Coding Bootcamp

©Simplilearn. All rights reserved.

simplilearn

TECHNOLOGY

MongoDB

©Simplilearn. All rights reserved.

TECHNOLOGY

MongoDB Concepts

©Simplilearn. All rights reserved.

simplilearn

Learning Objectives

By the end of this lesson, you will be able to:

- Define the basic concepts of MongoDB to provide a scalable, high-performance platform for handling diverse and unstructured data efficiently
- Analyze the concept of databases to provide a structured and efficient way to access, manage, and manipulate information
- Utilize MongoDB features such as indexing, sharding, and load balancing to optimize database performance
- Configure a MongoDB Atlas account, create a database, and manage collections and documents

©Simplilearn. All rights reserved.

Learning Objectives

By the end of this lesson, you will be able to:

- Assess the use of JSON and BSON data formats in MongoDB for data storage and manipulation
- Apply document schema validation for a collection in MongoDB to ensure data integrity
- Perform CRUD (Create, Read, Update, Delete) operations using Mongo Shell on a MongoDB collection

©Simplilearn. All rights reserved.

A Day in the Life of a Full Stack Developer

You are hired as a developer in an organization and have been assigned a task by your senior. You have been asked to write an ad hoc query, where the results may be discrete, depending on the variables in the query.

The idea is to optimize the performance of these ad hoc queries. To do so, you decide to use MongoDB.

To accomplish the task, you need to explore MongoDB, understand SQL and NoSQL databases, learn about BSON data types, perform CRUD operations, and more.

©Simplilearn. All rights reserved.

Introduction to MongoDB

©Simplilearn. All rights reserved.

What Is MongoDB?

MongoDB is a free, open-source document-oriented database developed by **MongoDB Inc.** It is designed to store large amounts of data that allows the users to work efficiently.

Unlike traditional relational databases that use tables and rows, MongoDB stores data in JSON-like documents, which makes it easier to work with data structures that are complex and change over time.

©Simplilearn. All rights reserved.

MongoDB: Features

01 Is categorized under NoSQL database management system

02 Stores unstructured and semi-structured data

MongoDB

03 Improves scalability, performance, and availability

04 Supports indexing and querying capabilities

©Simplilearn. All rights reserved.

Importance of MongoDB

Reasons why MongoDB can be a good choice for the application:

01 High availability

02 Better querying and indexing

03 Easy application

04 Good performance

05 Open-source database

06 Large community and ecosystem

©Simplilearn. All rights reserved.

MongoDB Structure

MongoDB structure contains the following components:

Database: Provides easily scalable and high-performance data access

Fields: Represents complex data structures

Collections: Stores query data in a flexible manner

MongoDB

Indexes: Stores a small subset of data from a collection

Documents: Creates a collection of key-value pairs

Embedded documents: Organizes related data within a single document

©Simplilearn. All rights reserved.

MongoDB Structure

MongoDB structure

©Simplilearn. All rights reserved.

simplilearn

MongoDB: Advantages

The following are the advantages of MongoDB:

01 Flexible data model

02 Scalability

03 High performance

04 Rich querying language

05 Low overhead

06 Flexible deployment options

07 Easy integration with other technologies

©Simplilearn. All rights reserved.

Where to Use MongoDB?

Following are some essential applications of MongoDB:

E-commerce

MongoDB handles diverse product catalogs and extensive customer data seamlessly. For example; Amazon utilizes MongoDB for its inventory management system, ensuring real-time updates.

Content Management Systems (CMS)

MongoDB is well-suited for CMS due to its ability to handle diverse and dynamic content types. For example; Forbes uses MongoDB to manage and deliver its vast array of articles, images, and multimedia content.

Internet of Things (IoT)

It manages high-volume sensor data and supports real-time device monitoring. For example; Bosch leverages MongoDB to store and analyze data from its connected devices.

©Simplilearn. All rights reserved.

Where to Use MongoDB?

Following are some essential applications of MongoDB:

Healthcare

It stores dynamic patient records and wearable health device data for proactive care. For example; Mount Sinai Health System uses MongoDB to manage patient records, research data, and clinical applications.

Social networks

It stores complex user profiles and delivers real-time activity feeds. For example; uberVu, a social media analytics platform, uses MongoDB to perform efficient range queries over time frames for content.

