Managing Large Databases

Using MongoDB

Table of Contents

Sessions

Session 1 – Introduction to MongoDB

Session 2 – MongoDB Databases

Session 3 – MongoDB Operators

Session 4 – Aggregation Pipeline

Session 5 – Database Commands

Session 6 – MongoDB Shell Methods

Session 7 – MongoDB Indexing

Session 8 – MongoDB Replication and Sharding

Session 9 – Transaction Management

Session 1o – MongoDB Tools

Session 11 – MongoDB Cloud

Session 12 – MongoDB Database Connectivity With Python

Appendix – Case Studies

SESSION 1
INTRODUCTION TO MONGODB

Learning Objectives

In this session, students will learn to:

- Explain various types of databases
- Describe how data is stored in the MongoDB Server
- Explain the installation of MongoDB Server and MongoDB Shell

Databases have been in use for quite some time now. Most of the companies use structured databases, such as SQL Server and Oracle, to store information related to their products, customers, sales, employees, and so on. These databases grow continuously with more and more new information. Companies must invest in the required hardware to store the ever-growing data. As the databases explode, some of the existing database systems become very expensive to upgrade and maintain. In addition, the performance of these databases is affected due to large volume of data.

To overcome the issues that structured databases posed with increasing data, companies are now opting for NoSQL databases. Especially companies that host social media apps or gaming apps, which encounter an ever-increasing customer or player base. MongoDB is one of the popular NoSQL databases.

This session will provide an overview of the types of databases and the prevalence of NoSQL databases. This session also explains the architecture of MongoDB and illustrates the steps to install the MongoDB server and shell.

1.1 Databases

Data refers to any information and a database is a collection of data. An example of a database is a telephone directory or a product catalog. Depending on the way the data is stored, there are different types of databases. These are:

Hierarchical	Similar to a family tree, this type of database stores data in the form of parent-child nodes. **Example:** Windows Registry
Relational	This type of database stores data in the form of rows and columns that together form a table (relation). **Example:** MySQL, Oracle
Object-oriented	This type of database stores data in the form of objects similar to the objects used in an object-oriented programming language. **Example:** PostgreSQL
Network	This database stores data as multiple children and parent nodes that results in complex database structures. **Example:** RDM Server
NoSQL	This database does not use a relational model and is used for analyzing large sets of distributed data. **Example:** MongoDB

Before getting into the details of MongoDB, let us understand the difference between relational databases, which are quite popular, and NoSQL databases, which are becoming more and more popular.

Table 1.1 distinguishes the features of relational and NoSQL databases.

Attribute	Relational Database	NoSQL Database
Data storage	It stores data within rows and columns in tables.	It stores data in various formats, such as documents, graphs, or key-value pairs.
Schema	It requires every table in the database to have a defined schema and the schema cannot be altered after it is created.	It does not require data to have a defined schema. It supports dynamic schema, which allows alterations to the storage structure as required.
Scaling	To support increase in data volume and database users, the hardware of the server must be updated with RAM or processors, or a completely new server machine must be installed. Adding more resources to a server is referred to as vertical scaling.	To support increase in data volume and database users, multiple computers or servers with minimal configuration can be installed. Adding more servers that contain same set of databases with different sets of information is referred to as horizontal scaling.
Performance	The information related to a single entity is stored in multiple tables, based on normalization. Therefore, if data is to be retrieved for a single entity, multiple tables must be iterated through. This results in a decrease in performance, especially if the volume of data is large.	The information related to a single entity is stored together in a single location. Therefore, if data is to be retrieved for a single entity, a simple query can be used. This improves the performance of the database.
Consistency	Relational DataBase Management System (RDBMS) follows the Atomicity, Consistency, Isolation, and Durability (ACID) properties.	NoSQL Database does not follow the ACID properties.

Attribute	Relational Database	NoSQL Database
Application	RDBMS is a best fit when data items must be related and stored in a rules-based, consistent manner. Examples include e-commerce transactions and stock tracking.	NoSQL database is a best fit for applications that require huge data volumes and low latency. Examples include online gaming, social media, and shopping apps.

Table 1.1: Relational Databases vs. NoSQL Databases

1.2 Overview of MongoDB

Consider a social media mobile app where a movie star with a million followers has posted his latest movie stills. Several thousands of his followers are commenting on the movie stills and the movie star is also responding to these comments. Comments and responses are a mix of text messages, animations, pictures, and emojis. If this social media app is placed on RDBMS, there would be one table each to store:

- the details of the movie star,
- the details of the followers,
- the posts put up by the movie star, and
- the comments and responses.

All these tables will be related to each other based on one or more columns. In this case, the time taken for reading/writing such large, changing, and varied data to the server will be really long. Consequently, the user experience of the mobile app would not be very good. However, a NoSQL database would perfectly fit in such a scenario because all the data can be stored together. This data storage will speed up data retrieval leading to good user experience. One example of a NoSQL database is MongoDB.

Depending on how data is stored in the database, there are four types of NoSQL databases:

- Document databases: Store data as JSON, BSON, or XML documents
- Key-value databases: Store data with a label or attribute and the value
- Wide-column databases: Store data in columns

- Graph databases: Store each data element a node and relationship between the data elements as links or relationships

MongoDB is a NoSQL database program that stores information in formats similar to JavaScript Object Notation (JSON) called the Binary JavaScript Object Notation (BSON). MongoDB is an open-source software which works with large sets of distributed data. It provides high performance, high availability, and automatic scaling.

The JSON format is a standard human-readable format used to transmit data in the form of key-value pairs. BSON is used to store and access documents transmitted using JSON format. MongoDB stores documents in BSON format.

1.2.1 MongoDB Architecture

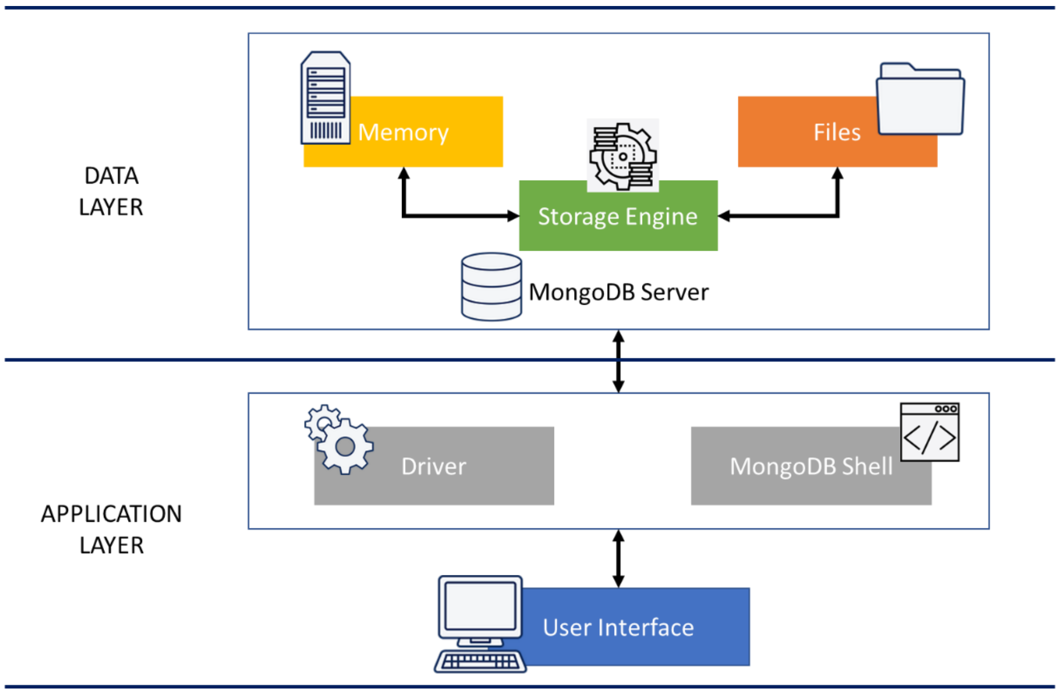
MongoDB operates in two layers: the application layer and the data layer.

The application layer is made up of:

- User interface that the user uses to interact with the database by using a mobile app (Android or iOS) or directly from the Web,
- MongoDB driver that helps to make the connection between the user interface and the MongoDB server, and
- MongoDB Shell which is the command-line interface that users can use to interact with the MongoDB server.

The data layer consists of the MongoDB server that receives the queries from the application layer and sends them on to the storage engine. The storage engine then reads or writes the data to the memory or the file. The storage engine determines how data is stored and the amount of data stored on the disk (file) and memory.

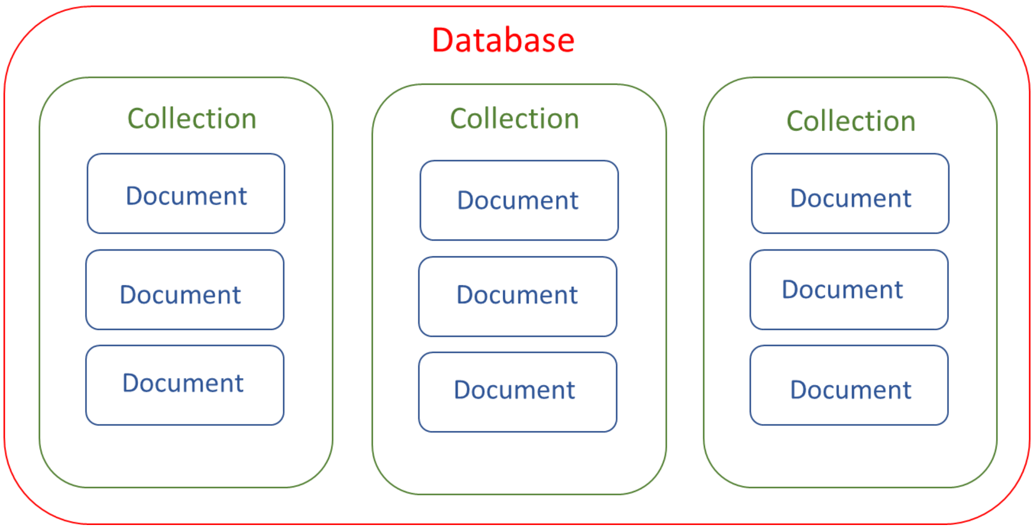
Thus, the vital components of MongoDB that are necessary for its function are:



- MongoDB supports drivers for C, C++, .Net, Go, C#, Python, Node.js, and so on.
- WiredTiger is the default MongoDB storage engine. However, MongoDB also uses other storage engines, such as the in-memory storage engine, MMAPv1 storage engine, and the encrypted storage engine.

1.2.2 Database, Collection, and Document

Data is stored in documents (rows and columns in RDBMS) on the MongoDB Server. Multiple documents together make up a collection (tables in RDBMS). Multiple collections make up a database.



MongoDB stores data in the document as BSON documents. BSON is a binary representation of JSON documents, although BSON supports more data types than JSON. The BSON format will look as follows:

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

A sample document with values in the fields will look as follows:

```
{
  _id: ObjectId("5099803df3f4948bd2f98391"),
  name: { first: "Larissa", last: "Smith" },
  birth: new Date('Jun 23, 2015'),
  grade: 6,
  marks: [{sub: “English”,
           score: 89},
          {sub: “Math”,
           score: 75},
          {sub: “Science”,
           score: 92},
          {sub: “Social Studies”
           score: 70},
}
```

The sample document shows key-value pairs which form the basic unit of data in MongoDB.

1.2.3 MongoDB Editions

MongoDB is available both as a server and as a service on the cloud. As a server, MongoDB is available as Community Server and Enterprise Server. While Community Server is a free edition, Enterprise Server is a commercial edition that is available on subscription. While the core server features are the same across the two editions, the Enterprise edition offers additional operational and management capabilities, an in-memory storage engine, and advanced security features.

As a cloud service, MongoDB is available as MongoDB Atlas. It offers Data as a Service (DaaS) hosting data in the cloud and provides all the features of cloud computing including high availability and scalability.

MongoDB Atlas includes products such as Atlas Database, Atlas Search, Atlas Data Federation, Atlas Charts, Atlas App Services, Atlas Device Sync, Atlas Data Lake, and so on.

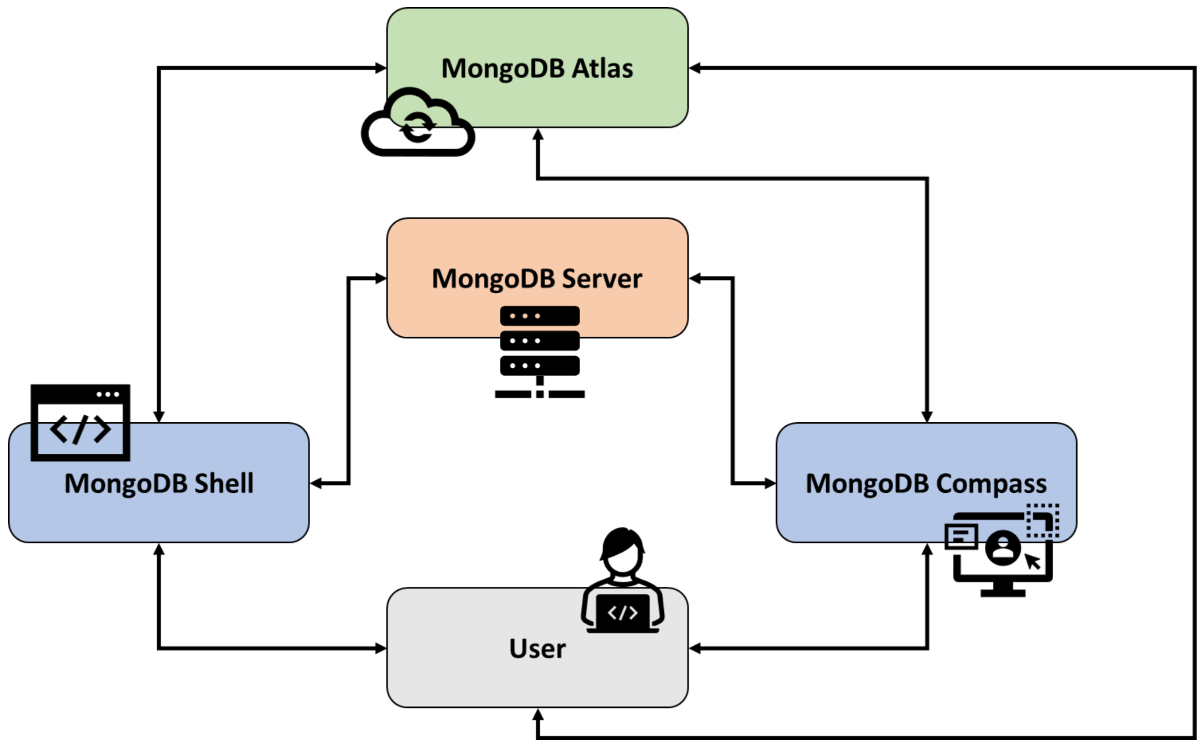
With MongoDB Atlas, users can:

- Save time and cost. MongoDB Atlas includes a fully managed database service that provides access to the database resources as required.
- Use a common API to serve any type of database requests.
- Perform multi-region deployments. Atlas Database is available in more than 80 regions across AWS, Google Cloud, and Azure.
- Focus on building rather than managing. Atlas provides automated infrastructure operations that guarantee availability, scalability, and security compliance.
- Work with data as code. In MongoDB, documents map directly to the objects in the code. As a result, data of any structure can be stored, and the schema can be modified when new features are added to the application.

MongoDB Atlas allows deployment of databases to any cloud provider rather than providing a single option.

The MongoDB Shell is a standalone command-line product that the user can use to connect to the server or the service. MongoDB Compass is Graphical User Interface (GUI) that the users can use to connect to the server or to the service.

Users can also directly access the Atlas platform and work on it without using MongoDB Shell or MongoDB Compass.



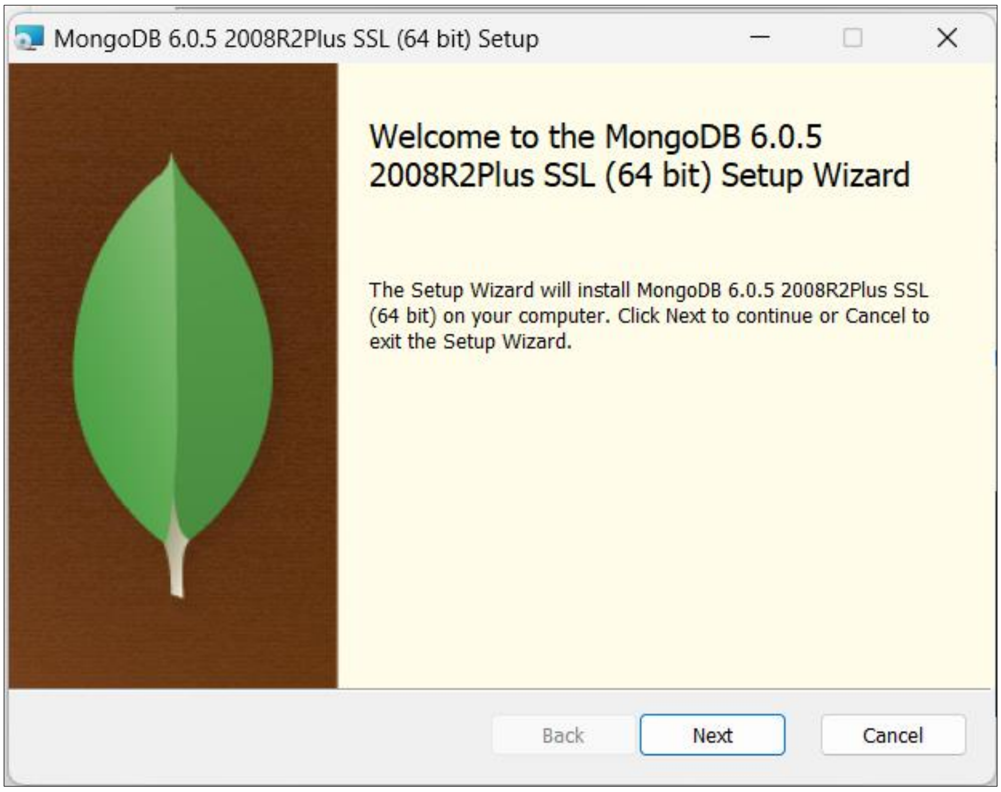
1.3 Installing MongoDB Community Server

This course is based on the MongoDB 6.0 Community edition.

To install the MongoDB 6.0 Community edition on Windows systems:

1. Open a browser and navigate to the URL:
   https://www.mongodb.com/try/download/community
   The MongoDB Community Server Download page opens.
2. On this page, scroll down and ensure that:
   - In the **Version** drop-down, **6.0.5(current)** is selected.
   - In the **Platform** drop-down, **Windows** is selected.
   - In the **Package** drop-down, **msi** is selected.
3. Click **Download**.
4. After the download is complete, run the installer.

   The MongoDB 6.0.5 2008R2Plus SSL (64-bit) Setup wizard opens as shown in Figure 1.1.

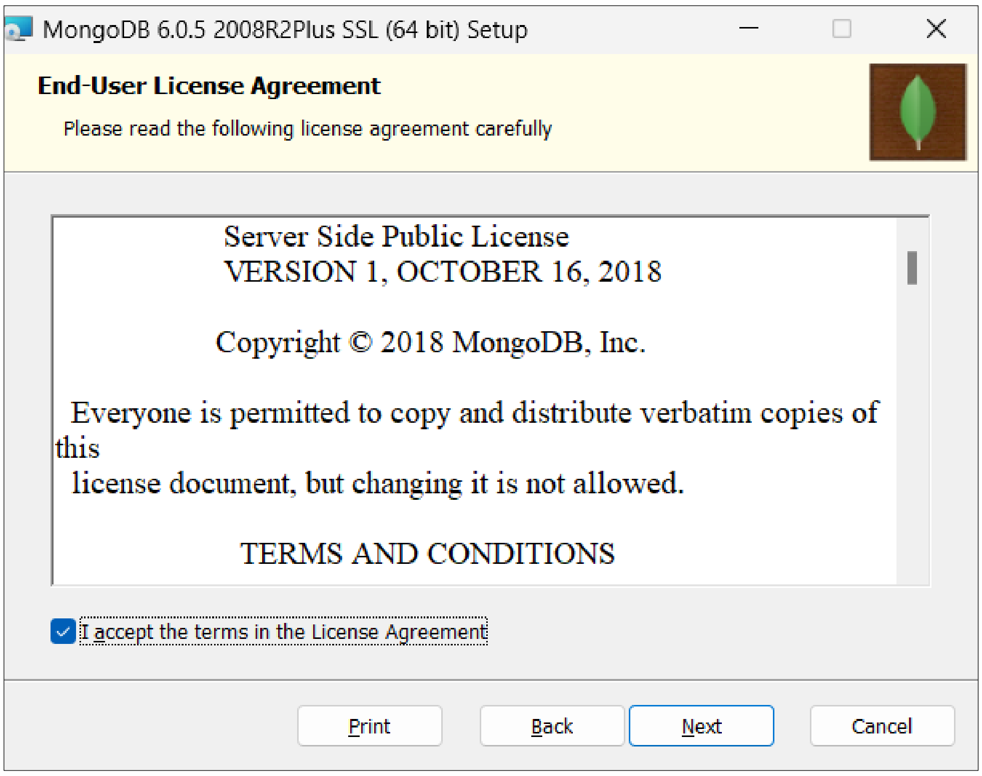

MongoDB 6.0.5 2008R2Plus SSL (64 bit) Setup

Welcome to the MongoDB 6.0.5 2008R2Plus SSL (64 bit) Setup Wizard

The Setup Wizard will install MongoDB 6.0.5 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Figure 1.1: Welcome Page of the MongoDB Installation Wizard

5. Click **Next**.
   The End-User License Agreement page of the wizard opens as shown in Figure 1.2.


MongoDB 6.0.5 2008R2Plus SSL (64 bit) Setup
End-User License Agreement
Please read the following license agreement carefully

Server Side Public License
VERSION 1, OCTOBER 16, 2018

Copyright © 2018 MongoDB, Inc.

Everyone is permitted to copy and distribute verbatim copies of this
license document, but changing it is not allowed.

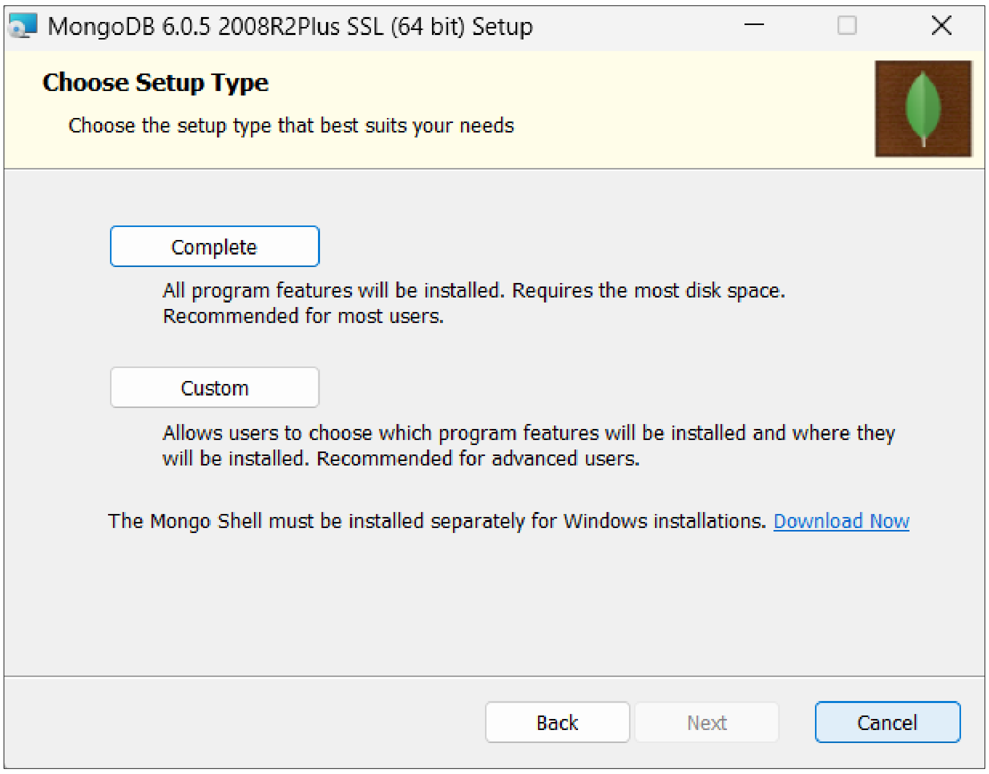
TERMS AND CONDITIONS

[✓] I accept the terms in the License Agreement

Figure 1.2: End-User License Agreement Page

6. To proceed, select the **I accept the terms in the License Agreement** check box.
7. Click **Next**.

   The Choose Setup Type page of the wizard opens as shown in Figure 1.3.


MongoDB 6.0.5 2008R2Plus SSL (64 bit) Setup

Choose Setup Type

Choose the setup type that best suits your needs

Complete

All program features will be installed. Requires the most disk space. Recommended for most users.

Custom

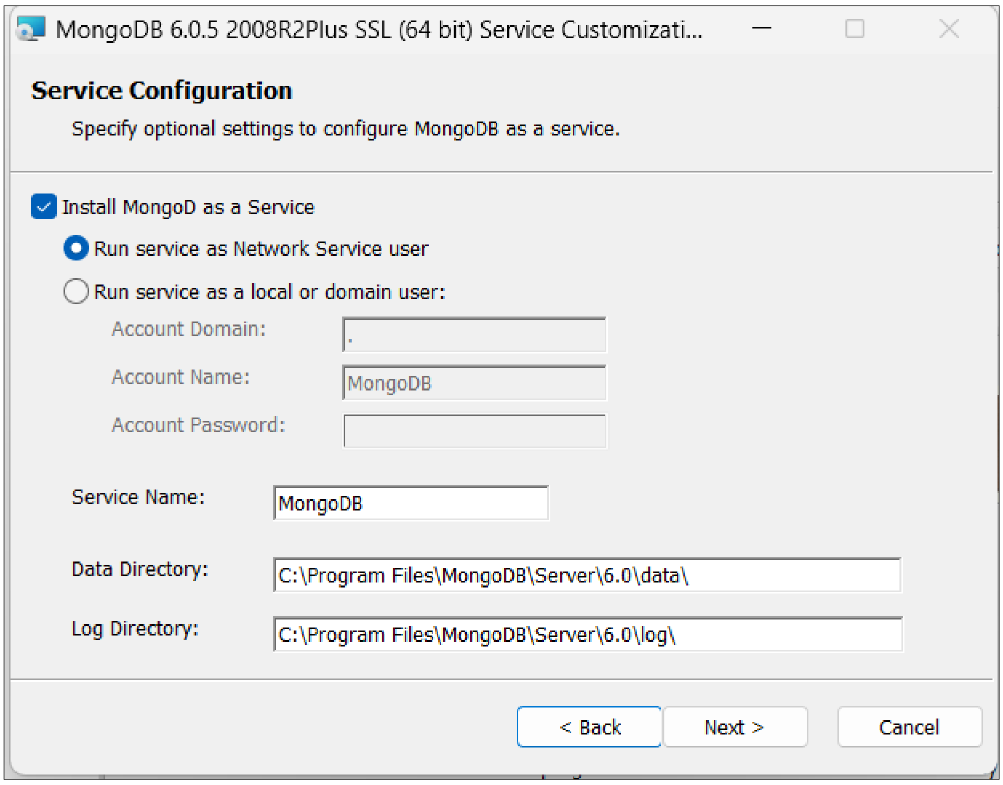
Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

The Mongo Shell must be installed separately for Windows installations. Download Now

Figure1.3: Choose Setup Type Page

8. Click **Complete**.

   The Service Configuration page of the wizard opens as shown in Figure 1.4.


