

## Unit-4

### MongoDB

#### Introduction to module overview in mongodb

In MongoDB, a module is a collection of related functions and data that can be reused in multiple applications. Modules are used to organize and encapsulate code, making it more modular and easier to maintain.

MongoDB provides a flexible module system that allows developers to define and use modules in their applications. Modules can be defined as JavaScript files or as compiled binary files, and can be loaded dynamically or statically depending on the requirements of the application.

The module system in MongoDB provides several benefits:

**Reusability:** Modules can be reused across multiple applications, reducing code duplication and improving code maintainability.

**Encapsulation:** Modules can encapsulate code and data, hiding implementation details and reducing complexity.

**Modularity:** Modules can be developed and tested independently, making it easier to manage large code bases.

**Flexibility:** MongoDB's module system is flexible and can be adapted to different use cases and application architectures.

To use a module in MongoDB, you can define it as a file with the .js extension or as a compiled binary file, and then import it into your application using the `require()` function. Once imported, you can access the functions and data defined in the module using the dot notation.

#### Document Database Overview in mongodb

- Document data bases earlier we work is with relational data bases, such kind of data bases, data is stored in tables. Each table contains rows and columns.
- In MongoDB, we can use document data bases, data is stored in separate documents and one document is not dependent on the other documents.
- Each data is stored in separate documents. One document is having one structure and another document having another structure.
- MongoDB is a document-oriented database that stores data in flexible, semi-structured documents in BSON (Binary JSON) format.
- In MongoDB, a document is a data structure composed of field-value pairs, similar to a JSON(JavaScript Object Notation) object.
- A document can have various fields with various data types like strings, numbers and soon. And also a document can have nested sub-documents, arrays, and arrays of sub-documents, allowing for complex and flexible data models.

MongoDB's document database model provides several benefits:

**Flexible schema:** Documents in MongoDB can have varying fields, allowing for flexible schema design and reducing the need for migrations.

**Rich query language:** MongoDB provides a rich query language that supports complex queries, aggregations, and indexing for efficient data retrieval.

**Horizontal scalability:** MongoDB supports sharding, allowing for horizontal scalability of the database across multiple nodes.

**High performance:** MongoDB is designed to be high-performance, with support for in-memory caching, automatic sharding, and parallel processing of queries.

**Developer productivity:** MongoDB's flexible schema and document model allows developers to work with data in a way that is natural and intuitive, reducing development time and increasing productivity.

MongoDB is commonly used for a variety of applications, including web and mobile applications, content management systems, and analytics. Its flexible schema and document model make it a good choice for applications with constantly changing data requirements and complex data models.

### **Understanding JSON in mongodb**

- JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy to read and write, and can be easily converted to other data formats.
- In MongoDB, data is stored in BSON (Binary JSON) format, which is a binary-encoded serialization of JSON data. BSON is similar to JSON, but includes additional data types such as Date and Binary, and supports more efficient serialization and deserialization.
- MongoDB uses JSON-like syntax for defining and manipulating documents, which are the primary unit of data in MongoDB.
- Documents in MongoDB are similar to JSON objects, consisting of field-value pairs, and can be nested to create complex data structures.

### **JSON Format:**

Each JSON object is a set of Key-Value pairs, place in a set of curly braces ({ }). Each Key-Value pair is separated with comma(,) and each key must be placed in quotes(" "). If we remove quotes for key name, it is invalid syntax in JSON. After each key name, there should be a colon(:), after that value comes.

### **JSON Syntax:**

```
{  
  "name1": value1,  
  "name2": value2,  
  "name3": value3,  
  ....  
  ....  
  ....  
  "nameN": valueN  
}
```

### **Example:**

```
{  
  "String" : "That is a sample string",  
  "number" : 6,  
  "boolean" : true,  
  "array" : ["first", "second", "third"],  
  "object" : {  
    "key1" : 1,  
    "key2" : "sample",  
    "key3" : true,  
  }  
}
```

```
    "key4" : [1,2,3],
    "key5" : {
      "nestedobjkey1" : 10,
      "nestedobjkey2" : false
    },
    "null":null
  }
```

JSON Supports 6 Datatypes:

- String
- Number
- Boolean
- Array
- Object
- Null

### MongoDB Structure and Architecture

#### MongoDB Structure:

- Each MongoDB database consists of Collections and inside of each collection, there are set of documents. Documents are grouped into one collection using common characteristics.
- For example, you can group people into persons collection, and each document in this collection represents one person.
- Usually, each document in the collection having same set of fields. In some documents, some fields are common and some may be different.
- It is not mandatory to have same set of fields in all documents inside one collection.

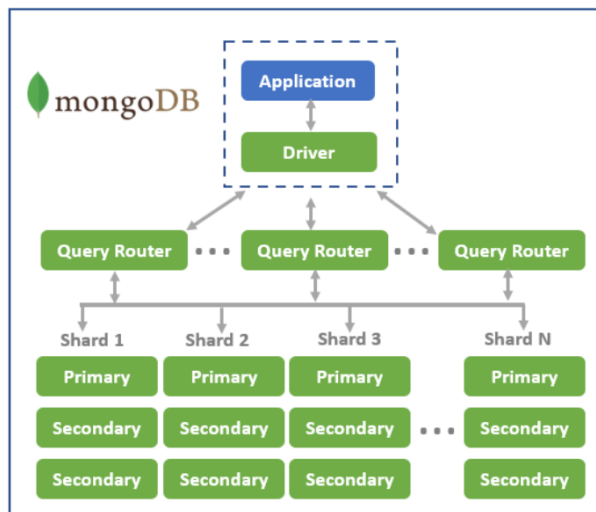


#### MongoDB Architecture:

MongoDB is a distributed, document-oriented database that is designed to scale horizontally across multiple nodes. MongoDB's architecture is based on a master-slave replication model, with each node serving a specific purpose in the cluster.

#### MongoDB Application:

MongoDB is an **open source NoSQL database management program**. NoSQL is used as an alternative to traditional relational databases. NoSQL databases are quite useful for working with large sets of distributed data. MongoDB is a tool that can manage document-oriented information, store or retrieve information.



### MongoDB Driver:

A database driver is a **computer program that implements a protocol (ODBC or JDBC) for a database connection**. The official MongoDB Node.js driver **allows Node.js applications to connect to MongoDB and work with data**. The driver features an asynchronous API which allows you to interact with MongoDB using traditional callbacks.

### Query Router:

Query Router Sharding is transparent to applications; whether there is one or one hundred shards, the application code for querying MongoDB is the same. **Applications issue queries to a query router that dispatches the query to the appropriate shards.**

### Shards:

MongoDB supports sharding, which is a method for horizontally scaling the database across multiple nodes. Sharding involves partitioning the data into multiple shards, each of which is stored on a separate node. MongoDB's sharding architecture is based on a range-based partitioning scheme, where each shard is responsible for a range of values based on the shard key.

### Replica sets:

MongoDB's replication architecture is based on replica sets, which consist of multiple nodes that replicate data to ensure high availability and fault tolerance. Each replica set has one primary node that handles all write operations, and multiple secondary nodes that replicate data from the primary node.

### Config servers:

MongoDB's sharding architecture requires a separate set of servers, called config servers, to manage metadata and configuration information for the sharded cluster.

### Clients:

Applications interact with MongoDB through drivers and APIs that provide a simple and intuitive interface for working with the database. MongoDB supports a wide range of programming languages and platforms, making it easy to integrate with existing applications and services.

### **MongoDB Remote Management**

We can connect MongoDB server using mongoDB shell located on the same server. But the server is in remote location, we have to manage it.

So, mongo shell is located on your local MAC of the CPU. Usually, mongo shell is in built on MongoDB server. It means we need to install MongoDB server, no need to install MongoDB shell.

We can also use mongo shell embedded in GUI applications such as robust mongoDB compass. But if you want to interact with mongoDB server from another server. For example, your backend or frontend server. In this case, you use mongoDB driver.

Different drivers are available for different servers such as Node .js, Java, .NET, Ruby and Python and so on. In all cases, you need to connect with mongoDB server remotely, in order to perform CRUD operations such as Create, Read, Update and Delete.

If you need to connect to MongoDB server remotely, make sure that you use SSL Encryption. Otherwise, all data for mongoDB server will be sent as plain text. Both mongoDB shell and mongoDB server will support SSL Encryption.

MongoDB Remote Management is a set of tools and services that enable users to monitor, manage, and secure MongoDB instances from a remote location. MongoDB Remote Management includes the following components:

**MongoDB Cloud Manager:** Cloud Manager for monitoring, automating, and backing up MongoDB instances. Cloud Manager includes features such as alerts, performance monitoring, automation, and backup and recovery.

**MongoDB Atlas:** fully-managed cloud database service that provides an easy-to-use interface for deploying, scaling, and managing MongoDB clusters. Atlas includes features such as automated backups, point-in-time recovery, and automated scaling, making it easy to manage MongoDB instances from a remote location.

**Ops Manager:** for managing MongoDB instances, including monitoring, automation, and backup and recovery. Ops Manager includes features such as advanced monitoring, automation, and alerting, making it an ideal solution for managing large-scale MongoDB deployments.

**Security:** to secure MongoDB instances from a remote location. These features include access control, encryption, and auditing.

### **Installing MongoDB on the local computer (Mac or Windows)**

We can install MongoDB in two ways,

1. By using Homebrew
2. Manual installation

#### **By using Homebrew:**

1. Click installing with Homebrew option in MongoDB website.
2. Click on Homebrew link in home page.
3. At Install Homebrew option, there is a link. Copy the link and open your terminal(command prompt)

4. At terminal, paste the above link, press enter. It will ask you for password, you can enter your password and click Enter.