©Simplilearn. All rights reserved.

Capabilities of MongoDB

©Simplilearn. All rights reserved.

Ad Hoc Queries

Ad-hoc queries in MongoDB are short-lived commands whose output depends on variables that can change, making their results different each time they are run.

The ad hoc query is a command which takes value from a variable.

Whenever the ad hoc query is executed, the results may be discrete, depending on the variables in the query.

They are essential in scenarios where the nature of data requests cannot be fully anticipated during database schema design.

©Simplilearn. All rights reserved.

Indexing for Faster Execution of the Query

Indexing in MongoDB helps to enhance operations like search and filters.

MongoDB indexes may be created on run time.

A database will take longer to respond if multiple documents are queried without the correct indices.

MongoDB can also be declared on any field within any of the user documents.

©Simplilearn. All rights reserved.

Data Replication

When there is only one database to store the data, it can encounter multiple performance issues.

©Simplilearn. All rights reserved.

Data Replication

Every organization has to deploy multiple servers for recovery and backup.
This process is called data replication.

Horizontal scaling across multiple servers that store the same records improves record availability.

©Simplilearn. All rights reserved.

Data Replication

Data replication also increases the load capacity of the servers because the customer's request will be distributed among all the servers.

- A master server or node accepts all write operations and applies those operations to all of its slaves.
- A slave server can function as a master server if the primary or master server fails.

©Simplilearn. All rights reserved.

Real-Time Analytics

Real-time analytics involves processing and analyzing data as it becomes available, enabling timely insights and decision-making. The following are the key features of MongoDB facilitating real-time analysis:

- **Ad hoc queries:** MongoDB's ability to execute dynamic queries enables the rapid retrieval of relevant data.
- **Flexible schema:** MongoDB's flexible schema, can accommodate diverse data structures for real-time data ingestion and querying.
- **Indexing:** Efficient indexing mechanisms in MongoDB ensure fast query responses, essential for real-time data processing.
- **Aggregation framework:** This enables complex data computations within the database, reducing the need for external data movement for analysis.

©Simplilearn. All rights reserved.

Sharding

It is a process that splits a large dataset into many datasets.

©Simplilearn. All rights reserved.

Sharding

Sharding is necessary for a software to perform well.

Horizontal scalability is also possible by sharding through replication sets in MongoDB.

©Simplilearn. All rights reserved.

Sharding

By bringing together distributed server shards, a unified and comprehensive database is created to handle periods of system downtime.

Operations in a sharding environment are handled through a light-weight procedure called Mongos

Queries can be made using Mongo to the precise shard

©Simplilearn. All rights reserved.

Load Balancing

It is an important feature of large-scale database management for developing enterprise applications.

MongoDB supports big-scale load balancing via horizontal scaling functions like replication and sharding.

©Simplilearn. All rights reserved.

Load Balancing

The platform can deal with multiple concurrent read and write requests for identical data using load balancers

©Simplilearn. All rights reserved.

SQL and NoSQL Databases

©Simplilearn. All rights reserved.

SQL Databases

SQL is used to manage data in a relational database management system (RDBMS) using domain-specific languages.

©Simplilearn. All rights reserved.

SQL Databases

SQL is used to manage structured data.

Data manipulation language (DML)

Data query language (DQL)

Data control language (DCL)

Data definition language (DDL)

©Simplilearn. All rights reserved.

SQL Databases

The scope of the SQL query includes data query, data manipulation, data definition, and data access control.

```
{
  "_id": "5cf0029caff5056591b0ce7d",
  "firstname": "Jane",
  "lastname": "Wu",
  "address": {
    "street": "1 Circle Rd",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90404"
  }
  "hobbies": ["surfing", "coding"]
}
```

©Simplilearn. All rights reserved.

NoSQL Databases

NoSQL databases provide a mechanism for storing data in a non-tabular form.

Not Only SQL

NoSQL databases are used intensively in big data and real-time web applications.

©Simplilearn. All rights reserved.

NoSQL Databases

NoSQL databases provide:

©Simplilearn. All rights reserved.

NoSQL Databases

The data structures used by NoSQL databases are:

Key–value pair

Wide column

Graph

Document

©Simplilearn. All rights reserved.

