to store and manage data. It stores data in JSON-like format called BSON (Binary JSON).

➢ **JSON (JavaScript Object Notation):** Transmit data between server and a web application.



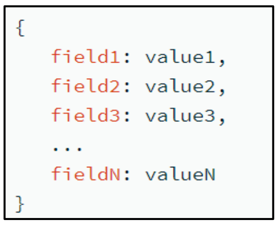
```
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false,
  "hobbies": ["reading", "coding", "traveling"],
  "address": {
    "city": "Exampleville",
    "country": "Wonderland"
  }
}
```

➢ **Features of MongoDB:**

1. **Schema-flexibility:** MongoDB is a schema-less database, meaning that documents in a collection can have different structures.
2. **Scalability:** MongoDB is designed to scale horizontally, allowing for the distribution of data across multiple servers or clusters.
3. **Query Language:** MongoDB provides a rich query language with support for complex queries, indexing, and aggregation.
4. **Performance:** MongoDB uses a variety of optimization techniques, including indexing and caching, to provide high-performance read and write operations.

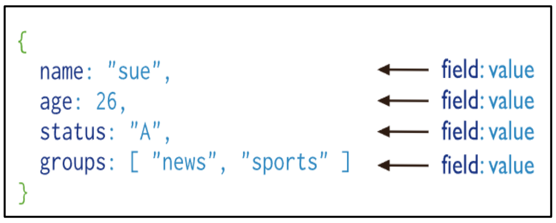
➢ **MongoDB Database, Collection and Documents:**
MongoDB stores data records as **documents** which are gathered together in **collections**. A **database** stores one or more collections of documents.

Document Structure:



```
{
   field1: value1,
   field2: value2,
   field3: value3,
   ...
   fieldN: valueN
}
```

A MongoDB document:



```
{
   name: "sue",
   age: 26,
   status: "A",
   groups: [ "news", "sports" ]
}
```

➢ MongoDB is composed of various components and tools that work together to provide a flexible, scalable, and high-performance NoSQL database system:

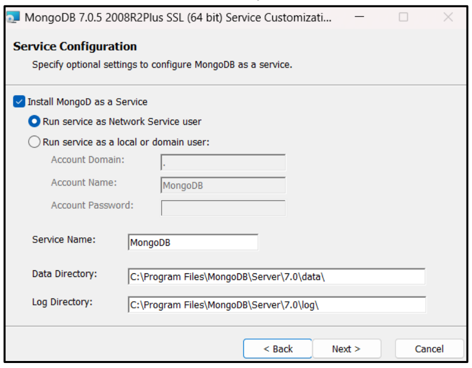
1. **MongoDB Server:** It is the core database server that stores and manages data.
2. **MongoDB Shell:** It is an interactive JavaScript interface to MongoDB.
3. **MongoDB Compass:** It is a graphical user interface (GUI) for MongoDB.
4. **MongoDB Atlas:** It is MongoDB's cloud database service.It offers a fully managed database platform, allowing you to deploy, manage, and scale MongoDB databases in the cloud.

INSTALLATIONS

➢ **Installing MongoDB Community Server:**

1. **Download the installer:**
   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.

2. **Follow the Installation wizard:**
   The wizard steps you through the installation of "MongoDB" and "MongoDB Compass".
   a. Choose Setup Type
   Choose Complete Setup type (recommended for most users).
   b. Service Configuration
   Choose to install MongoDB as a service. This allows MongoDB to start automatically with the Windows system.

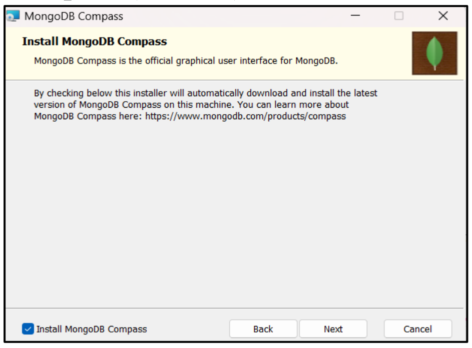

MongoDB 7.0.5 2008R2Plus SSL (64 bit) Service Customizati...

Service Configuration
Specify optional settings to configure MongoDB as a service.

☑ Install MongoD as a Service
● Run service as Network Service user
○ Run service as a local or domain user:
Account Domain: .
Account Name: MongoDB
Account Password:

Service Name: MongoDB
Data Directory: C:\Program Files\MongoDB\Server\7.0\data\
Log Directory: C:\Program Files\MongoDB\Server\7.0\log\

3. **Install MongoDB Compass:**
   Optional. To have the wizard install MongoDB Compass, select Install MongoDB Compass (Default).

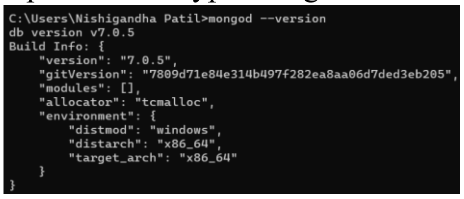

MongoDB Compass

Install MongoDB Compass
MongoDB Compass is the official graphical user interface for MongoDB.

By checking below this installer will automatically download and install the latest version of MongoDB Compass on this machine. You can learn more about MongoDB Compass here: https://www.mongodb.com/products/compass

☑ Install MongoDB Compass

4. **Verify the Installation:**
   Open cmd → type mongod --version.

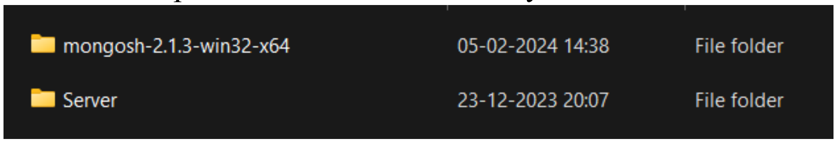


```
C:\Users\Nishigandha Patil>mongod --version
db version v7.0.5
Build Info: {
    "version": "7.0.5",
    "gitVersion": "7809d71e84e314b497f282ea8aa06d7ded3eb205",
    "modules": [],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "windows",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}
```

➢ **Installing MongoDB Shell:**

1. **Download the installer:**
   Go to: https://www.mongodb.com/try/download/shell
   Select Version → Platform → Package(zip) → Download.

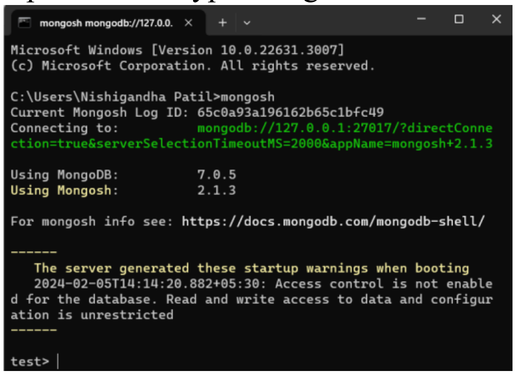
2. **Extract the zip file:**
   Extract the zip file to the location where you have installed the MongoDB server.



mongosh-2.1.3-win32-x64	05-02-2024 14:38	File folder
Server	23-12-2023 20:07	File folder

   Open mongosh-2.1.3 folder→ Go to the bin folder → Copy path → Search for 'Edit environment variable' → System variable → Path → Edit → Paste the bin folder path → OK.

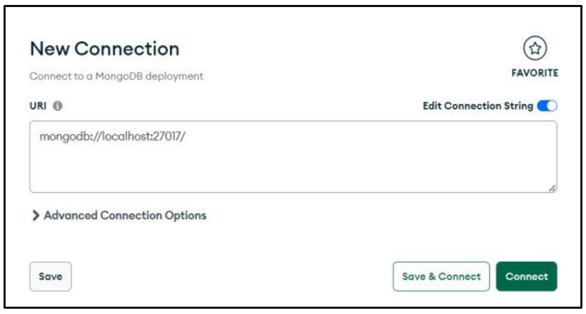
3. **Connect to Localhost:**
   Open cmd → type mongosh.



```
Microsoft Windows [Version 10.0.22631.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Nishigandha Patil>mongosh
Current Mongosh Log ID: 65c0a93a196162b65c1bfc49
Connecting to:          mongodb://127.0.0.1:27017/?directConne
ction=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.3

Using MongoDB:          7.0.5
Using Mongosh:          2.1.3

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

------
   The server generated these startup warnings when booting
   2024-02-05T14:14:20.882+05:30: Access control is not enable
d for the database. Read and write access to data and configur
ation is unrestricted
------

test> 
```

➢ **Installing MongoDB Compass:**

1. Go to: https://www.mongodb.com/try/download/compass → Download **exe** file → Run the installer.
2. **Connect to Localhost:**



```
mongodb://localhost:27017/
```

➢ **MongoDB Atlas:** No need to install it.
Go to https://www.mongodb.com/cloud/atlas and signup for free.

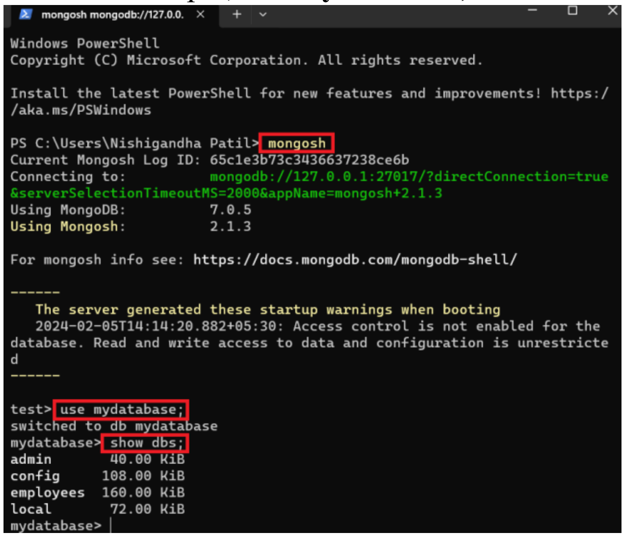
MONGODB CRUD OPERATIONS

MongoDB supports CRUD (Create, Read, Update, Delete) operations for interacting with data.