### What is Homebrew?

Homebrew is a MAC OS package manager that simplifies the process of installing internal and external packages that are not installed by default.

Homebrew knows all the current regions of various software packages.

- a. To install homebrew, at your terminal, enter command **"brew update"**.  
This updates the current Homebrew package with latest version.
- b. To install MongoDB, enter command **"brew install mongodb"** at terminal.
- c. Open a terminal window and create a data directory for MongoDB data using the command: **"sudo mkdir -p /data/db"**, after that **enter your password**.
- d. Then, give the write permissions to the directory using command:  
**sudo chmod 777 /data/db**
- e. After that, start the MongoDB daemon process using command: **mongod**
- f. Then, finally it connects and displays that **"waiting for connections on the port 27017"**.
- g. Now, try to connect mongoDB using mongo shell.
- h. In order to launch mongo shell, we use the command **"mongo"** at prompt.
- i. After that you can use database by using various queries.

### Example:

```
>show dbs
```

```
Admin      0.000GB
```

```
Local      0.000GB
```

To show the mongoDB version, use command:

```
>db.version()
```

```
3.6.3
```

### For Windows:

Go to the MongoDB website and download the appropriate version of MongoDB for Windows.

Open the downloaded file and follow the installation wizard.

During the installation, choose the "Complete" option, which includes MongoDB Compass (a graphical user interface for MongoDB).

Once the installation is complete, open a Command Prompt window. Here we need to create MongoDB data directory using the following command:

```
>md \data\db
```

Now, you can see the data directory is created in C drive.

After that navigate to the MongoDB bin directory and copy the link at command prompt: **>C:\Program Files\MongoDB\Server<version>\bin\mongod.exe**

The above command connects(launching) mongoDB server and display the message **"waiting for the connections on port 27017"**.

Now, we need to launch the mongo shell, we use the command at command prompt:

```
>C:\Program Files\MongoDB\Server<version>\bin\mongo.exe
```

After that, you can use database by using various queries.

**Example:**

To check the default databases,

```
>show dbs
```

```
Admin          0.000GB
```

```
Config         0.000GB
```

```
Local          0.000GB
```

To show the mongoDB version, use command:

```
>db.version()
```

```
3.6.3
```

**Create MongoDB Atlas Cluster**

MongoDB Atlas offers a cloud-based platform for hosting MongoDB databases, and the free tier allows developers to explore and experiment with the database without incurring any costs. Follow the steps below to set up your own MongoDB Atlas cluster.

**Steps**

To create a free MongoDB Atlas cluster, follow these steps:

**Step 1:** Visit the MongoDB Atlas Website. To get started, go to the MongoDB Atlas website at <https://www.mongodb.com/cloud/atlas>. Click on the “Get started free/sign up” button to proceed.

**Step 2:** Create a MongoDB Atlas Account Provide the required information, including your name, email address, and password, to create a MongoDB Atlas account. Click “Get Started” to move forward. You can also create an account using google email.



## Create your account

Have an account? [Log in now](#)



Or with email and password

### Email Address

We recommend using your work email

### First Name

### Last Name

### Password

### Company Name

Optional

☐ I accept the [Privacy Policy](#) and the [Terms of Service](#)

☐ I'm not a robot



Sign up

**Step 3:** Choose a Project Name Select a suitable project name for your cluster. This helps you organize and manage your MongoDB Atlas resources effectively. Once you've chosen a name, click "Next" to continue.

**Step 4:** Select Cloud Provider and Region You can choose between AWS, Google Cloud, or Azure as your cloud provider. Pick the region that is closest to your location to ensure optimal performance. Click "Create New Cluster" to proceed. This steps is important if you want to integrate a VPC peering with your cloud provider.



**MongoDB**

### 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**  
For production applications with sophisticated workload requirements.

STORAGE	RAM	vCPU
10 GB	2 GB	2 vCPUs

**SERVERLESS** **\$0.09/1M reads**  
For application development and testing, or workloads with variable traffic.

STORAGE	RAM	vCPU
Up to 1 TB	Auto-scale	Auto-scale

**M0** **FREE**  
For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM	vCPU
512 MB	Shared	Shared

**Provider**

☐ AWS ☒ Google Cloud ☐ Azure

**Region** ★ Recommended region ⓘ

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

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

**Step 5:** Select the Free Tier (M0 Sandbox) On the following page, scroll down to the “Cluster Tier” section. Here, you can select the free tier option called “M0 Sandbox”. This tier provides a free cluster with limited resources, perfect for development and testing purposes. Click “Create Cluster” to initiate the cluster creation process.

**Step 6:** Wait for Cluster Creation MongoDB Atlas will now start setting up your cluster. This process may take a few minutes.

**Step 7:** Explore Cluster Configuration Once the cluster is created, you’ll be redirected to the cluster overview page. Here, you can explore various configuration options for your cluster. The default settings are generally sufficient for most development needs. You can configure your database username and password. Let’s configure **appUser** and **0lR040clSGnMy1Ag** for testing purpose. Finally click on “Create User”.

Data Services

App Services

Charts

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

Username and Password

Certificate

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

appUser

Password

0lR040clSGnMylAg

Autogenerate Secure Password

Copy

Create User

2 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

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.

IP Address

Enter IP Address

Add My Current IP Address

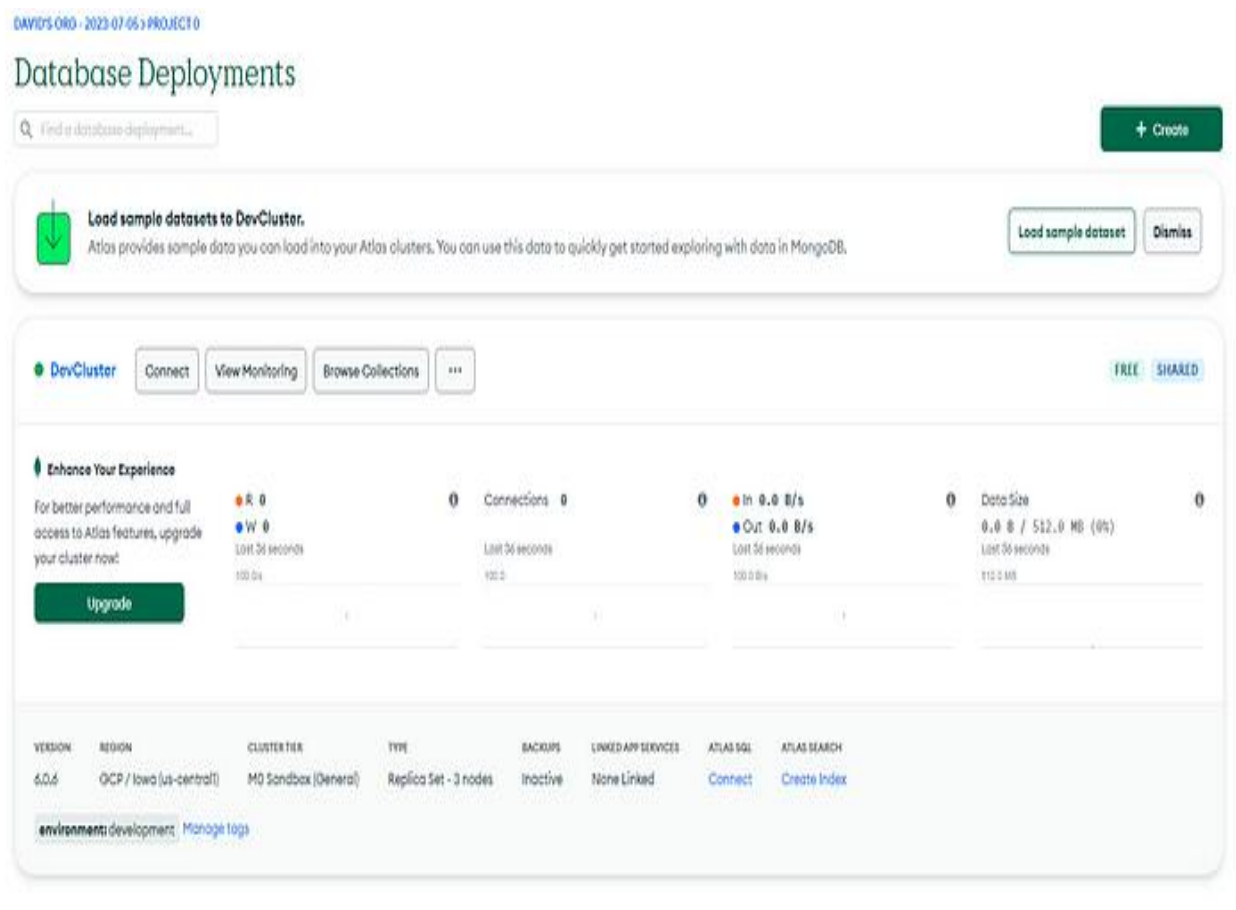
Description

Enter description

Add Entry

VLITS, Vadlamudi.

Page 10



**Fig: MongoDB Cluster created**

**Step 8:** (Optional) Configure IP Whitelist In the “Cluster Overview” page, you can find the “IP Whitelist” section. You can add specific IP addresses to allow access to your cluster. By default, your current IP address is added to the whitelist. This step is optional but can be useful if you need to access the cluster from specific IP addresses.

**Step 9:** Connect to the Cluster To connect to your newly created cluster, click the “Connect” button on the cluster overview page. In the “Connect to Cluster” modal, choose the “Connect your application” option.