SQL and NoSQL Databases

The difference between SQL and NoSQL databases are as follows:

Features	SQL Database	NoSQL Database
Data storage model	Fixed rows and columns in tables	JSON document, key-value pair, wide-column, graph
Schemas	Fixed	Flexible
Scaling	Vertical (scale-up with larger server)	Horizontal
Multi-record ACID	Supported	Not supported
Joins	Typically required	Typically not required
Data to object mapping	Requires ORM (Object-Relational Mapping)	Does not require ORM. It maps directly to data structures in most programming languages.

©Simplilearn. All rights reserved.

JSON and BSON Data Formats

©Simplilearn. All rights reserved.

Introduction to JSON

JavaScript Object Notation, also known as JSON, is one of the most popular data transition formats in MongoDB.

(JavaScript Object Notation)

JSON

It is a text-based and lightweight composition.

©Simplilearn. All rights reserved.

simplilearn

Properties of JSON

- It is a **data interchange format** that is easy to read and write for humans and machines.
- It is based on a subset of the JavaScript programming language standard.
- It transmits data between a client and a server in a web application.
- It is text-based, which makes it easy to transmit over network connections and to read in a plain text editor.

©Simplilearn. All rights reserved.

Importance of JSON

Following are the reasons to use JSON:

- Transmits data between a server and web applications
- Helps web services and APIs to provide public data
- Transmits data between a client and a server in web applications
- Consists of extremely lightweight and language independent structure
- Serializes and transmits structured data over a network

©Simplilearn. All rights reserved.

JSON: Format

Following are the basic rules for JSON format:

- JSON represents data objects in a structured format.
- The object is surrounded by curly braces {}.
- A key-value pair consists of a key and a value, separated by a colon (:).

```
{

Field1: Value1
Field2: Value2
.....
FieldN: ValueN

}
```

©Simplilearn. All rights reserved.

JSON: Structure

The following is a structural diagram of JSON:

©Simplilearn. All rights reserved.

JSON: Example

Here is an example of a simple JSON object:

```
{
  "name": "John Smith",
  "age": 30,
  "isStudent": true,
  "hobbies": ["reading", "hiking", "photography"],
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "state": "CA",
    "zip": "12345"
  }
}
```

©Simplilearn. All rights reserved.

Drawbacks of JSON

Supports a limited number of basic data types

Lacks support for dates and binary data

Slows down the traversal as the objects and properties don’t have a fixed length

©Simplilearn. All rights reserved.

Introduction to BSON

BSON (Binary JavaScript Object Notation) is a computer data interchange format. It is a binary file format that stores serialized JSON documents in a binary-encoded format.

©Simplilearn. All rights reserved.

Importance of BSON

The reasons to use BSON are:

- More compact and efficient than JSON
- Lightweight
- Highly traversable
- Less spacious and faster scan speed
- Easily and quickly parsed
- Wide range of data types

©Simplilearn. All rights reserved.

BSON Format to Store Data

BSON documents consist of a sequence of fields, where each field is a key-value pair.

The key is a string that identifies the field, and the value can be of any BSON data type.

©Simplilearn. All rights reserved.

BSON

The following are the datatypes supported by BSON (Binary JSON):

- Double
- String
- Object
- Array
- Binary data
- ObjectID
- Boolean
- Date
- Null
- Regular expression
- JavaScript code
- Symbol
- JavaScript code with scope
- 32-bit integer
- Timestamp
- 64-bit integer
- Decimal128

©Simplilearn. All rights reserved.

simplilearn

BSON Document: Example

Here is an example of a BSON document that represents a simple employee record:

```
\x2A\x00\x00\x00         // length of document
\x08                     // type of field (string)
\x5F\x69\x64\x00         // field name (_id)
\xB7\x1D\x14\xF8\x53\x2E // ObjectId value
\x10                     // type of field (int32)
\x61\x67\x65\x00         // field name (age)
\x1E\x00\x00\x00         // int32 value (30)
\x02                     // type of field (string)
\x66\x69\x72\x73\x74\x4E // field name (first_name)
\x05\x00\x00\x00\x4A\x6F\x68\x6E\x00 // string value (John)
\x02                     // type of field (string)
\x6C\x61\x73\x74\x5F\x6E\x61\x6D\x65 // field name (last_name)
\x04\x00\x00\x00\x44\x6F\x65\x00 // string value (Doe)
```

©Simplilearn. All rights reserved.