MongoDB 6.0.5 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:

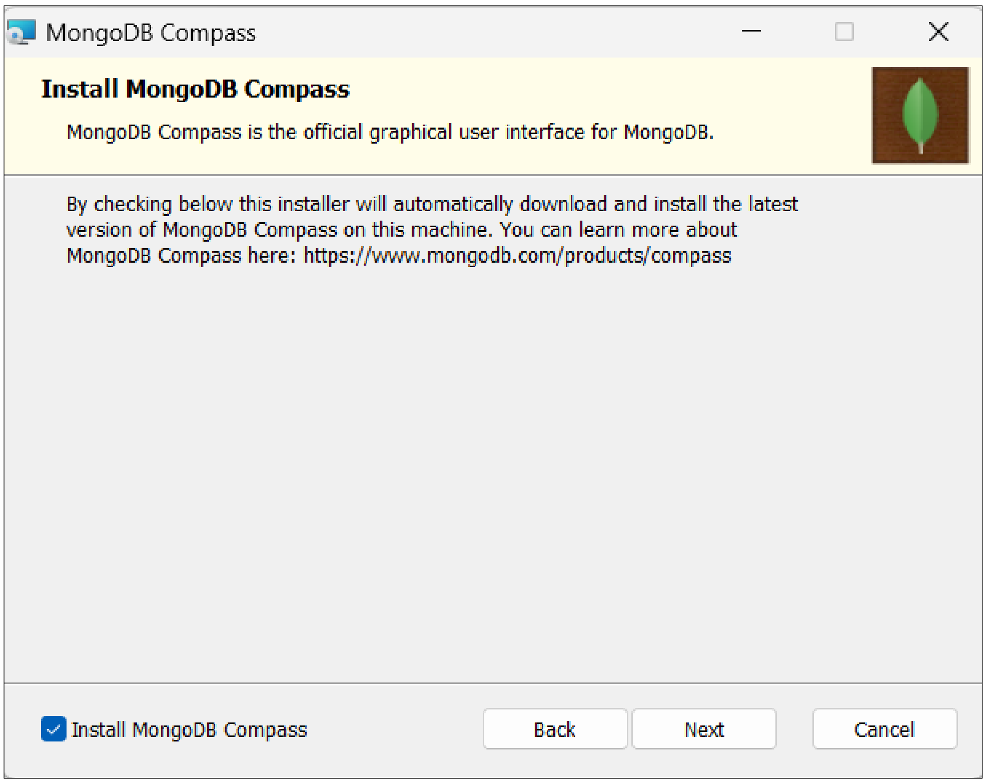
Service Name: MongoDB

Data Directory: C:\Program Files\MongoDB\Server\6.0\data\

Log Directory: C:\Program Files\MongoDB\Server\6.0\log\

Figure 1.4: Service Configuration Page

9. Click **Next**.
   The Install MongoDB Compass page of the wizard opens as shown in Figure 1.5.


MongoDB Compass

Install MongoDB Compass

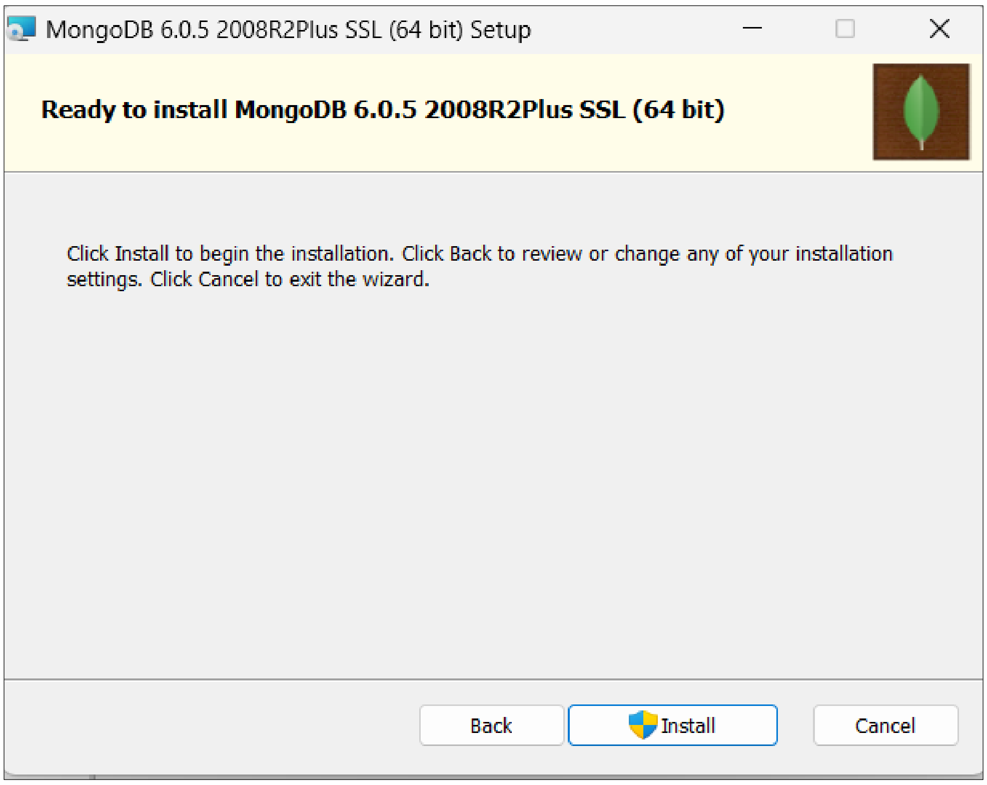
MongoDB Compass is the official graphical user interface for MongoDB.

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

Figure 1.5: Install MongoDB Compass Page

10. Ensure that the **Install MongoDB Compass** check box is selected.
11. Click **Next**.

    The Ready to install MongoDB 6.0.5 2008R2Plus SSL (64-bit) page of the wizard opens as shown in Figure 1.6.



MongoDB 6.0.5 2008R2Plus SSL (64 bit) Setup

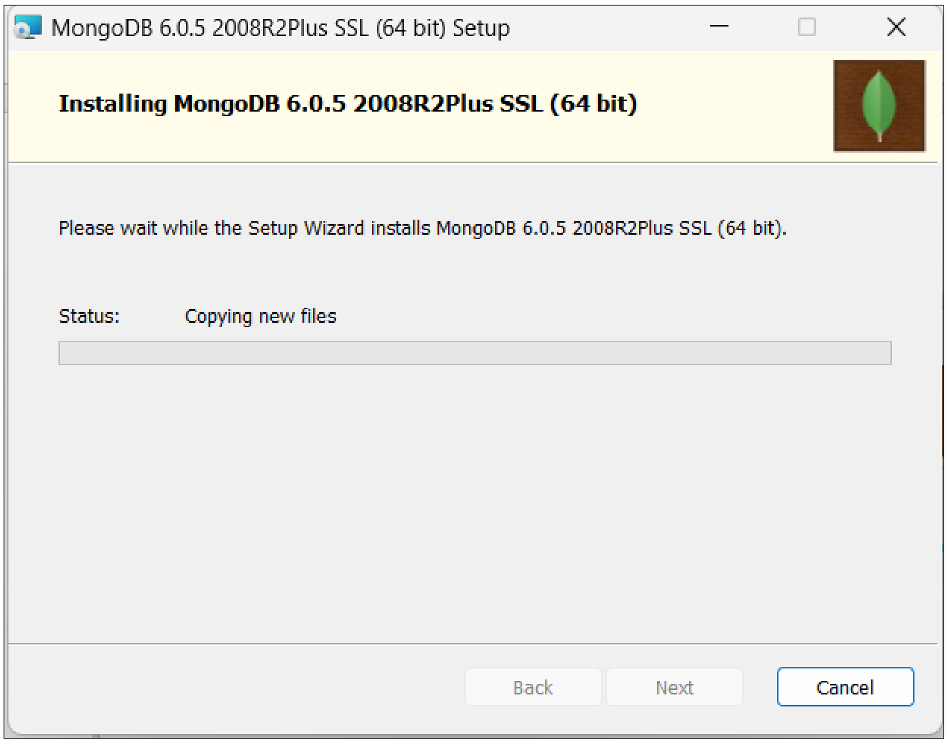
Ready to install MongoDB 6.0.5 2008R2Plus SSL (64 bit)

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Figure 1.6: Screenshot of the Ready to Install Page

12. Click **Install**.

    The Installing MongoDB 6.0.5 2008R2Plus SSL (64-bit) page of the wizard opens as shown in Figure 1.7.


MongoDB 6.0.5 2008R2Plus SSL (64 bit) Setup

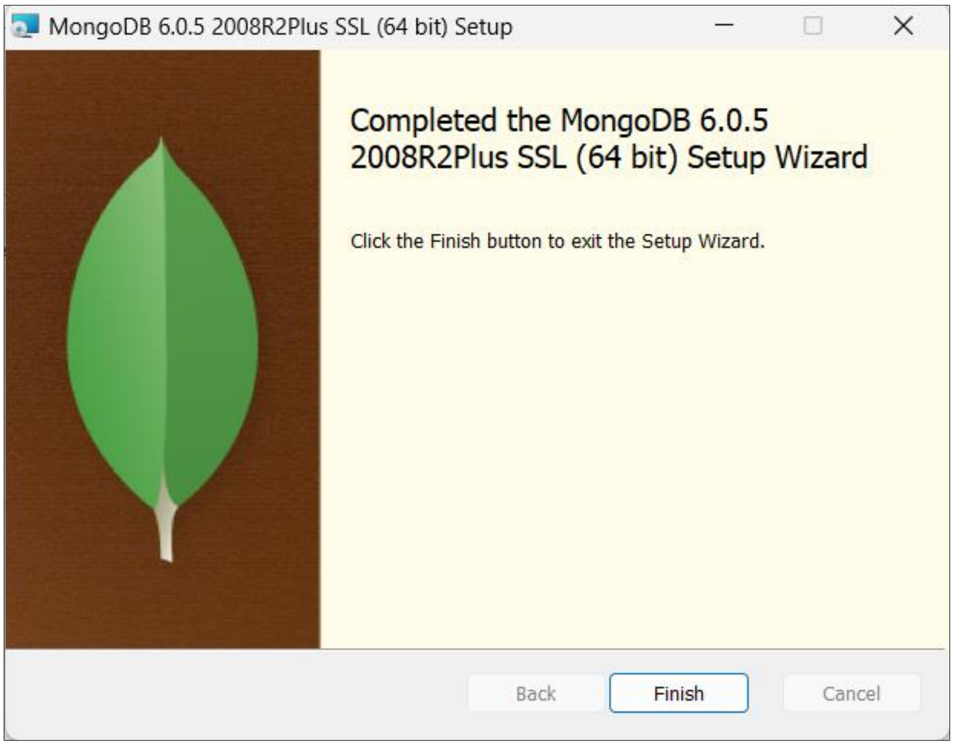
Installing MongoDB 6.0.5 2008R2Plus SSL (64 bit)

Please wait while the Setup Wizard installs MongoDB 6.0.5 2008R2Plus SSL (64 bit).

Status: Copying new files

Figure 1.7: Installation in Progress Page

The Completed the MongoDB 6.0.5 2008R2Plus SSL (64-bit) Setup Wizard page of the wizard opens as shown in Figure 1.8.


MongoDB 6.0.5 2008R2Plus SSL (64 bit) Setup

Completed the MongoDB 6.0.5 2008R2Plus SSL (64 bit) Setup Wizard

Click the Finish button to exit the Setup Wizard.

Figure 1.8: Installation Completion Page

13. Click **Finish**.

MongoDB Compass is also installed along with MongoDB Community Server. The MongoDB Compass window opens by default. Close this window.

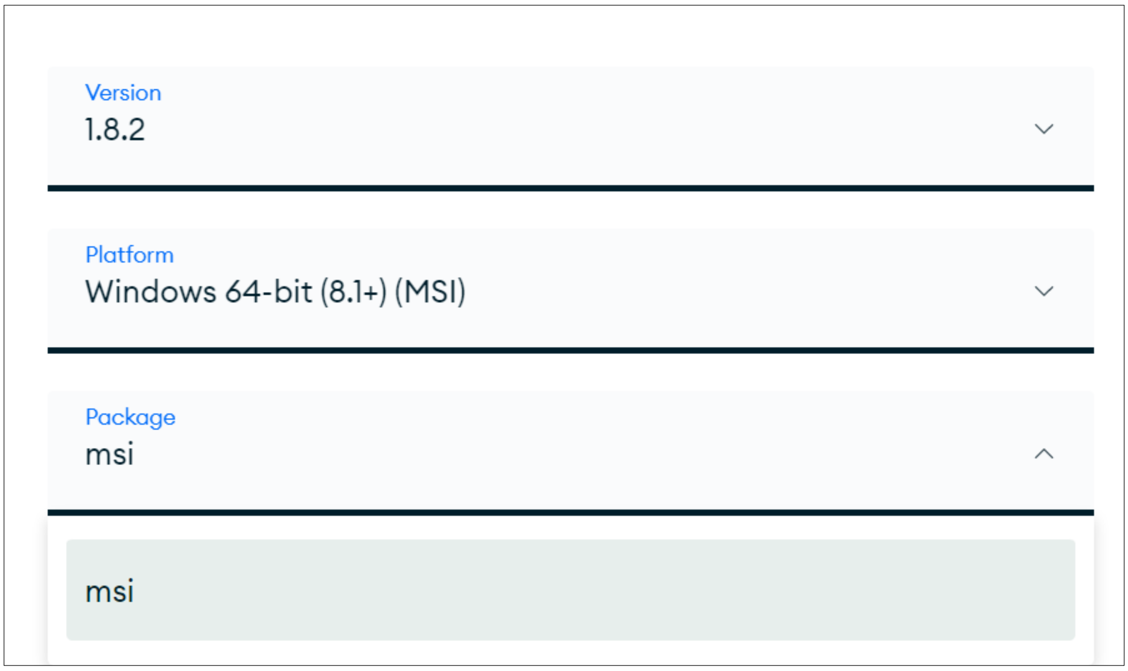
1.4 Installing MongoDB Shell

installing the MongoDB Server, install the MongoDB Shell.

To install the MongoDB Shell:

1. Open a browser and navigate to the URL: https://www.mongodb.com/try/download/shell

The MongoDB Shell Download page opens as shown in Figure 1.9.



Version
1.8.2

Platform
Windows 64-bit (8.1+) (MSI)

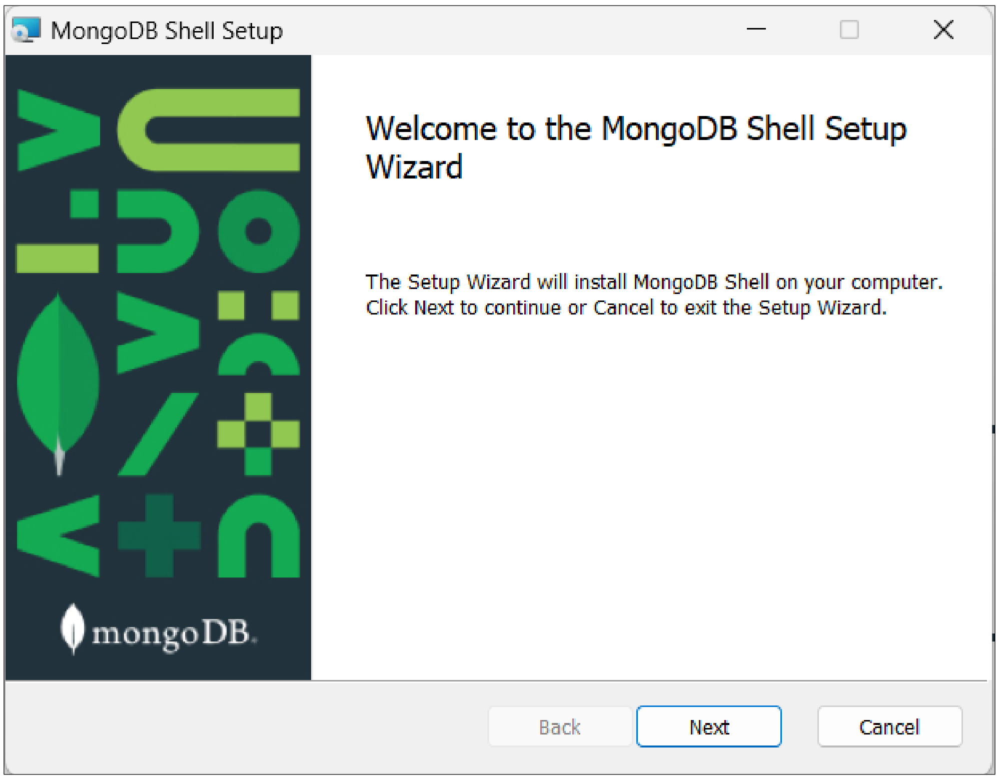
Package
msi

msi

Figure 1.9: Version, Platform, and Package Selection

2. On this page, scroll down and ensure that:
   - In the **Version** drop-down, **1.8.2** is selected.
   - In the **Platform** drop-down, **Windows 64-bit (8.1+) (MSI)** is selected.
   - In the **Package** drop-down, **msi** is selected.
3. Click **Download**.
4. After the download is complete, run the installer.

The MongoDB Shell Setup wizard opens as shown in Figure 1.10.

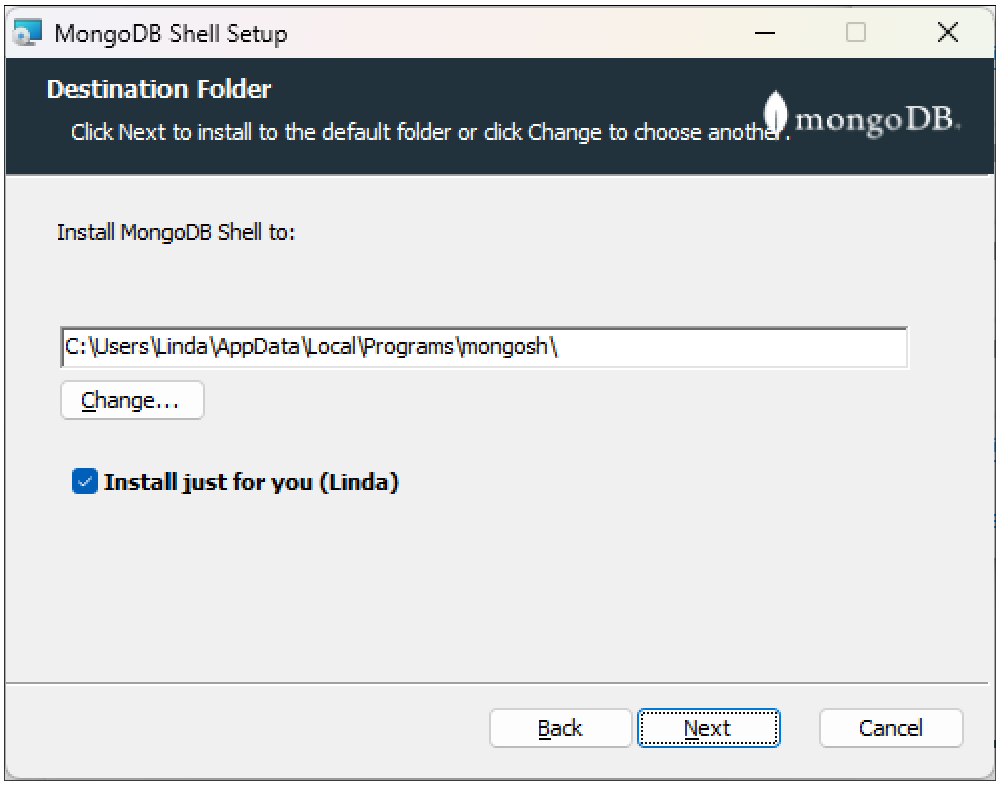

MongoDB Shell Setup

Welcome to the MongoDB Shell Setup Wizard

The Setup Wizard will install MongoDB Shell on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Figure 1.10: Screenshot of MongoDB Shell Setup Wizard

5. Click **Next**.
   The Destination Folder page of the wizard opens as shown in Figure 1.11.


MongoDB Shell Setup

Destination Folder

Click Next to install to the default folder or click Change to choose another.

Install MongoDB Shell to:

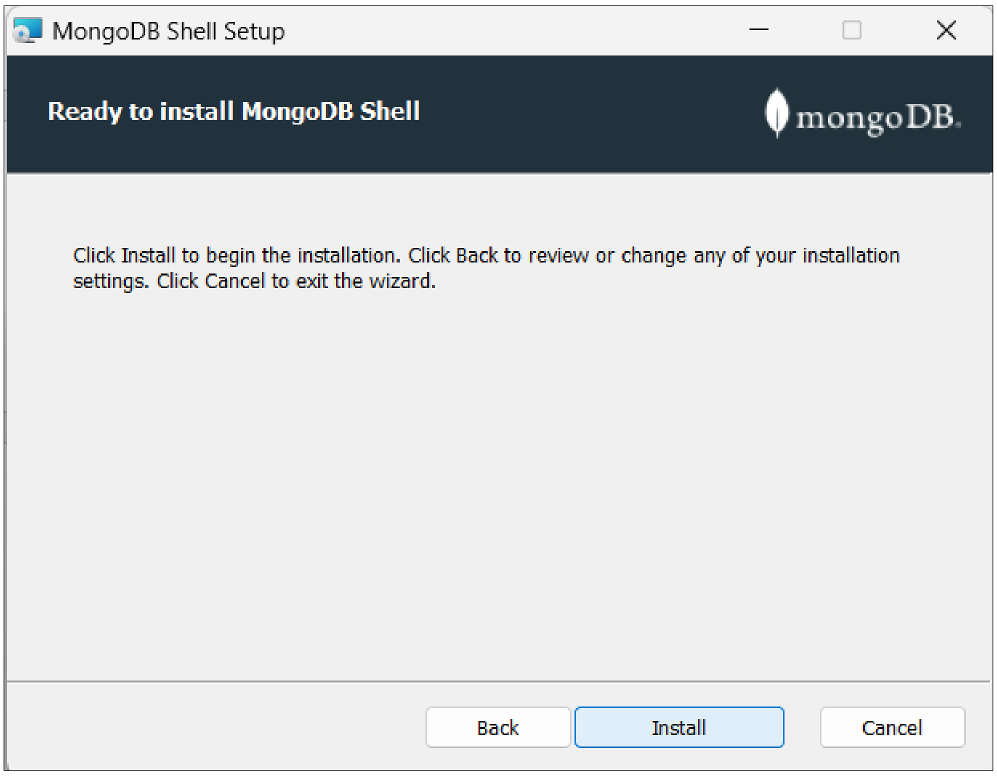
C:\Users\Linda\AppData\Local\Programs\mongosh\

☑ Install just for you (Linda)

Figure 1.11: Destination Folder Page

6. Check the folder path and click **Next**.

The Ready to install MongoDB Shell page of the wizard opens as shown in Figure 1.12.


MongoDB Shell Setup

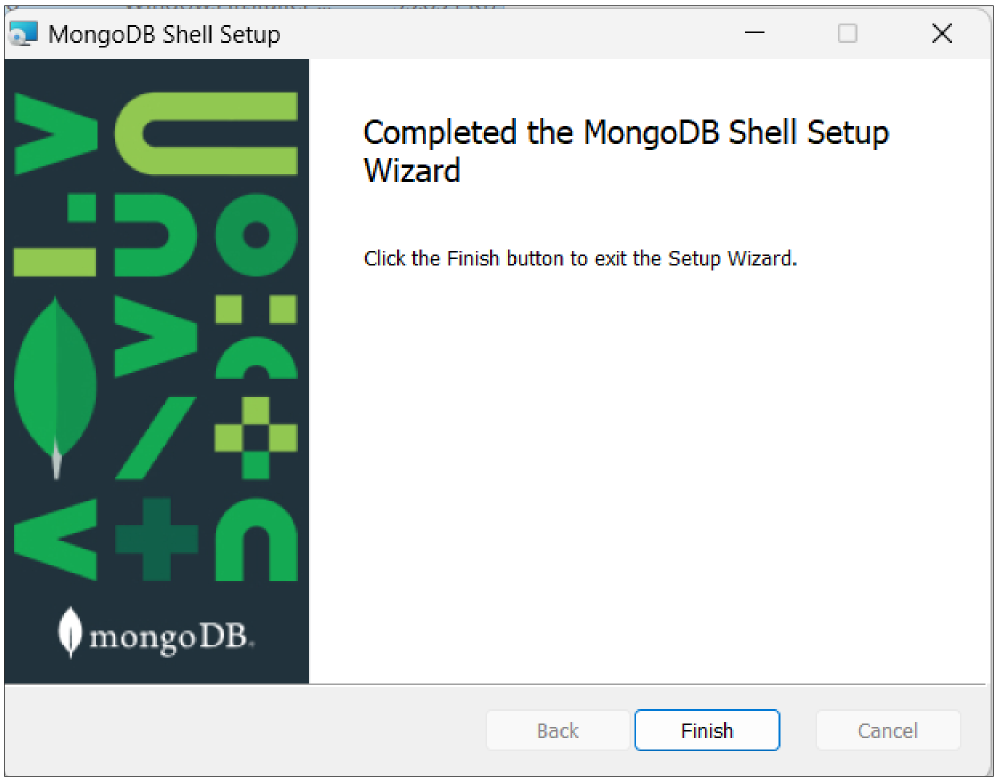
Ready to install MongoDB Shell

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Figure 1.12: Ready to Install MongoDB Shell Page

7. Click **Install**.

The MongoDB Shell Setup page of the wizard opens as shown in Figure 1.13.



Completed the MongoDB Shell Setup Wizard

Click the Finish button to exit the Setup Wizard.

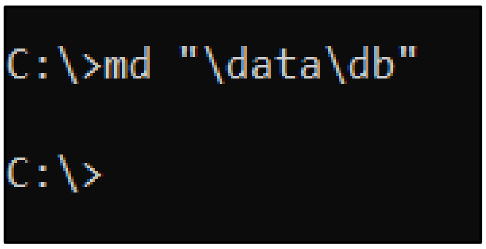
Figure 1.13: Screenshot of the MongoDB Shell Setup Page

8. Click **Finish**.

1.5 Setting up the Environment

To set up the environment:

1. Open a Windows command prompt (cmd.exe) as an **Administrator**.
2. Create the data directory where MongoDB will store all the data, as shown in Figure 1.14.



```
C:\>md "\data\db"

C:\>
```

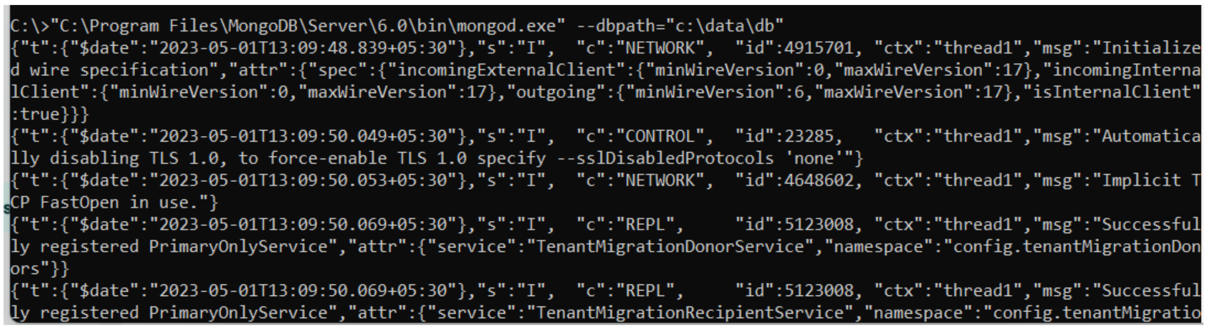
Figure 1.14: Creating Directories

MongoDB listens for connections from clients only on port 27017, by default. The default storage path for data is /data/db directory.

To start the **MongoDB** database, run the command:

```
"C:\Program Files\MongoDB\Server\6.0\bin\mongod.exe" --
dbpath="c:\data\db"
```

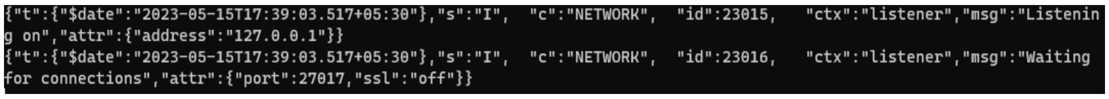
The command is executed as shown in Figure 1.15.



```
C:\>"C:\Program Files\MongoDB\Server\6.0\bin\mongod.exe" --dbpath="c:\data\db"
{"t":{"$date":"2023-05-01T13:09:48.839+05:30"},"s":"I",  "c":"NETWORK",  "id":4915701, "ctx":"thread1","msg":"Initialize
d wire specification","attr":{"spec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":17},"incomingInterna
lClient":{"minWireVersion":0,"maxWireVersion":17},"outgoing":{"minWireVersion":6,"maxWireVersion":17},"isInternalClient"
:true}}}
{"t":{"$date":"2023-05-01T13:09:50.049+05:30"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx":"thread1","msg":"Automatica
lly disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2023-05-01T13:09:50.053+05:30"},"s":"I",  "c":"NETWORK",  "id":4648602, "ctx":"thread1","msg":"Implicit T
CP FastOpen in use."}
{"t":{"$date":"2023-05-01T13:09:50.069+05:30"},"s":"I",  "c":"REPL",     "id":5123008, "ctx":"thread1","msg":"Successful
ly registered PrimaryOnlyService","attr":{"service":"TenantMigrationDonorService","namespace":"config.tenantMigrationDon
ors"}}
{"t":{"$date":"2023-05-01T13:09:50.069+05:30"},"s":"I",  "c":"REPL",     "id":5123008, "ctx":"thread1","msg":"Successful
ly registered PrimaryOnlyService","attr":{"service":"TenantMigrationRecipientService","namespace":"config.tenantMigratio
```

Figure 1.15: Starting the MongoDB Database

The `--dbpath` option shows the database directory. If the MongoDB database server is running correctly, the message: **waiting for connections** is displayed as shown in Figure 1.16.



```
{"t":{"$date":"2023-05-15T17:39:03.517+05:30"},"s":"I",  "c":"NETWORK",  "id":23015,   "ctx":"listener","msg":"Listenin
g on","attr":{"address":"127.0.0.1"}}
{"t":{"$date":"2023-05-15T17:39:03.517+05:30"},"s":"I",  "c":"NETWORK",  "id":23016,   "ctx":"listener","msg":"Waiting
for connections","attr":{"port":27017,"ssl":"off"}}
```

Figure 1.16: Waiting for Connections Message

Now, the MongoDB instance `mongod` is successfully running.

Add C:\Program Files\MongoDB\Server\6.0\bin to the system path. This allows to run the `mongod` instance instantly without having to navigate to this folder each time and then run the instance.