Connect to DevCluster

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://appUser:<password>@devcluster.iiejdkx.mongodb.net/

You will be prompted for the password for the **appUser** user's (Database User) username. When entering your password, make sure that any special characters are [URL encoded](#).

RESOURCES

[Connect with Compass](#) [Import and Export Data](#)  
[Access your Database Users](#) [Troubleshoot Connections](#)

Go Back Close

Our string connection is **mongodb+srv://appUser:<password>@devcluster.iiejdkx.mongodb.net/** we will need to replace password with the real password.

**Step 10:** Download and install Mongo Compass.

**Step 11:** Once you have installed Mongo Compass, go and create a new connection and paste the connection string from above, remember to replace the password. Finally click on save & connect.

Mongo Atlas Free Cluster

Connect to a MongoDB deployment

FAVORITE

URI Edit Connection String

mongodb+srv://appUser:\*\*\*\*\*@devcluster.iiejdkx.mongodb.net/

Advanced Connection Options

General Authentication TLS/SSL Proxy/SSH In-Use Encryption Advanced

Connection String Scheme

mongodb mongodb+srv

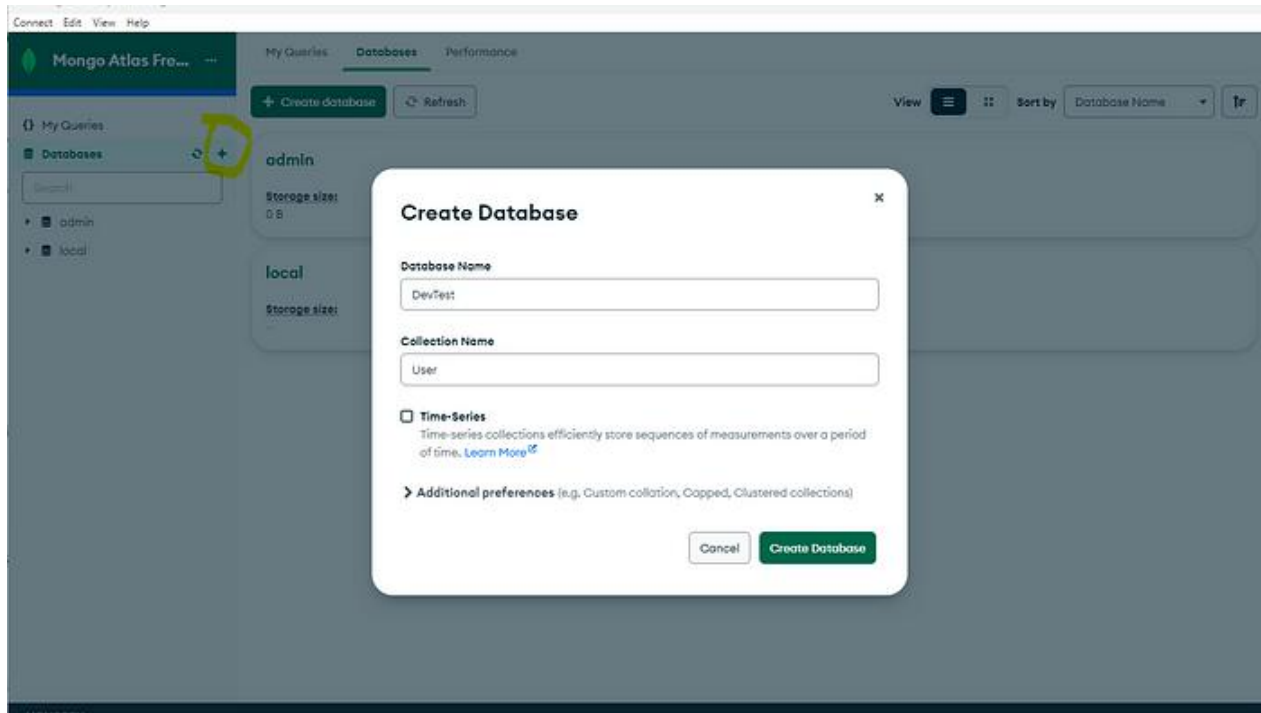
DNS Seed List Connection Format. The +srv indicates to the client that the hostname that follows corresponds to a DNS SRV record.

Hostname

devcluster.iiejdkx.mongodb.net

Save Connect

**Step 12:** Click on create new database (yellow circle)

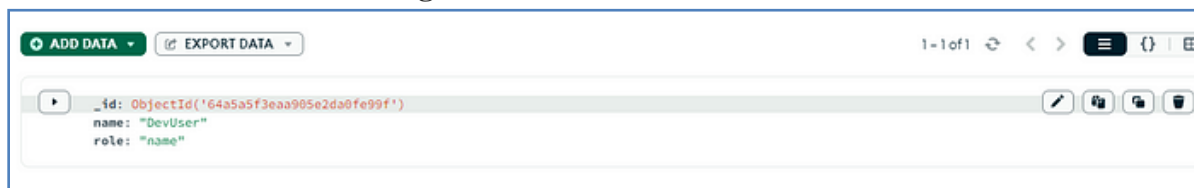


**Fig: Create new database& collection using Mongo Compass**

**Step 13:** Click on “add data” button, then select insert document and paste the following Json:

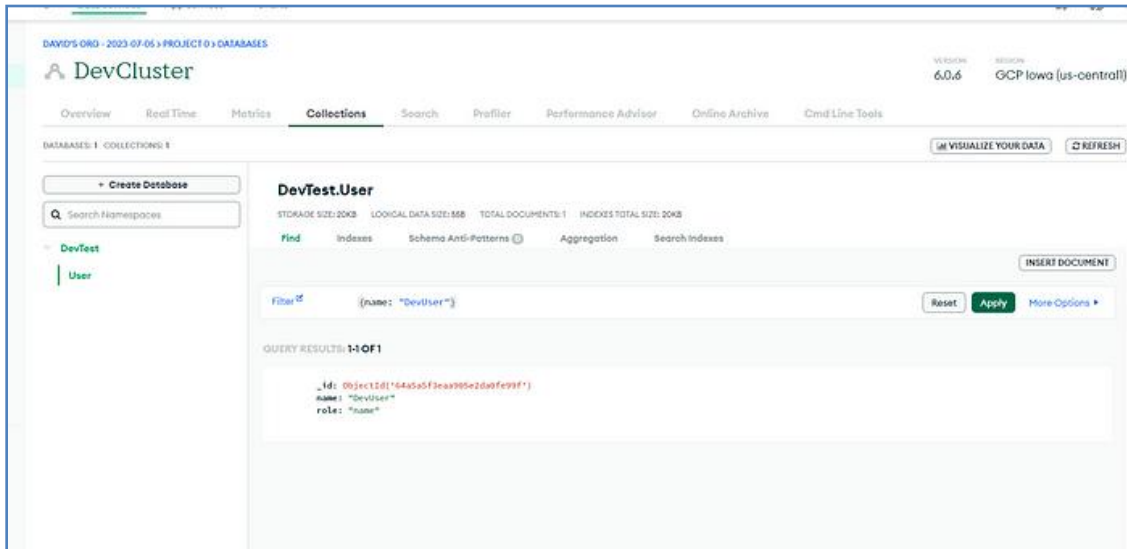
```
/**
 * Paste one or more documents here
 */
{
  "_id": {
    "$oid": "64a5a5f3eaa905e2da0fe99f"
  },
  "name": "DevUser",
  "role": "name"
}
```

**Fig: Click on insert**



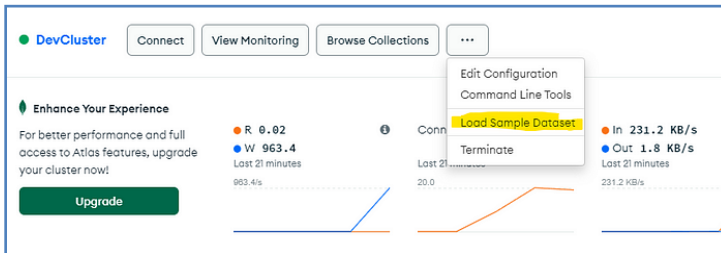
**Fig: Inserted document using Mongo Compass**

You can also check this from the cluster manager, click on Browser collections:



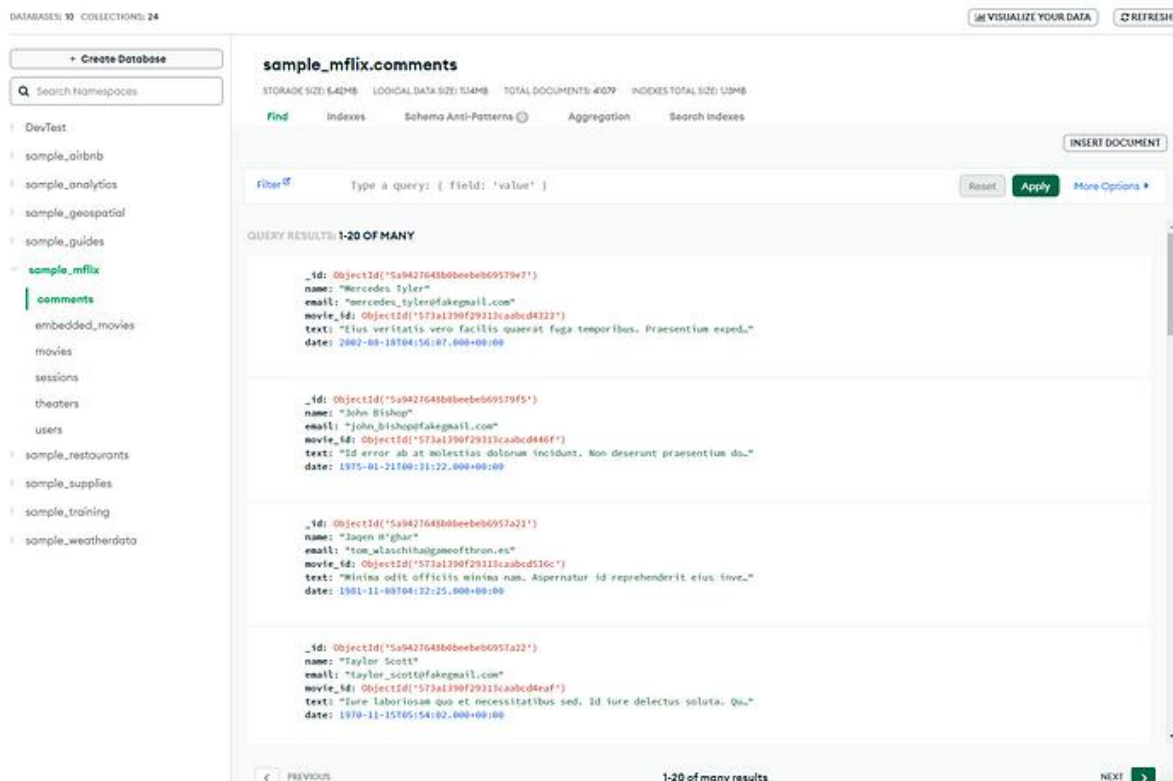
**Fig: Querying from Mongo Atlas cluster manager**

If you are looking for more real data just to practice queries, you can import a sample dataset.



**Fig: Load sample dataset**

Then go to your collections and now you will see more collections and data.



## GUI tools Overview in mongodb

### What is a MongoDB GUI Client?

While MongoDB provides a command-line interface for accessing and querying data, it can be overwhelming for less technical users and database administrators. That's where MongoDB GUI (Graphical User Interface) tools come in.

**A MongoDB GUI client is a graphical interface that allows users to interact with MongoDB databases using a visual representation of the data.** It provides a user-friendly way to view, query, and manipulate data, making it easier to work with MongoDB for developers, administrators, and non-technical users alike. These tools eliminate the need for writing complex command-line queries and provide a more intuitive and efficient way to work with MongoDB.

### MongoDB GUI Features:

Feature	Description
Intuitive Interface	User-friendly interface for effortless database management.
Query Builder	Construct complex queries without delving into MQL syntax.
Schema Visualization	Visual representation of data schema for clarity.
Data Manipulation	Easily insert, update, and delete documents.
Index Management	Simplified index creation and management for improved performance.
Real-time Monitoring	Monitor database operations in real-time for performance insights.
Import/Export Functionality	Support for data import/export in multiple formats for seamless data transfer.

### GUI Tools:

**MongoDB Compass:** A visual tool that provides a graphical interface for working with MongoDB databases. Compass includes features such as schema analysis, query building, and data visualization, making it easy to explore and interact with your data.

#### Pros:

- Official MongoDB GUI Tool
- User-Friendly Interface
- Seamless Integration with MongoDB

#### Cons:

- Some Advanced Features Require a Paid License
- Limited Customization Options Compared to Other GUI Tools

**Studio 3T:** A powerful IDE for MongoDB that provides a range of tools and features for developing, debugging, and managing MongoDB databases. Studio 3T includes features such as a SQL import/export tool, a visual query builder, index management and a schema explorer.

#### Pros:

- Comprehensive Feature Set
- Intuitive User Interface
- Excellent Customer Support
- SQL Migration Feature for Data Import

#### Cons:

- Limited Functionality in Free Version
- Potential Pricing Constraints for Some Users

**Robo 3T:** A lightweight GUI tool for MongoDB that provides a simple and easy-to-use interface for managing MongoDB databases. Robo 3T includes features such as a JSON editor, a visual query builder, and a schema explorer.



**Pros:**

- Open-Source and Free
- Simple and Intuitive Interface
- Suitable for Basic MongoDB Management Tasks

**Cons:**

- Limited Advanced Features Compared to Other GUI Tools
- May Not Be Ideal for Handling Complex Database Management Tasks

**NoSQLBooster for MongoDB:** A full-featured GUI tool for MongoDB that provides a range of tools and features for managing and monitoring MongoDB databases. NoSQLBooster includes features such as a visual Query builder, a schema explorer, data import/ export, Intellisense embedded JavaScript engine, and Real time data visualization.

**Pros:**

- Rich Feature Set
- User-Friendly Interface
- Outstanding Query Autocomplete and Suggestion

**Cons:**

- Some Advanced Features Require a Paid License
- Steeper Learning Curve Compared to Other GUI Tools

**TablePlus:**

TablePlus serves as an on-premise database management client, extending support to various database types, including MongoDB features like secure connection, quick interface and data import/ export etc.

**Pros:**

- Cross-Platform Support
- User-Friendly Interface
- Customizable UI for Personalized Experience

**Cons:**

- May Lack Advanced Features Compared to Specialized MongoDB GUI Tools
- May Require Additional Setup for MongoDB-Specific Functionalities

**Other tools:**

Humongous.io, NoSQL Manager and ToolJet.

**Install and Configure MongoDB Compass**

**MongoDB**, the most popular NoSQL database, is an open-source document-oriented database. The term 'NoSQL' means 'non-relational'. It means that MongoDB isn't based on the table-like relational database structure but provides an altogether different mechanism for storage and retrieval of data. This format of storage is called BSON (similar to JSON format).

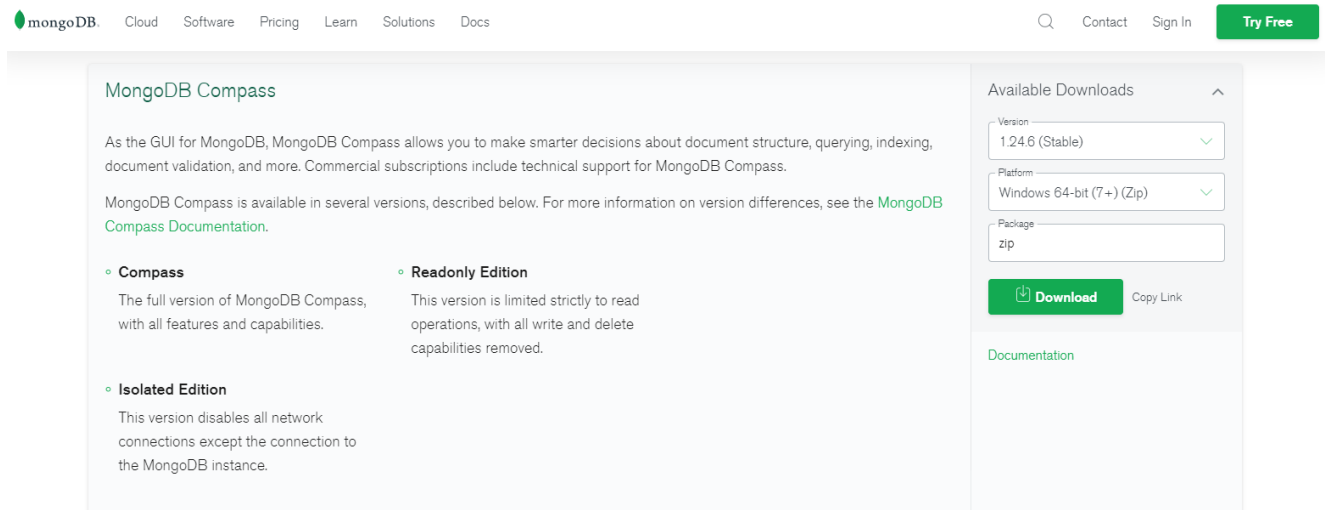
**MongoDB Compass** is a graphical interface to interact with the MongoDB database management system. It comes in handy as it does not require prior knowledge of MongoDB query syntax.

**Installation on Windows:*****Software Requirements:***

- 64-bit version of Microsoft Windows 7 or higher.
- MongoDB 3.6 or higher.
- Microsoft .NET Framework version 4.5 or higher.

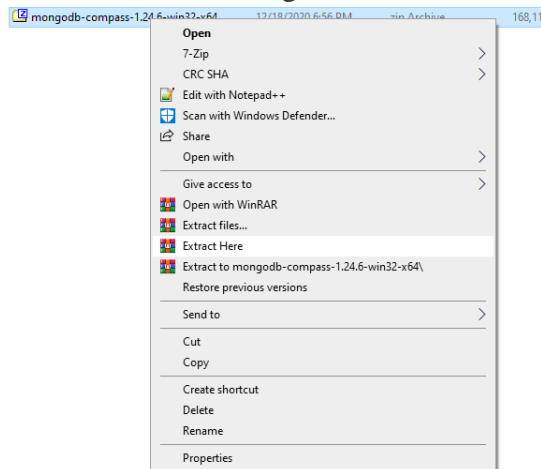
**Step 1:** Firstly go [MongoDb website](https://www.mongodb.com/compass) and download MongoDB Compass.



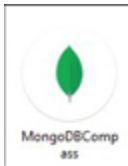


The screenshot shows the MongoDB Compass download page. On the left, there's a description of MongoDB Compass as the GUI for MongoDB, highlighting its features like document structure, querying, indexing, and validation. It lists three editions: Compass (full version), Readonly Edition (limited to read operations), and Isolated Edition (disables network connections). On the right, the 'Available Downloads' section shows the selected version (1.24.6 Stable), platform (Windows 64-bit), and package type (zip). A green 'Download' button and a 'Copy Link' option are visible. A 'Documentation' link is also present at the bottom of the download section.