- **Creating Database and Collection in MongoDB Shell (mongosh):**

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. For example, **use mydatabase;**



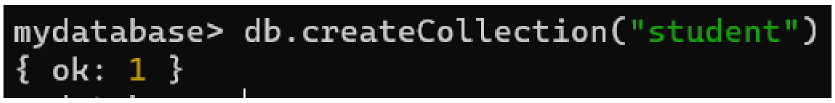
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https:/
/aka.ms/PSWindows

PS C:\Users\Nishigandha Patil> mongosh
Current Mongosh Log ID: 65c1e3b73c3436637238ce6b
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true
&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.3
Using MongoDB:          7.0.5
Using Mongosh:          2.1.3

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

------
   The server generated these startup warnings when booting
   2024-02-05T14:14:20.882+05:30: Access control is not enabled for the
database. Read and write access to data and configuration is unrestricte
d
------

test> use mydatabase;
switched to db mydatabase
mydatabase> show dbs;
admin        40.00 KiB
config      108.00 KiB
employees   160.00 KiB
local        72.00 KiB
mydatabase>
```

To see all available databases, in your terminal type **show dbs**.
Notice that "mydatabase" is not listed. This is because the database is empty. An empty database is essentially non-existant.

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

2. Create Collection:

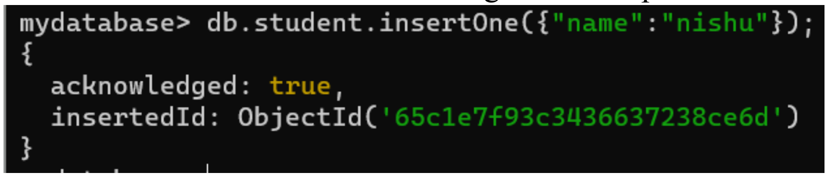
- **Method1:** You can create a collection using the **createCollection()** database method.



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

This will create "student" collection.

- **Method2:**
You can also create a collection during the **insert** process.



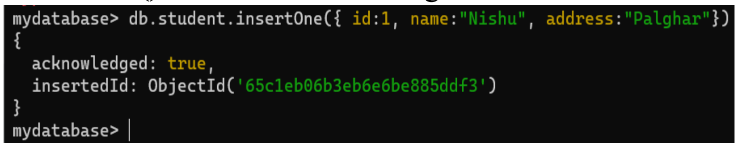
```
mydatabase> db.student.insertOne({"name":"nishu"});
{
  acknowledged: true,
  insertedId: ObjectId('65c1e7f93c3436637238ce6d')
}
```

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

- **CREATE:**

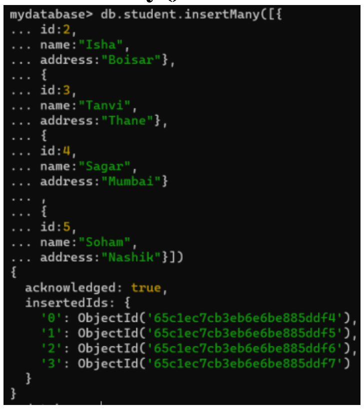
  There are 2 methods to insert documents into a MongoDB database.

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



```
mydatabase> db.student.insertOne({ id:1, name:"Nishu", address:"Palghar"})
{
  acknowledged: true,
  insertedId: ObjectId('65c1eb06b3eb6e6be885ddf3')
}
mydatabase> |
```

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

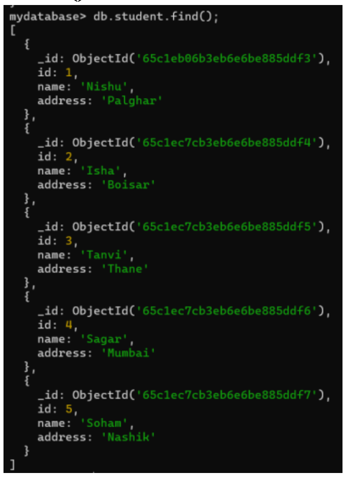


```
mydatabase> db.student.insertMany([{
... id:2,
... name:"Isha",
... address:"Boisar"},
... {
... id:3,
... name:"Tanvi",
... address:"Thane"},
... {
... id:4,
... name:"Sagar",
... address:"Mumbai"}
... ,
... {
... id:5,
... name:"Soham",
... address:"Nashik"}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65c1ec7cb3eb6e6be885ddf4'),
    '1': ObjectId('65c1ec7cb3eb6e6be885ddf5'),
    '2': ObjectId('65c1ec7cb3eb6e6be885ddf6'),
    '3': ObjectId('65c1ec7cb3eb6e6be885ddf7')
  }
}
```

- **READ (Query Documents):**

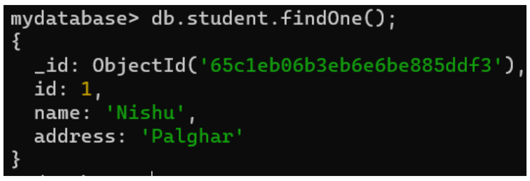
  There are 2 methods to find and select data from a MongoDB collection:

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



```
mydatabase> db.student.find();
[
  {
    _id: ObjectId('65c1eb06b3eb6e6be885ddf3'),
    id: 1,
    name: 'Nishu',
    address: 'Palghar'
  },
  {
    _id: ObjectId('65c1ec7cb3eb6e6be885ddf4'),
    id: 2,
    name: 'Isha',
    address: 'Boisar'
  },
  {
    _id: ObjectId('65c1ec7cb3eb6e6be885ddf5'),
    id: 3,
    name: 'Tanvi',
    address: 'Thane'
  },
  {
    _id: ObjectId('65c1ec7cb3eb6e6be885ddf6'),
    id: 4,
    name: 'Sagar',
    address: 'Mumbai'
  },
  {
    _id: ObjectId('65c1ec7cb3eb6e6be885ddf7'),
    id: 5,
    name: 'Soham',
    address: 'Nashik'
  }
]
```

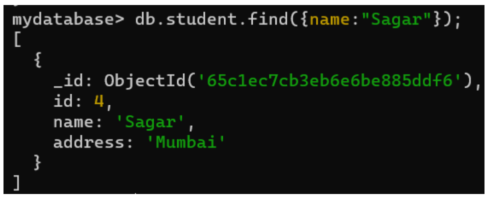
  2. **findOne():** Used to select only one document. If left empty, it will return the first document it finds.



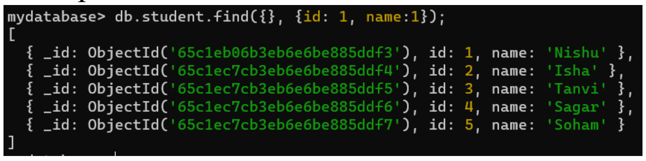
```
mydatabase> db.student.findOne();
{
  _id: ObjectId('65c1eb06b3eb6e6be885ddf3'),
  id: 1,
  name: 'Nishu',
  address: 'Palghar'
}
```

  use **.count()** to count the no. of documents. Example: **db.student.find().count()**

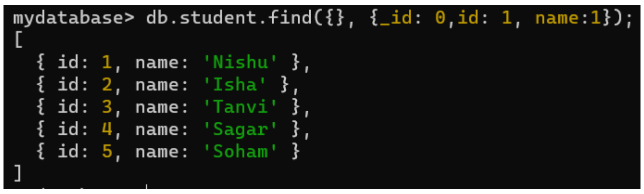
- **Querying Data:** To query, or filter, data we can include a query in our find() or findOne() methods.


```
mydatabase> db.student.find({name:"Sagar"});
[
  {
    _id: ObjectId('65c1ec7cb3eb6e6be885ddf6'),
    id: 4,
    name: 'Sagar',
    address: 'Mumbai'
  }
]
```

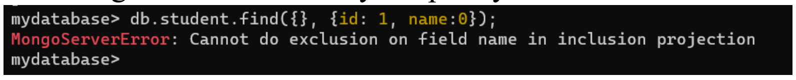
- **Projection:** Both find methods accept a second parameter called ‘projection’. This parameter is an object that describes which fields to include in the results.
  Note: This parameter is optional. If omitted, all fields will be included in the results.
  Example:


```
mydatabase> db.student.find({}, {id: 1, name:1});
[
  { _id: ObjectId('65c1eb06b3eb6e6be885ddf3'), id: 1, name: 'Nishu' },
  { _id: ObjectId('65c1ec7cb3eb6e6be885ddf4'), id: 2, name: 'Isha' },
  { _id: ObjectId('65c1ec7cb3eb6e6be885ddf5'), id: 3, name: 'Tanvi' },
  { _id: ObjectId('65c1ec7cb3eb6e6be885ddf6'), id: 4, name: 'Sagar' },
  { _id: ObjectId('65c1ec7cb3eb6e6be885ddf7'), id: 5, name: 'Soham' }
]
```

  _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.


```
mydatabase> db.student.find({}, {_id: 0,id: 1, name:1});
[
  { id: 1, name: 'Nishu' },
  { id: 2, name: 'Isha' },
  { id: 3, name: 'Tanvi' },
  { id: 4, name: 'Sagar' },
  { id: 5, name: 'Soham' }
]
```

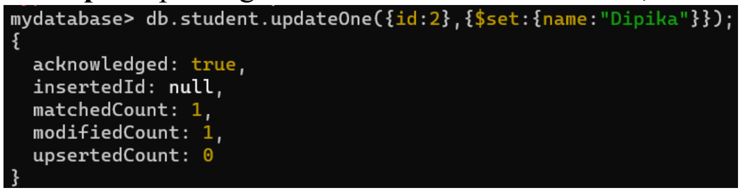
  Note: You cannot use both 0 and 1 in the same object. The only exception is the **_id** field. You should either specify the fields you would like to include or the fields you would like to exclude.
  We will get an error if we try to specify both 0 and 1 in the same object.


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

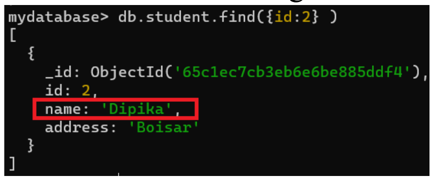
➢ **UPDATE:**

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 be updated. The second parameter is an object defining the updated data.

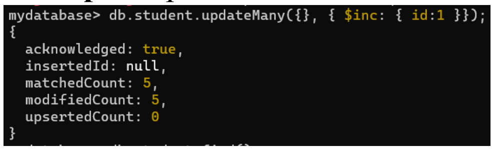
1. updateOne(): It will update the first document that is found matching the provided query.
Example: updating the "name" of id:2. To do this, we need to use the **$set** operator.


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

Check the document again to see that the "name" has been updated.

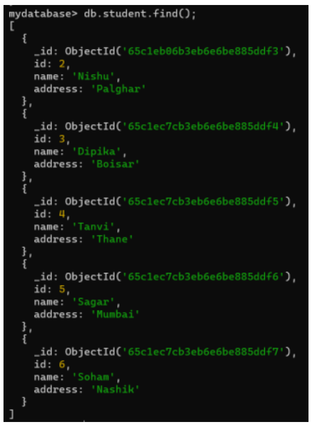

```
mydatabase> db.student.find({id:2} )
[
  {
    _id: ObjectId('65c1ec7cb3eb6e6be885ddf4'),
    id: 2,
    name: 'Dipika',
    address: 'Boisar'
  }
]
```

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:



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

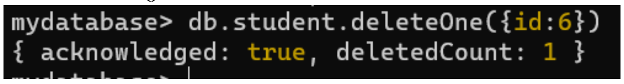
   You will see that all the ids have been incremented by 1.