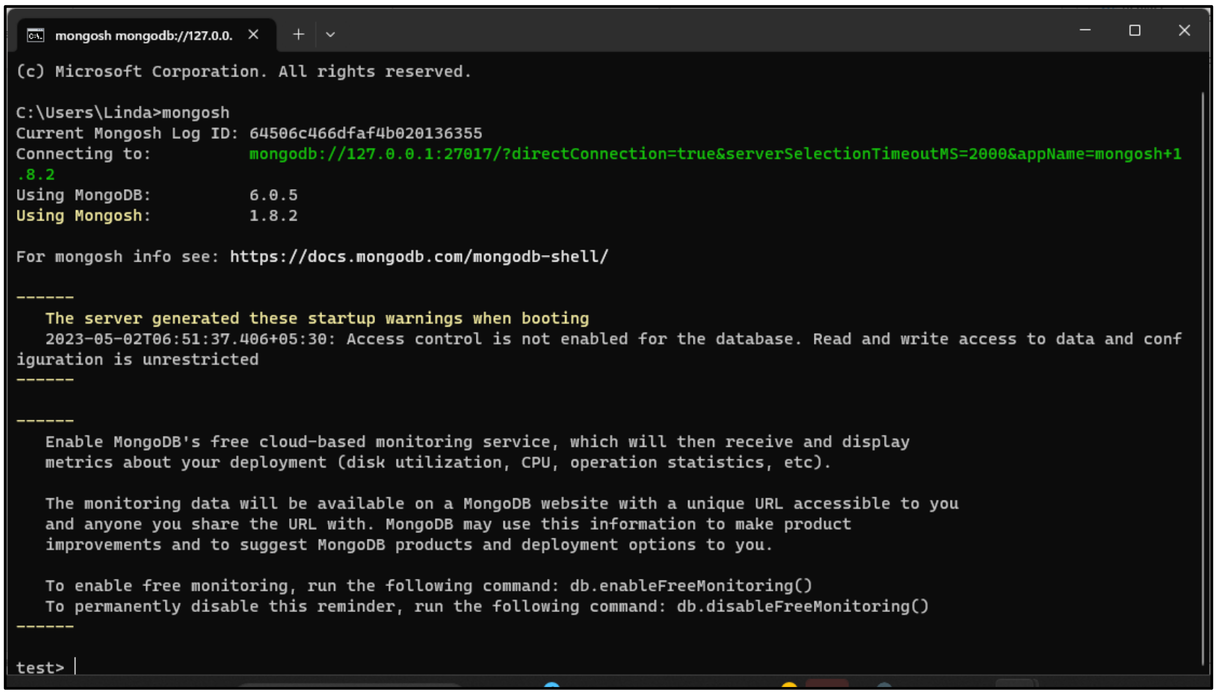
1.6 Connecting MongoDB Server and MongoDB Shell

To connect the MongoDB Shell with MongoDB Server:

1. Open another Windows command prompt/interpreter.
2. To connect the MongoDB Shell to a MongoDB deployment running on **localhost** with the default port 27017, run the command:

```
mongosh
```

The command is executed as shown in Figure 1.17.



```
(c) Microsoft Corporation. All rights reserved.

C:\Users\Linda>mongosh
Current Mongosh Log ID: 64506c466dfaf4b020136355
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1
.8.2
Using MongoDB:          6.0.5
Using Mongosh:          1.8.2

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

------
   The server generated these startup warnings when booting
   2023-05-02T06:51:37.406+05:30: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------

------
   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()
------

test> |
```

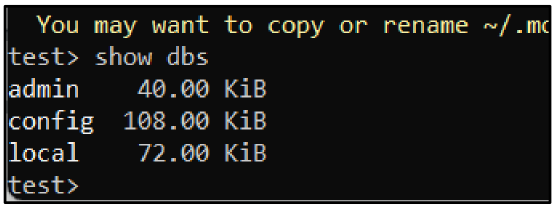
Figure 1.17: Output of the Mongosh Command

MongoDB Shell is now successfully connected to the MongoDB server.

3. To check the connection, run the command:

```
show dbs
```

List of exisitng databases is displayed as shown in Figure 1.18



```
  You may want to copy or rename ~/.mo
test> show dbs
admin     40.00 KiB
config   108.00 KiB
local     72.00 KiB
test>
```

Figure 1.18: Output of the `show dbs` Command

1.7 Summary

- A NoSQL database is a non-relational data that can support huge volumes of unstructured data.
- MongoDB is an open-source NoSQL database that stores data in the BSON format.
- The drivers, shell, and storage engine are vital elements for MongoDB to function.
- MongoDB stores data in the form of databases, collections, and documents.
- MongoDB works in two layers: application layer and data layer.
- Mongo DB Compass is the GUI for interacting with the MongoDB Server.
- MongoDB Atlas is the product MongoDB offers to compute on the cloud.
- To set up the MongoDB environment, first, install the MongoDB Server, then install the MongoDB Shell, and then use `mongosh` to connect the server to the shell.

Test Your Knowledge

1. Which of the following options is the data storage model of NoSQL databases?

    a. Document
    b. Tables with fixed rows and columns
    c. Key-value
    d. Graph

2. Which of the following statements are not true about NoSQL databases?

    a. It has flexible schema.
    b. It supports Horizontal scaling.
    c. The data architectures offered by NoSQL databases are more versatile than the data architectures offered by the relational database tables.
    d. It offers only read scalability.

3. Document database stores data in which of the following format?

    a. JSON
    b. BSON
    c. Google Docs
    d. XML

4. Which of the following is the default port for MongoDB which listens to the client for connection?

    a. 27017
    b. 20717
    c. 21071
    d. 20171

5. Which of the following service does MongoDB cloud Atlas provides?

    a. Infrastructure as a service
    b. Platform as a service
    c. Data as a service
    d. Function as a service

Answers to Test Your Knowledge

1	a, c, d
2	d
3	a, b, d
4	a
5	c

Try it Yourself

1. Install the **MongoDB 6.0** Community edition on your Windows.
2. Install **Mongo Shell 1.8.2** on your Windows system.
3. Using command prompt, set up the environment for MongoDB to execute `mongod` instance and connect `mongosh` with `mongod` instance. Execute a command to view the databases.

SESSION 2
MONGODB DATABASES

Learning Objectives

In this session, students will learn to:

- Explain the installation of MongoDB Database Tools and set the environment variable for MongoDB Database Tools
- Explain the method to load a sample dataset into MongoDB server
- Describe different datatypes in MongoDB
- Explain the methods to create database and collection
- Describe the ways to insert, query, update, and delete documents, collections, and databases

This session will provide an overview of installing MongoDB Database Tools and setting the environment variables for it. It will also explain how to load a sample database into MongoDB server. This session will explore the methods provided by MongoDB to create databases and collections. It will also explain the methods to insert, query, update, and delete documents, collections, and databases.

2.1 Install MongoDB Database Tools

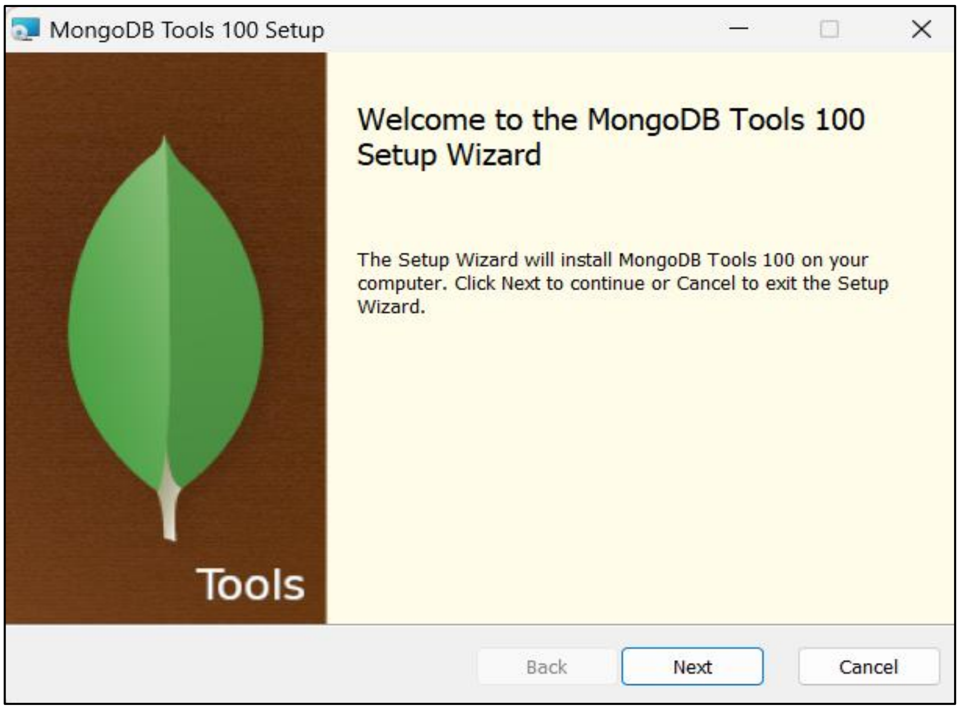
MongoDB provides a collection of tools that facilitate the users to work with and manage databases. Table 2.1 describes some of the common MongoDB tools.

Tool Name	Description
`mongodump`	Exports and creates a binary backup of the contents of a database
`mongorestore`	Restores backup data from mongodump into a database and loads standard input data into a mongod or mongos instance
`mongoexport`	Exports and creates a JavaScript Object Notation (JSON) or Comma-Separated Values (CSV) backup of the contents stored in a MongoDB instance
`mongoimport`	Imports data from CSV, JSON, and Tab-Separated Values (TSV) files into a MongoDB instance
`bsondump`	Converts BSON(Binary JavaScript Object Notation) files into JSON files
`mongotop`	Provides data on the amount of time spent by a MongoDB instance on reading and writing
`mongofiles`	Facilitates the manipulation of files stored in a MongoDB instance
`mongostat`	Provides statistics such as the number of inserts, deletes, and update queries executed per second and the amount of virtual memory used by the process

Table 2.1: Some Common MongoDB Tools

To install MongoDB database tools for Windows:

1. Open the browser and navigate to the Website https://www.mongodb.com/try/download/database-tools
2. On this page, scroll down:
    - In the **Version** drop-down, ensure that **100.7.0** is selected.
    - In the **Platform** drop-down, ensure that **Windows x86_64** is selected.
    - In the **Package** drop-down, select the **msi** option.
3. Click **Download**.
4. After the download is complete, run the installer.
   The **MongoDB Tools 100 Setup** wizard opens as shown in Figure 2.1.

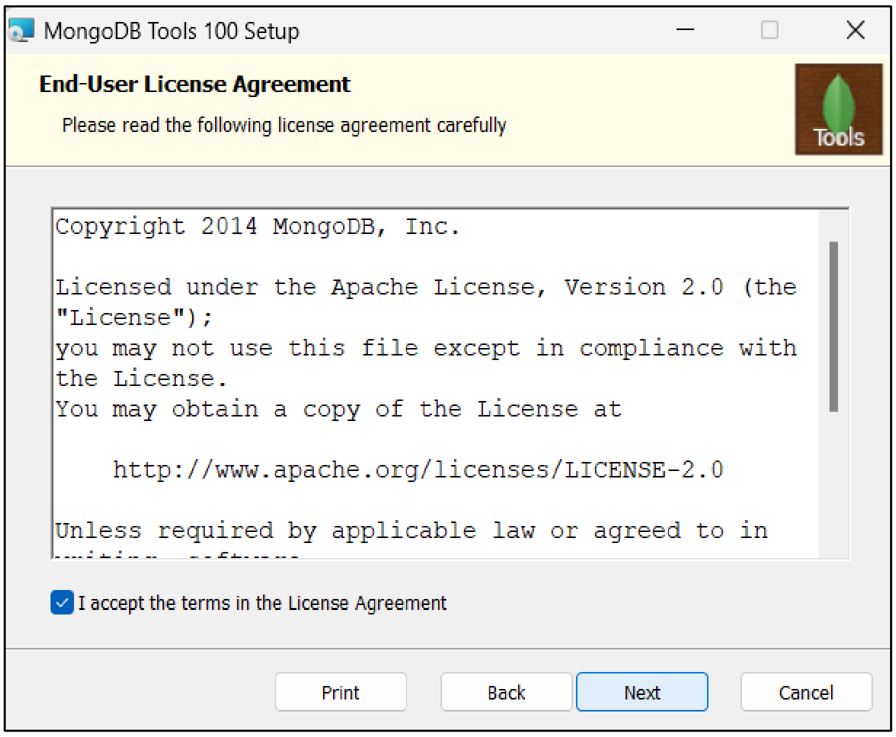


Welcome to the MongoDB Tools 100 Setup Wizard

The Setup Wizard will install MongoDB Tools 100 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Figure 2.1: MongoDB Tools Setup wizard

5. Click **Next**.
   The **End-User License Agreement** page opens.
6. Select the **I accept the terms in the License Agreement** check box as shown in Figure 2.2.



End-User License Agreement

Please read the following license agreement carefully

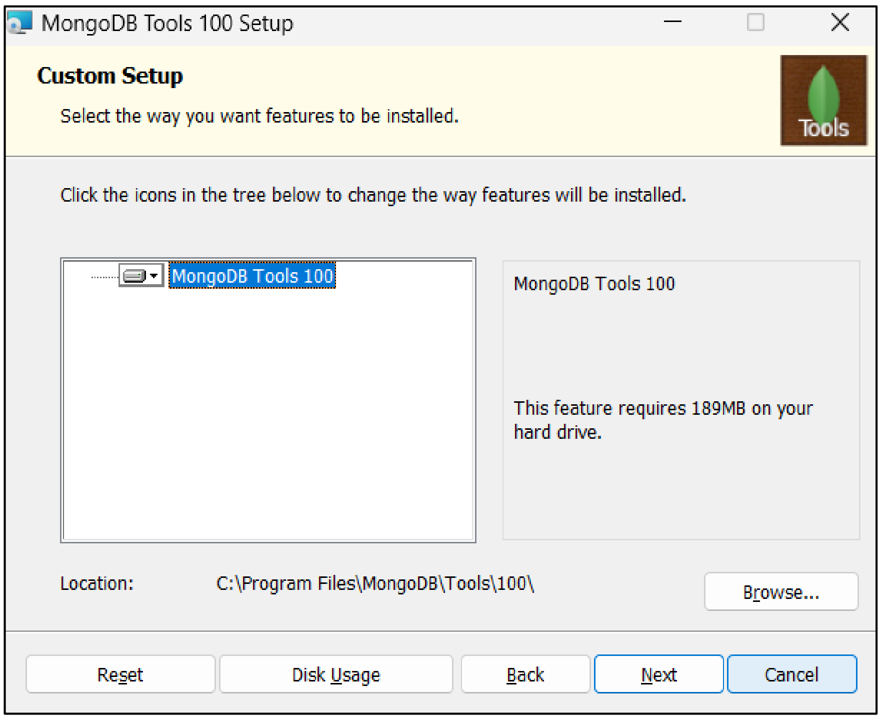
```
Copyright 2014 MongoDB, Inc.

Licensed under the Apache License, Version 2.0 (the
"License");
you may not use this file except in compliance with
the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in
```

✓ I accept the terms in the License Agreement

Figure 2.2: End-User License Agreement

7. Click **Next**.

The **Custom Setup** page opens as shown in Figure 2.3.


MongoDB Tools 100 Setup

Custom Setup

Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

MongoDB Tools 100

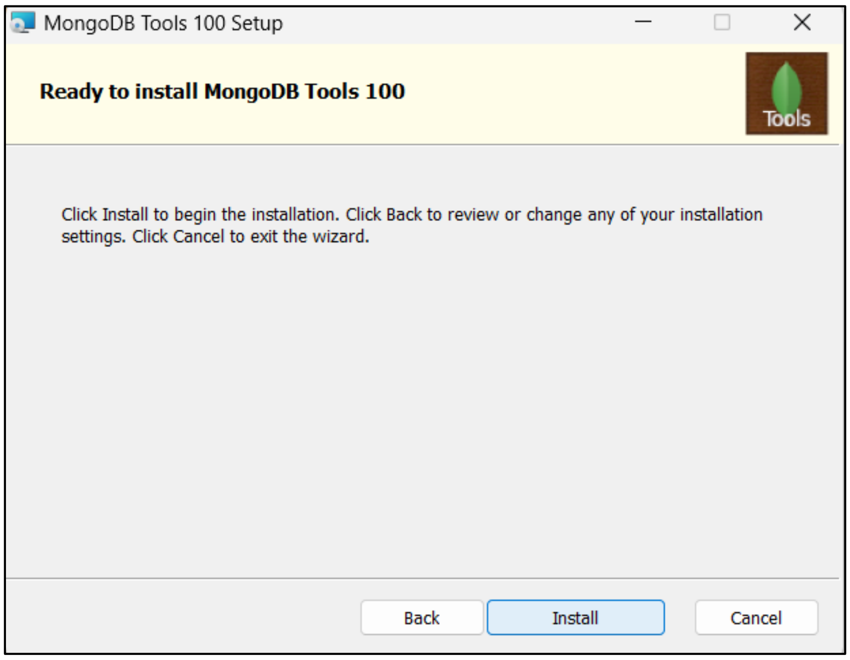
MongoDB Tools 100

This feature requires 189MB on your hard drive.

Location: C:\Program Files\MongoDB\Tools\100\

Figure 2.3: Custom Setup Page

8. Click **Next**.
   The **Ready to Install MongoDB Tools 100** page opens as shown in Figure 2.4.

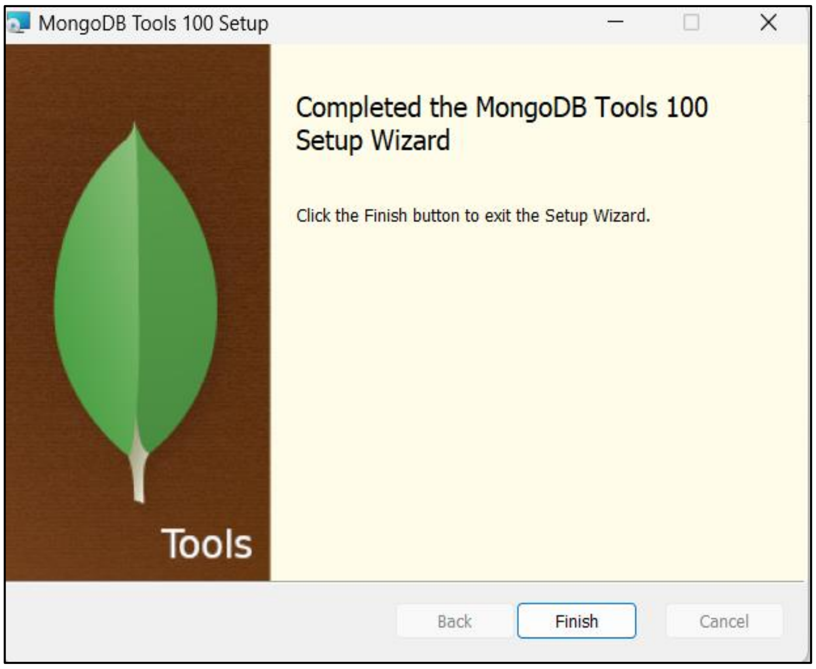

MongoDB Tools 100 Setup

Ready to install MongoDB Tools 100

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Figure 2.4: Ready to Install MongoDB Tools 100

9. Click **Install**.
   The **Completed the MongoDB Tools 100 Setup Wizard** page opens as shown in Figure 2.5.



Completed the MongoDB Tools 100 Setup Wizard

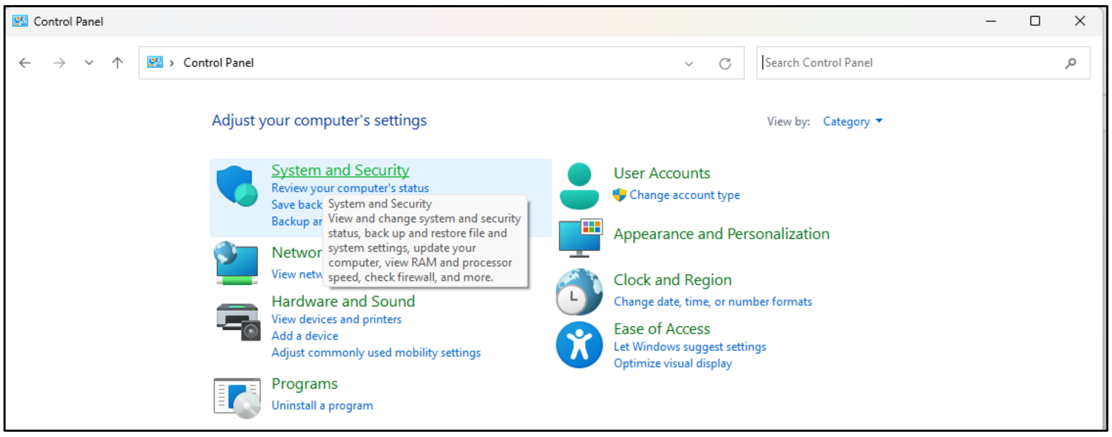
Click the Finish button to exit the Setup Wizard.

Figure 2.5: Completed the MongoDB Tools 100 Setup Wizard

10. Click **Finish**.

Now, MongoDB Database Tools are installed. The next step is to set the environment variables for the executable file. To set the environment variable:

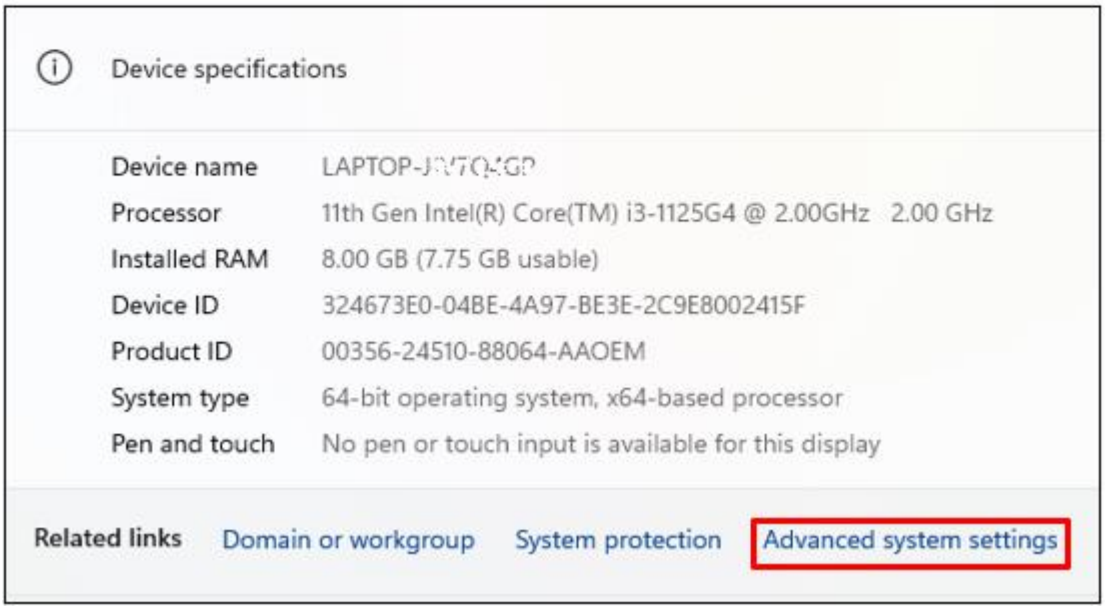
1. Open the **Control Panel** in **Windows**.
2. In the **Control Panel** window, select **System and Security** as shown in Figure 2.6.



System and Security
View and change system and security status, back up and restore file and system settings, update your computer, view RAM and processor speed, check firewall, and more.

Figure 2.6: Control Panel

3. In the **System and Security** window that opens, click **System**.
4. In the **System** → **About** window that opens, select **Advanced system settings** as shown in Figure 2.7.



Device specifications

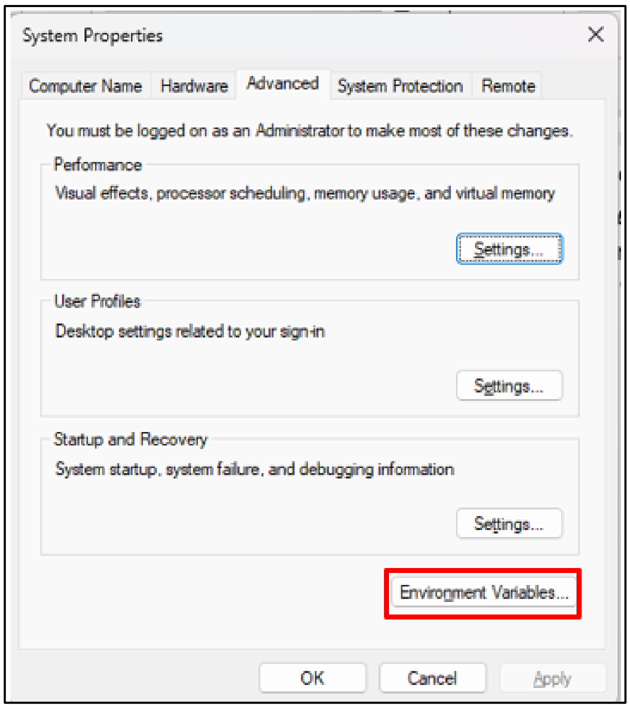
Device name	LAPTOP-J[illegible]
Processor	11th Gen Intel(R) Core(TM) i3-1125G4 @ 2.00GHz 2.00 GHz
Installed RAM	8.00 GB (7.75 GB usable)
Device ID	324673E0-04BE-4A97-BE3E-2C9E8002415F
Product ID	00356-24510-88064-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Related links Domain or workgroup System protection Advanced system settings

Figure 2.7: Advanced System Settings

The **System Properties** window opens.

5. In the **System Properties** window, select **Environment Variables** as shown in Figure 2.8.



System Properties

Computer Name Hardware Advanced System Protection Remote

You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

Settings...

User Profiles

Desktop settings related to your sign-in

Settings...

Startup and Recovery

System startup, system failure, and debugging information

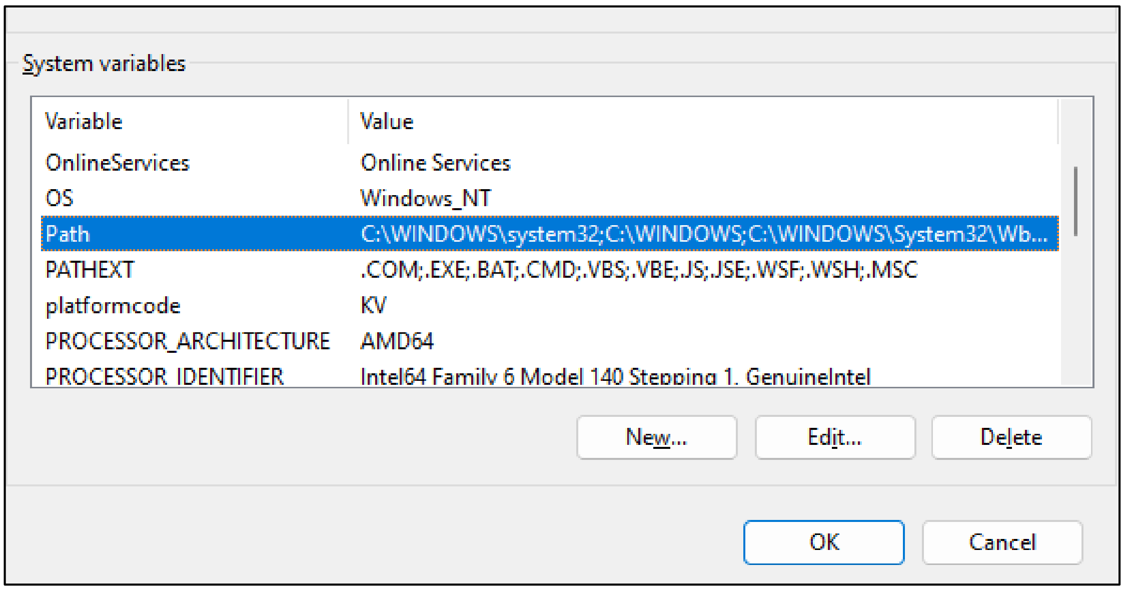
Settings...

Environment Variables...

OK Cancel Apply

Figure 2.8: System Properties

6. In the **Environment Variables** window that opens, under **System variables** section, select **Path** and click **Edit** as shown in Figure 2.9.



System variables

Variable	Value
OnlineServices	Online Services
OS	Windows_NT
Path	C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wb...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
platformcode	KV
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 140 Stepping 1, GenuineIntel

Figure 2.9: Path

7. In the **Edit environment variable** window that opens, click **New**, and then click **Browse** to navigate to the folder where the database tools are installed.
8. Click **OK** in all the windows opened in Steps 5 through 7.

The environment variable is now set for MongoDB tools.