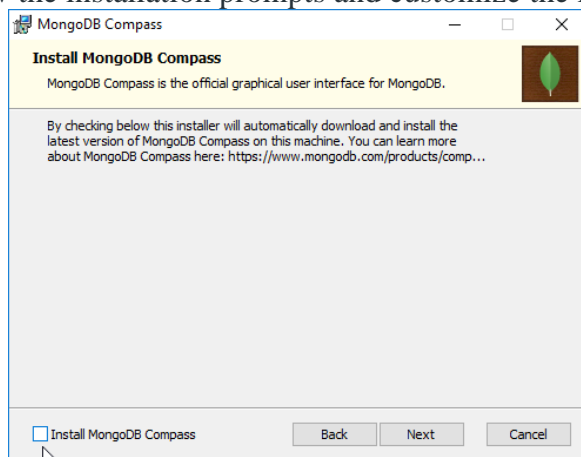
**Step 2: Unzip File after downloading.**



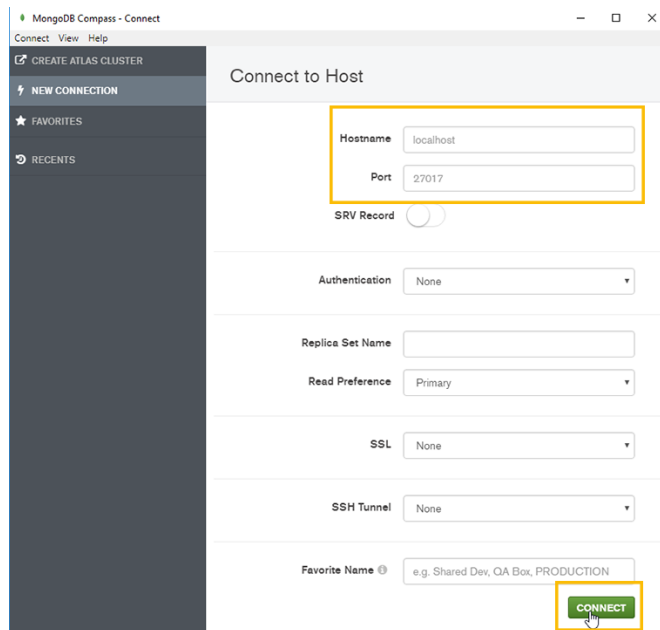
**Step 3: Double click the installer icon.**



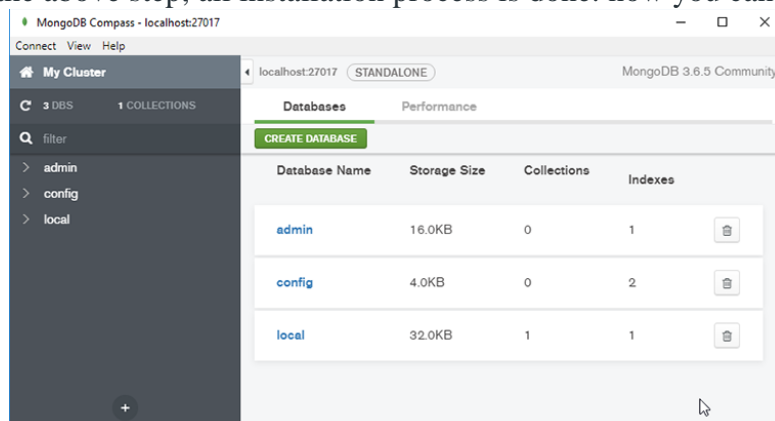
**Step 4: Follow the installation prompts and customize the installation according to your need.**



**Step 5:** At this stage, a prompt will pop which can be used to configure the setting of the MongoDB Compass.



**Step 6:** After the above step, all installation process is done. now you can work with your databases.



After installation, you will find the MongoDB compass icon on your desktop.

### Introduction to the MongoDB Shell

The MongoDB shell is a command-line interface that allows you to interact with MongoDB databases and perform a wide range of operations. With the MongoDB shell, you can:

- Connect to a MongoDB database server and authenticate with it
- Create and manage databases and collections
- Insert, update, and delete documents in a collection
- Query and aggregate data using MongoDB's powerful query language
- Create and manage indexes to optimize your queries
- Execute JavaScript functions and scripts
- Perform administrative tasks such as backup and restore

The MongoDB shell is built on top of the JavaScript interpreter, which means that you can use JavaScript syntax to interact with MongoDB databases. You can run the MongoDB shell by starting a command-line

interface and typing "mongo". Once you're connected to a MongoDB database, you can start executing commands and interacting with your data. The MongoDB shell is a powerful tool for managing and interacting with MongoDB databases, and is often used by developers and administrators who want to perform advanced operations and automate tasks.

### MongoDB shell Commands:

Below is the list of frequently used MongoDB Shell commands with their actions:

- **mongo:** This command returns information related to the MongoDB shell.

```
PS C:\Users\Dark_knight> mongo
MongoDB shell version v4.2.8
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("6c1d5cb1-a0f9-4872-8589-91ff38697cdc") }
MongoDB server version: 4.2.8
Server has startup warnings:
2020-12-10T06:48:27.951+0530 I CONTROL [initandlisten]
2020-12-10T06:48:27.951+0530 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2020-12-10T06:48:27.951+0530 I CONTROL [initandlisten] **      Read and write access to data and configuration is
unrestricted.
2020-12-10T06:48:27.953+0530 I CONTROL [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>
```

- **show dbs:** This command returns all available databases in the current MongoDB database.

```
> show dbs
UserQuery      0.000GB
admin           0.000GB
auth            0.000GB
config          0.000GB
database        0.000GB
email           0.000GB
feedback        0.000GB
fulldb          0.000GB
hostal          0.000GB
local           0.000GB
osms            0.000GB
pms             0.000GB
session         0.000GB
userProfiler    0.000GB
>
```

- **use<DATABASE\_NAME>:** This command is used to select a specific database. If the mentioned database doesn't exist, mongoDB will create a new database.

```
> use feedback
switched to db feedback
>
```

- **show collections:** This command returns all collections in the selected database.

```
> show collections
feeds
> db.feeds.find()
{ "_id" : ObjectId("5fd5175130d4b32890859ac1"), "name" : "albart", "email" : "albart@gmail.com", "feed" : "my feedback",
  "__v" : 0 }
{ "_id" : ObjectId("5fd583714eba820ed4531c58"), "name" : "alert", "email" : "aldsjf@gmail.com", "feed" : "new", "__v" :
  0 }
{ "_id" : ObjectId("5fd583e8485b7d26b8929020"), "name" : "s.p Singh", "email" : "spsinghjio@gmail.com", "feed" : "new",
  "__v" : 0 }
{ "_id" : ObjectId("5fd5b5f736451772804d12a73"), "name" : "gfg", "email" : "gfg@testmail.com", "feed" : "test", "__v" : 0
  }
>
```

- **db.<collection\_name>.find():** This command returns the documents in the collection.

```
> db.feeds.find()
{ "_id" : ObjectId("5fd5175130d4b32890859ac1"), "name" : "albart", "email" : "albart@gmail.com", "feed" : "my feedback",
  "__v" : 0 }
{ "_id" : ObjectId("5fd583714eba820ed4531c58"), "name" : "alert", "email" : "aldsjf@gmail.com", "feed" : "new", "__v" :
  0 }
{ "_id" : ObjectId("5fd583e8485b7d26b8929020"), "name" : "s.p Singh", "email" : "spsinghjio@gmail.com", "feed" : "new",
  "__v" : 0 }
{ "_id" : ObjectId("5fd5bf736451772804d12a73"), "name" : "gfg", "email" : "gfg@testmail.com", "feed" : "test", "__v" : 0
}
>
```

### MongoDB Shell JavaScript Engine

The MongoDB shell is built on top of the JavaScript engine, which allows you to execute JavaScript code and interact with MongoDB databases. The JavaScript engine used by MongoDB is called V8, which is the same engine used by the Google Chrome browser. V8 is a high-performance JavaScript engine that is optimized for running JavaScript code quickly and efficiently.

When you start the MongoDB shell, you're essentially starting a JavaScript interpreter that is preloaded with the MongoDB API. This means that you can use JavaScript syntax to interact with MongoDB databases and perform a wide range of operations, including creating and managing databases and collections, inserting and updating documents, and querying and aggregating data.

You can also use the MongoDB shell to execute JavaScript functions and scripts, making it a powerful tool for automating tasks and performing advanced operations.

#### Write Scripts:

You can write scripts for the MongoDB Shell that modify data in MongoDB or perform administrative operations. You may also want to package your scripts as snippets for easier distribution and management.

This tutorial introduces using the MongoDB Shell with JavaScript to access MongoDB.

#### Compatibility:

You can write scripts for the MongoDB Shell for deployments hosted in the following environments:

**MongoDB Atlas:** The fully managed service for MongoDB deployments in the cloud

**MongoDB Enterprise:** The subscription-based, self-managed version of MongoDB

**MongoDB Community:** The source-available, free-to-use, and self-managed version of MongoDB.

#### Execute a JavaScript File:

Execute a Script from Within mongosh.

You can execute a .js file from within the MongoDB Shell using the load() method.

#### File Paths:

The load() method accepts relative and absolute paths. If the current working directory of the MongoDB Shell is /data/db, and connect-and-insert.js is in the /data/db/scripts directory, then the following calls within the MongoDB Shell are equivalent:

```
load( "scripts/connect-and-insert.js" )

load( "/data/db/scripts/connect-and-insert.js" )
```

#### Example:

The following example creates and executes a script that:

- Connects to a local instance running on the default port.
- Connects to the myDatabase database.