JSON vs. BSON

Following are the differences between JSON and BSON:

JSON	BSON
• It is a text-based format.	• It is a binary-encoded format.
• It supports strings, numbers, objects, arrays, and boolean values.	• It supports the same data types as JSON, plus additional types.
• It is easily parsed and serialized with standard libraries and tools.	• It requires specialized libraries and tools.
• It is suitable for data interchange between web applications and APIs.	• It is suitable for storage and retrieval of data in MongoDB.
• It is human-readable.	• It is not human-readable.

©Simplilearn. All rights reserved.

MongoDB GUI Elements

©Simplilearn. All rights reserved.

MongoDB GUI Tools

MongoDB offers several graphical user interface (GUI) tools that provide a user-friendly interface for managing MongoDB databases. Some of the MongoDB GUI tools are:

MongoDB Compass

01

It visualizes the schema, creates and edits documents, and runs ad-hoc queries.

Robo 3T

02

It builds the query, edits documents, and monitors the server.

Robo Mongo

03

It provides support for multiple tabs and connections.

HumongouS.io

04

It performs as a cloud-based MongoDB GUI tool.

©Simplilearn. All rights reserved.

MongoDB GUI Tools

Following are some of the MongoDB GUI tools:

Studio 3T

05

It is a commercial MongoDB GUI tool used for database administrators and developers.

NoSQL Booster for MongoDB

06

It is a tool used for importing and exporting data in various formats.

MongoDB Management Service (MMS)

07

It is a tool used for data exploration and visualization.

©Simplilearn. All rights reserved.

Major MongoDB GUI Tools

Some of the important MongoDB GUI tools are:

Studio 3T

Robo 3T

MongoDB Compass

©Simplilearn. All rights reserved.

Studio 3T

Studio 3T is made for growing professional teams. It offers a variety of ways to view and interrogate data collections, including sophisticated aggregations.

It enables teams to create, manage, and visualize their MongoDB databases with ease.

©Simplilearn. All rights reserved.

Studio 3T: Features

Some of the key features of Studio 3T are:

- Visual query builder
- Aggregation editor
- Schema explorer
- SQL importer/exporter
- IntelliShell
- SQL query
- Data compare and sync

©Simplilearn. All rights reserved.

Robo 3T

Robo 3T is a lightweight, open-source MongoDB GUI with an embedded Mongo shell and a real auto-completion.

It helps developers and database administrators to work with MongoDB databases easily.

©Simplilearn. All rights reserved.

Robo 3T: Features

Some of the key features of Robo 3T are:

- Intuitive user interface
- Visual query builder
- Shell integration

- JSON editor
- MongoDB Atlas integration
- Export and import
- Schema visualization

©Simplilearn. All rights reserved.

MongoDB Compass

MongoDB Compass is a graphical user interface (GUI) for working with MongoDB databases.

It allows users to easily visualize and manipulate data and perform administrative tasks on their databases.

©Simplilearn. All rights reserved.

MongoDB Compass: Features

Some of the key features of MongoDB Compass are:

- Data visualization
- Query building
- Schema analysis
- Index management
- Data manipulation

©Simplilearn. All rights reserved.

Databases, Collections, and Documents

©Simplilearn. All rights reserved.

Introduction

The fundamental building blocks for storing data in MongoDB are databases, collections, and documents.

©Simplilearn. All rights reserved.

Databases in MongoDB

A **database** is a container for the collection of documents.

Key concepts of database in MongoDB:

- Create a database
- View available databases
- Switch databases
- Drop a database
- Manage collections within a database

©Simplilearn. All rights reserved.

Collections in MongoDB

A **collection** is a group of MongoDB documents equivalent to a table in a relational database.

Key concepts of collections in MongoDB:

- Create a collection
- View available collections
- Insert documents into a collection
- Query documents in a collection
- Update documents in a collection
- Remove documents from a collection

©Simplilearn. All rights reserved.

Documents in MongoDB

A **document** is a set of key-value pairs representing a single object in a collection.

Key concepts of documents in MongoDB:

- Structure a document
- Insert a document
- Update a document
- Query a document
- Delete a document

©Simplilearn. All rights reserved.