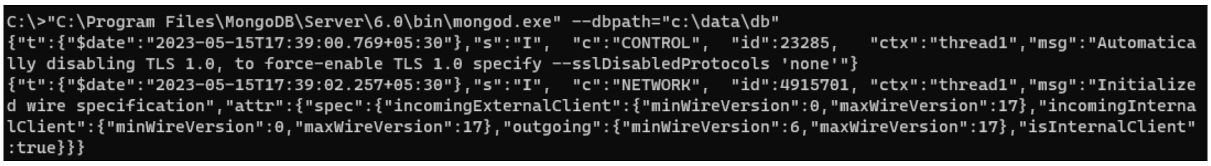
2.2 Load Sample Dataset into MongoDB Server

To load a sample dataset into MongoDB Server using the `mongoimport` tool:

1. Copy the sample dataset named `Sample_dataset` to the C drive.
2. Open the command prompt.
3. To run the `mongod` instance, execute the command:

```
"C:\Program Files\MongoDB\Server\6.0\bin\mongod.exe" --
dbpath="c:\data\db"
```

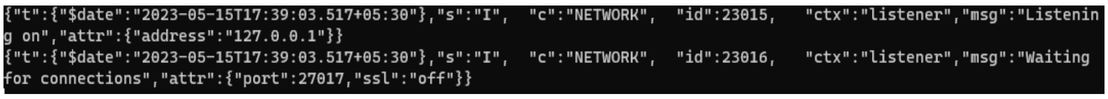
The command is executed as shown in Figure 2.10.



```
C:\>"C:\Program Files\MongoDB\Server\6.0\bin\mongod.exe" --dbpath="c:\data\db"
{"t":{"$date":"2023-05-15T17:39:00.769+05:30"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx":"thread1","msg":"Automatica
lly disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2023-05-15T17:39:02.257+05:30"},"s":"I",  "c":"NETWORK",  "id":4915701, "ctx":"thread1","msg":"Initialize
d wire specification","attr":{"spec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":17},"incomingInterna
lClient":{"minWireVersion":0,"maxWireVersion":17},"outgoing":{"minWireVersion":6,"maxWireVersion":17},"isInternalClient"
:true}}}
```

Figure 2.10: Starting the MongoDB Database

The `--dbpath` option shows the database directory. If the MongoDB database server is running correctly, the message waiting for connections is displayed as shown in Figure 2.11.



```
{"t":{"$date":"2023-05-15T17:39:03.517+05:30"},"s":"I",  "c":"NETWORK",  "id":23015,   "ctx":"listener","msg":"Listenin
g on","attr":{"address":"127.0.0.1"}}
{"t":{"$date":"2023-05-15T17:39:03.517+05:30"},"s":"I",  "c":"NETWORK",  "id":23016,   "ctx":"listener","msg":"Waiting
for connections","attr":{"port":27017,"ssl":"off"}}
```

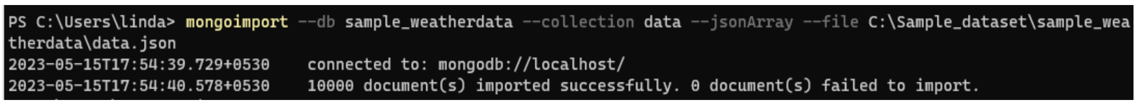
Figure 2.11: Waiting for Connections Message

Now, the MongoDB instance `mongod` is successfully running.

4. Open a new command prompt and type the command as:

```
mongoimport --db sample_weatherdata --collection data -
-jsonArray --file
C:\Sample_dataset\sample_weatherdata\data.json
```

The command executes as shown in Figure 2.12.



```
PS C:\Users\linda> mongoimport --db sample_weatherdata --collection data --jsonArray --file C:\Sample_dataset\sample_wea
therdata\data.json
2023-05-15T17:54:39.729+0530    connected to: mongodb://localhost/
2023-05-15T17:54:40.578+0530    10000 document(s) imported successfully. 0 document(s) failed to import.
```

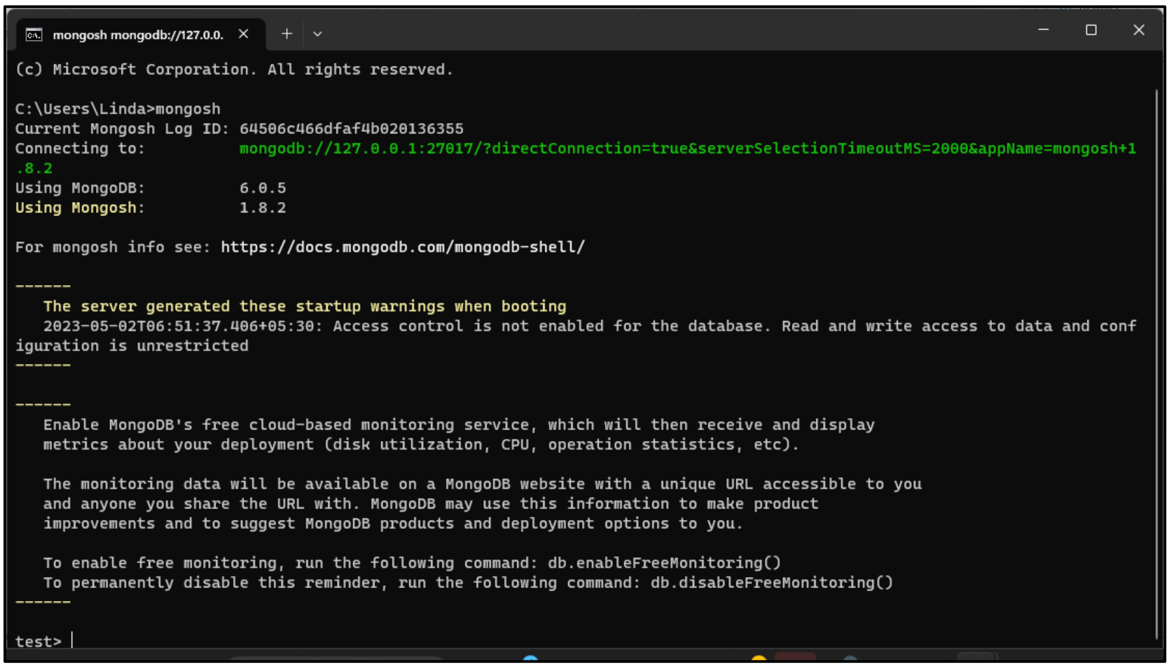
Figure 2.12: Import Sample Data

The sample dataset is now imported into the mongod instance.

5. To verify this, run the MongoDB shell command as:

```
mongosh
```

The command executes as shown in Figure 2.13.



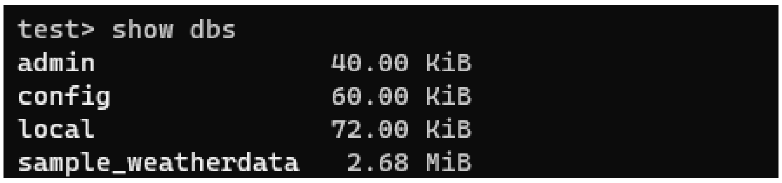
```
(c) Microsoft Corporation. All rights reserved.

C:\Users\Linda>mongosh
Current Mongosh Log ID: 64506c466dfaf4b020136355
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1
.8.2
Using MongoDB:          6.0.5
Using Mongosh:          1.8.2

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

------
   The server generated these startup warnings when booting
   2023-05-02T06:51:37.406+05:30: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------

------
   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()
------

test> |
```

Figure 2.13: `mongosh` Command

6. To view the databases available in this mongod instance, execute the command as:

```
show dbs
```

The command executes as shown in Figure 2.14.



```
test> show dbs
admin                40.00 KiB
config               60.00 KiB
local                72.00 KiB
sample_weatherdata    2.68 MiB
```

Figure 2.14: `show dbs` Command

This confirms that the sample data named `sample_weatherdata` is imported into the `mongod` instance.

The `show dbs` command would display a database only when the database contains at least one document.

2.3 Datatypes in MongoDB

MongoDB stores data internally in BSON format. BSON is an extension of JSON. It supports all the data types that JSON supports. Additionally, it supports other datatypes such as date, timestamp, and binary data. Table 2.2 describes the datatypes that are available in MongoDB.

Datatype	Description	Example
`String`	A `String` is used to store text in MongoDB. BSON strings are of Universal Coded Character Set (UCS) Transformation Format (UTF – 8).	Store the name of a student or employee, designation of an employee, and address
`Integer`	An `Integer` is used to store numeric values. MongoDB offers two variations of integer datatype—32-bit signed integer and 64-bit signed integer.	Store numeric values such as marks of a student, salary of an employee, and count of products in inventory
`Double`	A `Double` is used to store numeric values that contain 8-byte floating point values.	Store temperature, revenue, and price of an item
`Decimal`	Decimal is used to store 128-bit floating point decimal values.	Store financial and scientific computations
`Boolean`	`Boolean` is used to store true or false as values for the fields.	Store the value of `'Is_married'`, `'Is_fulltime_employee'`, and `'Is_passed'`
`Null`	`Null` is used to store NULL value.	Store the driver's license number of people who do not own a driver's license
`Array`	`Array` is a collection of multiple values that are of same type or different types.	Store marks of a student in five subjects
`Object`	`Object` is used to store embedded documents.	Store the description of an item which would include the item's shape, color, and dimensions
`ObjectID`	`ObjectID` is a unique identifier for each document in a collection. It is a hexadecimal string of 12-bytes that includes: • Timestamp value (4-bytes) • Random value(5-bytes with 3-bytes of machine ID and 2-bytes of process ID) • Counter (3-bytes)	A hexadecimal ID that MongoDB assigns to each new field in MongoDB, for example, `'64617a0420cf1016275bd88f'`
`Date`	Stores date as a 64-bit signed integer which is a Universal Time Co-ordinated (UTC)	`String date:` Mon May 15 2023 01:03:25 GMT-0500 (EST)

Datatype	Description	Example
	datetime format. This BSON date format allows storage of dates before and after Jan 1, 1970, 00:00:00 using the negative and positive sign-bits, respectively. When the user creates a date variable in MongoDB, it returns the date in string format. When the user creates a date variable in MongoDB using the `new` keyword, MongoDB returns it in the International Standards Organization Date (`ISODate`) format.	`ISODate`: 2023-05-15T04:25:48.512Z

Table 2.2: Datatypes in MongoDB

2.4 Working with Databases and Collections

Consider that the user wants to create a database in MongoDB and perform certain operations in it. MongoDB offers a set of commands that facilitates the user to create and drop databases and collections. MongoDB also provides commands that can be used to insert, query, update, and delete documents in a collection.

2.4.1 Create Database

Consider that the user wants to create a database named `Student_detail`. The syntax to create a database is:

```
use DATABASE_NAME
```

To create a database named `Student_detail`:

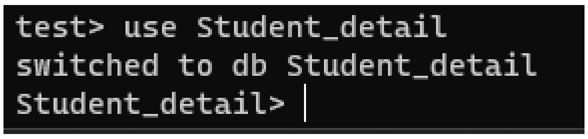
1. Ensure that the `mongod` instance is running.
2. To run the MongoDB Shell, at the command prompt, execute the command as:

```
mongosh
```

3. To create a database named `Student_detail`, run the command as:

```
use Student_detail
```

The command executes as shown in Figure 2.15.



```
test> use Student_detail
switched to db Student_detail
Student_detail>
```

Figure 2.15: Create Database

The database named `Student_detail` has been created.

2.4.2 Create Collection

Consider that the user wants to create a collection named `Studentinfo` in the `Student_detail` database. The syntax to create a collection or a view is:

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

Table 2.3 describes the parameters of the `createCollection` command.

Parameter	Type	Description
`name`	`String`	This parameter specifies the name of the collection to be created. This is a mandatory parameter.
`options`	`Document`	This parameter is optional. It includes fields that specify whether the entity to be created must be a: • **Capped collection:** Capped collections are collections of fixed size. If the maximum size of the collection is reached, then the oldest documents are erased to create space for new entries. The size parameter and the maximum number of documents that can be created must be specified for this option. • **Timeseries collection:** Timeseries collections are collections that store sequences of data points (key-value pairs) over a period along with the time at which the data points were recorded. • **Clustered collection:** Clustered collections are collections that are stored in the order of a clustered index—a key value that specifies the order of arrangement of the documents in a collection. The parameter includes fields such as key value field, uniqueness of the clustered index, and name of the index. • **View:** View is a virtual collection. If a view must be created, the name of the source collection or view must be specified. This parameter also includes a field named `validator` that allows the users to create collections with user-defined validation rules.

Table 2.3: Parameters of `createCollection()` Method

To create a collection named `Studentinfo` in the database named `Student_detail`:

1. To switch to the `Student_detail` database, execute the command as:

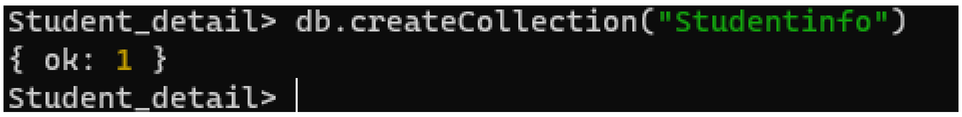
```
use Student_detail
```

2. To create a collection named `Studentinfo`, execute the command

as:

```
db.createCollection("Studentinfo")
```

The command executes as shown in Figure 2.16.



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

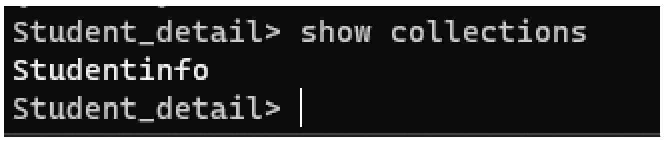
Figure 2.16: Create Collection

The `Studentinfo` collection is created.

3. To view the collection inside the `Student_detail` database, execute the command as:

```
show collections
```

The command is executed as shown in Figure 2.17.



```
Student_detail> show collections
Studentinfo
Student_detail>
```

Figure 2.17: Show Collections

2.4.3 Insert Document

Documents can be inserted into a collection in a database. MongoDB provides options to insert a single document or multiple documents into a collection.

Insert a Single Document

The syntax to insert a single document into a collection is:

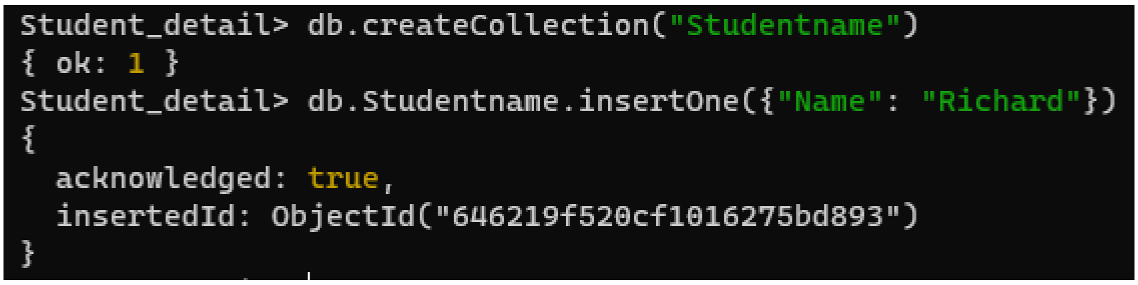
```
db.collection.insertOne()
```

Consider that the user wants to insert a document into the `Studentname` collection in the `Student_detail` database.

To perform this task, execute the commands as:

```
db.createCollection("Studentname")
db.Studentname.insertOne({"Name": "Richard"})
```

The command executes as shown in Figure 2.18.



```
Student_detail> db.createCollection("Studentname")
{ ok: 1 }
Student_detail> db.Studentname.insertOne({"Name": "Richard"})
{
  acknowledged: true,
  insertedId: ObjectId("646219f520cf1016275bd893")
}
```

Figure 2.18: Insert Single Document

A document is inserted into the `Studentname` collection.

The `insertedID` value in the output is the unique identifier generated by MongoDB for each document. This identifier is of the `ObjectId` datatype.

If the user tries to insert a document into a collection that does not exist in the database, then MongoDB automatically creates the collection.

Insert Multiple Documents

The syntax to insert multiple documents in a collection is:

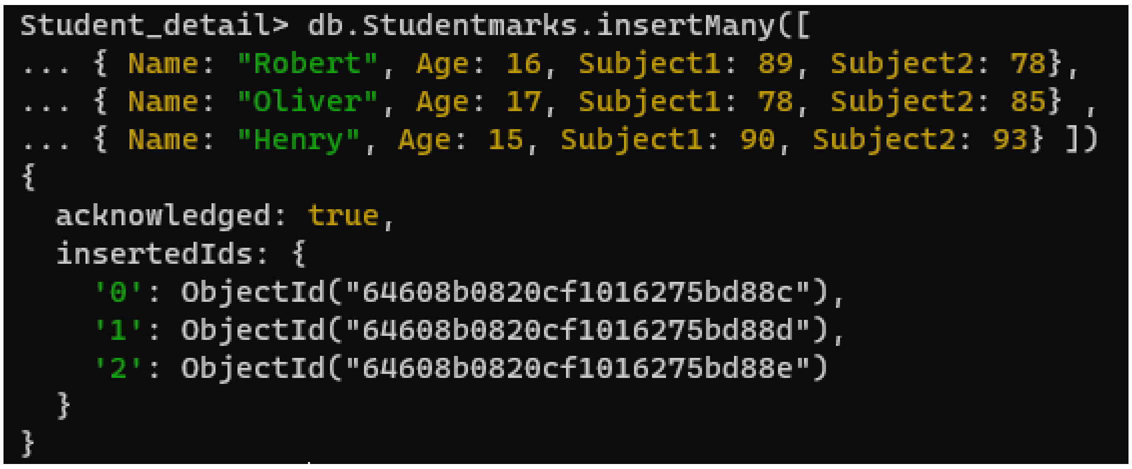
```
db.collection.insertMany()
```

Consider that the user wants to insert three documents into the `Studentmarks` collection in the `Student_detail` database.

To perform this task, execute the command as:

```
db.Studentmarks.insertMany([
{ Name: "Robert", Age: 16, Subject1: 89, Subject2: 78},
{ Name: "Oliver", Age: 17, Subject1: 78, Subject2: 85} ,
{ Name: "Henry", Age: 15, Subject1: 90, Subject2: 93} ])
```

The command executes as shown in Figure 2.19.



```
Student_detail> db.Studentmarks.insertMany([
... { Name: "Robert", Age: 16, Subject1: 89, Subject2: 78},
... { Name: "Oliver", Age: 17, Subject1: 78, Subject2: 85} ,
... { Name: "Henry", Age: 15, Subject1: 90, Subject2: 93} ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("64608b0820cf1016275bd88c"),
    '1': ObjectId("64608b0820cf1016275bd88d"),
    '2': ObjectId("64608b0820cf1016275bd88e")
  }
}
```

Figure 2.19: Insert Many Documents

2.4.4 Query Document

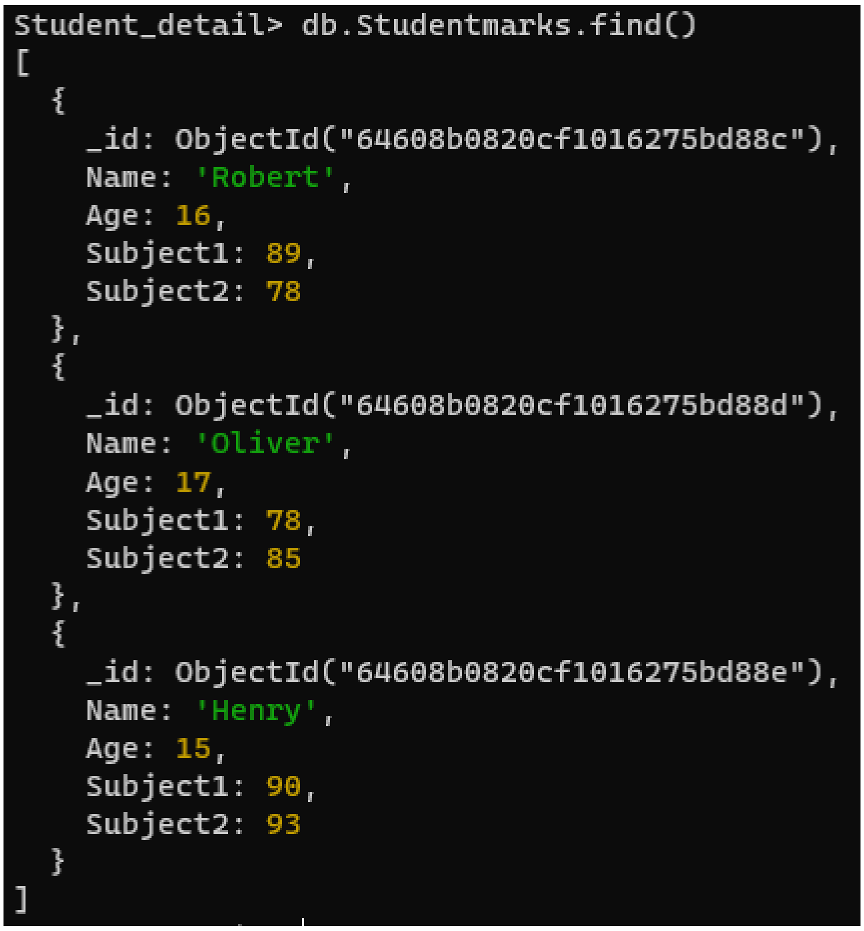
MongoDB provides commands to retrieve the documents from a collection. The syntax to retrieve all the documents in a collection is:

```
db.collection.find()
```

Consider that the user wants to retrieve all the documents in the `Studentmarks` collection. To perform this task, execute the command as:

```
db.Studentmarks.find()
```

The command executes as shown in Figure 2.20.



```
Student_detail> db.Studentmarks.find()
[
  {
    _id: ObjectId("64608b0820cf1016275bd88c"),
    Name: 'Robert',
    Age: 16,
    Subject1: 89,
    Subject2: 78
  },
  {
    _id: ObjectId("64608b0820cf1016275bd88d"),
    Name: 'Oliver',
    Age: 17,
    Subject1: 78,
    Subject2: 85
  },
  {
    _id: ObjectId("64608b0820cf1016275bd88e"),
    Name: 'Henry',
    Age: 15,
    Subject1: 90,
    Subject2: 93
  }
]
```

Figure 2.20: Retrieve Documents in a Collection

All the documents in the `Studentmarks` collection are retrieved and displayed.

2.4.5 Update Document

MongoDB provides two different methods to update documents in a collection. These methods can be used to:

- Update single document
- Update multiple documents

Update Single Document

The syntax to update a single document is:

```
db.collection.updateOne(filter, update, options)
```

Table 2.4 describes the parameters of the `updateOne` method.

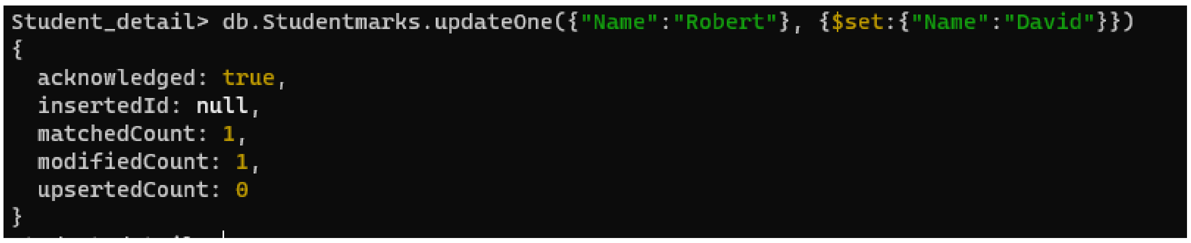
Parameter	Type	Description
`filter`	Document	This parameter specifies the selection criteria based on which the update must be performed. If multiple documents satisfy the criteria, then only the first document is updated. This parameter is mandatory.
`update`	Document	This parameter specifies the change that must be applied to the filtered documents. This parameter is mandatory.
`options`	Document	This parameter specifies some additional options—`upsert`, `writeconcern`, `collation`, `arrayfilters`, and `hint`—for the update method. These options will be explored in detail in later sessions. This parameter is optional.

Table 2.4: Parameters of `updateOne()` Method

Consider that the user wants to update the value of name field as "`David`" where the value in the `Name` field is "`Robert`". To perform this task, execute the command as:

```
db.Studentmarks.updateOne({"Name":"Robert"},
{$set:{"Name":"David"}})
```

The command executes as shown in Figure 2.21.



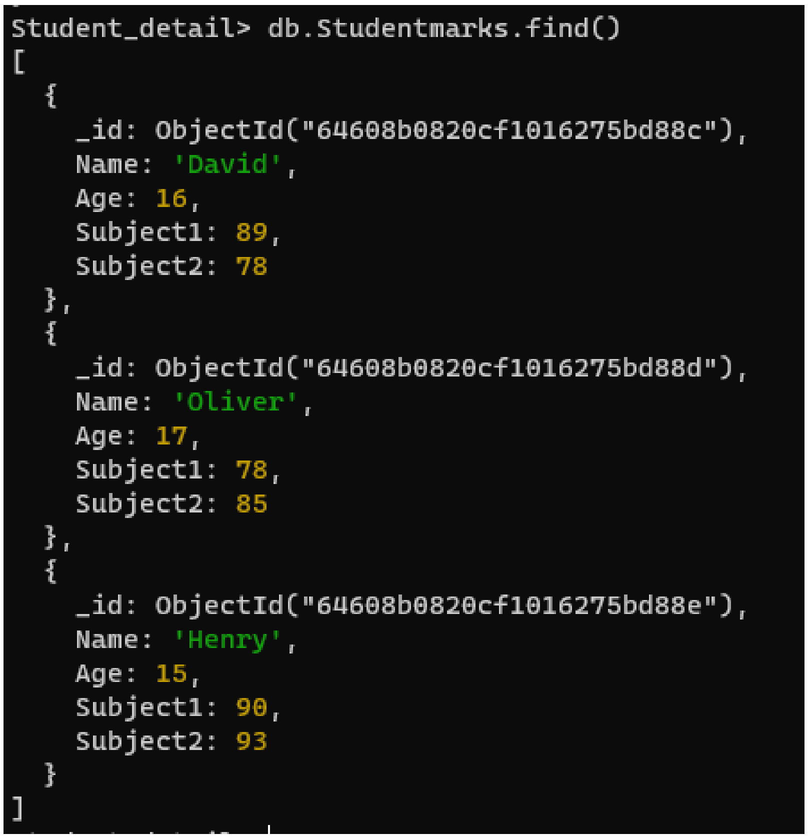
```
Student_detail> db.Studentmarks.updateOne({"Name":"Robert"}, {$set:{"Name":"David"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Figure 2.21: Method to Update Single Document

The document is updated. To view the updated document, execute the command as:

```
db.Studentmarks.find()
```

The command executes as shown in Figure 2.22.



```
Student_detail> db.Studentmarks.find()
[
  {
    _id: ObjectId("64608b0820cf1016275bd88c"),
    Name: 'David',
    Age: 16,
    Subject1: 89,
    Subject2: 78
  },
  {
    _id: ObjectId("64608b0820cf1016275bd88d"),
    Name: 'Oliver',
    Age: 17,
    Subject1: 78,
    Subject2: 85
  },
  {
    _id: ObjectId("64608b0820cf1016275bd88e"),
    Name: 'Henry',
    Age: 15,
    Subject1: 90,
    Subject2: 93
  }
]
```

Figure 2.22: Updated Document

Update Multiple Documents

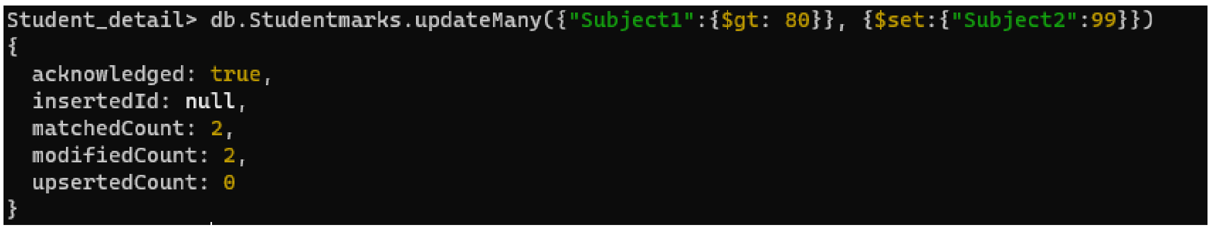
The syntax to update multiple documents is:

```
db.collection.updateMany(filter, update, options)
```

Consider that the user wants to update and set the value of `Subject2` as `99` in all documents where the value of `Subject1` is greater than `80`. To perform this task, execute the command as:

```
db.Studentmarks.updateMany({"Subject1":{$gt: 80}},
{$set:{"Subject2":99}})
```

The command executes as shown in Figure 2.23.