```
mydatabase> db.student.find();
[
  {
    _id: ObjectId('65c1eb06b3eb6e6be885ddf3'),
    id: 2,
    name: 'Nishu',
    address: 'Palghar'
  },
  {
    _id: ObjectId('65c1ec7cb3eb6e6be885ddf4'),
    id: 3,
    name: 'Dipika',
    address: 'Boisar'
  },
  {
    _id: ObjectId('65c1ec7cb3eb6e6be885ddf5'),
    id: 4,
    name: 'Tanvi',
    address: 'Thane'
  },
  {
    _id: ObjectId('65c1ec7cb3eb6e6be885ddf6'),
    id: 5,
    name: 'Sagar',
    address: 'Mumbai'
  },
  {
    _id: ObjectId('65c1ec7cb3eb6e6be885ddf7'),
    id: 6,
    name: 'Soham',
    address: 'Nashik'
  }
]
```

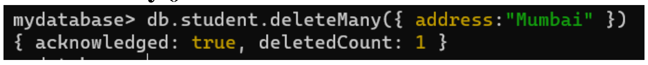
➢ **DELETE:**

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



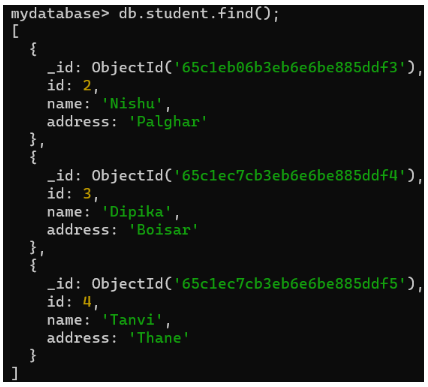
```
mydatabase> db.student.deleteOne({id:6})
{ acknowledged: true, deletedCount: 1 }
```

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



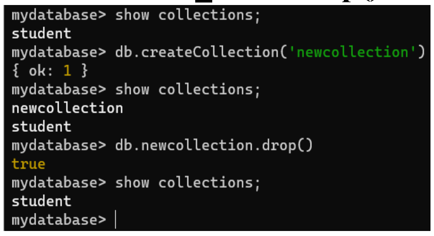
```
mydatabase> db.student.deleteMany({ address:"Mumbai" })
{ acknowledged: true, deletedCount: 1 }
```

Final document:



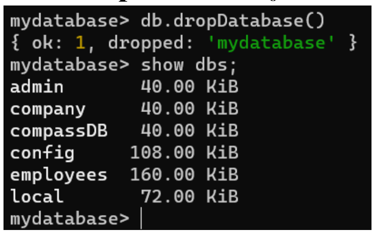
```
mydatabase> db.student.find();
[
  {
    _id: ObjectId('65c1eb06b3eb6e6be885ddf3'),
    id: 2,
    name: 'Nishu',
    address: 'Palghar'
  },
  {
    _id: ObjectId('65c1ec7cb3eb6e6be885ddf4'),
    id: 3,
    name: 'Dipika',
    address: 'Boisar'
  },
  {
    _id: ObjectId('65c1ec7cb3eb6e6be885ddf5'),
    id: 4,
    name: 'Tanvi',
    address: 'Thane'
  }
]
```

Drop Collection:
>db.collection_name.drop()



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

Drop Database:
>db.dropDatabase()

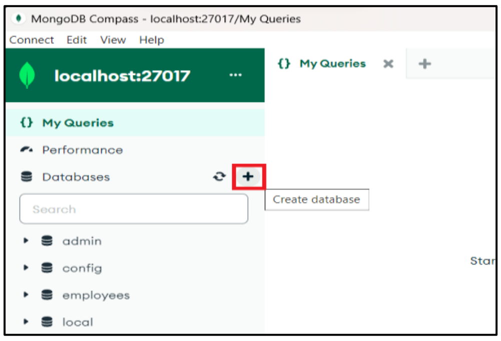


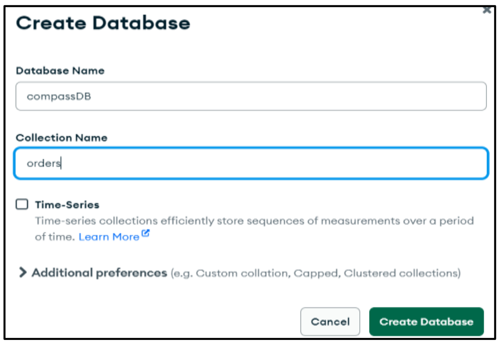
```
mydatabase> db.dropDatabase()
{ ok: 1, dropped: 'mydatabase' }
mydatabase> show dbs;
admin        40.00 KiB
company      40.00 KiB
compassDB    40.00 KiB
config      108.00 KiB
employees   160.00 KiB
local        72.00 KiB
mydatabase> 
```

➢ **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.




Create Database

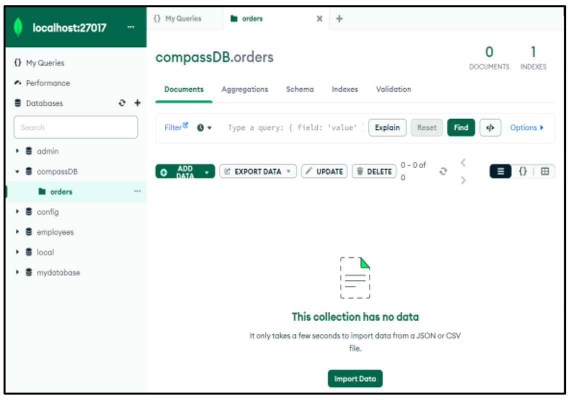
Database Name: compassDB

Collection Name: orders

☐ Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More

› Additional preferences (e.g. Custom collation, Capped, Clustered collections)

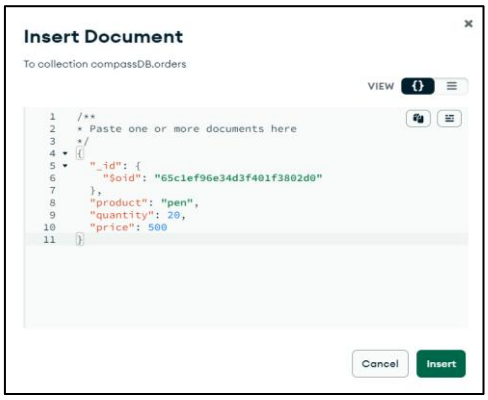
Click on "ADD DATA" →Insert Document →Add items → Insert.


compassDB.orders

0 DOCUMENTS 1 INDEXES

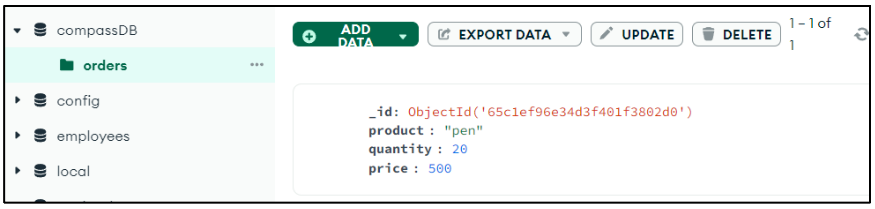
This collection has no data

It only takes a few seconds to import data from a JSON or CSV file.


Insert Document

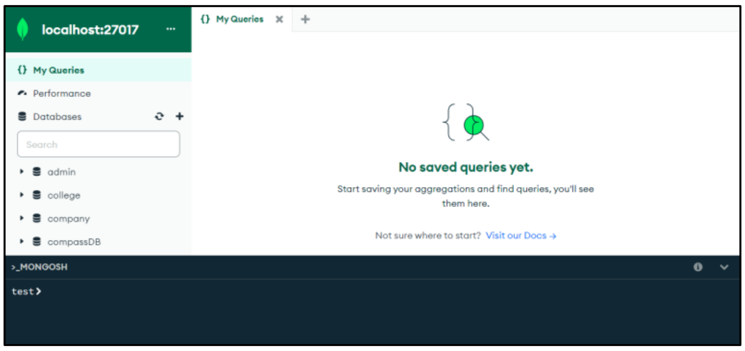
To collection compassDB.orders

```
/**
* Paste one or more documents here
*/
{
  "_id": {
    "$oid": "65c1ef96e34d3f401f3802d0"
  },
  "product": "pen",
  "quantity": 20,
  "price": 500
}
```


```
_id: ObjectId('65c1ef96e34d3f401f3802d0')
product : "pen"
quantity : 20
price : 500
```

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:


No saved queries yet.

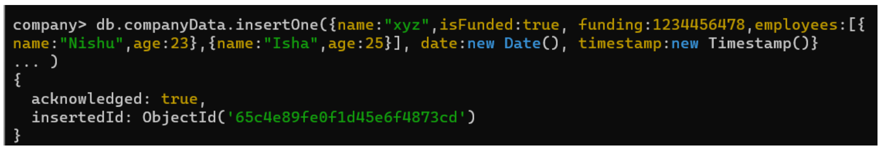
Start saving your aggregations and find queries, you'll see them here.

Not sure where to start? Visit our Docs →

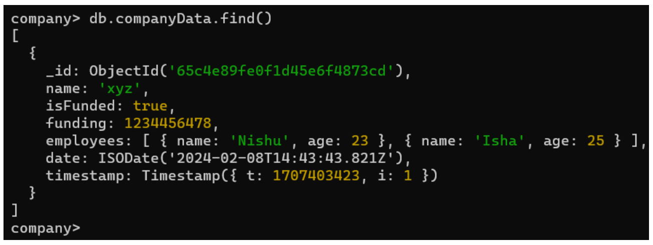
```
>_MONGOSH
test>
```

MongoDB Datatypes and Operators

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