MongoDB Shell and Drivers

The following are the essential tools for working with MongoDB:

MongoDB

MongoDB Shell

MongoDB Drivers

©Simplilearn. All rights reserved.

MongoDB Shell and Drivers

MongoDB Shell is a command-line interface that allows users to interact with MongoDB databases.

It supports powerful tools for developers and database administrators.

©Simplilearn. All rights reserved.

MongoDB Shell and Drivers

MongoDB shell is available for the following operating systems:

macOS

Windows

Linux

macOS

©Simplilearn. All rights reserved.

MongoDB Shell and Drivers

MongoDB drivers are software components that enable applications to communicate with MongoDB databases.

To interact with MongoDB, users can use different programming languages and drivers.

©Simplilearn. All rights reserved.

MongoDB Shell and Drivers

Here are some details on MongoDB drivers for different programming languages:

Java

MongoDB allows developers to connect to MongoDB databases from Java applications. The driver is compatible with all major Java frameworks and supports asynchronous programming.

Python

MongoDB provides a Python driver that allows developers to connect to MongoDB databases from Python applications. PyMongo is the official MongoDB driver for Python.

©Simplilearn. All rights reserved.

MongoDB Shell and Drivers

Here are some details on MongoDB drivers for different programming languages:

C++

MongoDB provides a C++ driver that allows developers to connect to MongoDB databases from C++ applications. The driver is compatible with all major C++ compilers and supports synchronous and asynchronous programming.

C#

MongoDB provides a C# driver that allows developers to connect to MongoDB databases from .NET applications. The driver is compatible with all major .NET frameworks.

©Simplilearn. All rights reserved.

MongoDB Shell and Drivers

Here are some details on MongoDB drivers for different programming languages:

Ruby

MongoDB provides a Ruby driver that allows developers to connect to MongoDB databases from Ruby applications. The driver is compatible with Ruby 1.9 and later versions and supports asynchronous programming.

Node.js

MongoDB provides a Node.js driver that allows developers to connect to MongoDB databases from Node.js applications. The driver is compatible with Node.js 8.x and later versions and supports asynchronous programming.

©Simplilearn. All rights reserved.

Data Modeling

©Simplilearn. All rights reserved.

Data Modeling

Data modeling is used to boost the performance of the database engine and provides the best way for data retrieval.

The following points must be considered while building a data model for an application:

Queries

Updates

Processing

©Simplilearn. All rights reserved.

Flexible Schema

MongoDB's collections do not require documents to adhere to a fixed schema, allowing for flexibility in data structure.

A single document does not need to have similar fields; the data types for a field can differ.

©Simplilearn. All rights reserved.

Flexible Schema

The schema of the documents in a collection can be changed by:

Adding a new field

Removing the field

Updating the data type of the field

©Simplilearn. All rights reserved.

simplilearn

Flexible Schema

The flexible schema facilitates the mapping of the BSON documents to an object.

Every document can pair with the data fields of the described entity.

These documents in a collection will share a similar structure by enforcing the document validation rule.

©Simplilearn. All rights reserved.

Document Structure

Building the data model for MongoDB applications revolves around:

Schema of the BSON document

Relations between the data

©Simplilearn. All rights reserved.

Document Structure

MongoDB allows embedding related data within a single document.

Embedded data documents record relationships between data by storing relevant data in a single BSON document schema.

©Simplilearn. All rights reserved.

Document Structure

Embedded data models in MongoDB enable storing related information in the same BSON document.

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Embedded sub-document

Embedded sub-document

©Simplilearn. All rights reserved.

Document Structure

The embedded data model can be used for:

Implementing a relationship between the BSON documents

Building a one-to-many relationship between the documents in the database

©Simplilearn. All rights reserved.

Document Structure

Below are the advantages of document embedding:

- Boosts the performance of the read operations
- Boosts the performance of the ability to request and fetch relevant data
- Enables the user to update the related data in a single write operation

©Simplilearn. All rights reserved.

Document Structure

There are several ways to work with embedded documents in MongoDB.

Use dot notation

Example: If a document contains an embedded document for address, use the following syntax: **document.address.city**

Use index notation

Example: If a document contains an array of embedded documents, it can be accessed by the following syntax: **document.arrayName[0]**

©Simplilearn. All rights reserved.

Document Structure

Query

A query can be performed using dot notation or index notation.