```
Student_detail> db.Studentmarks.updateMany({"Subject1":{$gt: 80}}, {$set:{"Subject2":99}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
```

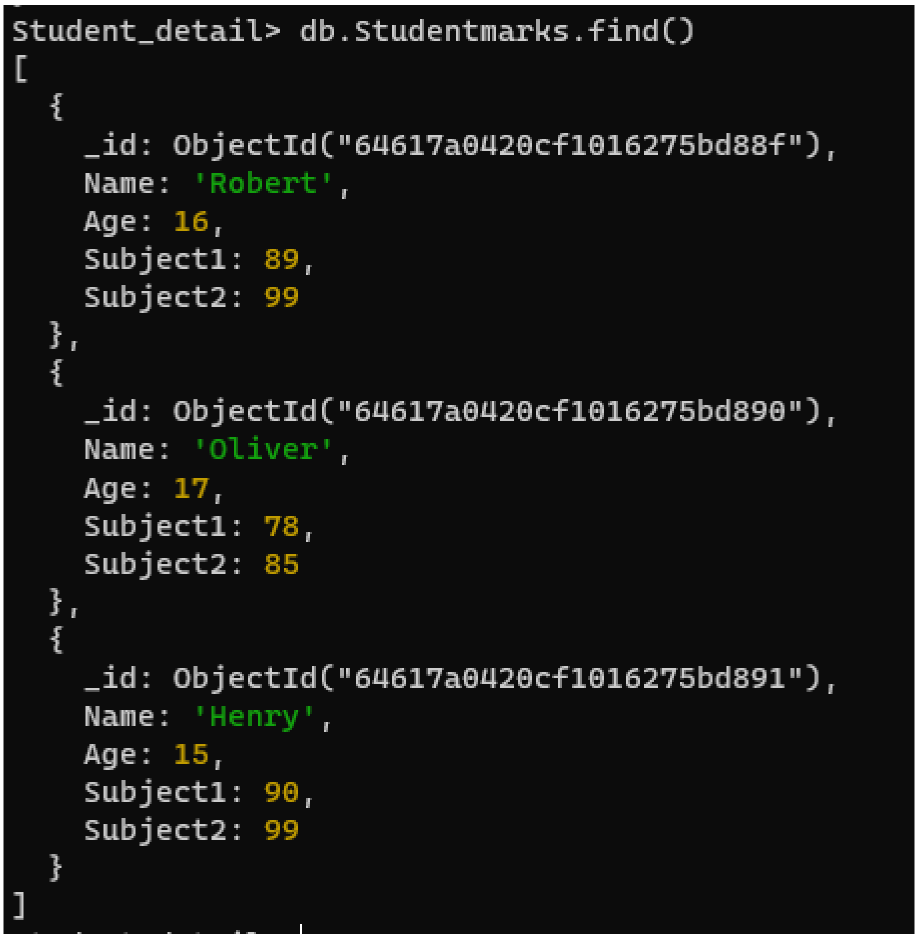
Figure 2.23: Method to Update Many Documents

The document is updated. In this command, `$gt` is a comparison operator that is used to filter out only those documents with `Subject1` greater than `80`.

To view the updated documents, execute the command as:

```
db.Studentmarks.find()
```

The command executes as shown in Figure 2.24.



```
Student_detail> db.Studentmarks.find()
[
  {
    _id: ObjectId("64617a0420cf1016275bd88f"),
    Name: 'Robert',
    Age: 16,
    Subject1: 89,
    Subject2: 99
  },
  {
    _id: ObjectId("64617a0420cf1016275bd890"),
    Name: 'Oliver',
    Age: 17,
    Subject1: 78,
    Subject2: 85
  },
  {
    _id: ObjectId("64617a0420cf1016275bd891"),
    Name: 'Henry',
    Age: 15,
    Subject1: 90,
    Subject2: 99
  }
]
```

Figure 2.24: Updated Documents

Here, two documents are updated as they satisfy the specified criteria.

2.4.6 Delete Document

MongoDB provides two different methods to delete documents in a collection. These methods can be used to:

- Delete single document
- Delete many documents

Delete Single Document
The syntax to delete single document is:

```
db.collection.deleteOne(filter, options)
```

Table 2.5 describes the parameters of the `deleteOne` method.

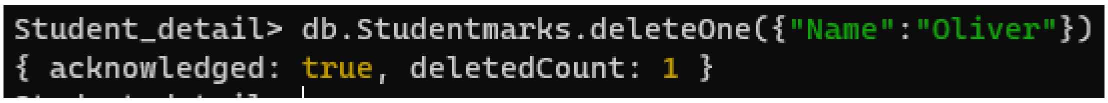
Parameter	Type	Description
`filter`	`Document`	This parameter specifies the selection criteria based on which the delete must be performed. If an empty criterion is mentioned, then the first document returned in the collection is deleted. This parameter is mandatory.
`options`	`Document`	This parameter specifies some additional options—writeconcern, collation, and hint—for the delete method. These options will be explored in detail in later sessions. This parameter is optional.

Table 2.5: Parameters of `deleteOne` Method

Consider that the user wants to delete the document where the value of the `Name` field is "`Oliver`". To perform this task, execute the command as:

```
db.Studentmarks.deleteOne({"Name":"Oliver"})
```

The command executes as shown in Figure 2.25.



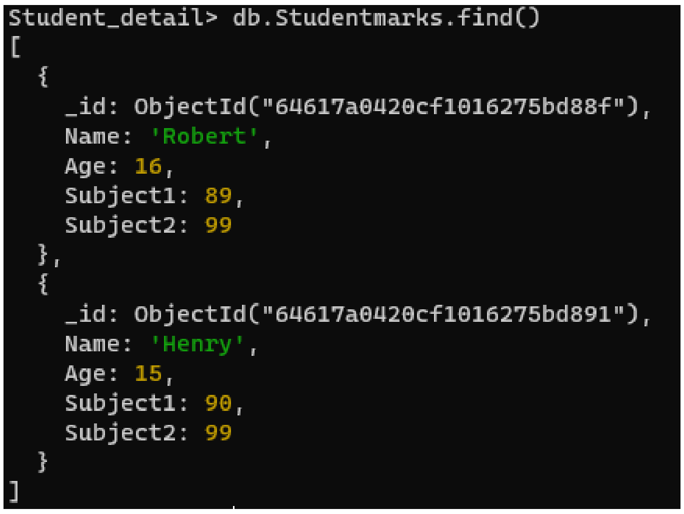
```
Student_detail> db.Studentmarks.deleteOne({"Name":"Oliver"})
{ acknowledged: true, deletedCount: 1 }
```

Figure 2.25: Method to Delete Single Document

The document where the value of the name field is "`Oliver`" is deleted. To verify this, execute the command as:

```
db.Studentmarks.find()
```

The command executes as shown in Figure 2.26.



```
Student_detail> db.Studentmarks.find()
[
  {
    _id: ObjectId("64617a0420cf1016275bd88f"),
    Name: 'Robert',
    Age: 16,
    Subject1: 89,
    Subject2: 99
  },
  {
    _id: ObjectId("64617a0420cf1016275bd891"),
    Name: 'Henry',
    Age: 15,
    Subject1: 90,
    Subject2: 99
  }
]
```

Figure 2.26: Output of `deleteOne` Method

Delete Multiple Documents

The syntax to delete multiple documents is:

```
db.collection.deleteMany(filter, options)
```

Table 2.6 describes the parameters of the `deleteMany` method.

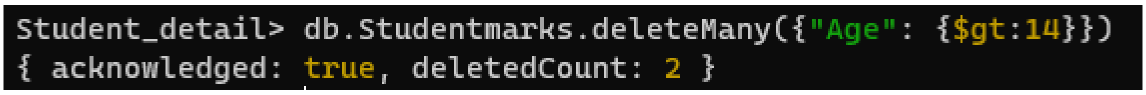
Parameter	Type	Description
`filter`	`Document`	This parameter specifies the selection criteria based on which the delete must be performed. If an empty criterion is mentioned, then all the documents in the collection are deleted. This parameter is mandatory.
`options`	`Document`	This parameter specifies some additional options—writeconcern, collation, and hint—for the delete method. These options will be explored in detail in later sessions. This parameter is optional.

Table 2.6: Parameters of `deleteMany` Method

Consider that the user wants to delete all the documents where the value in the `Age` field is greater than `14`. To perform this task, execute the command as:

```
db.Studentmarks.deleteMany({"Age": {$gt:14}})
```

The command executes as shown in Figure 2.27.



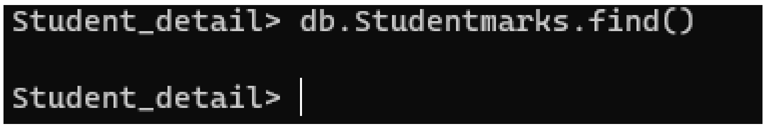
```
Student_detail> db.Studentmarks.deleteMany({"Age": {$gt:14}})
{ acknowledged: true, deletedCount: 2 }
```

Figure 2.27: **Method to Delete Many Documents**

Those documents where the value of the `Age` field is greater than `14` are deleted. To view the output, execute the command as:

```
db.Studentmarks.find()
```

The command executes as shown in Figure 2.28.



```
Student_detail> db.Studentmarks.find()

Student_detail>
```

Figure 2.28: Output of `deleteMany` Method

2.4.7 Drop Collection

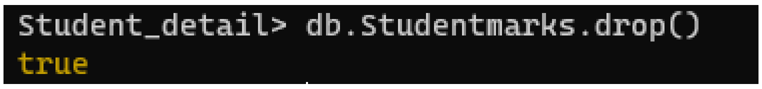
MongoDB provides a method to remove a collection from the database. The syntax to drop a collection is:

```
db.collection.drop()
```

Consider that the user wants to drop the collection named `Studentmarks` from the `Student_detail` database. To perform this task, execute the command as:

```
db.Studentmarks.drop()
```

The command executes as shown in Figure 2.29.



```
Student_detail> db.Studentmarks.drop()
true
```

Figure 2.29: Dropping a Collection

The `Studentmarks` collection is dropped. In MongoDB, when a collection is dropped, all the indexes associated with it are also dropped.

2.4.8 Drop Database

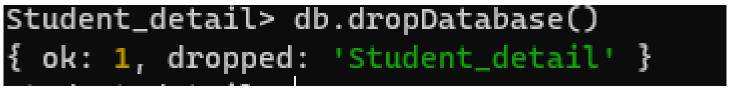
MongoDB provides a method to drop a database. The syntax to drop a database is:

```
db.dropDatabase()
```

This method drops the current database. Consider that the user wants to drop the `Student_detail` database. To perform this task, execute the command as:

```
db.dropDatabase()
```

The command is executed as shown in Figure 2.30.



```
Student_detail> db.dropDatabase()
{ ok: 1, dropped: 'Student_detail' }
```

Figure 2.30: Method to Drop a Database

The `Student_detail` database is dropped. When a database is dropped, all the collections and data files associated with it are also dropped. Here, the `Studentinfo` and `Studentname` collections are also dropped.

2.5 Summary

- MongoDB provides a set of tools that helps users to manage the databases in the MongoDB environment.
- To start using the commands in MongoDB, install MongoDB Database Tools and set the environment variable for it.
- A sample dataset can be loaded into MongoDB server using the `mongoimport` tool.
- MongoDB stores data in BSON format. It supports various datatypes such as string, integer, double, Boolean, null, array, object, objectID, and date.
- MongoDB provides different commands and methods to create and delete databases and collections. It also provides methods to insert, query, update, and delete documents and collections.

Test Your Knowledge

1. Which of the following file formats can be imported into MongoDB instance using the `mongoimport` database tool?

   a. CSV
   b. JSON
   c. DBF
   d. TSV

2. Which of the following commands is used to view the databases on the MongoDB server?

   a. show databases
   b. view dbs
   c. show dbs
   d. view databases

3. Consider that the user wants to create a database named `Employee_detai`l. Which of the following commands must be used to create the database?

   a. `use Employee_detail`
   b. `create Employee_detail`
   c. `create database Employee_detail`
   d. `use database Employee_detail`

4. Which of the following datatypes is used to store embedded documents in MongoDB?

   a. Array
   b. Object
   c. String object
   d. Array Object

5. Which of the following is the correct syntax to delete a collection from the current database?

   a. `db.collection.drop`
   b. `db.drop`
   c. `db.collection.delete`
   d. `db.delete`

Answers to Test Your Knowledge

1	a, b, d
2	c
3	a
4	b
5	a

Try it Yourself

1. Install MongoDB Database Tools and set the environment variable for MongoDB Database Tools.
2. Load the sample dataset `sample_training` into the MongoDB server.
3. Create a database named `Employee` and a collection named `Employee_detail`.
4. Insert the following documents into the `Employee_detail` collection.

```
[{
  Emp_name: "Oliver Smith",
  Age: 23,
  Designation: "Manager"
}
{
  Emp_name: "David Michael",
  Age: 32,
  Designation: "Software Engineer"
}]
```

5. Update the `designation` of "`Oliver Smith`" to "`Accountant`".
6. Delete the document in `Employee_detail` collection where the name of the employee is "`David Michael`".
7. Drop the collection `Employee_detail`.
8. Drop the database `Employee`.

SESSION 3
MONGODB OPERATORS

Learning Objectives

In this session, students will learn to:

- Describe the types of MongoDB operators
- Explain how to use the query and projection operators
- Explain how to modify field and array data using update operators

MongoDB provides various operators which enable users to effectively interact with the database.

This session describes the types of MongoDB operators and their use in general. This session also describes in detail the function of the more commonly used operators with real-life examples.

3.1 Introduction to MongoDB Operators

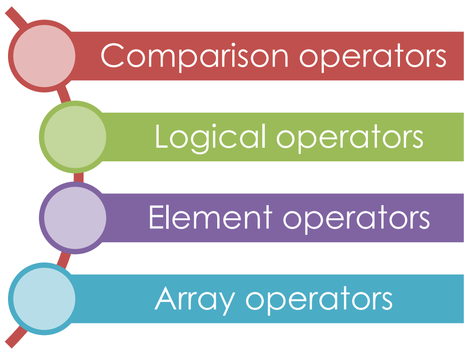
Operators are reserved words or symbols that instruct the compiler or interpreter to perform specific mathematical or logical operations on the dataset as required.

Some of the types of operators in MongoDB are:

- Query operators
- Projection operators
- Update operators
- Miscellaneous operators

3.2 Query Operators

Query operators are used to retrieve data from a database. Different types of query operators are:



3.2.1 Comparison Operators

Comparison operators compare the field's value with the specified value and return the documents that match the value. Table 3.1 describes the comparison operators in MongoDB.

Operator	Description
`$eq`	Used to retrieve, update, or delete documents where the field value is equal to a specified value
`$gt`	Used to retrieve, update, or delete documents where the field value is greater than a specified value
`$gte`	Used to retrieve, update, or delete documents where the field value is greater than or equal to a specified value
`$in`	Used to retrieve, update, or delete documents where the field value is any of the values specified in an array
`$lt`	Used to retrieve, update, or delete documents where the field value is less than a specified value
`$lte`	Used to retrieve, update, or delete documents where the field value is less than or equal to a specified value
`$ne`	Used to retrieve, update, or delete documents where the field value is not equal to a specified value
`$nin`	Used to retrieve, update, or delete documents where the field value is none of the values specified in an array.

Table 3.1: Comparison Operators in MongoDB

Let us take a look at how to use some of these comparison operators.

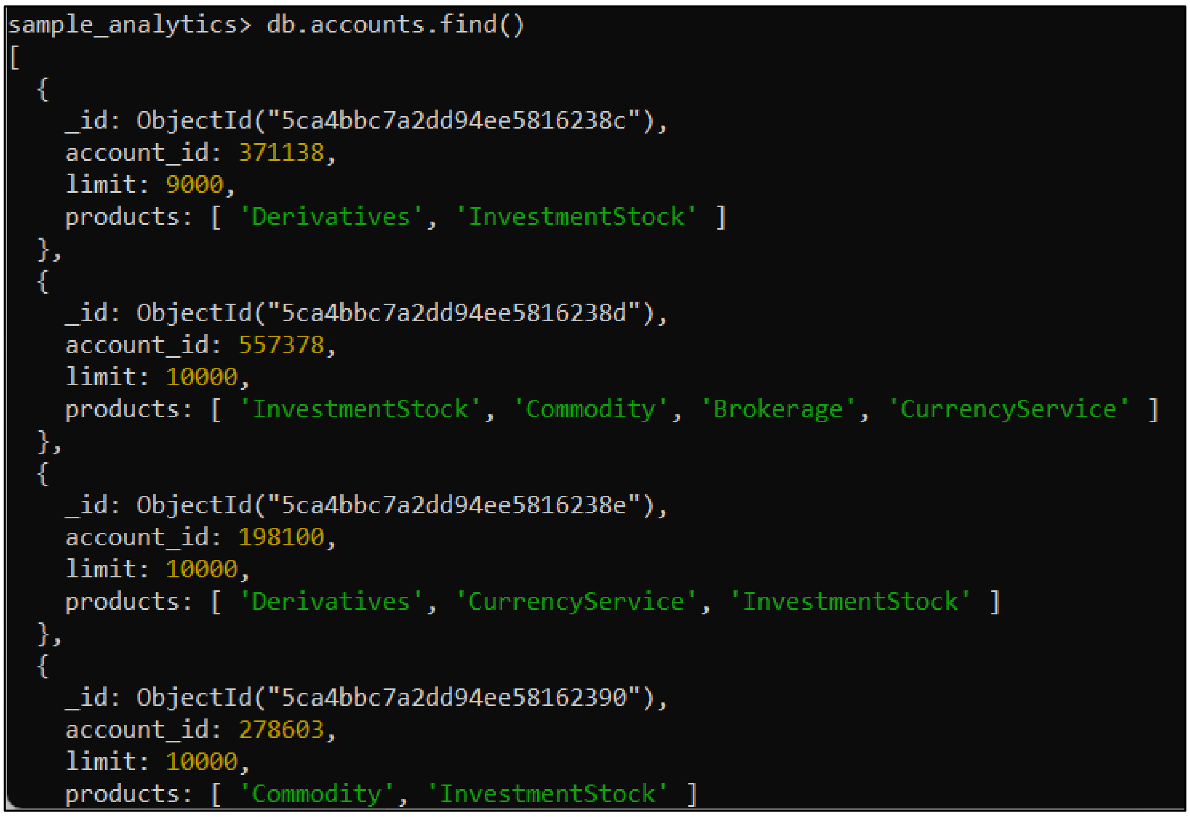
$eq Operator

This operator checks whether the value of a field equals the specified value. The syntax for this operator is:

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

Let us take a look at how to use this operator.

Figure 3.1 shows the documents stored in the `accounts` collection in the sample database named `sample_analytics`.



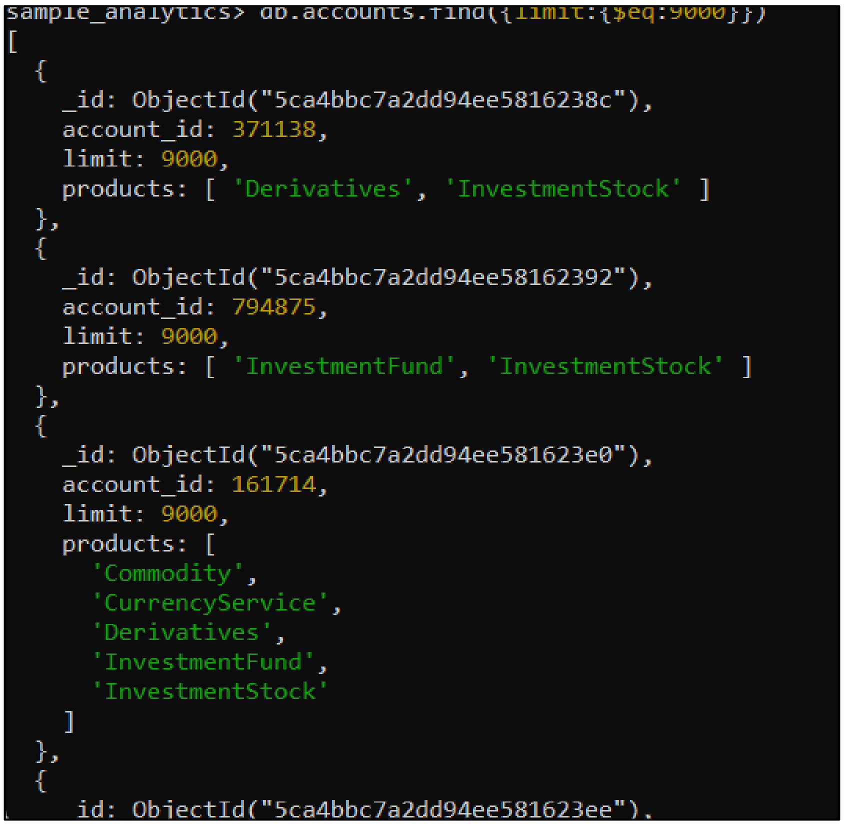
```
sample_analytics> db.accounts.find()
[
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816238c"),
    account_id: 371138,
    limit: 9000,
    products: [ 'Derivatives', 'InvestmentStock' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816238d"),
    account_id: 557378,
    limit: 10000,
    products: [ 'InvestmentStock', 'Commodity', 'Brokerage', 'CurrencyService' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816238e"),
    account_id: 198100,
    limit: 10000,
    products: [ 'Derivatives', 'CurrencyService', 'InvestmentStock' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee58162390"),
    account_id: 278603,
    limit: 10000,
    products: [ 'Commodity', 'InvestmentStock' ]
```

Figure 3.1: Documents in the Accounts Collection

Now, consider that the user wants to view the `accounts` which have the `limit` value as `9000`. To do so, the user can execute a query that uses the `$eq` operator as:

```
db.accounts.find({limit:{$eq:9000}})
```

Figure 3.2 shows the output of this query.



```
sample_analytics> db.accounts.find({limit:{$eq:9000}})
[
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816238c"),
    account_id: 371138,
    limit: 9000,
    products: [ 'Derivatives', 'InvestmentStock' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee58162392"),
    account_id: 794875,
    limit: 9000,
    products: [ 'InvestmentFund', 'InvestmentStock' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee581623e0"),
    account_id: 161714,
    limit: 9000,
    products: [
      'Commodity',
      'CurrencyService',
      'Derivatives',
      'InvestmentFund',
      'InvestmentStock'
    ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee581623ee"),
```

Figure 3.2: Output of the Query with `$eq` Operator

`$lt` Operator

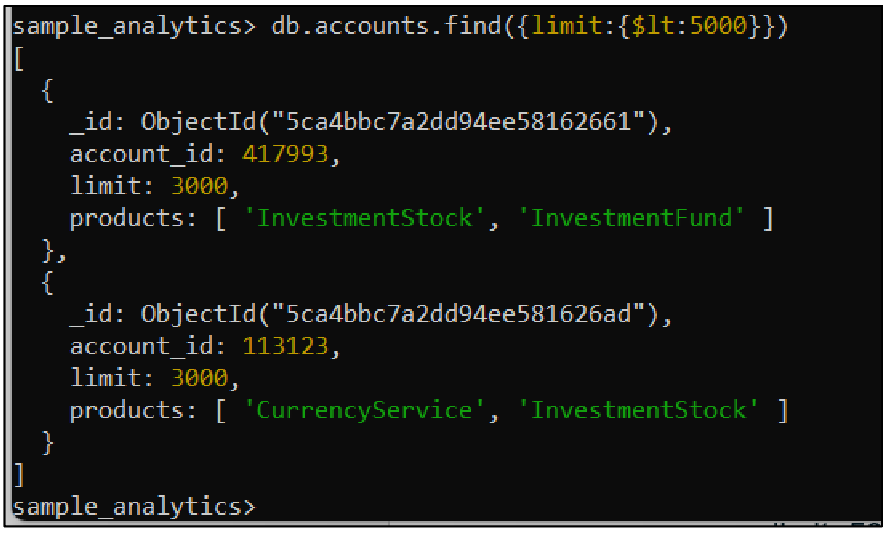
This operator checks whether the value of a field is less than the specified value. The syntax for this operator is:

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

For example, consider that in the `accounts` collection, the user wants to view the documents where the `limit` value is less than `5000`. To do so, the user can execute a query that uses the `$lt` operator as:

```
db.accounts.find({limit:{$lt:5000}})
```

Figure 3.3 shows the output of the query.



```
sample_analytics> db.accounts.find({limit:{$lt:5000}})
[
  {
    _id: ObjectId("5ca4bbc7a2dd94ee58162661"),
    account_id: 417993,
    limit: 3000,
    products: [ 'InvestmentStock', 'InvestmentFund' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee581626ad"),
    account_id: 113123,
    limit: 3000,
    products: [ 'CurrencyService', 'InvestmentStock' ]
  }
]
sample_analytics>
```

Figure 3.3: Output of the Query with `$lt` Operator

`$in` Operator

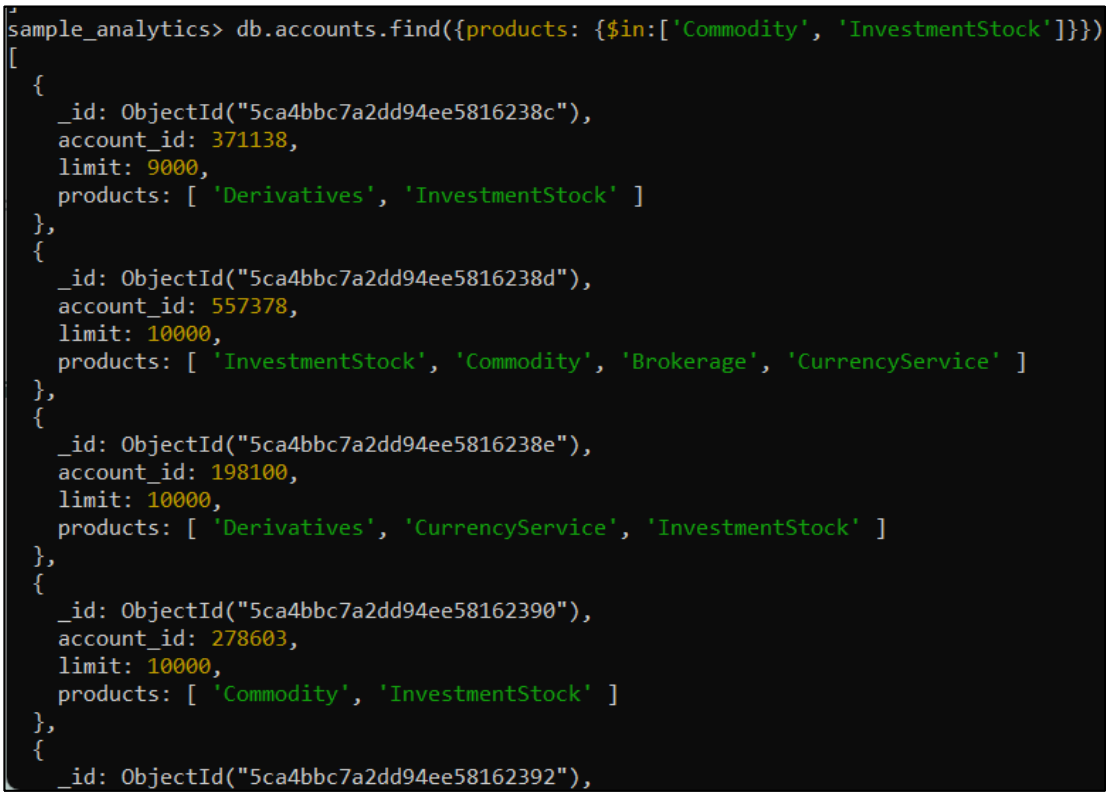
This operator checks whether the value of a field matches any of the values in the specified array. The syntax for this operator is:

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

For example, consider that from the `accounts` collection, the user wants to view only the documents where the product is either a '`Commodity`' or an '`InvestmentStock`'. To do so, the user can execute a query that uses the `$in` operator as:

```
db.accounts.find({products:
{$in:['Commodity', 'InvestmentStock']}})
```

Figure 3.4 shows the output of the query.

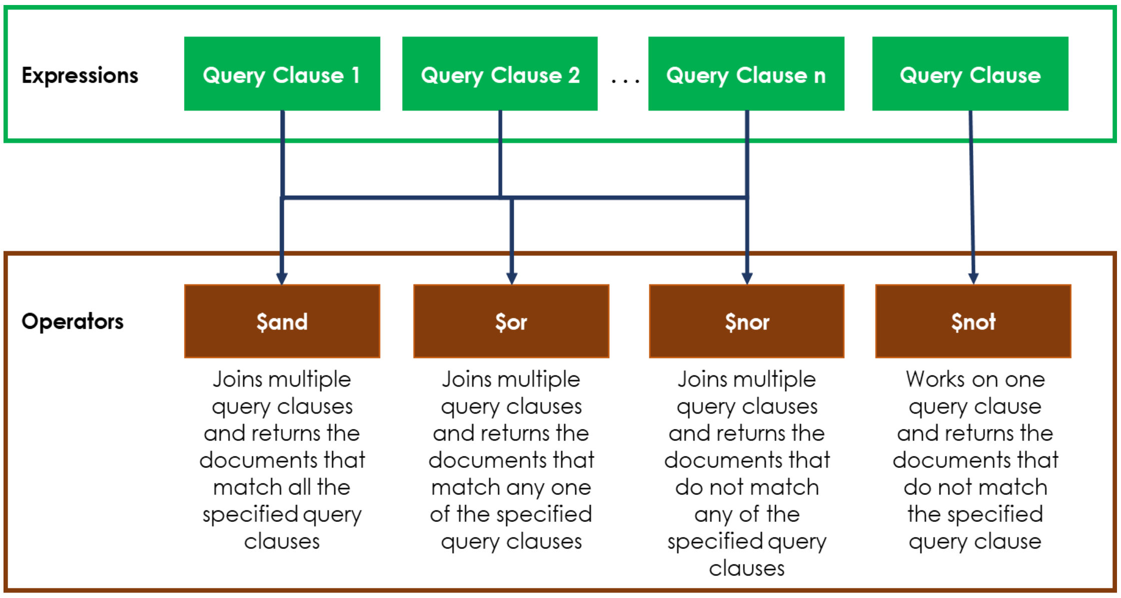


```
sample_analytics> db.accounts.find({products: {$in:['Commodity', 'InvestmentStock']}})
[
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816238c"),
    account_id: 371138,
    limit: 9000,
    products: [ 'Derivatives', 'InvestmentStock' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816238d"),
    account_id: 557378,
    limit: 10000,
    products: [ 'InvestmentStock', 'Commodity', 'Brokerage', 'CurrencyService' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816238e"),
    account_id: 198100,
    limit: 10000,
    products: [ 'Derivatives', 'CurrencyService', 'InvestmentStock' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee58162390"),
    account_id: 278603,
    limit: 10000,
    products: [ 'Commodity', 'InvestmentStock' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee58162392"),
```

Figure 3.4: Output of the Query with `$in` Operator

3.2.2 Logical Operators

Logical operators evaluate multiple expressions and return a Boolean value (true, false). Four logical operators are: `$and, $or, $nor, and $not`:


Expressions
Query Clause 1
Query Clause 2
...
Query Clause n
Query Clause
Operators
$and
Joins multiple query clauses and returns the documents that match all the specified query clauses
$or
Joins multiple query clauses and returns the documents that match any one of the specified query clauses
$nor
Joins multiple query clauses and returns the documents that do not match any of the specified query clauses
$not
Works on one query clause and returns the documents that do not match the specified query clause

`$and` Operator

This operator, as the name suggests, performs a logical AND operation. This means that the operator lists only the documents that match all the expressions in the specified array.

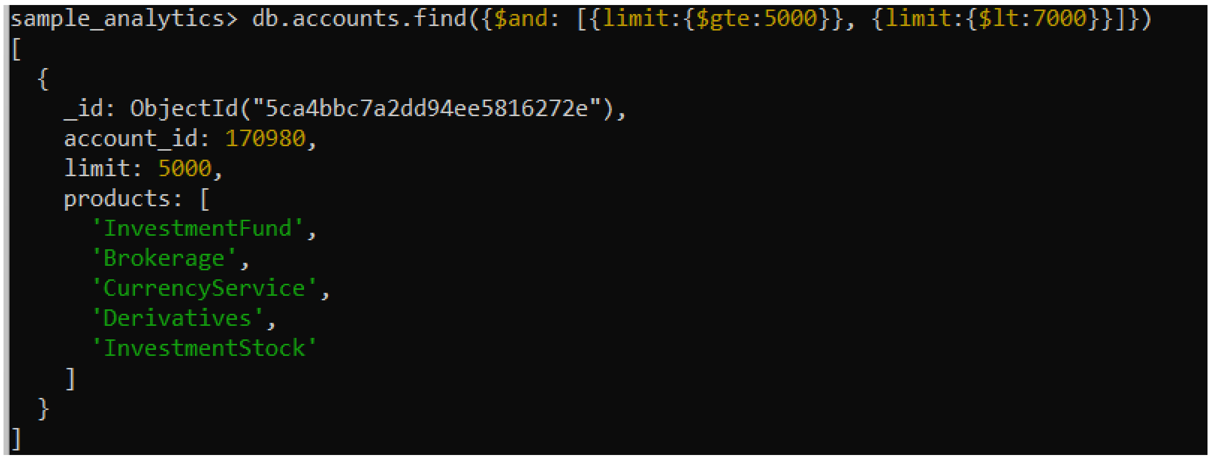
The syntax for this operator is:

```
{ $and: [ { <expression1> }, { <expression2> } , ... ,
              { <expressionN> } ] }
```

Consider that from the `accounts` collection, the user wants to view only the documents where the `limit` is greater than or equal to `5000` and less than `7000`. To do so, the user can execute a query that uses the `$and` operator as:

```
db.accounts.find({$and: [{limit:{$gte:5000}},
{limit:{$lt:7000}}]})
```

Figure 3.5 shows the output of this query.