```
company> db.companyData.insertOne({name:"xyz",isFunded:true, funding:1234456478,employees:[{
name:"Nishu",age:23},{name:"Isha",age:25}], date:new Date(), timestamp:new Timestamp()}
... )
{
  acknowledged: true,
  insertedId: ObjectId('65c4e89fe0f1d45e6f4873cd')
}
```

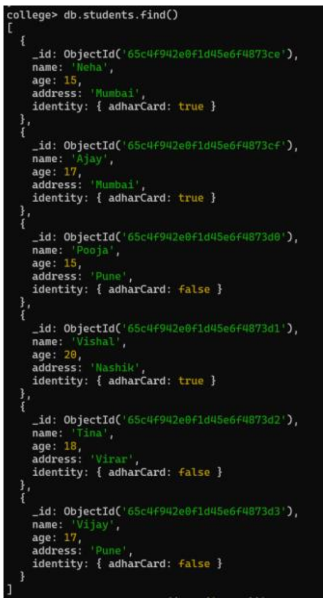


```
company> db.companyData.find()
[
  {
    _id: ObjectId('65c4e89fe0f1d45e6f4873cd'),
    name: 'xyz',
    isFunded: true,
    funding: 1234456478,
    employees: [ { name: 'Nishu', age: 23 }, { name: 'Isha', age: 25 } ],
    date: ISODate('2024-02-08T14:43:43.821Z'),
    timestamp: Timestamp({ t: 1707403423, i: 1 })
  }
]
company>
```

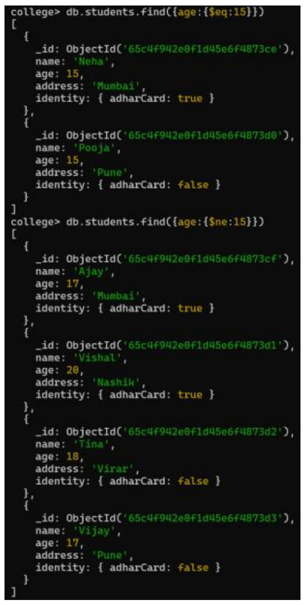
➢ **Query Operators:** There are many query operators that can be used to compare and reference document fields.

1. **Comparison:**

Operators	**Description**
$eq	Values are equal
$ne	Values are not equal
$gt	Value is greater than another value
$gte	Value is greater than or equal to another value
$lt	Value is less than another value
$lte	Value is less than or equal to another value
$in	Value is matched within an array



```
college> db.students.find()
[
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873ce'),
    name: 'Neha',
    age: 15,
    address: 'Mumbai',
    identity: { adharCard: true }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873cf'),
    name: 'Ajay',
    age: 17,
    address: 'Mumbai',
    identity: { adharCard: true }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873d0'),
    name: 'Pooja',
    age: 15,
    address: 'Pune',
    identity: { adharCard: false }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873d1'),
    name: 'Vishal',
    age: 20,
    address: 'Nashik',
    identity: { adharCard: true }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873d2'),
    name: 'Tina',
    age: 18,
    address: 'Virar',
    identity: { adharCard: false }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873d3'),
    name: 'Vijay',
    age: 17,
    address: 'Pune',
    identity: { adharCard: false }
  }
]
```

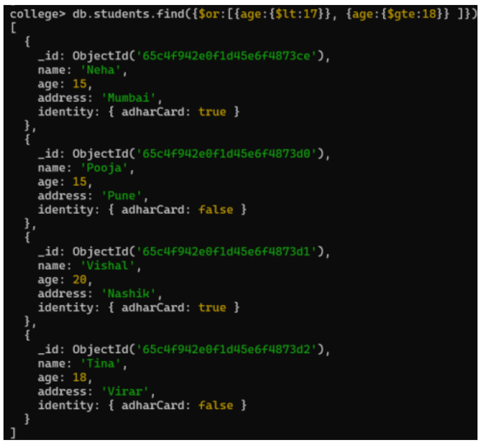


```
college> db.students.find({age:{$eq:15}})
[
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873ce'),
    name: 'Neha',
    age: 15,
    address: 'Mumbai',
    identity: { adharCard: true }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873d0'),
    name: 'Pooja',
    age: 15,
    address: 'Pune',
    identity: { adharCard: false }
  }
]
college> db.students.find({age:{$ne:15}})
[
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873cf'),
    name: 'Ajay',
    age: 17,
    address: 'Mumbai',
    identity: { adharCard: true }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873d1'),
    name: 'Vishal',
    age: 20,
    address: 'Nashik',
    identity: { adharCard: true }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873d2'),
    name: 'Tina',
    age: 18,
    address: 'Virar',
    identity: { adharCard: false }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873d3'),
    name: 'Vijay',
    age: 17,
    address: 'Pune',
    identity: { adharCard: false }
  }
]
```

2. **Logical:**

Operators	**Description**
$and	Returns documents where both queries match

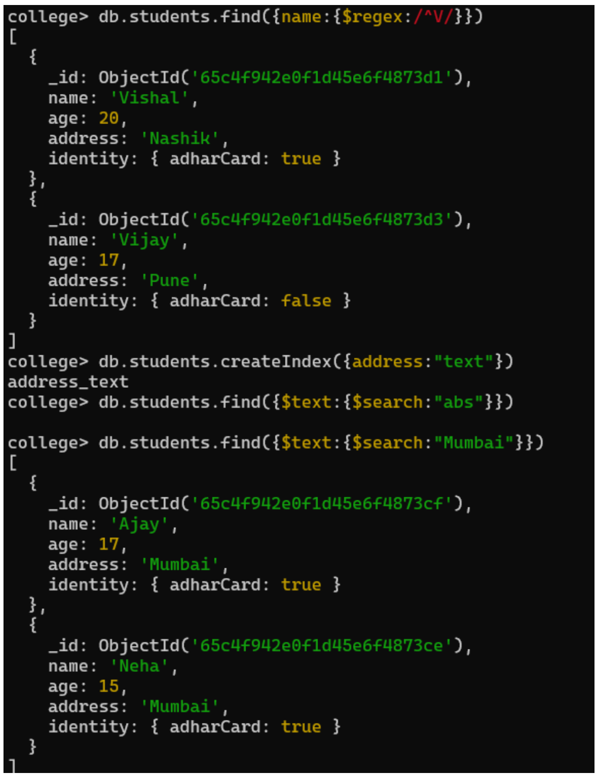
$or	Returns documents where either query matches
$nor	Returns documents where both queries fail to match
$not	Returns documents where the query does not match



```
college> db.students.find({$or:[{age:{$lt:17}}, {age:{$gte:18}} ]})
[
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873ce'),
    name: 'Neha',
    age: 15,
    address: 'Mumbai',
    identity: { adharCard: true }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873d0'),
    name: 'Pooja',
    age: 15,
    address: 'Pune',
    identity: { adharCard: false }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873d1'),
    name: 'Vishal',
    age: 20,
    address: 'Nashik',
    identity: { adharCard: true }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873d2'),
    name: 'Tina',
    age: 18,
    address: 'Virar',
    identity: { adharCard: false }
  }
]
```

3. Evaluation:

Operators	Description
$regex	Allows the use of regular expressions when evaluating field values
$text	Performs a text search

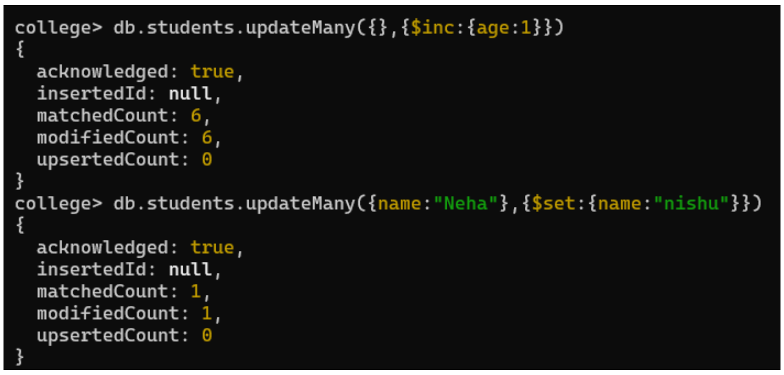


```
college> db.students.find({name:{$regex:/^V/}})
[
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873d1'),
    name: 'Vishal',
    age: 20,
    address: 'Nashik',
    identity: { adharCard: true }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873d3'),
    name: 'Vijay',
    age: 17,
    address: 'Pune',
    identity: { adharCard: false }
  }
]
college> db.students.createIndex({address:"text"})
address_text
college> db.students.find({$text:{$search:"abs"}})

college> db.students.find({$text:{$search:"Mumbai"}})
[
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873cf'),
    name: 'Ajay',
    age: 17,
    address: 'Mumbai',
    identity: { adharCard: true }
  },
  {
    _id: ObjectId('65c4f942e0f1d45e6f4873ce'),
    name: 'Neha',
    age: 15,
    address: 'Mumbai',
    identity: { adharCard: true }
  }
]
```

➢ Update Operators:

1. Fields:

Operators	Description
$inc	Increments the field value
$set	Sets the value of a field

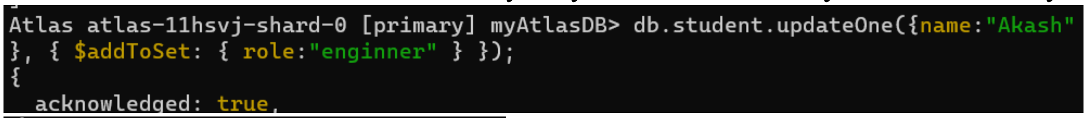


```
college> db.students.updateMany({},{$inc:{age:1}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 6,
  modifiedCount: 6,
  upsertedCount: 0
}
college> db.students.updateMany({name:"Neha"},{$set:{name:"nishu"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

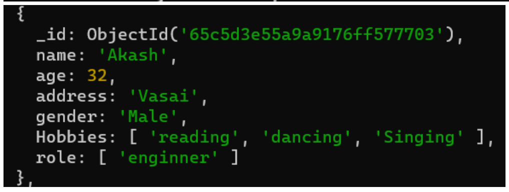
2. Array:

Operators	Description
$addToSet	Adds distinct elements to an array
$pop	Removes the first or last element of an array
$pull	Removes all elements from an array that match the query
$push	Adds an element to an array

$addToset: Adds an element to an array only if it doesn't already exist in the array.