Example: To query all the documents where the city field in the embedded address document is "New York," use the following syntax:

db.collection.find({ "address.city": "New York" })

©Simplilearn. All rights reserved.

Document Structure

Update

The update can be done using the **$set** operator.

Example: To update the city field in the embedded address document, use the following syntax:

db.collection.updateOne({ "_id": ObjectId("123") }, { "$set": { "address.city": "San Francisco" } })

©Simplilearn. All rights reserved.

Document Structure

Delete

The embedded document can be deleted by setting its value to null using the **$unset** operator.

Example: To delete the address field in a document, use the following syntax:

```
db.collection.updateOne({ "_id": ObjectId("123") }, { "$unset": { "address": "" } })
```

©Simplilearn. All rights reserved.

Document Schema Validation

©Simplilearn. All rights reserved.

Document Schema Validation

Validation rules are **db.createCollection()** command with the **validator** option that specifies the validation rules.

To add document validation to an already created collection, use the **collMod** command with the validator option.

©Simplilearn. All rights reserved.

Document Schema Validation

MongoDB also provides related validation options, which include:

ValidationLevel option:

Illustrates how strictly MongoDB applies the rule of validation to existing BSON documents

ValidationAction option:

Illustrates whether the MongoDB should throw an error or reject BSON documents

©Simplilearn. All rights reserved.

Document Schema Validation

MongoDB supports the JSON schema validation rules, for example:

```
db.createCollection("employee", {
 validator: {
            $jsonSchema: {
             bsonType: "object",
             required: [ "name", "designation", "qualification", "address" ],
             properties: {
             name: {
             bsonType: "string",
             description: "name is required"
             },
             designation: {
             bsonType: "string",
             description: "designation is required"
             },
```

The $jsonSchema operator is used as a validator option while creating a new collection.

©Simplilearn. All rights reserved.

Document Schema Validation

MongoDB supports the JSON schema validation rules, for example:

```
qualification: {
        enum: [ "B.Tech", "BBA", "MBA", "B.Com", null ],
description: "choose one of the enum values and qualification is required“ },
address: {
        bsonType: "object",
        required: [ "city" ],
        properties: {
                street: {
                bsonType: "string",
                description: "must be a string"
                },
```

©Simplilearn. All rights reserved.

Document Schema Validation

MongoDB supports the JSON schema validation rules, for example:

```
city: {
                    bsonType: "string",
                    description: "city is required"
                    },

required"
                    },
                    zipcode: {
                    bsonType: int,
                    description: "must be a int and is required"
                    }
          }
          }
          }}}})
```

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

MongoDB Cloud Atlas is:

- A fully-handled cloud database for enterprise applications
- A useful way to run MongoDB as it is a modern and popular database system
- It is widely available in 70+ regions across other platforms

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

An Atlas account is created using two methods:

1. Signing in with a Google account
2. Signing in with an email address

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Google account steps are:

1. Click the **Sign Up with Google** button on the registration page
2. Enter one of the parameters like the user's email address or mobile number
3. Click the **Next** button and enter the password
4. Click the **Next** button and review and accept the **Terms of Service**
5. Click the **Submit** button

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

The following information need to be filled:

- Email Address
- First Name
- Last Name
- Password
- Company Name
- Job Function
- Country

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

The following are the rules for user password:

- Contain at least eight characters
- Contain special characters, like ! ” # $ % & ’ ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~
- Not contain the user’s email address and the Atlas username
- Be different from the last four passwords

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

The following are the final steps:

Review and accept the Terms of Service and Privacy Policy of the MongoDB Atlas

Click on **Submit** to create a MongoDB Atlas

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 1: Go to MongoDB Atlas **Login page** and log in

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 2: Click on **Build a Database**

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 3: Choose the shared database type, which is the free database type

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 4: Select either AWS, GCP, or Azure as your cloud provider and specify the region for the user database or cluster

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 5: Choose user cluster tier

Cluster Tier

M0 Sandbox (Shared RAM, 512 MB Storage)
Encrypted

Base hourly rate is for a MongoDB replica set with **3 data bearing servers**.