```
sample_analytics> db.accounts.find({$and: [{limit:{$gte:5000}}, {limit:{$lt:7000}}]})
[
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816272e"),
    account_id: 170980,
    limit: 5000,
    products: [
      'InvestmentFund',
      'Brokerage',
      'CurrencyService',
      'Derivatives',
      'InvestmentStock'
    ]
  }
]
```

Figure 3.5: Output of the Query with `$and` Operator

`$or` Operator

The `$or` operator performs a logical OR operation. This operator lists documents that satisfy at least one of the expressions in the specified array.

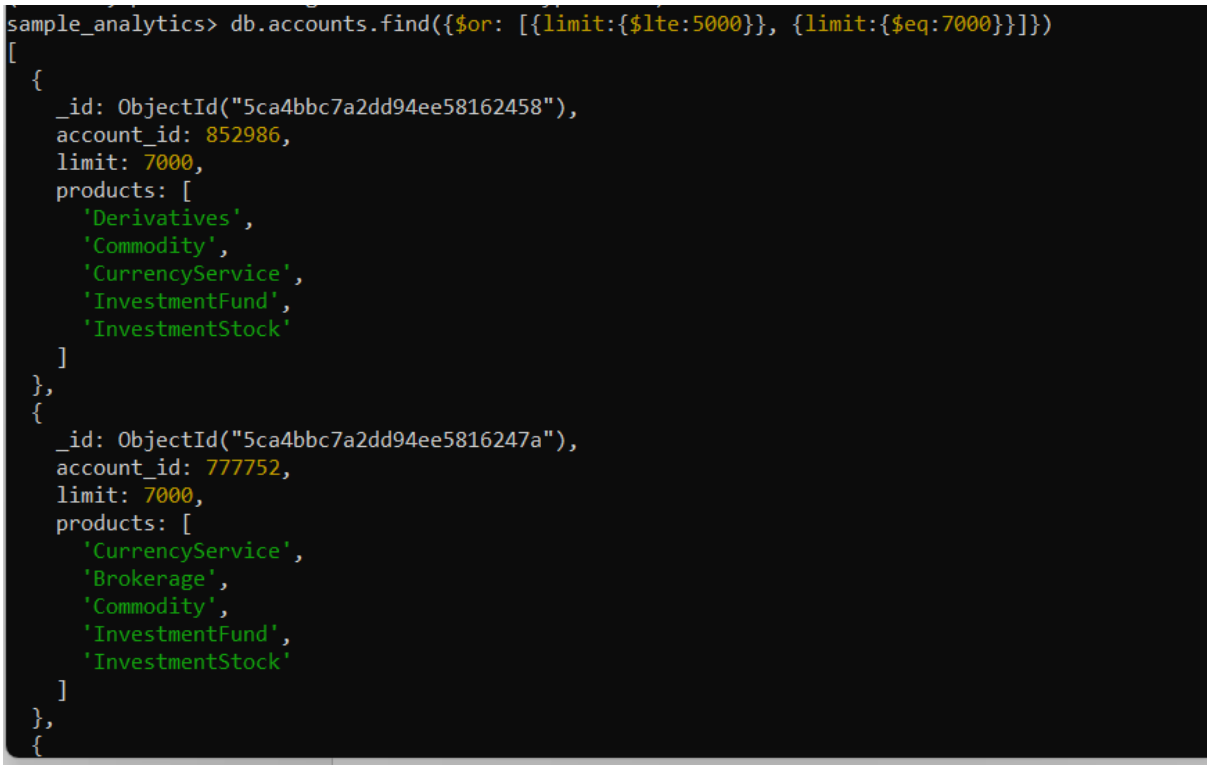
The syntax for this operator is:

```
{ $or: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }
```

Consider that from the `account` collection, the user wants to view all the documents where the `limit` is less than or equal to `5000` or equal to `7000`. To do, the user can execute a query that uses the `$or` operator as:

```
db.accounts.find({$or: [{limit:{$lte:5000}},
{limit:{$eq:7000}}]})
```

Figure 3.6 shows the output of the query.



```
sample_analytics> db.accounts.find({$or: [{limit:{$lte:5000}}, {limit:{$eq:7000}}]})
[
  {
    _id: ObjectId("5ca4bbc7a2dd94ee58162458"),
    account_id: 852986,
    limit: 7000,
    products: [
      'Derivatives',
      'Commodity',
      'CurrencyService',
      'InvestmentFund',
      'InvestmentStock'
    ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816247a"),
    account_id: 777752,
    limit: 7000,
    products: [
      'CurrencyService',
      'Brokerage',
      'Commodity',
      'InvestmentFund',
      'InvestmentStock'
    ]
  },
  {
```

Figure 3.6: Output of the Query with `$or` Operator

`$nor` Operator

This operator performs a logical NOR operation. It lists the documents that do not match any of the expressions in the specified array of expressions.

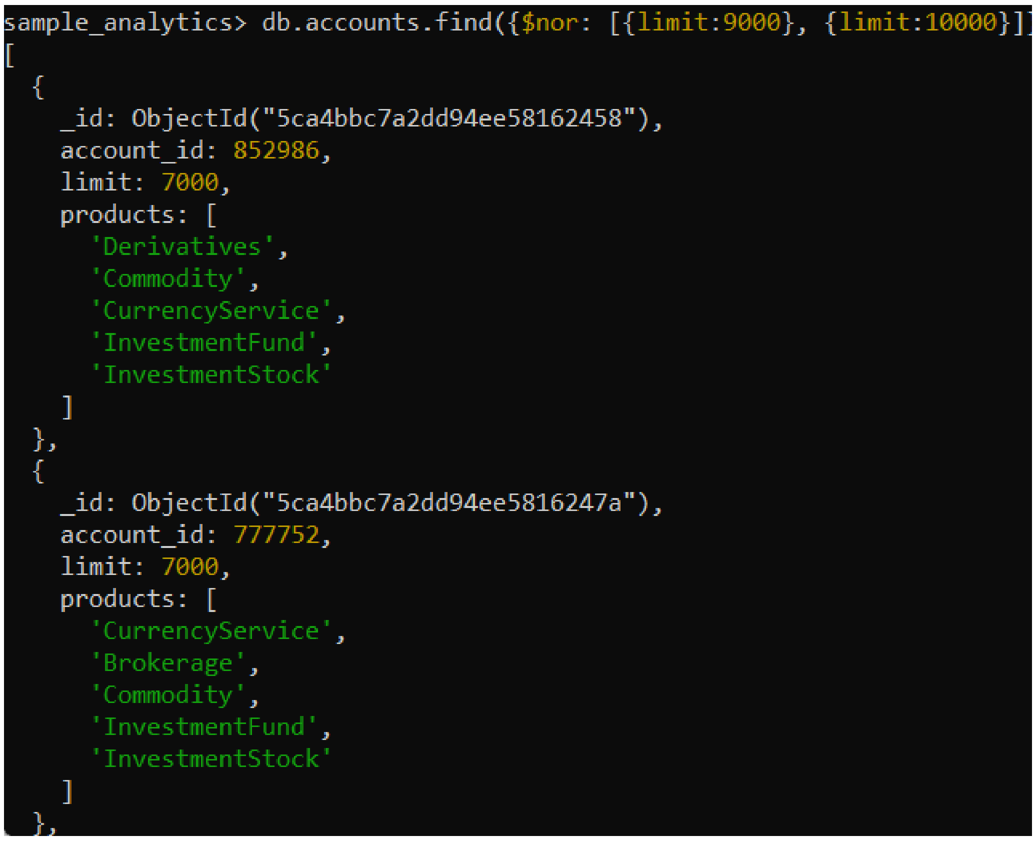
The syntax for this operator is:

```
{ $nor: [ { <expression1> }, { <expression2> }, ... { <expressionN> } ] }
```

For example, consider that from the `accounts` collection, the user wants to view all the documents where the `limit` is neither `9000` nor `10000`. To do so, the users can execute a query that uses the `$nor` operator as:

```
db.accounts.find({$nor: [{limit:9000}, {limit:10000}]})
```

Figure 3.7 shows the output of this query.



```
sample_analytics> db.accounts.find({$nor: [{limit:9000}, {limit:10000}]}
[
  {
    _id: ObjectId("5ca4bbc7a2dd94ee58162458"),
    account_id: 852986,
    limit: 7000,
    products: [
      'Derivatives',
      'Commodity',
      'CurrencyService',
      'InvestmentFund',
      'InvestmentStock'
    ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816247a"),
    account_id: 777752,
    limit: 7000,
    products: [
      'CurrencyService',
      'Brokerage',
      'Commodity',
      'InvestmentFund',
      'InvestmentStock'
    ]
  },
```

Figure 3.7: Output of the Query with `$nor` Operator

`$not` Operator

This operator performs a logical NOT operation. It lists the documents that do not match the specified expression.
The syntax for this operator is:

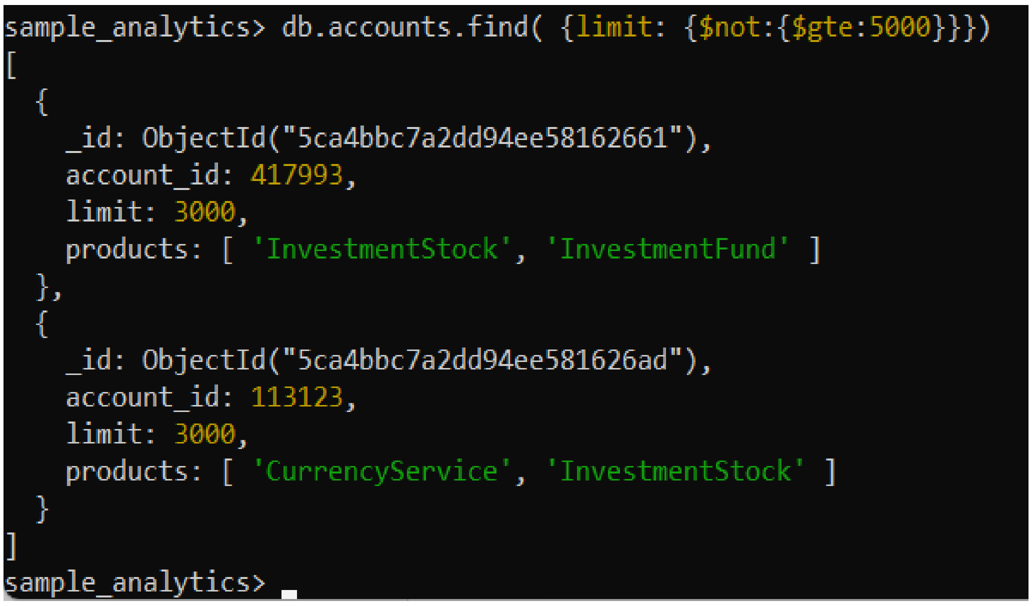
```
{ field: { $not: { <operator-expression> } } }
```

For example, consider that from the `accounts` collection, the user wants to view all the documents where `limit` is not greater than or equal to `5000`.

To do so, the user can execute a query that uses the `$not` operator as:

```
db.accounts.find( {limit: {$not:{$gte:5000}}})
```

Figure 3.8 shows the output of this query.



```
sample_analytics> db.accounts.find( {limit: {$not:{$gte:5000}}})
[
  {
    _id: ObjectId("5ca4bbc7a2dd94ee58162661"),
    account_id: 417993,
    limit: 3000,
    products: [ 'InvestmentStock', 'InvestmentFund' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee581626ad"),
    account_id: 113123,
    limit: 3000,
    products: [ 'CurrencyService', 'InvestmentStock' ]
  }
]
sample_analytics> _
```

Figure 3.8: Output of the Query with `$not` Operator

3.2.3 Element Operators

Element operators are used to check if a particular field is available in a MongoDB document or if a particular field with a specified type is available in the document. These operators are used with either of the two values: true or false. If these operators are used with the true value, the query will return all the documents that include the specified field or the specified field with the specified datatype. If used with the false value, the query will return all the documents that do not include the specified field or the specified field with the specified datatype.

MongoDB supports two element operators:

- `$exists`
- `$type`

`$exists` Operator

The `$exists` operator checks if the specified field is present in a document. If it is used with the true value, `$exists` returns the documents that contain the specified field, including documents where the field value is `null`. If `$exists` is used with the false value, `$exists` returns only the documents that do not contain the specified field.

The syntax for this operator is:

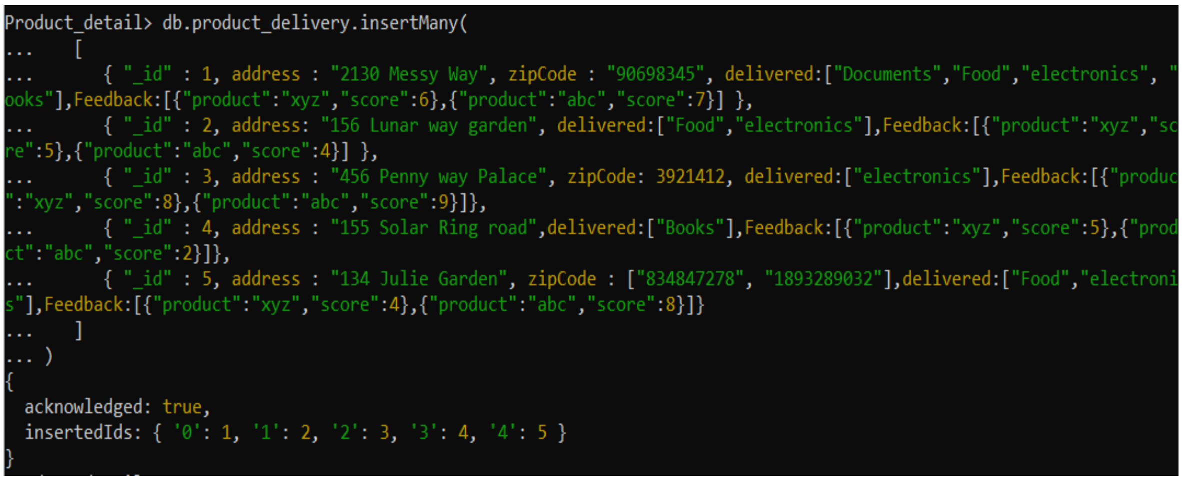
```
{ field: { $exists: <boolean> } }
```

In this syntax, `<Boolean>` takes two values: true or false.

Before understanding how this operator works, let us create a database named `Product_detail` and a collection named `product_delivery`. Let us also insert the five documents into the `product_delivery` collection using the query as:

```
db.product_delivery.insertMany(
   [  { "_id" : 1, address : "2130 Messy Way", zipCode :
"90698345", delivered:["Documents","Food","electronics",
"Books"],Feedback:[{"product":"xyz","score":6},{"product":"abc
","score":7}] },
      { "_id" : 2, address: "156 Lunar way garden",
delivered:["Food","electronics"],Feedback:[{"product":"xyz","s
core":5},{"product":"abc","score":4}] },
      { "_id" : 3, address : "456 Penny way Palace", zipCode:
3921412,
delivered:["electronics"],Feedback:[{"product":"xyz","score":8
},{"product":"abc","score":9}]},
      { "_id" : 4, address : "155 Solar Ring
road",delivered:["Books"],Feedback:[{"product":"xyz","score":5
},{"product":"abc","score":2}]},
      { "_id" : 5, address : "134 Julie Garden", zipCode :
["834847278",
"1893289032"],delivered:["Food","electronics"],Feedback:[{"pro
duct":"xyz","score":4},{"product":"abc","score":8}]}
   ])
```

Figure 3.9 shows the output of this query.



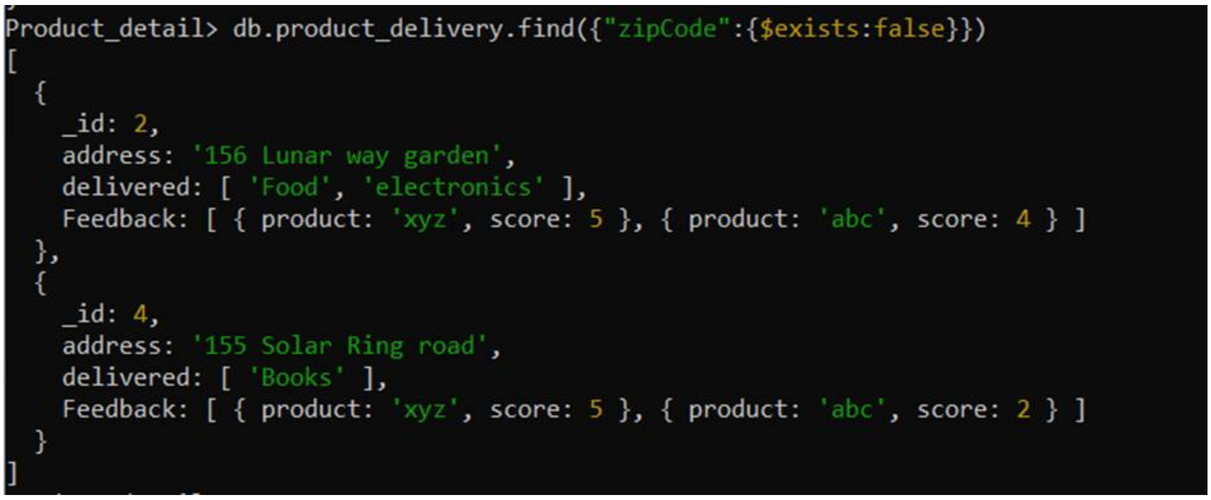
```
Product_detail> db.product_delivery.insertMany(
...     [
...         { "_id" : 1, address : "2130 Messy Way", zipCode : "90698345", delivered:["Documents","Food","electronics", "
ooks"],Feedback:[{"product":"xyz","score":6},{"product":"abc","score":7}] },
...         { "_id" : 2, address: "156 Lunar way garden", delivered:["Food","electronics"],Feedback:[{"product":"xyz","sc
re":5},{"product":"abc","score":4}] },
...         { "_id" : 3, address : "456 Penny way Palace", zipCode: 3921412, delivered:["electronics"],Feedback:[{"produc
":"xyz","score":8},{"product":"abc","score":9}]},
...         { "_id" : 4, address : "155 Solar Ring road",delivered:["Books"],Feedback:[{"product":"xyz","score":5},{"prod
ct":"abc","score":2}]},
...         { "_id" : 5, address : "134 Julie Garden", zipCode : ["834847278", "1893289032"],delivered:["Food","electroni
s"],Feedback:[{"product":"xyz","score":4},{"product":"abc","score":8}]}
...     ]
... )
{
  acknowledged: true,
  insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4, '4': 5 }
}
```

Figure 3.9: Output of the `insertMany` Query

Now, consider that from the `product_delivery` collection, the user wants to view documents that do not have a zip code. To do, so, the user can execute a query that uses the `$exists` operator as:

```
db.product_delivery.find({"zipCode":{$exists:false}})
```

Figure 3.10 shows the output of this query.



```
Product_detail> db.product_delivery.find({"zipCode":{$exists:false}})
[
  {
    _id: 2,
    address: '156 Lunar way garden',
    delivered: [ 'Food', 'electronics' ],
    Feedback: [ { product: 'xyz', score: 5 }, { product: 'abc', score: 4 } ]
  },
  {
    _id: 4,
    address: '155 Solar Ring road',
    delivered: [ 'Books' ],
    Feedback: [ { product: 'xyz', score: 5 }, { product: 'abc', score: 2 } ]
  }
]
```

Figure 3.10: Output of the Query with `$exists` Operator

`$type` Operator

This operator returns the documents where the datatype of the specified field in the document matches the specified datatype. The `$type` operator is quite useful when dealing with highly unstructured data where the collection is complex and the prediction of the datatypes of any field is difficult.

The syntax for this operator is:

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

The `$type` operator accepts aliases and numbers in place of the datatype. Table 3.2 lists the datatype and its corresponding number and alias.

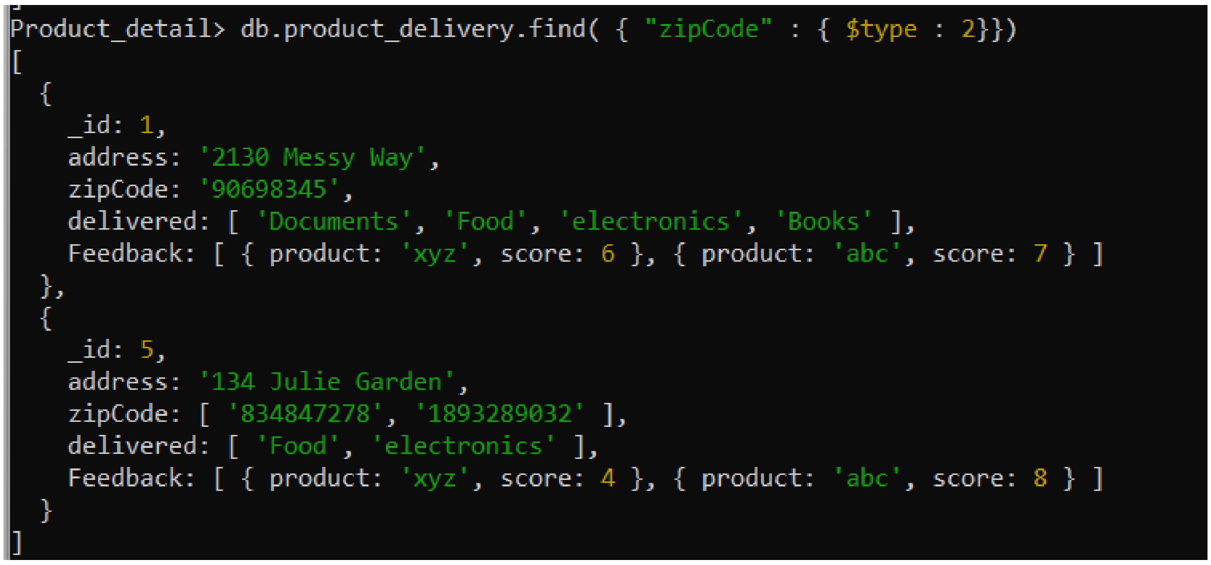
Datatype	Number	Alias
Double	1	"double"
String	2	"string"
Object	3	"object"
Array	4	"array"
Binary data	5	"binData"
ObjectId	7	"objectId"
Boolean	8	"bool"
Date	9	"date"
Null	10	"null"
Regular Expression	11	"regex"
JavaScript	13	"javascript"
32-bit integer	16	"int"
Timestamp	17	"timestamp"
64-bit integer	18	"long"
Decimal128	19	"decimal"
Min key	-1	"minKey"
Max key	127	"maxKey"

Table 3.2: Datatypes and their Corresponding Numbers and Aliases

Now, consider that from the `product_delivery` collection, the user wants to view the `zipCode` fields which have the string type. To do so, the user can execute a query that uses the `$type` operator as:

```
db.product_delivery.find( { "zipCode" : { $type : 2}})
```

Figure 3.11 shows the output of the query.



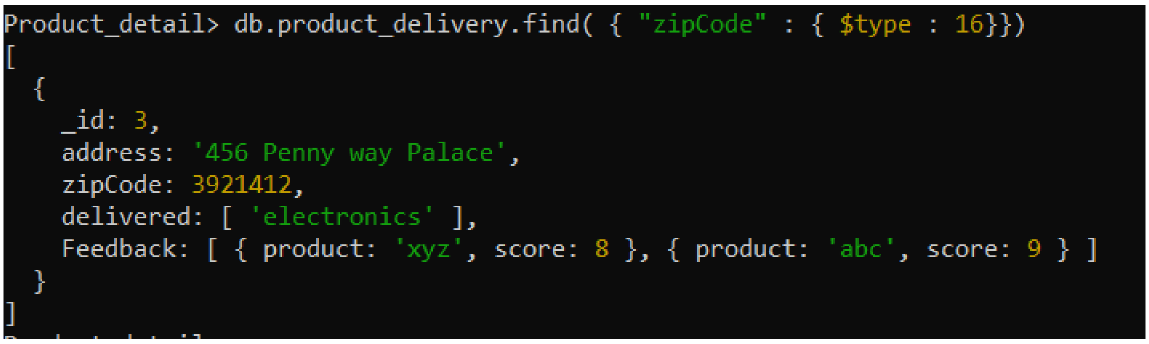
```
Product_detail> db.product_delivery.find( { "zipCode" : { $type : 2}})
[
  {
    _id: 1,
    address: '2130 Messy Way',
    zipCode: '90698345',
    delivered: [ 'Documents', 'Food', 'electronics', 'Books' ],
    Feedback: [ { product: 'xyz', score: 6 }, { product: 'abc', score: 7 } ]
  },
  {
    _id: 5,
    address: '134 Julie Garden',
    zipCode: [ '834847278', '1893289032' ],
    delivered: [ 'Food', 'electronics' ],
    Feedback: [ { product: 'xyz', score: 4 }, { product: 'abc', score: 8 } ]
  }
]
```

Figure 3.11: Output of the Query with `$type` Operator

Now, consider that from the `product_delivery` collection, the user wants to view the `zipCode` fields which are of the `string` type. To do so, the user can execute a query that uses the `$type` operator as:

```
db.product_delivery.find( { "zipCode" : { $type : 16}})
```

Figure 3.12 shows the output of the query.