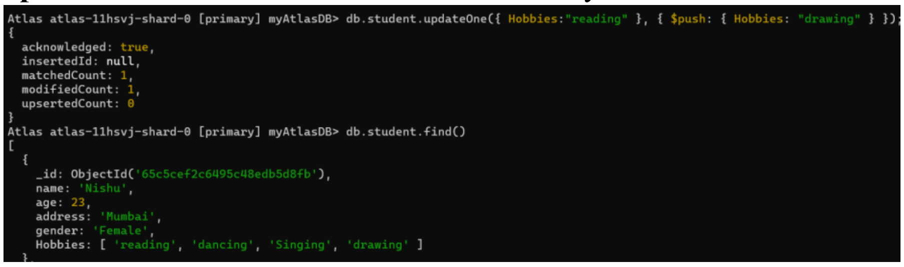


```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.updateOne({name:"Akash"
}, { $addToSet: { role:"enginner" } });
{
  acknowledged: true,
```



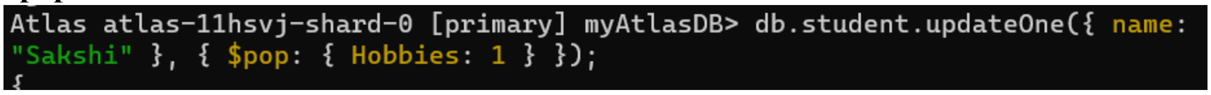
```
 {
   _id: ObjectId('65c5d3e55a9a9176ff577703'),
   name: 'Akash',
   age: 32,
   address: 'Vasai',
   gender: 'Male',
   Hobbies: [ 'reading', 'dancing', 'Singing' ],
   role: [ 'enginner' ]
 },
```

$push: Adds an element to the end of an array**.**



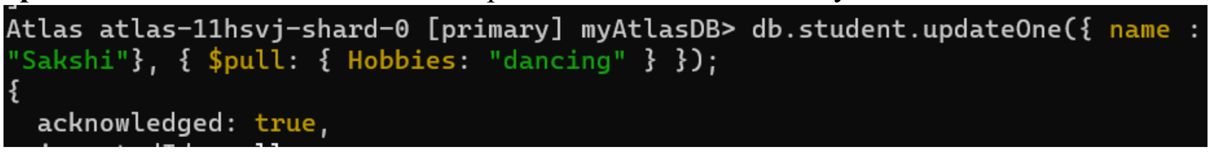
```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.updateOne({ Hobbies:"reading" }, { $push: { Hobbies: "drawing" } });
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.find()
[
  {
    _id: ObjectId('65c5cef2c6495c48edb5d8fb'),
    name: 'Nishu',
    age: 23,
    address: 'Mumbai',
    gender: 'Female',
    Hobbies: [ 'reading', 'dancing', 'Singing', 'drawing' ]
```

$pop: Use **1** to remove last element and **-1** to remove first element.



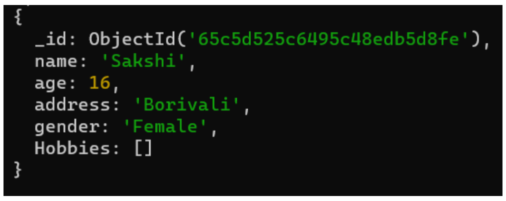
```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.updateOne({ name:
"Sakshi" }, { $pop: { Hobbies: 1 } });
```

$pull: Removes all occurrences of a specified value from an array.



```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.updateOne({ name :
"Sakshi"}, { $pull: { Hobbies: "dancing" } });
{
  acknowledged: true,
```

Final Document:

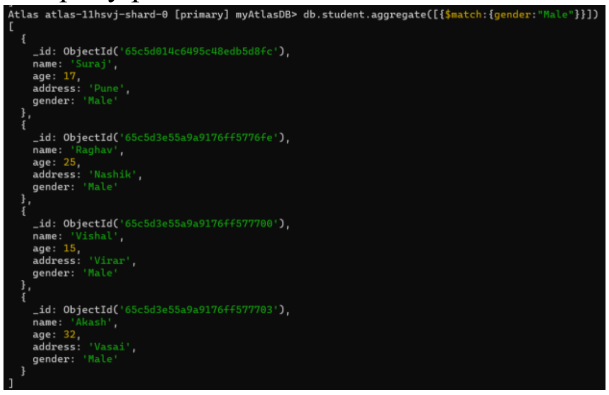


```
{
  _id: ObjectId('65c5d525c6495c48edb5d8fe'),
  name: 'Sakshi',
  age: 16,
  address: 'Borivali',
  gender: 'Female',
  Hobbies: []
}
```

MONGODB AGGREGATION PIPELINE

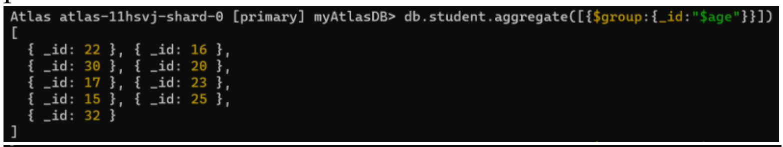
Aggregation operations allow you to group, sort, perform calculations, analyze data, and much more. Aggregation pipelines can have one or more "stages". The order of these stages is important. Each stage acts upon the results of the previous stage.

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

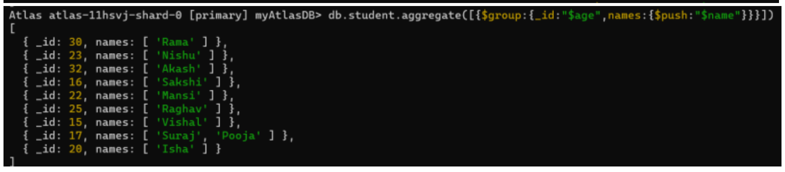


```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.aggregate([{$match:{gender:"Male"}}])
[
  {
    _id: ObjectId('65c5d014c6495c48edb5d8fc'),
    name: 'Suraj',
    age: 17,
    address: 'Pune',
    gender: 'Male'
  },
  {
    _id: ObjectId('65c5d3e55a9a9176ff5776fe'),
    name: 'Raghav',
    age: 25,
    address: 'Nashik',
    gender: 'Male'
  },
  {
    _id: ObjectId('65c5d3e55a9a9176ff577700'),
    name: 'Vishal',
    age: 15,
    address: 'Virar',
    gender: 'Male'
  },
  {
    _id: ObjectId('65c5d3e55a9a9176ff577703'),
    name: 'Akash',
    age: 32,
    address: 'Vasai',
    gender: 'Male'
  }
]
```

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



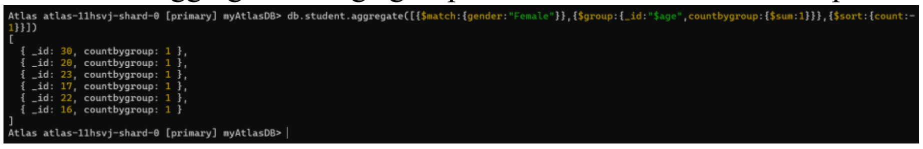
```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.aggregate([{$group:{_id:"$age"}}])
[
  { _id: 22 }, { _id: 16 },
  { _id: 30 }, { _id: 20 },
  { _id: 17 }, { _id: 23 },
  { _id: 15 }, { _id: 25 },
  { _id: 32 }
]
```



```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.aggregate([{$group:{_id:"$age",names:{$push:"$name"}}}])
[
  { _id: 30, names: [ 'Rama' ] },
  { _id: 23, names: [ 'Nishu' ] },
  { _id: 32, names: [ 'Akash' ] },
  { _id: 16, names: [ 'Sakshi' ] },
  { _id: 22, names: [ 'Mansi' ] },
  { _id: 25, names: [ 'Raghav' ] },
  { _id: 15, names: [ 'Vishal' ] },
  { _id: 17, names: [ 'Suraj', 'Pooja' ] },
  { _id: 20, names: [ 'Isha' ] }
]
```

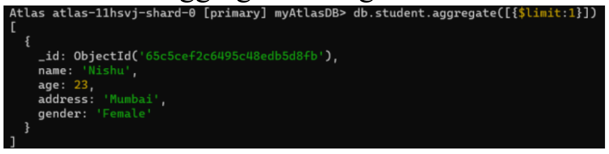
Use {$push:"$$ROOT"} to retrieve all documents.