- Populates the movies collection with sample documents.

Create a file named **connect-and-insert.js** with the following contents:

```
db = connect( 'mongodb://localhost/myDatabase' );
db.movies.insertMany( [
  {
    title: 'Titanic',
    year: 1997,
    genres: [ 'Drama', 'Romance' ]
  },
  {
    title: 'Spirited Away',
    year: 2001,
    genres: [ 'Animation', 'Adventure', 'Family' ]
  },
  {
    title: 'Casablanca',
    genres: [ 'Drama', 'Romance', 'War' ]
  }
] )
```

To load and execute the connect-and-insert.js file, use mongosh to connect to your deployment and run the following command:

```
load( "connect-and-insert.js" )
```

To confirm that the documents loaded correctly, use the myDatabase collection and query the movies collection.

```
use myDatabase
db.movies.find()
```

### **MongoDB Shell JavaScript Syntax**

The MongoDB shell uses JavaScript syntax to interact with MongoDB databases. Some of the most commonly used JavaScript syntax in the MongoDB shell include:

**Variables:** You can use the var keyword to declare and initialize variables in the MongoDB shell, just like you would in regular JavaScript code.

**Functions:** You can define and call JavaScript functions in the MongoDB shell, allowing you to encapsulate and reuse logic.

**Objects:** MongoDB documents are represented as JavaScript objects in the MongoDB shell, and you can use object syntax to create and manipulate documents.

**Arrays:** MongoDB arrays are also represented as JavaScript arrays in the MongoDB shell, and you can use array syntax to create and manipulate arrays.

**Control flow statements:** You can use control flow statements such as if-else statements and loops in the MongoDB shell to control the flow of your code.

**MongoDB API:** The MongoDB shell also provides a rich API for interacting with MongoDB databases, including functions for creating and managing databases and collections, inserting and querying data, and performing administrative tasks.

**Let us take a simple mathematical program:**

1. `>x= 100`
2. `100`
3. `>x/ 5;`
4. `20`

**You can also use the JavaScript libraries**

1. `> "Hello, World!".replace("World", "MongoDB");`  
Hello, MongoDB!

**You can even define and call JavaScript functions**

1. `> function factorial (n) {`  
    `if (n <= 1) return 1;`  
    `return n * factorial(n - 1);`  
    `}`
2. `> factorial (5);`

Output: 120

- When you press "Enter", the shell detect whether the JavaScript statement is complete or not.
- If the statement is not completed, the shell allows you to continue writing it on the next line. If you press "Enter" three times in a row, it will cancel the half-formed command and get you back to the `>` - prompt.

**Introduction to the MongoDB Data Types**

Data Types	Description
String	String is the most commonly used datatype. It is used to store data. A string is a sequence of UTF-8 characters.
Integer	Integer is used to store the numeric value. It can be 32 bit or 64 bit depending on the server you are using.
Boolean	This datatype is used to store boolean values. It just shows True/False values.
Double	Double datatype stores floating point values.
Min/Max Keys	This datatype compare a value against the lowest and highest bson elements.
Arrays	This datatype is used to store a list or multiple values into a single key.
Object	Object datatype is used for embedded documents.

Null	It is used to store null values.
Symbol	It is generally used for languages that use a specific type.
Date	This datatype stores the current date or time. It makes you possible to specify your own date time by creating object of date and pass the value of date, month, year into it.

### Introduction to the CRUD Operations on documents in mongodb

#### MongoDB insert documents:

In MongoDB, the **db.collection.insert()** method is used to add or insert new documents into a collection in your database.

#### **Upsert**

There are also two methods "**db.collection.update()**" method and "**db.collection.save()**" method used for the same purpose. These methods add new documents through an operation called upsert.

Upsert is an operation that performs either an update of existing document or an insert of new document if the document to modify does not exist.

#### **Syntax**

```
>db.COLLECTION_NAME.insert(document)
```

**Example:** we insert a document into a collection named myCollection. This operation will automatically create a collection if the collection does not currently exist.

```
1. db.myCollection.insert(
2.   {
3.     course: "java",
4.     details: {
5.       duration: "6 months",
6.       Trainer: "Sonoo jaiswal"
7.     },
8.     Batch: [ { size: "Small", qty: 15 }, { size: "Medium", qty: 25 } ],
9.     category: "Programming language"
10.  }
11. )
```

After the successful insertion of the document, the operation will return a **WriteResult** object with its status.

#### **Output:**

```
WriteResult({ "nInserted" : 1 })
```

Here the **nInserted** field specifies the number of documents inserted. If an error is occurred then the **WriteResult** will specify the error information.

#### Check the inserted documents:

If the insertion is successful, you can view the inserted document by the following query.

```
>db.myCollection.find()
```

You will get the inserted document in return.

#### **Output:**



```
{ "_id" : ObjectId("56482d3e27e53d2dbc93cef8"), "course" : "java", "details" :  
{ "duration" : "6 months", "Trainer" : "Sonoo jaiswal" }, "Batch" :  
[ { "size" : "Small", "qty" : 15 }, { "size" : "Medium", "qty" : 25 } ],  
"category" : "Programming language" }
```

**Note:** Here, the ObjectId value is generated by MongoDB itself.

### **MongoDB insert multiple documents:**

If you want to insert multiple documents in a collection, you have to pass an array of documents to the `db.collection.insert()` method.

Create an array of documents

Define a variable named Allcourses that hold an array of documents to insert.

```
1. var Allcourses =  
2.   [  
3.     {  
4.       Course: "Java",  
5.       details: { Duration: "6 months", Trainer: "Sonoo Jaiswal" },  
6.       Batch: [ { size: "Medium", qty: 25 } ],  
7.       category: "Programming Language"  
8.     },  
9.     {  
10.      Course: ".Net",  
11.      details: { Duration: "6 months", Trainer: "Prashant Verma" },  
12.      Batch: [ { size: "Small", qty: 5 }, { size: "Medium", qty: 10 } ],  
13.      category: "Programming Language"  
14.    },  
15.    {  
16.      Course: "Web Designing",  
17.      details: { Duration: "3 months", Trainer: "Rashmi Desai" },  
18.      Batch: [ { size: "Small", qty: 5 }, { size: "Large", qty: 10 } ],  
19.      category: "Programming Language"  
20.    }  
21.  ];
```

### **Inserts the documents**

Pass this Allcourses array to the `db.collection.insert()` method to perform a bulk insert.

```
> db.myCollection.insert( Allcourses );
```

After the successful insertion of the documents, this will return a BulkWriteResult object with the status.

```
BulkWriteResult({  
  "writeErrors" : [ ],  
  "writeConcernErrors" : [ ],  
  "nInserted" : 3,  
  "nUpserted" : 0,  
  "nMatched" : 0,
```



```
"nModified" : 0,  
"nRemoved" : 0,  
"upserted" : [ ]  
})
```

**Note:** Here the `nInserted` field specifies the number of documents inserted. In the case of any error during the operation, the **BulkWriteResult** will specify that error.

You can check the inserted documents by using the following query:

```
>db.myCollection.find()
```

### Insert multiple documents with Bulk:

In its latest version of MongoDB (MongoDB 2.6) provides a `Bulk()` API that can be used to perform multiple write operations in bulk.

You should follow these steps to insert a group of documents into a MongoDB collection.

Initialize a bulk operation builder

First initialize a bulk operation builder for the collection `myCollection`.

```
var bulk = db.myCollection.initializeUnorderedBulkOp();
```

This operation returns an unordered operations builder which maintains a list of operations to perform .

Add insert operations to the bulk object

```
1. bulk.insert(  
2.   {  
3.     course: "java",  
4.     details: {  
5.       duration: "6 months",  
6.       Trainer: "Sonoo jaiswal"  
7.     },  
8.     Batch: [ { size: "Small", qty: 15 }, { size: "Medium", qty: 25 } ],  
9.     category: "Programming language"  
10.  }  
11.);
```

### Execute the bulk operation

Call the `execute()` method on the bulk object to execute the operations in the list.

```
bulk.execute();
```

After the successful insertion of the documents, this method will return a **BulkWriteResult** object with its status.

```
BulkWriteResult({  
  "writeErrors" : [ ],  
  "writeConcernErrors" : [ ],  
  "nInserted" : 1,  
  "nUpserted" : 0,  
  "nMatched" : 0,  
  "nModified" : 0,  
  "nRemoved" : 0,  
  "upserted" : [ ]  
})
```

Here the `nInserted` field specifies the number of documents inserted. In the case of any error during the operation, the **BulkWriteResult** will specify that error.

### MongoDB update documents:

In MongoDB, `update()` method is used to update or modify the existing documents of a collection.