Shared Clusters for development environments and low-traffic applications

Tier	RAM	Storage	vCPU	Base Price
M0 Sandbox	Shared	512 MB	Shared	Free forever
M0 clusters are best for getting started, and are not suitable for production environments.				
500 max connections | Low network performance | 100 max databases | 500 max collections				
M2	Shared	2 GB	Shared	$9 / MONTH
M5	Shared	5 GB	Shared	$25 / MONTH

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 6: Give a name to the user database

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 7: Click on the **Create Cluster** button to deploy the user database

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 8: Go to the MongoDB Atlas home page and then navigate to the **Network Access** option under the **Security** tab and click on **ADD IP ADDRESS**

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

An IP address is a unique identifier for a device that connects with the network.

Example: The 127.0.45.1 IP address ranges from 0.0.0.0 to 255.255.255.255.

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 9: Add an IP address and set the deletion time for the IP address

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 10: Create a database user for accessing the user database from the MongoDB Atlas

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

The difference between a database user and an Atlas user is as follows:

The database users can access databases that are hosted in MongoDB Atlas.

The Atlas users will not have access to the clusters or database.

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 11: Go to the MongoDB Atlas Home page and then navigate to the **Database Access** option under the **Security** tab

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 12: Click on the **Add New Database User** button to create a new user

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 13: A dialog will pop up in which the user has to select the authentication method like password, certificate, or AWS IAM user

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 14: If the user selects the password method, add a username and custom password or auto-generated secure password

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 15: Specify the privileges to the database user and assign roles to the user

©Simplilearn. All rights reserved.

MongoDB Cloud Atlas

Step 16: Click on the **Add User** button to add users

©Simplilearn. All rights reserved.

Creating an Account on MongoDB Atlas

Duration: 10 Min.

Problem Statement:

You have been assigned a task to create an account on MongoDB Atlas and build a MongoDB application.

Outcome:

By completing this task, learners will gain proficiency in setting up a MongoDB Atlas account and developing a MongoDB application. They will enhance their understanding of database management in a cloud environment and acquire practical skills in building scalable and efficient database-driven applications.

Note: Refer to the demo document for detailed steps: 01_Creating_an_Account_on_MangoDB_Atlas

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed:

1. Build a MongoDB application
2. Create a cluster

©Simplilearn. All rights reserved.

Connecting to Atlas Database

©Simplilearn. All rights reserved.

Connecting to Atlas Database

There are several ways to connect with the user Atlas database:

01 — Mongo shell (mongosh)

It is a command-line tool for interacting with your MongoDB deployment.

02 — Node.js driver

It is a native interface to interact with MongoDB databases from Node.js applications.

03 — PyMongo driver

It is the official Python driver for interacting with MongoDB databases from Python applications.

04 — MongoDB compass

It is a graphical user interface for MongoDB, allowing you to interact with your data visually.

©Simplilearn. All rights reserved.

Connecting to Atlas Database

A user would need:

- A database user with required roles or access permission for operations
- A MongoDB Atlas account
- An IP address added to the IP access list
- An organization and project added to a MongoDB Atlas account

©Simplilearn. All rights reserved.

Connecting to Atlas Database

The following are the general steps to connect to the Atlas database using Mongo shell:

01 Go to the mongo shell download page

02 Select the platform from the given list according to the user system specification and download the executable file of the Mongo shell

03 Navigate to the downloaded file and run the executable file to start the installation process of mongosh

04 Add Mongo shell binary path to the environment variables

©Simplilearn. All rights reserved.

simplilearn

Connecting to Atlas Database

The following are the steps to connect to the Atlas database using Mongo shell on a Linux system:

01 Go to the Mongo shell download page

02 Select the platform from the given list according to the user system specification and copy the download URL to the user clipboard

©Simplilearn. All rights reserved.

Connecting to Atlas Database

The following are the steps to connect to the Atlas database using Mongo shell on a Linux system:

03 Use curl to fetch the installer file from the uniform resource locator (URL) or paste the URL into the user's internet browser

04 Use the *dpkg* command to install the Mongo shell on a Linux system

©Simplilearn. All rights reserved.

Connecting to Atlas Database

The following command must be executed in the terminal to verify the installation of the MongoDB shell:

Syntax:

```
mongosh -version
```

Output:

0.0.0

©Simplilearn. All rights reserved.