```
Product_detail> db.product_delivery.find( { "zipCode" : { $type : 16}})
[
  {
    _id: 3,
    address: '456 Penny way Palace',
    zipCode: 3921412,
    delivered: [ 'electronics' ],
    Feedback: [ { product: 'xyz', score: 8 }, { product: 'abc', score: 9 } ]
  }
]
```

Figure 3.12: Output of the Query with `$type` Operator

3.2.4 Array Operators

Array operators are used when the user wants to filter data by specifying conditions for array fields.

There are three array operators:

`$all`	`$elemMatch`	`$size`
Used to retrieve, update, or delete documents that have an array field that has all the elements specified in the query	Used to retrieve, update, or delete documents that have an array field that has at least one of the elements specified in the query	Used to retrieve, update, or delete documents that have an array field with the same size as the size specified in the query

`$all` Operator

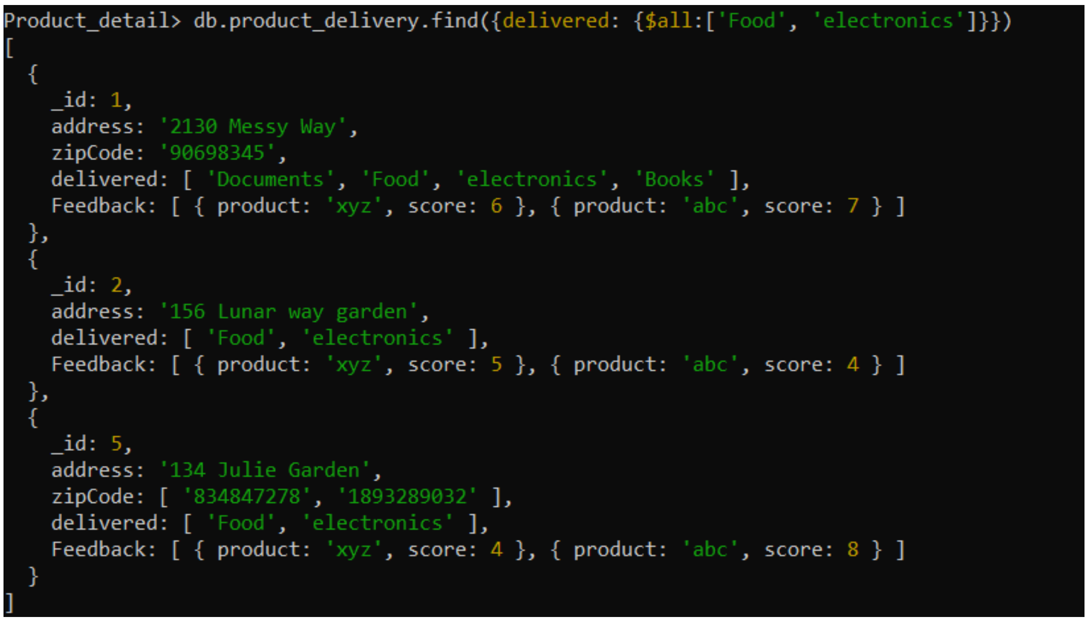
The syntax for this operator is:

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

For example, consider that from the `product_delivery` collection, the user wants to view the documents that include the value '`Food`' and '`electronics`' in the `delivered` field. To do so, the user can execute a query that uses the `$all` operator as:

```
db.product_delivery.find({delivered:
{$all:['Food', 'electronics']}})
```

Figure 3.13 shows the output of the query.



```
Product_detail> db.product_delivery.find({delivered: {$all:['Food', 'electronics']}})
[
  {
    _id: 1,
    address: '2130 Messy Way',
    zipCode: '90698345',
    delivered: [ 'Documents', 'Food', 'electronics', 'Books' ],
    Feedback: [ { product: 'xyz', score: 6 }, { product: 'abc', score: 7 } ]
  },
  {
    _id: 2,
    address: '156 Lunar way garden',
    delivered: [ 'Food', 'electronics' ],
    Feedback: [ { product: 'xyz', score: 5 }, { product: 'abc', score: 4 } ]
  },
  {
    _id: 5,
    address: '134 Julie Garden',
    zipCode: [ '834847278', '1893289032' ],
    delivered: [ 'Food', 'electronics' ],
    Feedback: [ { product: 'xyz', score: 4 }, { product: 'abc', score: 8 } ]
  }
]
```

Figure 3.13: Output of the Query with `$all` Operator

`$elemMatch` Operator

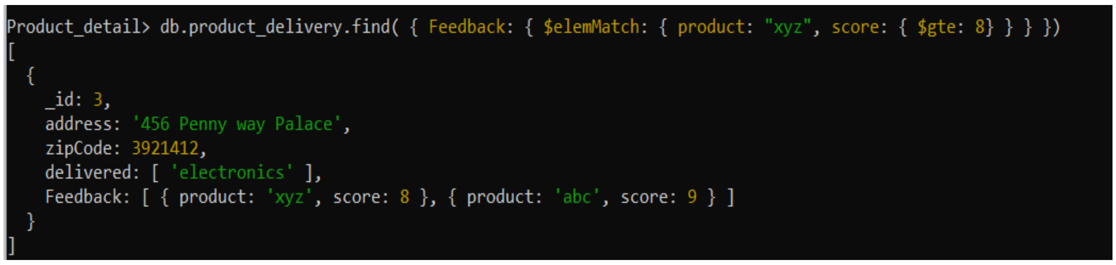
The syntax for this operator is:

```
{ <field>: { $elemMatch: { <query1>, <query2>, ... } } }
```

For example, consider that from the `product_delivery` collection, the user wants to view the documents that have the value `product: "xyz"` or `score greater than 8` in the `Feedback` field. To do so, the user can execute a query that uses the `$elemMatch` operator as:

```
db.product_delivery.find( { Feedback: { $elemMatch:
{ product: "xyz", score: { $gte: 8} } } })
```

Figure 3.14 shows the output of the query.



```
Product_detail> db.product_delivery.find( { Feedback: { $elemMatch: { product: "xyz", score: { $gte: 8} } } })
[
  {
    _id: 3,
    address: '456 Penny way Palace',
    zipCode: 3921412,
    delivered: [ 'electronics' ],
    Feedback: [ { product: 'xyz', score: 8 }, { product: 'abc', score: 9 } ]
  }
]
```

Figure 3.14: Output of the Query with `$elemMatch` Operator

`$size` Operator

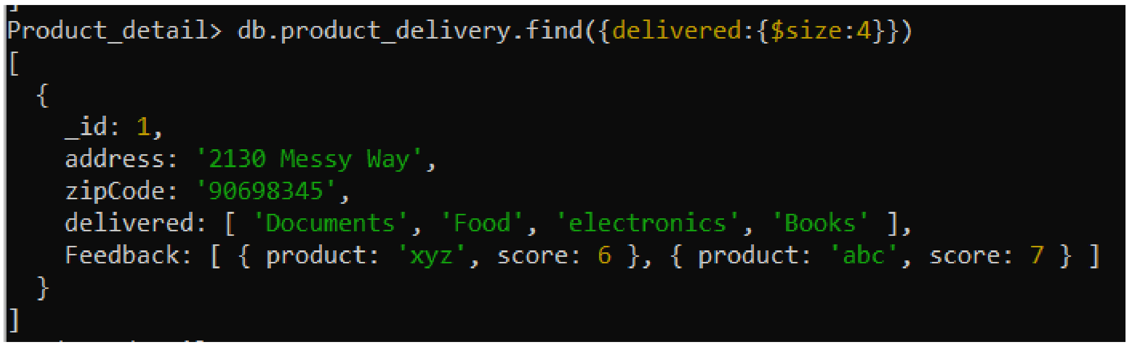
The syntax for this operator is:

```
{ <field>: { $size: n} }
```

For example, consider that from the `product_delivery` collection, the user wants to view the documents that have four products in the `delivered` array. To do so, the user can execute a query that uses the `$size` operator as:

```
db.product_delivery.find({delivered:{$size:4}})
```

Figure 3.15 shows the output of the query.



```
Product_detail> db.product_delivery.find({delivered:{$size:4}})
[
  {
    _id: 1,
    address: '2130 Messy Way',
    zipCode: '90698345',
    delivered: [ 'Documents', 'Food', 'electronics', 'Books' ],
    Feedback: [ { product: 'xyz', score: 6 }, { product: 'abc', score: 7 } ]
  }
]
```

Figure 3.15: Output of the Query with `$size` Operator

3.3 Projection Operators

The `find()` operator in MongoDB returns all the data in the collection that matches the specified condition. What if the user wants to view specific fields of data from the collection? This is where project operators come in handy.

Projection operators can be used to specify the fields or elements to display in the query result based on the condition specified in the query.

Four projection operators in MongoDB are:

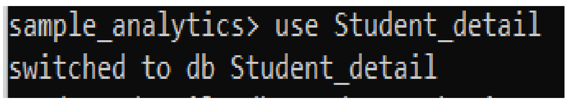
`$`	Used to return the first element in an array that matches the condition specified in the query or the first element in the array, if no query condition is specified
`$elemMatch`	Used to return the first element in an array that matches the condition specified by the `$elemMatch` operator
`$meta`	Used to retrieve the metadata related to a document in a collection
`$slice`	Used to specify the number of elements that must be returned by the query

$ Operator

The syntax for this operator is:

```
db.collection.find( { <array>: <condition> ... },
           { "<array>.$": 1 } )
```

To understand how this operator can be used, let us create a database named `Student_detail` and switch to that database as shown in Figure 3.16.



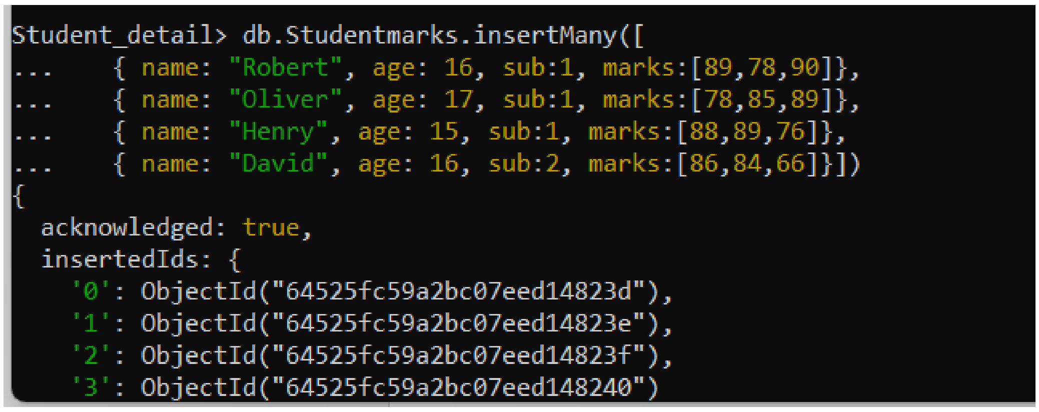
```
sample_analytics> use Student_detail
switched to db Student_detail
```

Figure 3.16: Switching to the `Student_detail` Database

Now, let us create a collection named `Studentmarks` in the database. Next, let us insert documents into the `Studentmarks` collection using the query as:

```
db.Studentmarks.insertMany([
   { name: "Robert", age: 16, sub:1, marks:[89,78,90]},
   { name: "Oliver", age: 17, sub:1, marks:[78,85,89]},
   { name: "Henry", age: 15, sub:1, marks:[88,89,76]},
   { name: "David", age: 16, sub:2, marks:[86,84,66]}])
```

Figure 3.17 shows the output of the query.



```
Student_detail> db.Studentmarks.insertMany([
...     { name: "Robert", age: 16, sub:1, marks:[89,78,90]},
...     { name: "Oliver", age: 17, sub:1, marks:[78,85,89]},
...     { name: "Henry", age: 15, sub:1, marks:[88,89,76]},
...     { name: "David", age: 16, sub:2, marks:[86,84,66]}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("64525fc59a2bc07eed14823d"),
    '1': ObjectId("64525fc59a2bc07eed14823e"),
    '2': ObjectId("64525fc59a2bc07eed14823f"),
    '3': ObjectId("64525fc59a2bc07eed148240")
```

Figure 3.17: Output of the `insertMany` Query

Now consider that the user wants to view the first element that is greater than or equal to `85` for the `marks` field, where the value of `sub` is `1`. The user also wants to view the `name` field and hide the `_id` field. To do so, the user can execute a query that uses the `$` projection operator as:

```
db.Studentmarks.find( { sub: 1, marks:
{ $gte: 85 }
},{_id:0,name:1,"marks.$": 1 } )
```

In this query, `"marks.$": 1` ensures that the `marks` column is displayed and `_id:0` ensures that the `ID` column is hidden in the query result.

Figure 3.18 shows the output of this query.

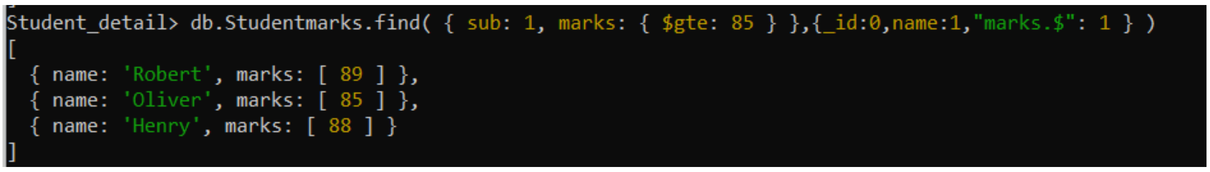

```
Student_detail> db.Studentmarks.find( { sub: 1, marks: { $gte: 85 } },{_id:0,name:1,"marks.$": 1 } )
[
  { name: 'Robert', marks: [ 89 ] },
  { name: 'Oliver', marks: [ 85 ] },
  { name: 'Henry', marks: [ 88 ] }
]
```

Figure 3.18: Output of the Query with $ Operator

`$slice` Operator

The syntax for this operator is:

```
db.collection.find( <query>,
{ <arrayField>: { $slice: <number> } } );
```

For example, consider that the user wants to view the first two elements of the `marks` field from the document where the `name` field has the value "`Robert`". To do so, the user can execute a query that uses the `$slide` operator as:

```
db.Studentmarks.find({ "name":"Robert" },
{ "marks": { $slice: 2}})
```

Figure 3.19 shows the output of this query.

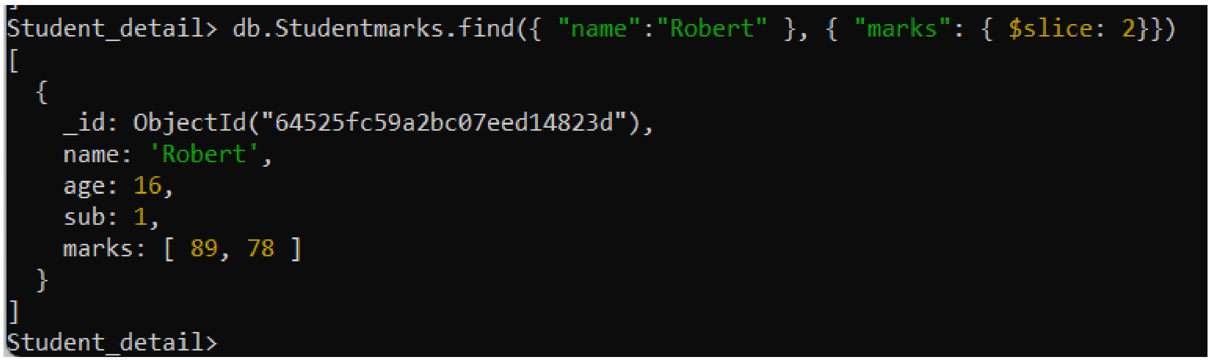

```
Student_detail> db.Studentmarks.find({ "name":"Robert" }, { "marks": { $slice: 2}})
[
  {
    _id: ObjectId("64525fc59a2bc07eed14823d"),
    name: 'Robert',
    age: 16,
    sub: 1,
    marks: [ 89, 78 ]
  }
]
Student_detail>
```

Figure 3.19: Output of the Query with `$slice` Operator

3.4 Update Operators

Update operators help users modify documents in a collection. There are separate update operators for modifying fields and arrays.

3.4.1 Field Update Operators

There are nine operators in MongoDB that allow updating the values of fields in the documents:

`$currentDate`
- Used when the value in a field must be set to the current date

`$inc`
- Used when the values in a field must be incremented by a specific number

`$min`
- Used when the values in a field must be updated only if they are more than a specific value

`$max`
- Used when the values in a field must be updated only if they are less than a specific value

`$mul`
- Used when the values in a field must be multiplied by a specific value

`$rename`
- Used when a field must be renamed

`$set`
- Used when the value of a field in the document must be set

`$unset`
- Used when a specific field must be removed from the document

`$setOnInsert`
- Used when the value of a field must be set only if a new document is inserted as a result of the update operation

`$currentDate` Operator

The syntax for this operator is:

```
{ $currentDate: { <field1>: <typeSpecification1>, … } }
```

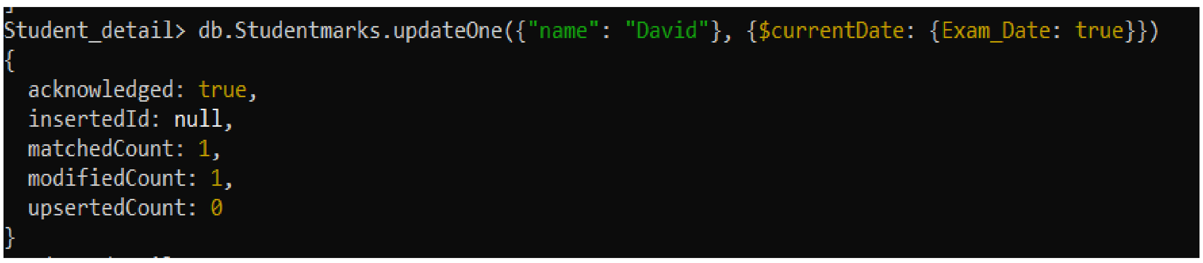
In this syntax, `<typeSpecification>` can be set to `true` if the user wants to input the current date in the `Date` type. If the user wants to insert the current date as a timestamp or a date type, then the user can set `<typeSpecification>` to `timestamp` or `date`, respectively. Note that these values must be used in lower case.

If the field specified in the query does not exist in the document, then this operator will add the field to the document.

For example, consider that the user wants to set the `Exam_Date` field in the `Studentmarks` collection to the current date as a `Date`. To do so, the user can execute and update query that uses the `$currentDate` operator as:

```
db.Studentmarks.updateOne({"name": "David"},
{$currentDate: {Exam_Date: true}})
```

Figure 3.20 shows the output of this query.



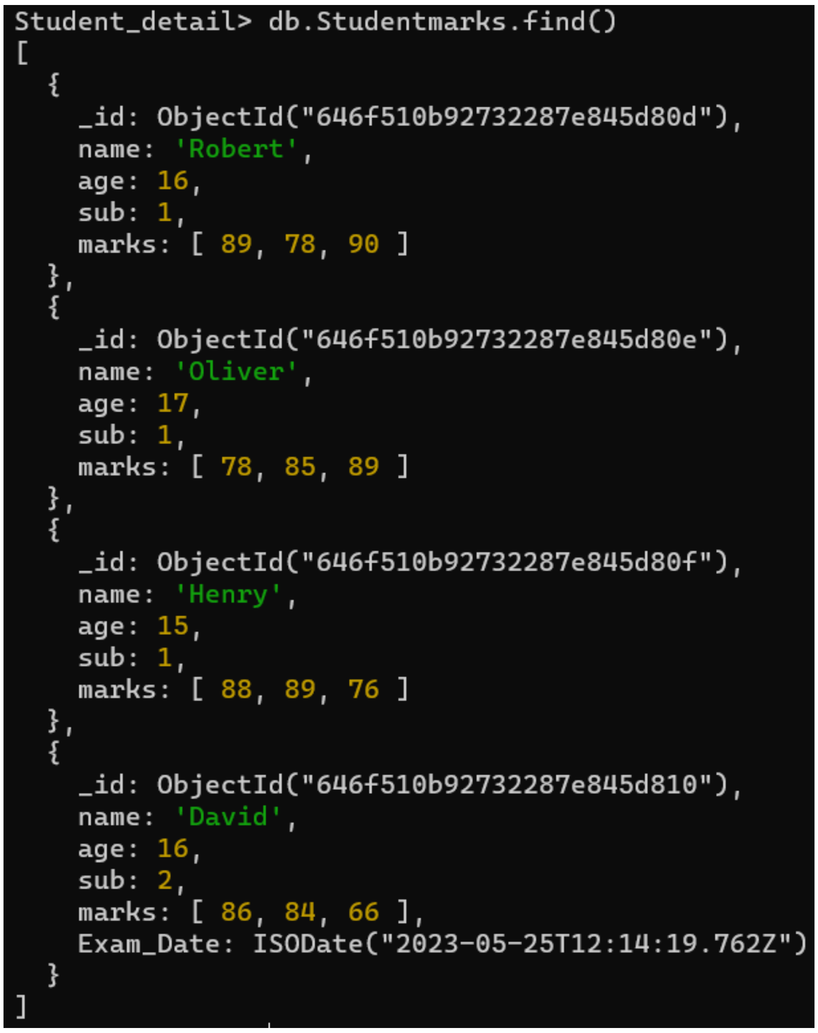
```
Student_detail> db.Studentmarks.updateOne({"name": "David"}, {$currentDate: {Exam_Date: true}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Figure 3.20: Output of the Query with `$currentDate` Operator

To view the documents in the updated `Studentmarks` collection, the user can execute the query as:

```
db.Studentmarks.find()
```

Figure 3.21 shows the output of this query.



```
Student_detail> db.Studentmarks.find()
[
  {
    _id: ObjectId("646f510b92732287e845d80d"),
    name: 'Robert',
    age: 16,
    sub: 1,
    marks: [ 89, 78, 90 ]
  },
  {
    _id: ObjectId("646f510b92732287e845d80e"),
    name: 'Oliver',
    age: 17,
    sub: 1,
    marks: [ 78, 85, 89 ]
  },
  {
    _id: ObjectId("646f510b92732287e845d80f"),
    name: 'Henry',
    age: 15,
    sub: 1,
    marks: [ 88, 89, 76 ]
  },
  {
    _id: ObjectId("646f510b92732287e845d810"),
    name: 'David',
    age: 16,
    sub: 2,
    marks: [ 86, 84, 66 ],
    Exam_Date: ISODate("2023-05-25T12:14:19.762Z")
  }
]
```

Figure 3.21: Updated Studentmarks Collection

`$inc` Operator

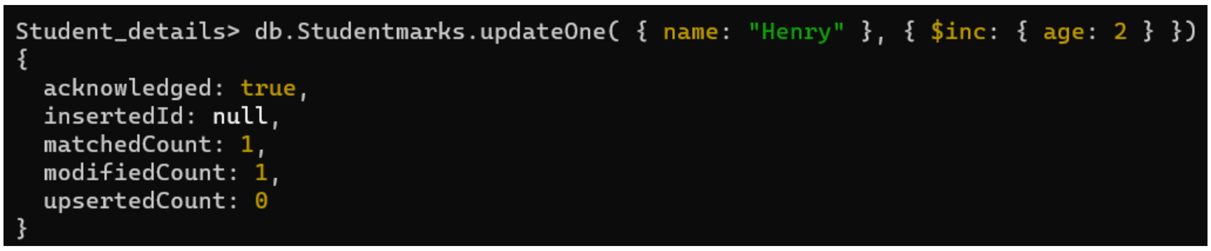
The syntax for this operator is:

```
{ $inc: { <field1>: <amount1>, <field2>:
<amount2>, ... } }
```

For example, consider that the user wants to increment the value in the `age` field by `2` where the `name` is "`Henry`". To do so, the user can execute a query that uses the `$inc` operator as:

```
db.Studentmarks.updateOne( { name: "Henry" },
{ $inc: { age: 2 } })
```

Figure 3.22 shows the output of this query.



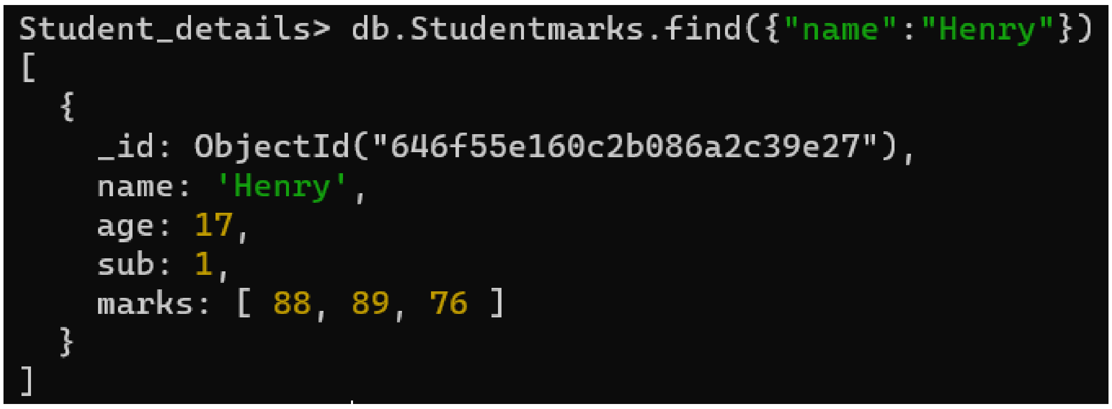
```
Student_details> db.Studentmarks.updateOne( { name: "Henry" }, { $inc: { age: 2 } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Figure 3.22: Output of the Query with `$inc` Operator

To view the updated document, the user can execute the query as:

```
db.Studentmarks.find({"name":"Henry"})
```

Figure 3.23 shows the output of this query.



```
Student_details> db.Studentmarks.find({"name":"Henry"})
[
  {
    _id: ObjectId("646f55e160c2b086a2c39e27"),
    name: 'Henry',
    age: 17,
    sub: 1,
    marks: [ 88, 89, 76 ]
  }
]
```

Figure 3.23: Updated Document

`$set` Operator

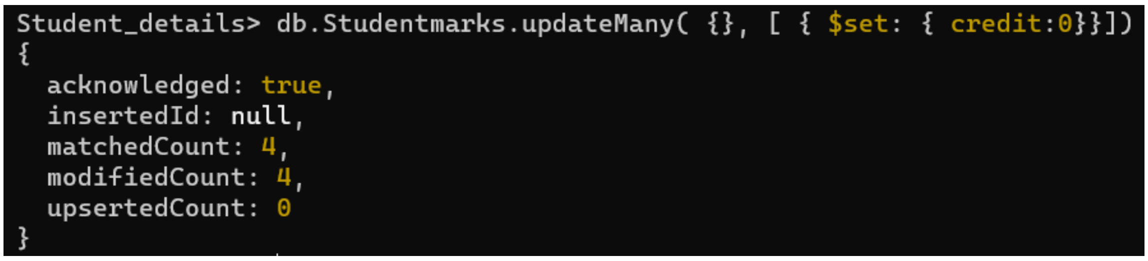
The syntax for this operator is:

```
{ $set: { <field1>: <value1>, ... } }
```

For example, consider that the user wants to add a new field named `credit` to the `Studentmarks` collection. To do this, the user can execute an update query that uses the `$set` operator as:

```
db.Studentmarks.updateMany( {}, [ { $set: {
credit:0}}])
```

Figure 3.24 shows the output of this query.



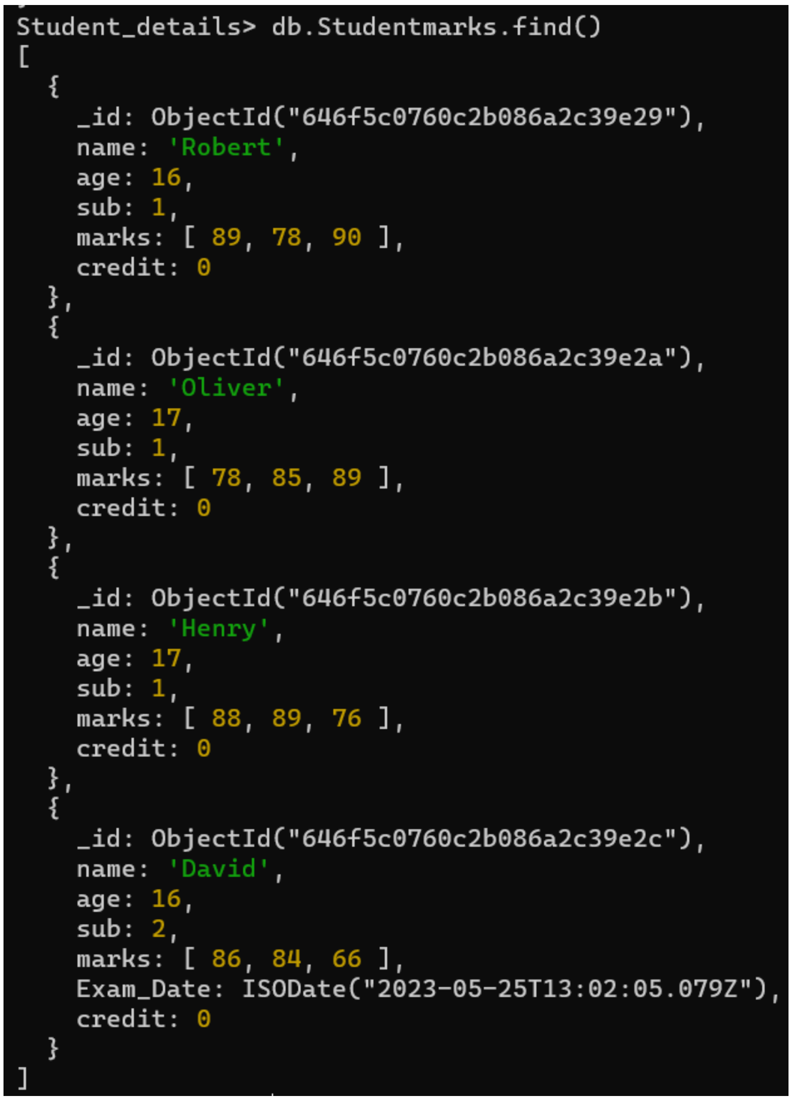
```
Student_details> db.Studentmarks.updateMany( {}, [ { $set: { credit:0}}])
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
```

Figure 3.24: Output of the Query with `$set` Operator

To view the updated document, the user can execute the query as:

```
db.Studentmarks.find()
```

Figure 3.25 shows the output of this query.