#### Syntax:

```
db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA, UPDATED_DATA)
```

#### Example

Consider an example which has a collection name `myCollection`. Insert the following documents in collection:

```
1. db.myCollection.insert(  
2.   {  
3.     course: "java",  
4.     details: {  
5.       duration: "6 months",  
6.       Trainer: "Sonoo jaiswal"  
7.     },  
8.     Batch: [ { size: "Small", qty: 15 }, { size: "Medium", qty: 25 } ],  
9.     category: "Programming language"  
10.  }  
11. )
```

After successful insertion, check the documents by following query:

```
>db.myCollection.find()
```

#### Output:

```
{ "_id" : ObjectId("56482d3e27e53d2dbc93cef8"), "course" : "java", "details" :  
  { "duration" : "6 months", "Trainer" : "Sonoo jaiswal" }, "Batch" :  
  [ { "size" : "Small", "qty" : 15 }, { "size" : "Medium", "qty" : 25 } ],  
  "category" : "Programming language" }
```

#### Update the existing course "java" into "android":

```
>db.myCollection.update({'course':'java'},{$set: {'course':'android'}})
```

#### Check the updated document in the collection:

```
>db.myCollection.find()
```

#### Output:

```
{ "_id" : ObjectId("56482d3e27e53d2dbc93cef8"), "course" : "android", "details" :  
  { "duration" : "6 months", "Trainer" : "Sonoo jaiswal" }, "Batch" :  
  [ { "size" : "Small", "qty" : 15 }, { "size" : "Medium", "qty" : 25 } ],  
  "category" : "Programming language" }
```

### MongoDB Delete documents:

In MongoDB, the `db.colloction.remove()` method is used to delete documents from a collection. The `remove()` method works on two parameters.

**1. Deletion criteria:** With the use of its syntax you can remove the documents from the collection.

**2. JustOne:** It removes only one document when set to true or 1.

#### Syntax:

```
db.collection_name.remove (DELETION_CRITERIA)
```

**Remove all documents:** If you want to remove all documents from a collection, pass an empty query document `{ }` to the `remove()` method. The `remove()` method does not remove the indexes.

Let's take an example to demonstrate the `remove()` method. In this example, we remove all documents from the "myCollection" collection.

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

#### **Remove all documents that match a condition**

If you want to remove a document that match a specific condition, call the `remove()` method with the `<query>` parameter.

The following example will remove all documents from the myCollection collection where the type field is equal to programming language.

```
db.myCollection.remove( { type : "programming language" } )
```

#### **Remove a single document that match a condition**

If you want to remove a single document that match a specific condition, call the `remove()` method with `justOne` parameter set to `true` or `1`.

The following example will remove a single document from the myCollection collection where the type field is equal to programming language.

```
db.myCollection.remove( { type : "programming language" }, 1 )
```

### **MongoDB Query documents**

- In MongoDB, the **db.collection.find()** method is used to retrieve documents from a collection. This method returns a cursor to the retrieved documents.
- The `db.collection.find()` method reads operations in mongoDB shell and retrieves documents containing all their fields.
- Note: You can also restrict the fields to return in the retrieved documents by using some specific queries. For example: you can use the `db.collection.findOne()` method to return a single document. It works same as the `db.collection.find()` method with a limit of 1.

#### **Syntax:**

```
db.COLLECTION_NAME.find({})
```

#### **Select all documents in a collection:**

To retrieve all documents from a collection, put the query document (`{ }`) empty. It will be like this:

```
db.COLLECTION_NAME.find()
```

**For example:** If you have a collection name "canteen" in your database which has some fields like foods, snacks, beverages, price etc. then you should use the following query to select all documents in the collection "canteen".

```
db.canteen.find()
```

### **MongoDB Create Database**

#### **Use Database method:**

- There is no create database command in MongoDB. Actually, MongoDB do not provide any command to create database.
- It may be look like a weird concept, if you are from traditional SQL background where you need to create a database, table and insert values in the table manually.
- Here, in MongoDB you don't need to create a database manually because MongoDB will create it automatically when you save the value into the defined collection at first time.

- You also don't need to mention what you want to create, it will be automatically created at the time you save the value into the defined collection.
- Here one thing is very remarkable that you can create collection manually by "db.createCollection()" but not the database.

**How and when to create database:**

If there is no existing database, the following command is used to create a new database.

**Syntax:**

```
use DATABASE_NAME
```

If the database already exists, it will return the existing database.

Let's take an example to demonstrate how a database is created in MongoDB. In the following example, we are going to create a database "myCollectiondb".

**See this example**

```
>use myCollectiondb  
Switched to db myCollectiondb
```

To **check the currently selected database**, use the command db:

```
>db  
myCollectiondb
```

To **check the database list**, use the command show dbs:

```
>show dbs  
local 0.078GB
```

Here, your created database "myCollectiondb" is not present in the list, **insert at least one document** into it to display database:

```
>db.movie.insert({"name":"myCollection"})  
WriteResult({ "nInserted": 1 })
```

```
>show dbs  
myCollectiondb 0.078GB  
local 0.078GB
```

**MongoDB Drop Database:**

The dropDatabase command is used to drop a database. It also deletes the associated data files. It operates on the current database.

**Syntax:**

```
db.dropDatabase()
```

This syntax will delete the selected database. In the case you have not selected any database, it will delete default "test" database.

To **check the database list**, use the command show dbs:

```
>show dbs  
myCollectiondb 0.078GB  
local 0.078GB
```

If you want to **delete the database "myCollectiondb"**, use the dropDatabase() command as follows:

```
>use myCollectiondb  
switched to the db myCollectiondb  
>db.dropDatabase()  
{ "dropped": "myCollectiondb", "ok": 1 }
```

Now check the list of databases:

```
>show dbs
local 0.078GB
```

### MongoDB Create Collection:

In MongoDB, `db.createCollection(name, options)` is used to create collection. But usually you don't need to create collection. MongoDB creates collection automatically when you insert some documents. It will be explained later. First see how to create collection:

#### Syntax:

```
db.createCollection(name, options)
```

Here,

**Name:** is a string type, specifies the name of the collection to be created.

**Options:** is a document type, specifies the memory size and indexing of the collection. It is an optional parameter.

Following is the list of options that can be used.

Field	Type	Description
Capped	Boolean	(Optional) If it is set to true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. If you specify true, you need to specify size parameter also.
AutoIndexID	Boolean	(Optional) If it is set to true, automatically create index on ID field. Its default value is false.
Size	Number	(Optional) It specifies a maximum size in bytes for a capped collection. If capped is true, then you need to specify this field also.
Max	Number	(Optional) It specifies the maximum number of documents allowed in the capped collection.

Let's take an **example to create collection**. In this example, we are going to create a collection name SSSIT.

```
>use test
switched to db test
>db.createCollection("SSSIT")
{ "ok" : 1 }
```

To **check the created collection**, use the command "show collections".

```
>show collections
SSSIT
```

### How does MongoDB create collection automatically:

MongoDB creates collections automatically when you insert some documents. For example: Insert a document named seomount into a collection named SSSIT. The operation will create the collection if the collection does not currently exist.

```
>db.SSSIT.insert({"name" : "seomount"})
>show collections
SSSIT
```

If you want to see the inserted document, use the `find()` command.

#### Syntax:

```
db.collection_name.find()
```

**MongoDB Drop collection:**

In MongoDB, `db.collection.drop()` method is used to drop a collection from a database. It completely removes a collection from the database and does not leave any indexes associated with the dropped collections.

The `db.collection.drop()` method does not take any argument and produce an error when it is called with an argument. This method removes all the indexes associated with the dropped collection.

**Syntax:**

```
db.COLLECTION_NAME.drop()
```

**MongoDB Drop collection example:**

Let's take an example to drop collection in MongoDB.

First **check the already existing collections** in your database.

```
>use mydb
Switched to db mydb

> show collections
SSSIT
system.indexes
```

**Note:** Here we have a collection named SSSIT in our database.

Now **drop the collection** with the name SSSIT:

```
>db.SSSIT.drop()
True
```

Now **check the collections** in the database:

```
>show collections
System.indexes
```

Now, there are no existing collections in your database.

**Note:** The drop command returns true if it successfully drops a collection. It returns false when there is no existing collection to drop.

**Introduction to MongoDB Queries:**

**MongoDB**, the most popular open-source document-oriented database is a NoSQL type of database. NoSQL database stands for Non-Structured Query Database. MongoDB stores the data in the form of the structure(field:value pair) rather than tabular form. It stores data in BSON (Binary JSON) format just like JSON format.

A simple example of a MongoDB database collection.

```
{
  "_id" : ObjectId("6009585d35cce6b7b8f087f1"),
  "title" : "Math",
  "author" : "Aditya",
  "level" : "basic",
  "length" : 230,
  "example" : 11
}
```

**What is MongoDB Query?**

MongoDB Query is a way to get the data from the MongoDB database. MongoDB queries provide the simplicity in process of fetching data from the database, it's similar to SQL queries in SQL Database

language. While performing a query operation, one can also use criteria or conditions which can be used to retrieve specific data from the database.

MongoDB provides the function names as `db.collection_name.find()` to operate query operation on database. In this post, we discussed this function in many ways using different methods and operators.

Here, we are working with:

**Database:** *geeksforgeeks*

**Collection:** *Article*

**Note:** Here “*pretty()*” query method is using for only better readability of Document Database.(It’s not necessary)

### Field selection:

The `find()` method displays the database collection in Non-Structured form(`{<Key> : <value>}`) including auto-created `<key> ” id ”` by MongoDB and collection data inserted by user or admin.