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



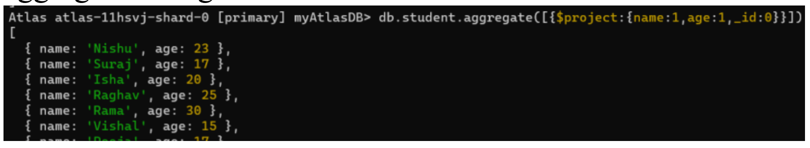
```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.aggregate([{$match:{gender:"Female"}},{$group:{_id:"$age",countbygroup:{$sum:1}}},{$sort:{count:-
1}}])
[
  { _id: 30, countbygroup: 1 },
  { _id: 20, countbygroup: 1 },
  { _id: 23, countbygroup: 1 },
  { _id: 17, countbygroup: 1 },
  { _id: 22, countbygroup: 1 },
  { _id: 16, countbygroup: 1 }
]
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> |
```

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



```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.aggregate([{$limit:1}])
[
  {
    _id: ObjectId('65c5cef2c6495c48edb5d8fb'),
    name: 'Nishu',
    age: 23,
    address: 'Mumbai',
    gender: 'Female'
  }
]
```

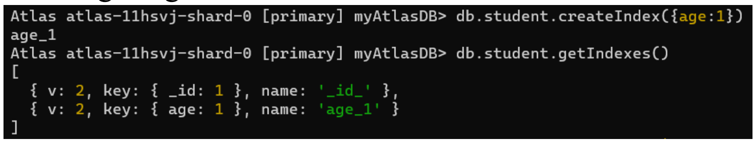
5. **$project:** This aggregation stage passes only the specified fields along to the next aggregation stage.



```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.aggregate([{$project:{name:1,age:1,_id:0}}])
[
  { name: 'Nishu', age: 23 },
  { name: 'Suraj', age: 17 },
  { name: 'Isha', age: 20 },
  { name: 'Raghav', age: 25 },
  { name: 'Rama', age: 30 },
  { name: 'Vishal', age: 15 },
```

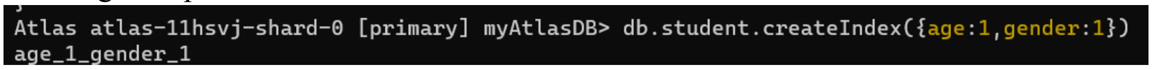
- **INDEXING:**
  In MongoDB, indexing is a technique used to improve the performance of queries by allowing the database to locate and access documents more efficiently. Indexes are data structures that store a small amount of data about the documents in a collection, and they provide a quick way to 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:



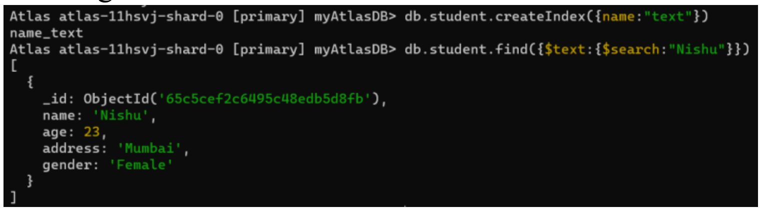
```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.createIndex({age:1})
age_1
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { age: 1 }, name: 'age_1' }
]
```

- Creating Compound Indexes:



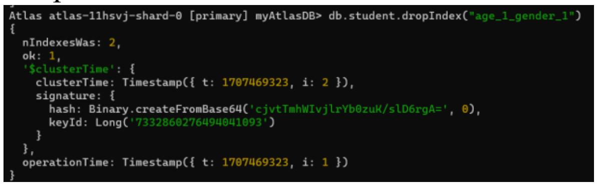
```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.createIndex({age:1,gender:1})
age_1_gender_1
```

- Creating Text Index:



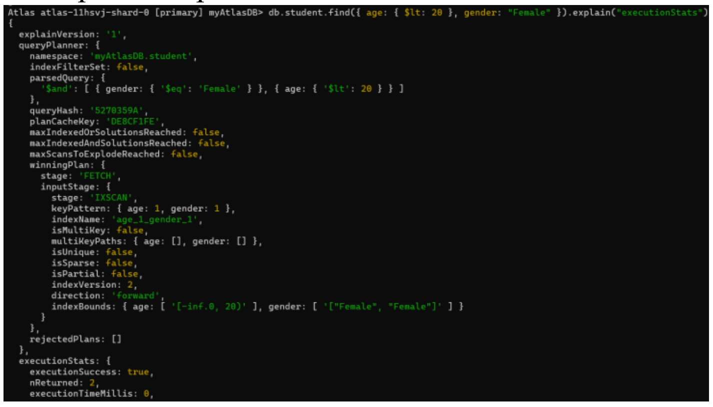
```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.createIndex({name:"text"})
name_text
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.find({$text:{$search:"Nishu"}})
[
  {
    _id: ObjectId('65c5cef2c6495c48edb5d8fb'),
    name: 'Nishu',
    age: 23,
    address: 'Mumbai',
    gender: 'Female'
  }
]
```

- Drop Index:



```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.dropIndex("age_1_gender_1")
{
  nIndexesWas: 2,
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1707469323, i: 2 }),
    signature: {
      hash: Binary.createFromBase64('cjvtTmhWIvjlrYb0zuK/slD6rgA=', 0),
      keyId: Long('7332860276494041093')
    }
  },
  operationTime: Timestamp({ t: 1707469323, i: 1 })
}
```

Example: Compound Indexes

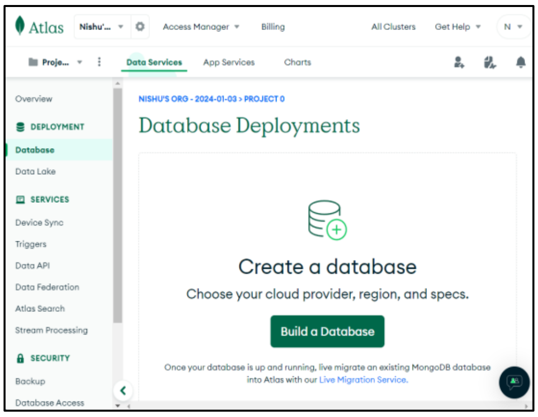


```
Atlas atlas-11hsvj-shard-0 [primary] myAtlasDB> db.student.find({ age: { $lt: 20 }, gender: "Female" }).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'myAtlasDB.student',
    indexFilterSet: false,
    parsedQuery: {
      '$and': [ { gender: { '$eq': 'Female' } }, { age: { '$lt': 20 } } ]
    },
    queryHash: '5270359A',
    planCacheKey: 'DE8CF1FE',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { age: 1, gender: 1 },
        indexName: 'age_1_gender_1',
        isMultiKey: false,
        multiKeyPaths: { age: [], gender: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { age: [ '[-inf.0, 20)' ], gender: [ '["Female", "Female"]' ] }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 2,
    executionTimeMillis: 0,
```

CLUSTER CREATION

Step 1: Visit: https://www.mongodb.com/cloud/atlas → Click on "Get Started Free" →Fill in Sign-Up Form → Click on Get Started.

Step 2: Configure a Cluster: Click on Build Database → Choose Plan (FREE) → Choose Cloud Provider, Region, and Cluster Name (Cluster0) → Create.



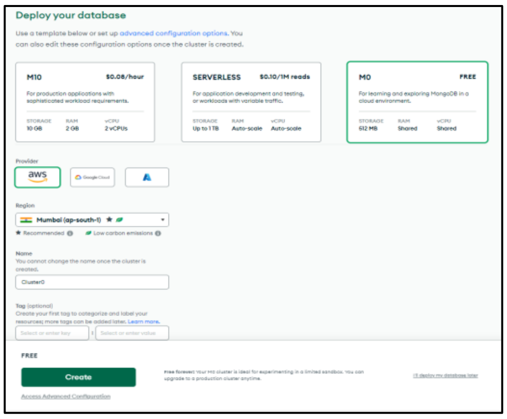
NISHU'S ORG - 2024-01-03 > PROJECT 0

Database Deployments

Create a database

Choose your cloud provider, region, and specs.

Once your database is up and running, live migrate an existing MongoDB database into Atlas with our Live Migration Service.



Deploy your database

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

M10	$0.08/hour	SERVERLESS	$0.10/1M reads	M0	FREE
For production applications with sophisticated workload requirements.		For application development and testing, or workloads with variable traffic.		For learning and exploring MongoDB in a cloud environment.	

	STORAGE	RAM	vCPU
M10	10 GB	2 GB	2 vCPUs
SERVERLESS	Up to 1 TB	Auto-scale	Auto-scale
M0	512 MB	Shared	Shared

Provider: aws

Region: Mumbai (ap-south-1)

★ Recommended   Low carbon emissions

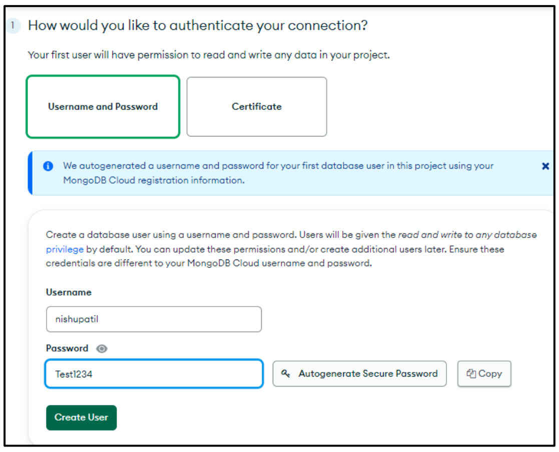
Name
You cannot change the name once the cluster is created.
Cluster0

Tag (optional)
Create your first tag to categorize and label your resources; more tags can be added later. Learn more.

FREE

Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Step 3: Create User: Add username and password → Create User → Click on Add My Current IP Address→ Finish and Close.



How would you like to authenticate your connection?

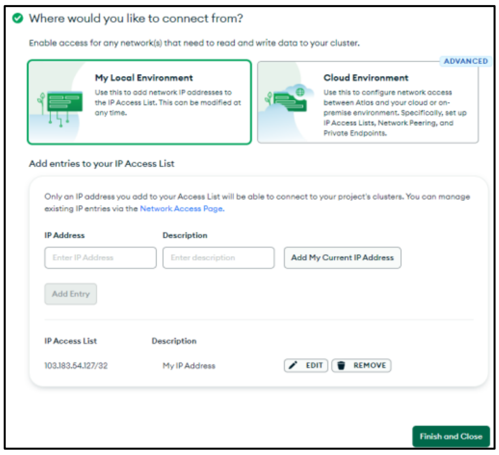
Your first user will have permission to read and write any data in your project.

We autogenerated a username and password for your first database user in this project using your MongoDB Cloud registration information.

Create a database user using a username and password. Users will be given the *read and write to any database* privilege by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password.

Username: nishupatil

Password: Test1234



Where would you like to connect from?

Enable access for any network(s) that need to read and write data to your cluster.

My Local Environment: Use this to add network IP addresses to the IP Access List. This can be modified at any time.

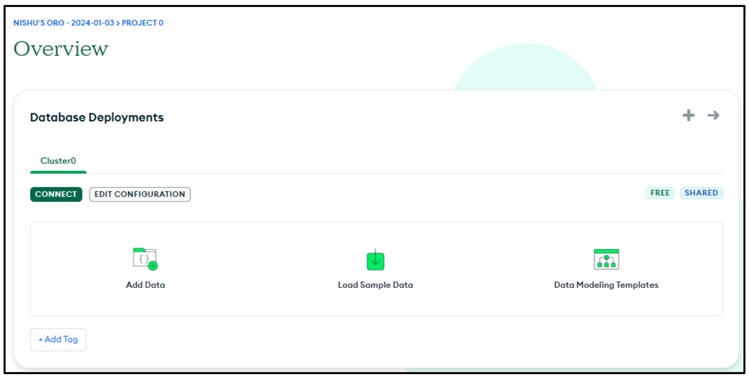
Cloud Environment (ADVANCED): Use this to configure network access between Atlas and your cloud or on-premise environment. Specifically, set up IP Access Lists, Network Peering, and Private Endpoints.