```
Student_details> db.Studentmarks.find()
[
  {
    _id: ObjectId("646f5c0760c2b086a2c39e29"),
    name: 'Robert',
    age: 16,
    sub: 1,
    marks: [ 89, 78, 90 ],
    credit: 0
  },
  {
    _id: ObjectId("646f5c0760c2b086a2c39e2a"),
    name: 'Oliver',
    age: 17,
    sub: 1,
    marks: [ 78, 85, 89 ],
    credit: 0
  },
  {
    _id: ObjectId("646f5c0760c2b086a2c39e2b"),
    name: 'Henry',
    age: 17,
    sub: 1,
    marks: [ 88, 89, 76 ],
    credit: 0
  },
  {
    _id: ObjectId("646f5c0760c2b086a2c39e2c"),
    name: 'David',
    age: 16,
    sub: 2,
    marks: [ 86, 84, 66 ],
    Exam_Date: ISODate("2023-05-25T13:02:05.079Z"),
    credit: 0
  }
]
```

Figure 3.25: Updated Document

`$rename` Operator

The syntax for this operator is:

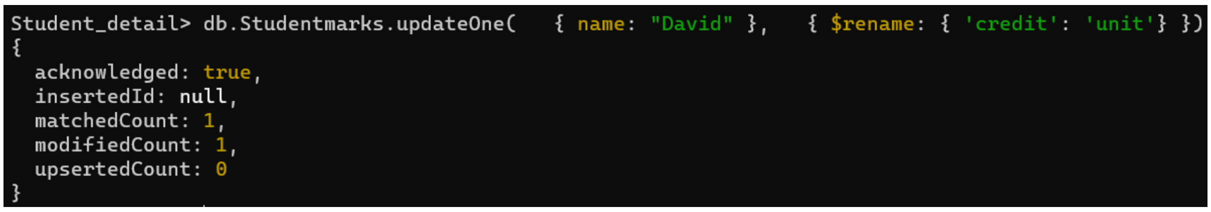
```
{$rename: { <field1>: <newName1>, <field2>:
        <newName2>, ... } }
```

The new field name must be different from the existing field name.

For example, consider that in the `Studentmarks` collection the user wants to change the name of `credit` field to `unit`. To do this, the user can execute an update query that uses the `$rename` operator as:

```
db.Studentmarks.updateOne(   { name: "David" },
{ $rename: { 'credit': 'unit'} })
```

Figure 3.26 shows the output of this query.



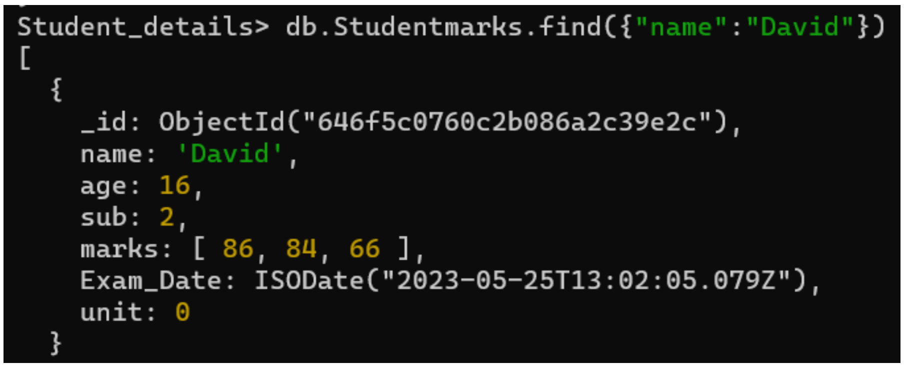
```
Student_detail> db.Studentmarks.updateOne(   { name: "David" },   { $rename: { 'credit': 'unit'} })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Figure 3.26: Output of the Query with `$rename` Operator

To view the updated document, the user can execute the query as:

```
db.Studentmarks.find({"name":"David"})
```

Figure 3.27 shows the output of this query.

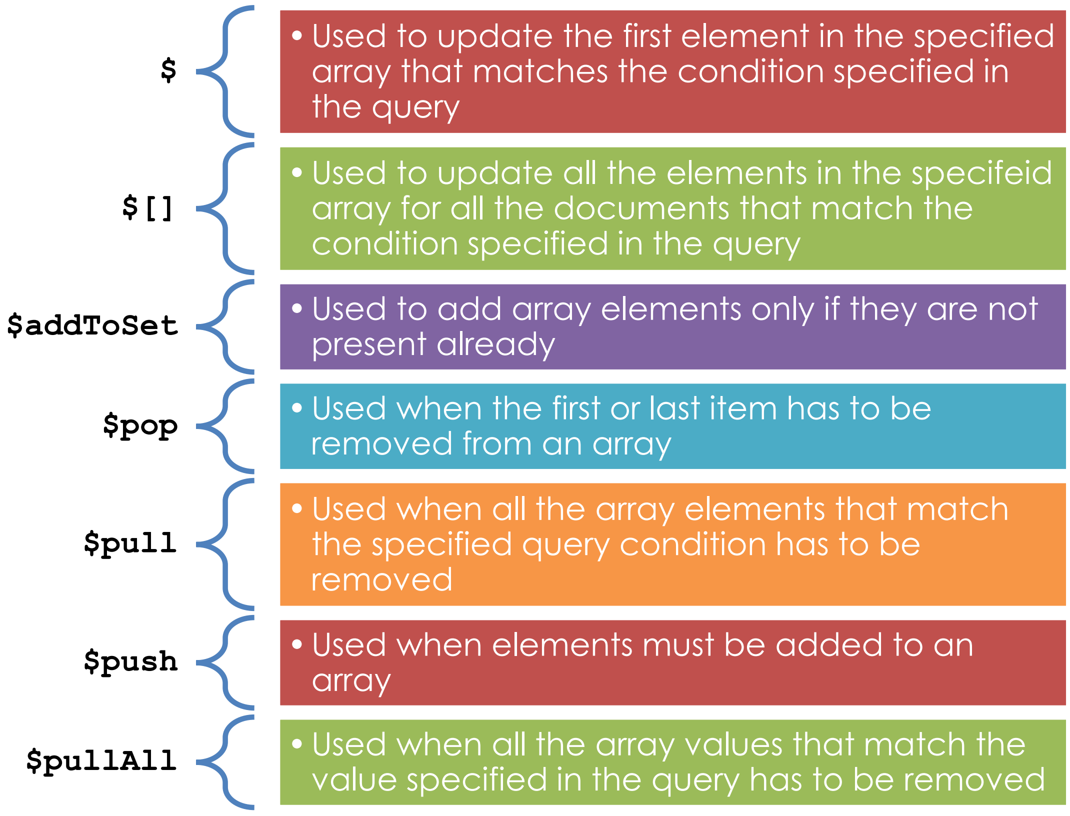


```
Student_details> db.Studentmarks.find({"name":"David"})
[
  {
    _id: ObjectId("646f5c0760c2b086a2c39e2c"),
    name: 'David',
    age: 16,
    sub: 2,
    marks: [ 86, 84, 66 ],
    Exam_Date: ISODate("2023-05-25T13:02:05.079Z"),
    unit: 0
  }
```

Figure 3.27: Updated Document

3.4.2 Array Update Operators

The array update operators supported by MongoDB are:



`$`	• Used to update the first element in the specified array that matches the condition specified in the query
`$[]`	• Used to update all the elements in the specifeid array for all the documents that match the condition specified in the query
`$addToSet`	• Used to add array elements only if they are not present already
`$pop`	• Used when the first or last item has to be removed from an array
`$pull`	• Used when all the array elements that match the specified query condition has to be removed
`$push`	• Used when elements must be added to an array
`$pullAll`	• Used when all the array values that match the value specified in the query has to be removed

$pop Operator

The syntax for this operator is:

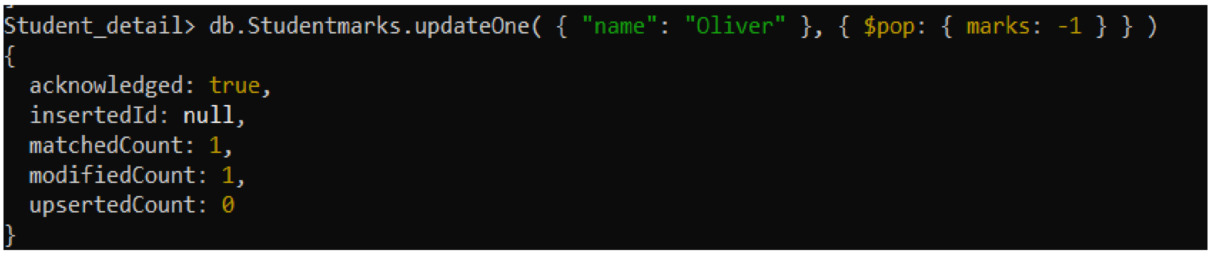
```
{ $pop: { <field>: <-1 | 1>, ... } }
```

In this syntax, `-1` is used to remove the first element and `1` is used to remove the last element in the array.

For example, consider that in the `Studentmarks` collection, the user wants to remove the first element from the `marks` array where `name` is "`Oliver`". To do so, the user can execute a query that uses the `$pop` operator as:

```
db.Studentmarks.updateOne( { "name": "Oliver" }, { $pop: {
marks: -1 } } )
```

Figure 3.28 shows the output of this query.



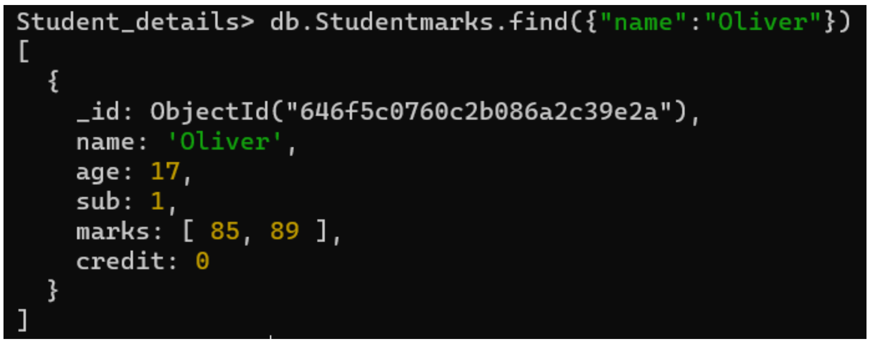
```
Student_detail> db.Studentmarks.updateOne( { "name": "Oliver" }, { $pop: { marks: -1 } } )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Figure 3.28: Output of Query with $pop Operator

To view the updated document, the user can execute the query as:

```
db.Studentmarks.find({"name":"Oliver"})
```

Figure 3.29 shows the output of this query.



```
Student_details> db.Studentmarks.find({"name":"Oliver"})
[
  {
    _id: ObjectId("646f5c0760c2b086a2c39e2a"),
    name: 'Oliver',
    age: 17,
    sub: 1,
    marks: [ 85, 89 ],
    credit: 0
  }
]
```

Figure 3.29: Updated Document

`$pull` Operator

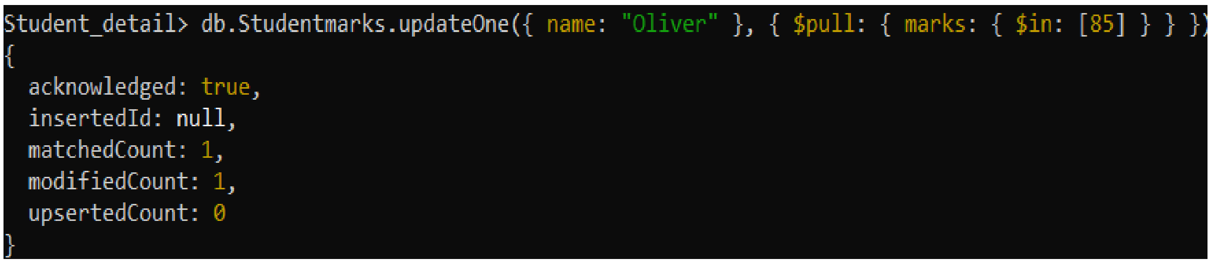
The syntax for this operator is:

```
{ $pull: { <field1>: <value|condition>,
  <field2>: <value|condition>, ... } }
```

Consider that in the `Studentmarks` collection, the user wants to remove the specific element from the `marks` array where `name` is "`Oliver`". To do so, the user can execute a query that uses the `$pull` operator as:

```
db.Studentmarks.updateOne({ name: "Oliver" },
{ $pull: { marks: { $in: [85] } } })
```

Figure 3.30 shows the output of this query.



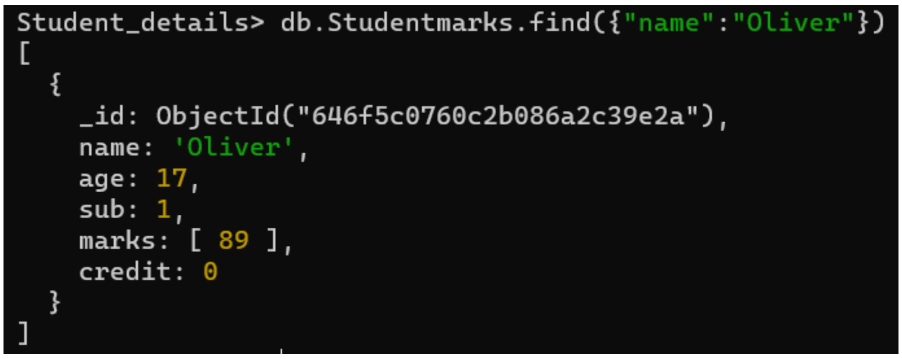
```
Student_detail> db.Studentmarks.updateOne({ name: "Oliver" }, { $pull: { marks: { $in: [85] } } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Figure 3.30: Output of Query with `$pull` Operator

To view the updated document, the user can execute the query as:

```
db.Studentmarks.find({"name":"Oliver"})
```

Figure 3.31 shows the output of this query.



```
Student_details> db.Studentmarks.find({"name":"Oliver"})
[
  {
    _id: ObjectId("646f5c0760c2b086a2c39e2a"),
    name: 'Oliver',
    age: 17,
    sub: 1,
    marks: [ 89 ],
    credit: 0
  }
]
```

Figure 3.31: Updated Document

`$push` Operator

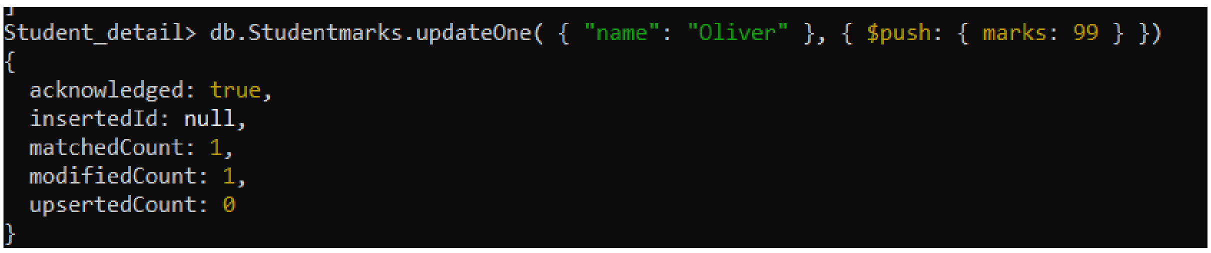
The syntax for this operator is:

```
{ $push: { <field1>: <value1>, ... } }
```

For example, consider that in the `Studentmarks` collection, the user wants to add a value to the `marks` array in the document where `name` is "`Oliver`". To do so, the user can execute a query that uses the `$push` operator as:

```
db.Studentmarks.updateOne( { "name":
"Oliver" }, { $push: { marks: 99 } })
```

Figure 3.32 shows the output of this query.



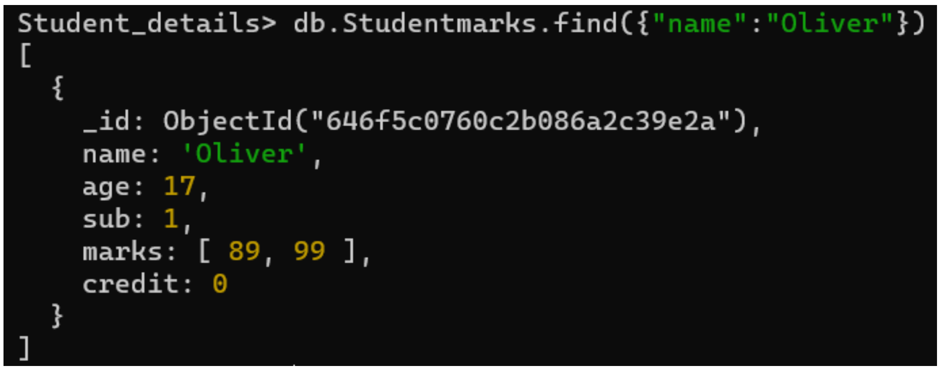
```
Student_detail> db.Studentmarks.updateOne( { "name": "Oliver" }, { $push: { marks: 99 } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Figure 3.32: Output of Query with `$push` Operator

To view the updated document, the user can execute the query as:

```
db.Studentmarks.find({"name":"Oliver"})
```

Figure 3.33 shows the output of this query.



```
Student_details> db.Studentmarks.find({"name":"Oliver"})
[
  {
    _id: ObjectId("646f5c0760c2b086a2c39e2a"),
    name: 'Oliver',
    age: 17,
    sub: 1,
    marks: [ 89, 99 ],
    credit: 0
  }
]
```

Figure 3.33: Updated Document

3.5 Other Operators

Three other operators that are commonly used in MongoDB are:

`$comment`

- Used to add comments to a query predictae to explain what the query is intended to do

`$rand`

- Used to generate a random floating number between 0 and 1

`$natural`

- Used to specify whether the documents must be iterated through in the natural order or the reverse order; takes value 1 for natural order and -1 for reverse order

`$comment` Operator

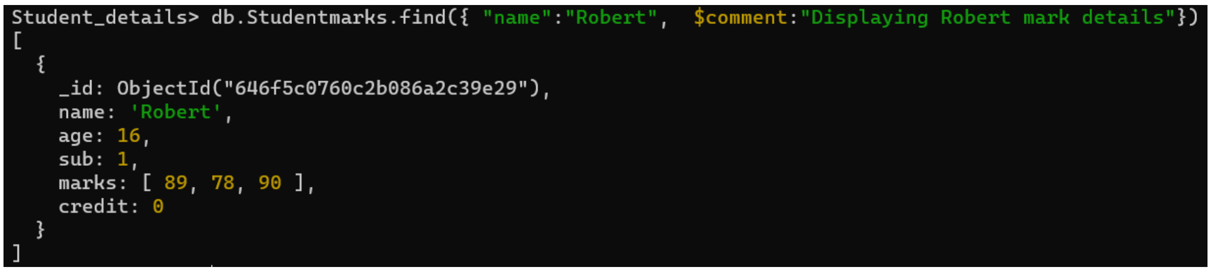
The syntax for this operator is:

```
db.collection.find( { <query>, $comment: <comment> } )
```

For example, consider that the user wants to add a comment for the document where the `name` is "`Robert`". To do so, the user can execute a query that uses the `$comment` operator as:

```
db.Studentmarks.find({ "name":"Robert",
$comment:"Displaying Robert mark details"})
```

Figure 3.34 shows the output of this query.



```
Student_details> db.Studentmarks.find({ "name":"Robert",  $comment:"Displaying Robert mark details"})
[
  {
    _id: ObjectId("646f5c0760c2b086a2c39e29"),
    name: 'Robert',
    age: 16,
    sub: 1,
    marks: [ 89, 78, 90 ],
    credit: 0
  }
]
```

Figure 3.34: Output of the Query with `$comment` Operator

The comment is not visible in Figure 3.34. This is because the comments are stored in the `system.profile` collection. This collection is automatically created in the MongoDB server if database profiling is enabled. To view the comment, user can use the `find()` query on the `system.profile` collection.

`$rand` Operator

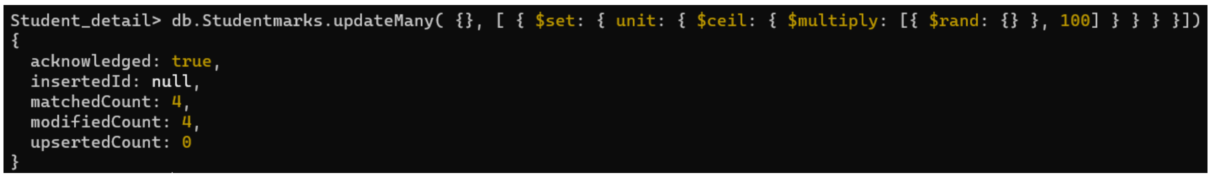
The syntax for this operator is:

```
{ $rand: {} }
```

For example, consider that in the `Studentmarks` collection, the user updates the field named `unit`. The value in this field will be automatically generated by the `$rand` operator. The value generated should be multiplied by `100` and rounded off to the highest integer. The query to achieve this objective is:

```
db.Studentmarks.updateMany( {}, [ { $set: { unit: {
$ceil: { $multiply: [{ $rand: {} }, 100] } } } }])
```

The empty update filter matches every document in the collection. Figure 3.35 shows the output of this query:



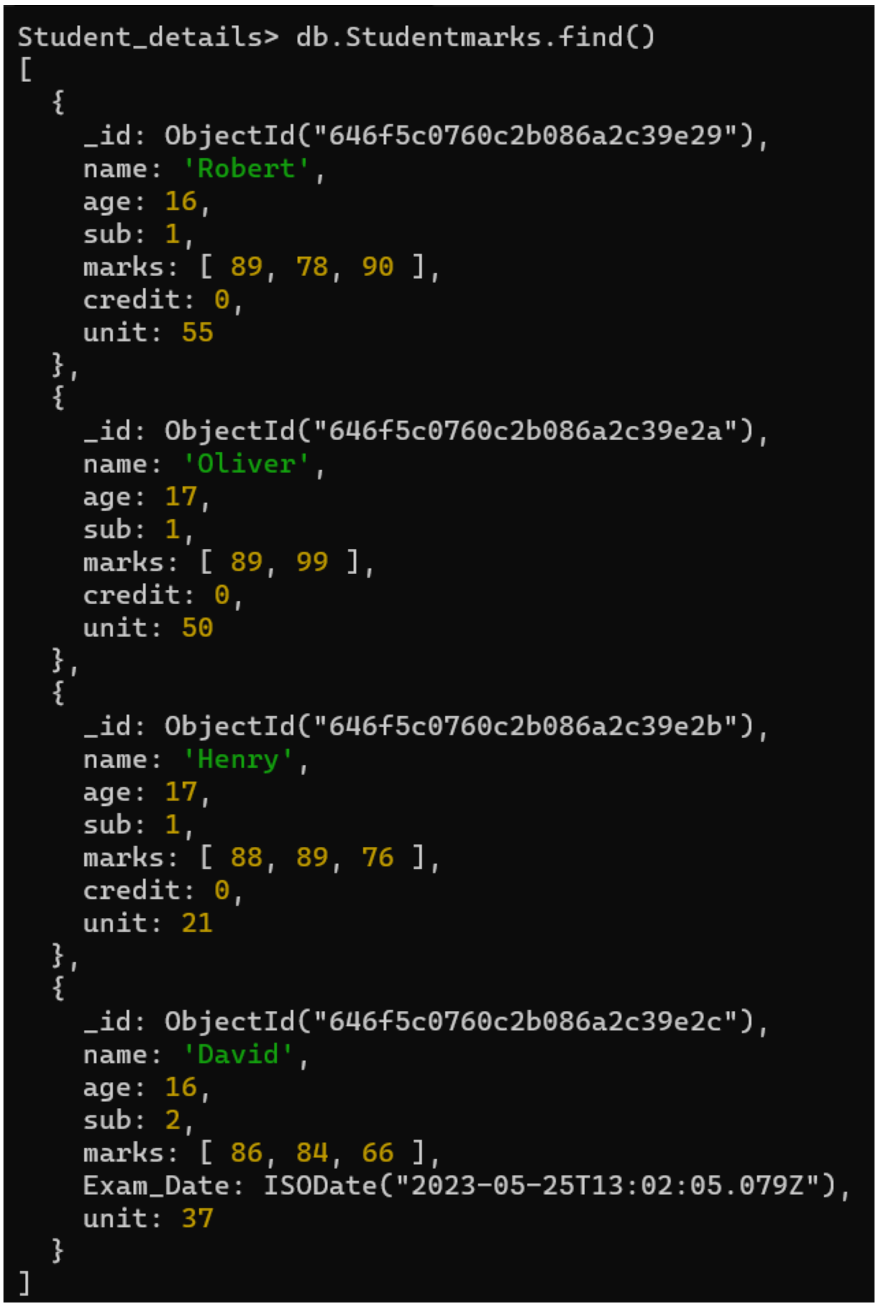
```
Student_detail> db.Studentmarks.updateMany( {}, [ { $set: { unit: { $ceil: { $multiply: [{ $rand: {} }, 100] } } } }])
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
```

Figure 3.35: Output of the Query with `$rand` Operator

To view the updated documents, the user can execute the query:

```
db.Studentmarks.find()
```

Figure 3.36 shows the output of this query.



```
Student_details> db.Studentmarks.find()
[
  {
    _id: ObjectId("646f5c0760c2b086a2c39e29"),
    name: 'Robert',
    age: 16,
    sub: 1,
    marks: [ 89, 78, 90 ],
    credit: 0,
    unit: 55
  },
  {
    _id: ObjectId("646f5c0760c2b086a2c39e2a"),
    name: 'Oliver',
    age: 17,
    sub: 1,
    marks: [ 89, 99 ],
    credit: 0,
    unit: 50
  },
  {
    _id: ObjectId("646f5c0760c2b086a2c39e2b"),
    name: 'Henry',
    age: 17,
    sub: 1,
    marks: [ 88, 89, 76 ],
    credit: 0,
    unit: 21
  },
  {
    _id: ObjectId("646f5c0760c2b086a2c39e2c"),
    name: 'David',
    age: 16,
    sub: 2,
    marks: [ 86, 84, 66 ],
    Exam_Date: ISODate("2023-05-25T13:02:05.079Z"),
    unit: 37
  }
]
```

Figure 3.36: Output of the `find()` Query

The `unit` field is updated with the random number generated for all the documents in the collection. Note that the `unit` field in the document named "`David`" is updated with the random number as the `credit` field was renamed to unit using the `rename` operator. For all the other records, a new field named `unit` is added and updated with the random number.

3.6 Summary

- MongoDB offers multiple operators to allow users to create complex queries.
- Main types of operators in MongoDB are query, projection, and update operators.
- Query operators allow operations that are based on comparison (both field and array), logic, and existence of a field.
- Projection operators help to specify what fields of a document a query must return.
- Update operators allow modification of both the field and array data.

Test Your Knowledge

1. Which of the following comparison operator will allow you to list only the documents that contain a field whose value matches with any value in the specified array?

   a. `$in`
   b. `$nin`
   c. `$ne`
   d. `$ni`

2. Consider that `marks` is a collection which consists of a field `score`. Which of the following option will you use to find only the score value of `datatype` string?

   a. `db.marks.find( { "Score" : { $type : 1}})`
   b. `db.marks.find( { "Score" : { $type : 2}})`
   c. `db.marks.find( { "Score" : { $type : 16}})`
   d. `db.marks.find( { "Score" : { $type : 3}})`

3. Consider that `marks` is a collection that consists of a field `subject_marks` which is an array. Which of the following option will list documents with `subject_marks` field of size `3`?

   a. `db.subject_marks.find({marks:{$size:3}})`
   b. `db.marks.find({subject_marks:{size:3}})`
   c. `db.marks.find({"$subject_marks":{$size:3}})`
   d. `db.marks.find({subject_marks:{$size:3}})`

4. Which of the following projection operator will project the first element in an array that matches the query condition?

   a. `$slice`
   b. `$first`
   c. `$`
   d. `$elementfirst`

5. Which of the following syntax allows you to remove the first element in the marks array?

   a. `{ $pop: { marks: -1 }}`
   b. `{ $pop: { marks: 1 }}`
   c. `{ $pop: { marks: 0 }`
   d. `{ $pop: { marks: $first}`

Answers to Test Your Knowledge

1	a
2	b
3	d
4	c
5	a

Try it Yourself

1. Create a database named `Student` and a collection named `Stud_mark`.
2. Insert the following four documents into the `Stud_mark` collection.

```
[
   { name: "Adam",
     gender:"M",
     subjects:["Java","C","Python"],
     marks:[89,78,90],
     average:85.6
   },
   { name: "Franklin",
     gender:"M",
     subjects:["C","VB","Python"],
     marks:[78,85,89],
     average:84
  },
   { name: "Michael",
     gender:"M",
     subjects:["Java", "PHP"],
     marks:[88,89],
     average:88.5
   },
   { name: "Amelia",
     gender:"F",
     subjects:["Ruby","C++"],
     marks:[86,87],
     average: 86.5
   }
]
```

Using the collection `stud_mark` perform the following tasks:

3. Find only the documents where the `average` value is equal to `84`.
4. Find only the documents where the `average` value is greater than `85`.
5. Display only the documents where the `subjects` array contains either `Java` or `C++`.
6. View only the documents where the `average` is greater than or equal to `87` and `average` is less than or equal to `90`.
7. View all the documents where the `subjects` array has the value `Java`.
8. Display only the documents where the first element in the `marks` array is less than `80`.

9. Display the details of the student named `Adam` where the `marks` array has only the first element and the second element.
10. Add a new date field `Date_of_exam` which shows the current date only for the student named “`Amelia`”.
11. Increase the `average` value by `2` for the student named “`Franklin`”.
12. Rename the field `Date_of_exam` as `Examination_date`.