### Syntax:

`db.collection_name.find()`

### Example:

```
db.article.find()
```

This method is used to display all the documents present in the article collection.

```
> db.article.find()
{ "_id" : ObjectId("6009585d35cce6b7b8f087f1"), "title" : "Math", "author" : "Aditya", "level" : "basic", "length" : 230, "example" : 11 }
{ "_id" : ObjectId("60095b8a3fc110f90873ce29"), "title" : "Array", "author" : "Aditya", "level" : "basic", "length" : 200, "example" : 5 }
{ "_id" : ObjectId("60095b8a3fc110f90873ce2a"), "title" : "Stack", "author" : "Rakesh", "level" : "easy", "length" : 400, "example" : 10 }
{ "_id" : ObjectId("60095b8a3fc110f90873ce2b"), "title" : "Queue", "author" : "Rakesh", "level" : "medium", "length" : 350, "example" : 2 }
{ "_id" : ObjectId("60095b8a3fc110f90873ce2c"), "title" : "Tree", "author" : "devil", "level" : "high", "length" : 1000, "example" : 10 }
{ "_id" : ObjectId("60095b8a3fc110f90873ce2d"), "title" : "Graph", "author" : "Aditya", "level" : "high", "length" : 1500, "example" : 15 }
{ "_id" : ObjectId("60095b8d3fc110f90873ce2e"), "title" : "Segment Tree", "author" : "devil", "level" : "very high", "length" : 500, "example" : 20 }
{ "_id" : ObjectId("60095fd4e5a7731b2a55a922"), "title" : "Trie", "author" : "Rakesh", "length" : 500, "example" : 20, "time" : 50 }
> |
```

### Finding a single document:

In MongoDB, we can find a single document using `findOne()` method, This method returns the first document that matches the given filter query expression.

### Syntax:

`db.collection_name.findOne()`

### Example:

```
db.article.findOne()
```

Here, we are going to display the first document of the article collection.

```
> db.article.findOne()
{
  "_id" : ObjectId("6009585d35cce6b7b8f087f1"),
  "title" : "Math",
  "author" : "Aditya",
  "level" : "basic",
  "length" : 230,
  "example" : 11
}
```

**Displaying documents in a formatted way:**

In MongoDB, we can display documents of the specified collection in well-formatted way using `pretty()` method.

**Syntax:**

```
db.collection_name.find().pretty()
```

**Example:**

```
db.article.find().pretty()
```

Here, we are going to display the documents of the article collection in a well-formatted way using `pretty()` method.

```
> db.article.find().pretty()
{
  "_id" : ObjectId("6009585d35cce6b7b8f087f1"),
  "title" : "Math",
  "author" : "Aditya",
  "level" : "basic",
  "length" : 230,
  "example" : 11
}
{
  "_id" : ObjectId("60095b8a3fc110f90873ce29"),
  "title" : "Array",
  "author" : "Aditya",
  "level" : "basic",
  "length" : 200,
  "example" : 5
}
{
  "_id" : ObjectId("60095b8a3fc110f90873ce2a"),
  "title" : "Stack",
  "author" : "Rakesh",
  "level" : "easy",
  "length" : 400,
  "example" : 10
}
```

**Equal filter query:**

The equality operator(`$eq`) is used to match the documents where the value of the field is equal to the specified value. In other words, the `$eq` operator is used to specify the equality condition.

**Syntax:**

```
db.collection_name.find({< key > : {$eq : < value >}})
```

**Example:**

```
db.article.find({author:{$eq:"devil"}}).pretty()
```

Here, we are going to display the documents that matches the filter query(i.e., `{author : {$eq : "devil"}}`) from the article collection.

```
> db.article.find({author:{$eq:"devil"}}).pretty()
{
  "_id" : ObjectId("60095b8a3fc110f90873ce2c"),
  "title" : "Tree",
  "author" : "devil",
  "level" : "high",
  "length" : 1000,
  "example" : 10
}
{
  "_id" : ObjectId("60095b8d3fc110f90873ce2e"),
  "title" : "Segment Tree",
  "author" : "devil",
  "level" : "very high",
  "length" : 500,
  "example" : 20
}
>
```



**Greater than filter query:**

To get the specific numeric data using conditions like greater than equal or less than equal use the `$gte` or `$lte` operator in the `find()` method.

**Syntax:**

```
db.collection_name.find({< key > : {$gte : < value >}})
```

or

```
db.collection_name.find({< key > : {$lte : < value >}})
```

**Example:**

```
db.article.find({length:{$gte:510}}).pretty()
```

Here, we are querying to get documented data which has the length attribute value greater than 510. So, we pass a filter query that is `{length: {$gte : 510}}` in the `find()` method.

```
> db.article.find({length : {$gte : 510 }}).pretty()
{
  "_id" : ObjectId("60095b8a3fc110f90873ce2c"),
  "title" : "Tree",
  "author" : "devil",
  "level" : "high",
  "length" : 1000,
  "example" : 10
}
{
  "_id" : ObjectId("60095b8a3fc110f90873ce2d"),
  "title" : "Graph",
  "author" : "Aditya",
  "level" : "high",
  "length" : 1500,
  "example" : 15
}
>
```

**Check the existence filter query:**

`$exists` operator shows all the collection documents if they exist on a given key.

**Syntax:**

```
db.collection_name.find({< key > : {$exists : < boolean >}})
```

**Example:**

```
db.article.find({time:{$exists:"true"}}).pretty()
```

Here, we are going to look all the documents which has the attribute named as time by passing a filter query that is `{time: {$exists : "true"}}` in the `find()` method.

```
> db.article.find({time : {$exists : "true" }}).pretty()
{
  "_id" : ObjectId("60095fd4e5a7731b2a55a922"),
  "title" : "Trie",
  "author" : "Rakesh",
  "length" : 500,
  "example" : 20,
  "time" : 50
}
> |
```

**Logical operator query:**

`$and` operator comes under the type of MongoDB logical operator which perform logical AND operation on the array of one or more expressions and select or retrieve only those documents that match all the given expression in the array.

**Syntax :**

```
db.collection_name.find({$and : [{< key > : {$eq : < value1 >}}, {< key > : {$exists : < boolean >}}]})
```

**Example:**

```
db.article.find({$and:[{level:{$eq:"high"}},{level:{$exists : "true"}}]}).pretty()
```

In this query example we are using and operator and given two condition which are highlighted following

- and operator: *{ \$and : [ first condition, second condition ] }*
- first condition (level == "high"): *{ level : { \$eq : "high" } }*
- second condition: *{ level : { \$exists : "true" } }*

```
> db.article.find({$and : [{ level : { $eq : "high" } }, { level : { $exists :
"true" } } ] }).pretty()
{
  "_id" : ObjectId("60095b8a3fc110f90873ce2c"),
  "title" : "Tree",
  "author" : "devil",
  "level" : "high",
  "length" : 1000,
  "example" : 10
}
{
  "_id" : ObjectId("60095b8a3fc110f90873ce2d"),
  "title" : "Graph",
  "author" : "Aditya",
  "level" : "high",
  "length" : 1500,
  "example" : 15
}
>
```

**Project query:**

This query returned the result whatever we specify in parameter using 1 and 0

- 1: It indicated to return the result
- 0: It indicates to not return the result

These parameters called **Projection Parameter**.

**Syntax:**

```
db. collection_name. find({< key > : < value >}, {<key> : < Projection_Parameter >})
```

**Example:**

```
db. article. find({ author : "devil" }, { title : 0 }).pretty()
```

This query example is requesting the data which have the author as named "devil" but in the record don't want to show the title attribute by specifying it as projection parameter 0

- Query to find a record having given attribute: *{ author : "devil" }*
- Projection Parameter: *{ title : 0 }*

```
db. article. find( { author : "devil" }, { title : 0 } ). pretty()
{
  "_id" : ObjectId("60095b8a3fc110f90873ce2c"),
  "author" : "devil",
  "level" : "high",
  "length" : 1000,
  "example" : 10
}
{
  "_id" : ObjectId("60095b8d3fc110f90873ce2e"),
  "author" : "devil",
  "level" : "very high",
  "length" : 500,
  "example" : 20
}
>
```

### Limit Query:

This query method specifies a maximum number of documents for a cursor to return.

#### **Syntax :**

```
db.collection_name.find(< { key } : < value > }).limit(< Integer_value >)
```

#### **Example:**

```
db.article.find({author : "devil" }). limit(2) . pretty()
```

This query method is simply the extension of the find method only provide a result at a maximum limited number(here is 2)

- Query to find record having given attribute: *find({author : "devil" })*
- Query to limit result: *limit( 2)*

#### **Output :**

```
db.article.find( { author : "devil" } ). limit(2) . pretty()
{
  "_id" : ObjectId("60095b8a3fc110f90873ce2c"),
  "title" : "Tree",
  "author" : "devil",
  "level" : "high",
  "length" : 1000,
  "example" : 10
}
{
  "_id" : ObjectId("60095b8d3fc110f90873ce2e"),
  "title" : "Segment Tree",
  "author" : "devil",
  "level" : "very high",
  "length" : 500,
  "example" : 20
}
```

### Sort the fields:

In MongoDB, It returns the finding result in sorted order that can be ascending or descending. The order specifies by given parameter like

- **1 (Positive One):** It indicates the Ascending order of having attribute value in the record.
- **-1 (Negative One):** It indicates the Descending order of having attribute value in the record.

#### **Syntax:**

```
db.collection_name.find(). sort(< { key } : 1})
```

#### **Example:**

```
db.article.find({author : "devil"}).sort({example : 1}).pretty()
```

This Example will show the resultant record which has the example attribute and it will show in ascending order because here we are passing value 1.

- Query to finding record having given attribute: *find ({author : "devil"})*
- Query to sort parameter: *sort({example : 1})*

```
db.article.find({ author : "devil" }).sort({ example : 1 }).pretty()
{
  "_id" : ObjectId("60095b8a3fc110f90873ce2c"),
  "title" : "Tree",
  "author" : "devil",
  "level" : "high",
  "length" : 1000,
  "example" : 10
}
{
  "_id" : ObjectId("60095b8d3fc110f90873ce2e"),
  "title" : "Segment Tree",
  "author" : "devil",
  "level" : "very high",
  "length" : 500,
  "example" : 20
}
```