Add entries to your IP Access List

Only an IP address you add to your Access List will be able to connect to your project's clusters. You can manage existing IP entries via the Network Access Page.

IP Access List	Description
103.183.54.127/32	My IP Address

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



NISHU'S ORG - 2024-01-03 > PROJECT 0

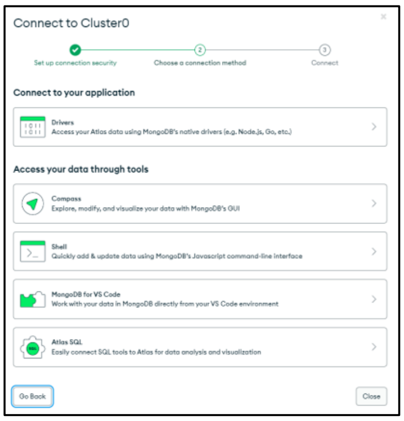
Overview

Database Deployments

Cluster0

You can connect MongoDB Atlas to following:

- **Connect Cluster0 to Compass:** Select Compass → Copy the Connection string.


Connect to Cluster0

Set up connection security · Choose a connection method · Connect

Connect to your application

Drivers
Access your Atlas data using MongoDB's native drivers (e.g. Node.js, Go, etc.)

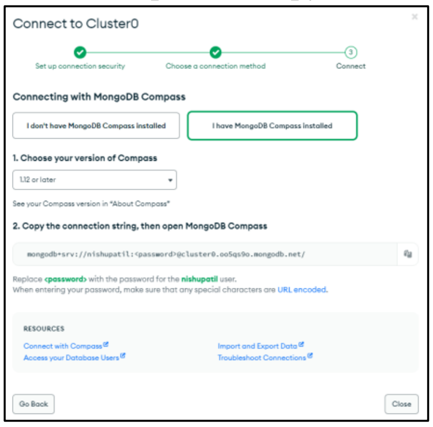
Access your data through tools

Compass
Explore, modify, and visualize your data with MongoDB's GUI

Shell
Quickly add & update data using MongoDB's Javascript command-line interface

MongoDB for VS Code
Work with your data in MongoDB directly from your VS Code environment

Atlas SQL
Easily connect SQL tools to Atlas for data analysis and visualization


Connect to Cluster0

Set up connection security · Choose a connection method · Connect

Connecting with MongoDB Compass

I don't have MongoDB Compass installed [✓ I have MongoDB Compass installed]

1. Choose your version of Compass

1.12 or later

See your Compass version in "About Compass"

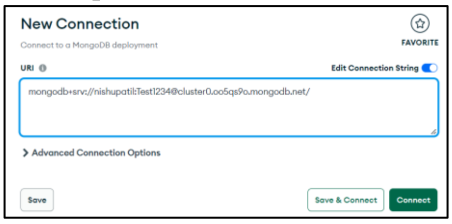
2. Copy the connection string, then open MongoDB Compass

```
mongodb+srv://nishupatil:<password>@cluster0.oo5qs9o.mongodb.net/
```

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

RESOURCES
Connect with Compass
Access your Database Users
Import and Export Data
Troubleshoot Connections

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

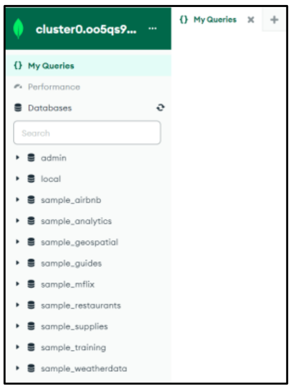

New Connection

Connect to a MongoDB deployment

URI

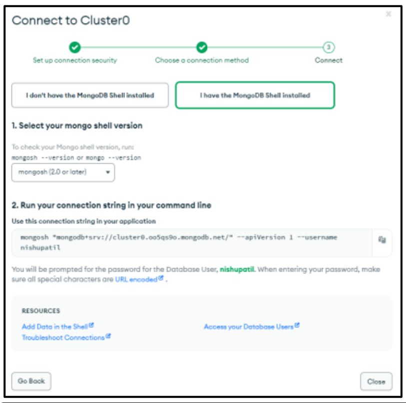
```
mongodb+srv://nishupatil:Test1234@cluster0.oo5qs9o.mongodb.net/
```

Databases in Cluster0:



admin
local
sample_airbnb
sample_analytics
sample_geospatial
sample_guides
sample_mflix
sample_restaurants
sample_supplies
sample_training
sample_weatherdata

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


Connect to Cluster0

Set up connection security · Choose a connection method · 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)

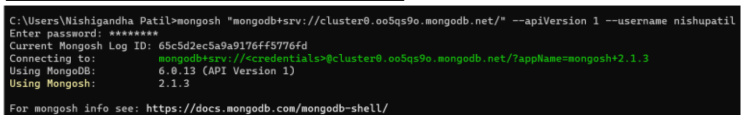
2. Run your connection string in your command line

Use this connection string in your application

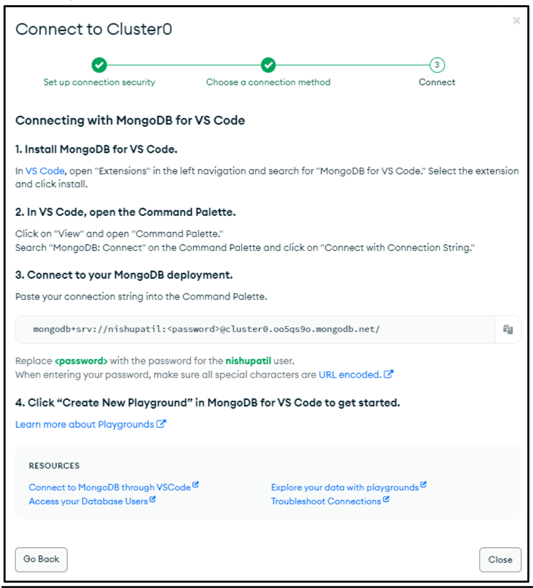
```
mongosh "mongodb+srv://cluster0.oo5qs9o.mongodb.net/" --apiVersion 1 --username
nishupatil
```

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

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


```
C:\Users\Nishigandha Patil>mongosh "mongodb+srv://cluster0.oo5qs9o.mongodb.net/" --apiVersion 1 --username nishupatil
Enter password: ********
Current Mongosh Log ID: 65c5d2ec5a9a9176ff5776fd
Connecting to:          mongodb+srv://<credentials>@cluster0.oo5qs9o.mongodb.net/?appName=mongosh+2.1.3
Using MongoDB:          6.0.13 (API Version 1)
Using Mongosh:          2.1.3

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

- **Connect Atlas to VS Code:** Open VS Code → 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.



Connect to Cluster0

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://nishupatil:<password>@cluster0.oo5qs9o.mongodb.net/
```

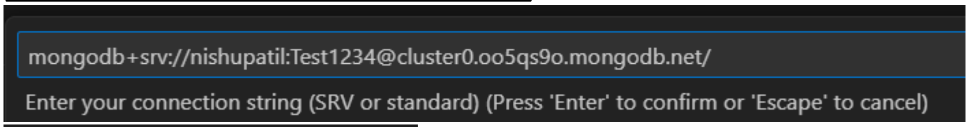
Replace **<password>** with the password for the **nishupatil** user.
When entering your password, make sure all special characters are URL encoded.

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

Learn more about Playgrounds

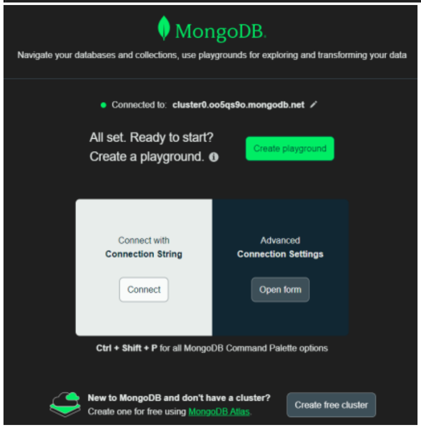
RESOURCES

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



```
mongodb+srv://nishupatil:Test1234@cluster0.oo5qs9o.mongodb.net/
```

Enter your connection string (SRV or standard) (Press 'Enter' to confirm or 'Escape' to cancel)



Navigate your databases and collections, use playgrounds for exploring and transforming your data

Connected to: **cluster0.oo5qs9o.mongodb.net**

All set. Ready to start?
Create a playground.

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.

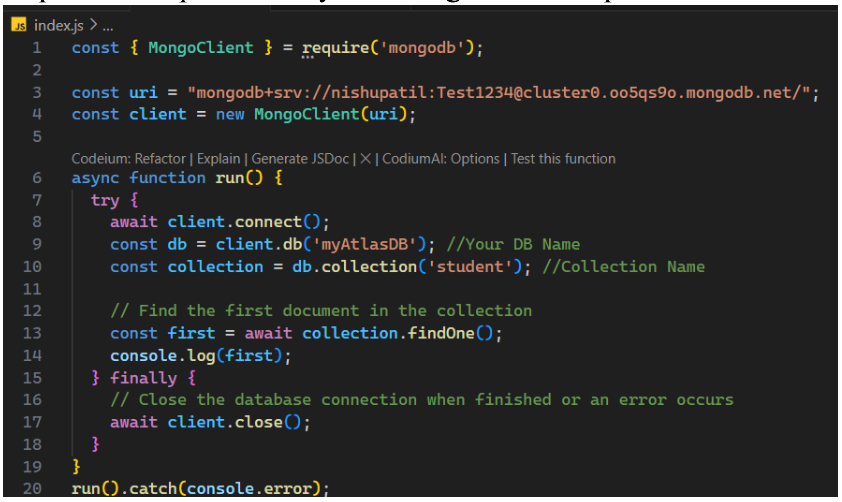
MONGODB DRIVERS

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

Following is the current officially supported drivers:

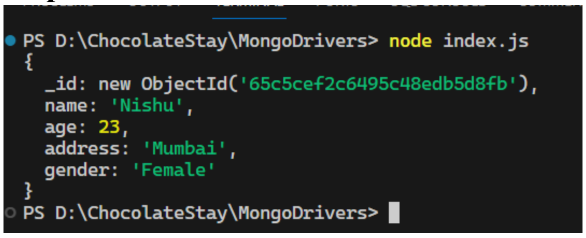
C	C++	C#	Go
Java	Node.js	PHP	Python
Ruby	Rust	Scala	Swift

- **Node.js Driver:**
  To use MongoDB with Node.js, you will need to install the mongodb package in your Node.js project.
  Step 1: Use **npm install mongodb** command in your project terminal.
  Step 2: Create an index.js file in your project directory.
  Step 3: Connection String
  Go to MongoDB Atlas → Go to Database → Connect → Copy Connection String
  Replace the <password>your MongoDB Atlas password.


index.js > ...
Codeium: Refactor | Explain | Generate JSDoc | × | CodiumAI: Options | Test this function

```javascript
const { MongoClient } = require('mongodb');

const uri = "mongodb+srv://nishupatil:Test1234@cluster0.oo5qs9o.mongodb.net/";
const client = new MongoClient(uri);

async function run() {
  try {
    await client.connect();
    const db = client.db('myAtlasDB'); //Your DB Name
    const collection = db.collection('student'); //Collection Name

    // Find the first document in the collection
    const first = await collection.findOne();
    console.log(first);
  } finally {
    // Close the database connection when finished or an error occurs
    await client.close();
  }
}
run().catch(console.error);
```

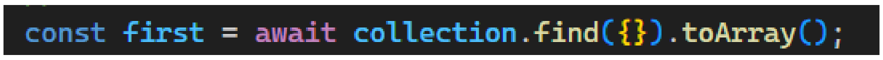
Step 4: Run the file in the terminal



```
PS D:\ChocolateStay\MongoDrivers> node index.js
{
  _id: new ObjectId('65c5cef2c6495c48edb5d8fb'),
  name: 'Nishu',
  age: 23,
  address: 'Mumbai',
  gender: 'Female'
}
PS D:\ChocolateStay\MongoDrivers> 
```

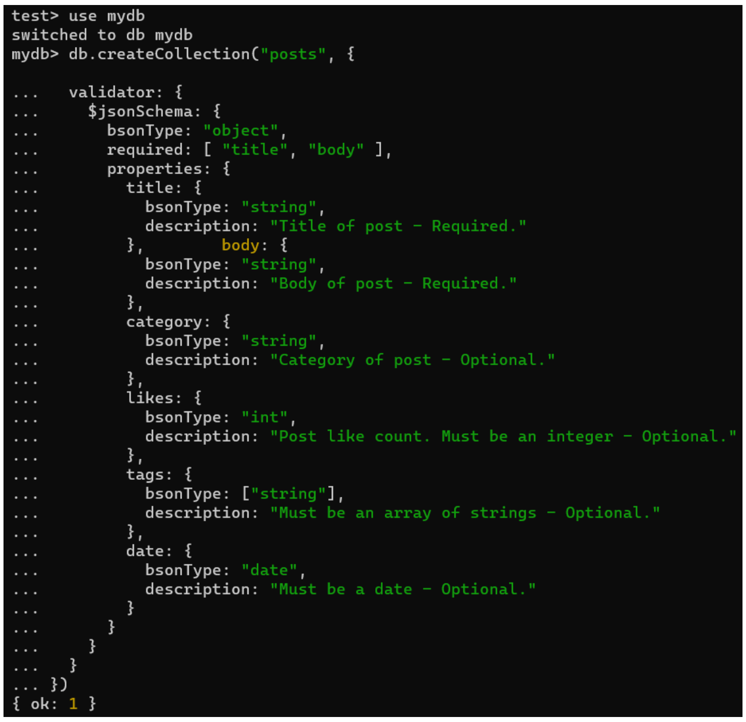
In above example we can replace the collection.findOne() with find(), insertOne(), insertMany(), updateOne(), updateMany(), deleteOne(), deleteMany(), or aggregate().

To find all documents:



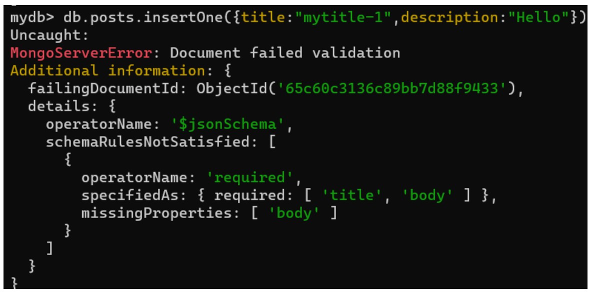
```javascript
const first = await collection.find({}).toArray();
```

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



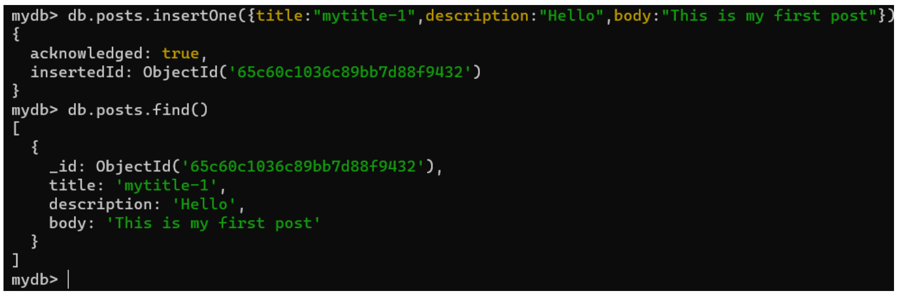
```
test> use mydb
switched to db mydb
mydb> db.createCollection("posts", {

...   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:



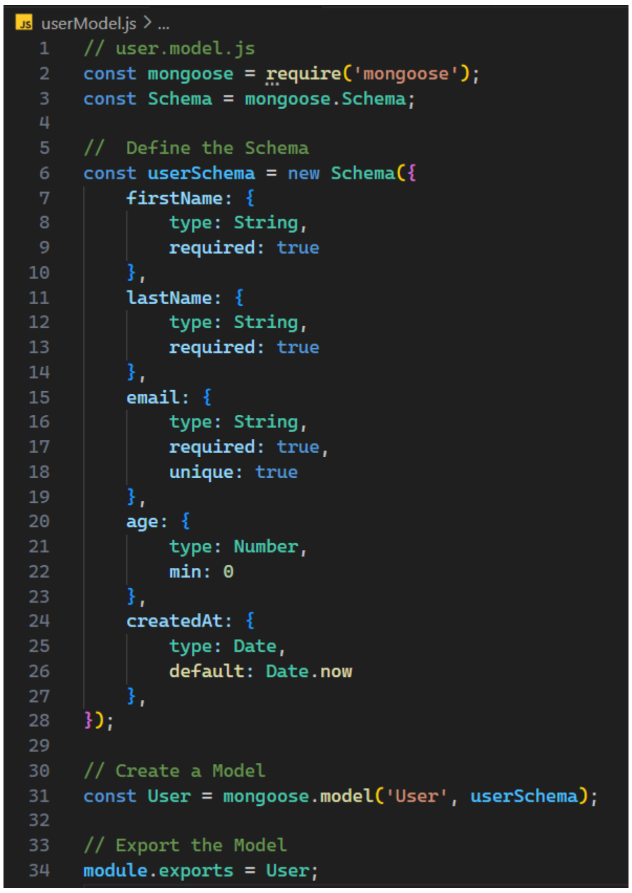
```
mydb> db.posts.insertOne({title:"mytitle-1",description:"Hello"})
Uncaught:
MongoServerError: Document failed validation
Additional information: {
  failingDocumentId: ObjectId('65c60c3136c89bb7d88f9433'),
  details: {
    operatorName: '$jsonSchema',
    schemaRulesNotSatisfied: [
      {
     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

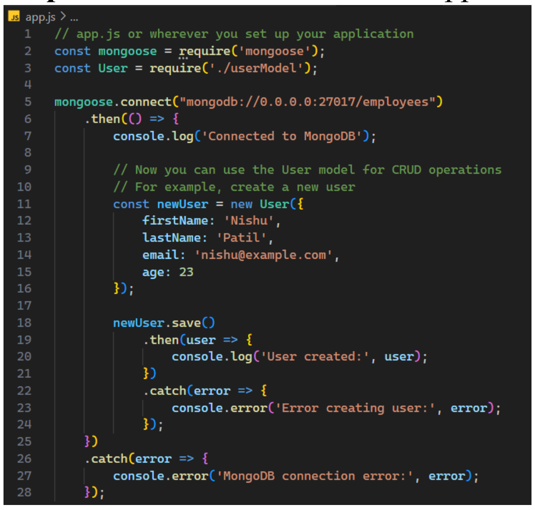
➢ **How to create Schema using mongoose:**
Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a higher-level, schema-based abstraction over the MongoDB driver, making it easier to interact with MongoDB databases using JavaScript or TypeScript.
Step 1: Install mongoose: **npm install mongoose**
Step 2: Run the MongoDB Server.
Step 3: Define the Schema (Create userModel.js)



userModel.js > ...

```javascript
// 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;
```

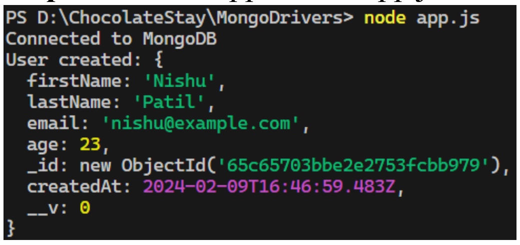
Step 4: Use the Schema in Your Application (Create app.js)



app.js > ...

```javascript
// 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: 'Nishu',
            lastName: 'Patil',
            email: 'nishu@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);
    });
```

Step 5: Run the app – node app.js



```
PS D:\ChocolateStay\MongoDrivers> node app.js
Connected to MongoDB
User created: {
  firstName: 'Nishu',
  lastName: 'Patil',
  email: 'nishu@example.com',
  age: 23,
  _id: new ObjectId('65c65703bbe2e2753fcbb979'),
  createdAt: 2024-02-09T16:46:59.483Z,
  __v: 0
}
```