Connecting to Atlas Database

On the macOS system the following should be performed:

1. Use the *Homebrew* command provided

2. Copy the command from the MongoDB Atlas user interface and run the *Homebrew* command in the terminal

©Simplilearn. All rights reserved.

Connecting to Atlas Database

After installing the MongoDB shell, go to the Atlas dashboard, select the database or cluster that the user wishes to connect to, and click on **Connect**

©Simplilearn. All rights reserved.

Connecting to Atlas Database

Select the preferred connection method for the user to connect their MongoDB Atlas cluster

©Simplilearn. All rights reserved.

Connecting to Atlas Database

Select the **Connect with the MongoDB Shell** option and copy the connection URL

Replace the myFirstDatabase with the database name user want to connect in the user MongoDB Atlas Cluster

©Simplilearn. All rights reserved.

Connecting to Atlas Database

In the terminal, paste the connection URL along with the **mongosh** command:

Authentication will fail if the user types the wrong password.

```
C:\Users\[redacted]>mongosh "mongodb+srv://[redacted].ezufc.mongodb.net/myFirstDatabase" --username [redacted]
Enter password: **************
Current Mongosh Log ID: 61b9a759c4f882a74e22c1fe
Connecting to:          mongodb+srv://[redacted].ezufc.mongodb.net/myFirstDatabase
MongoServerError: bad auth : Authentication failed.
```

©Simplilearn. All rights reserved.

Connecting to Atlas Database

If the user enters the correct password, the MongoDB Shell connects the user to the MongoDB Atlas cluster database.

```
C:\Users\[redacted]>mongosh "mongodb+srv://[redacted].ezufc.mongodb.net/myFirstDatabase" --username [redacted]
Enter password: **********
Current Mongosh Log ID: 61b9a7bc0b80e9d047b1f7a7
Connecting to:          mongodb+srv://[redacted].ezufc.mongodb.net/myFirstDatabase
Using MongoDB:          4.4.10
Using Mongosh:          1.0.7

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

Atlas atlas-7erw61-shard-0 [primary] myFirstDatabase>
```

©Simplilearn. All rights reserved.

Exploring MongoDB Atlas Database Collection and Documents

Duration: 25 Min.

Problem Statement:

You have been assigned a task to explore the MongoDB Atlas database collection and documents by creating a database user and connecting to the database.

Outcome:

By exploring the MongoDB Atlas database collection and documents, creating a database user, and connecting to the database, you will gain a comprehensive understanding of database management and user authentication processes.

Note: Refer to the demo document for detailed steps: 02_Exploring_MongoDB_Atlas_Database_Collection_and_Documents

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a database user
2. Connect to the database
3. Load the sample dataset
4. Insert a document
5. Edit the document

©Simplilearn. All rights reserved.

Performing CRUD Operations with Mongo Shell

Duration: 25 Min.

Problem Statement:

You have been assigned a task to perform CRUD operations with Mongo Shell by installing mongosh and performing update operations.

Outcome:

By completing this task, learners will gain proficiency in performing CRUD operations using Mongo Shell. They will learn how to install mongosh and execute update operations effectively, enhancing their skills in database management and manipulation.

Note: Refer to the demo document for detailed steps: 03_Performing_CRUD_Operations_with_Mongo_Shell

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed are:

1. Install mongosh
2. Install Mongo Shell
3. Connect to MongoDB
4. Execute commands
5. Create a document
6. Insert more documents
7. Perform update operations

©Simplilearn. All rights reserved.

Configuring MongoDB with a Java Project

Duration: 20 Min.

Problem Statement:

You have been assigned a task to configure MongoDB with Java project by creating a Java project and adding JAR files.

Outcome:

By successfully configuring MongoDB with a Java project and integrating JAR files, learners will gain comprehensive skills in setting up a robust database system for Java applications. This hands-on experience will empower them to proficiently manage data interactions and enhance the functionality of their Java projects with MongoDB's capabilities.

Note: Refer to the demo document for detailed steps: 04_Configuring_MongoDB_with_a_Java_Project

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed are:

1. Create a Java project
2. Create a new class
3. Download JAR files
4. Add an external JAR
5. Create a new project
6. Add JAR files
7. Create a database

©Simplilearn. All rights reserved.

Inserting Documents

©Simplilearn. All rights reserved.

Inserting Documents

MongoDB supports inserting single and multiple documents at a time.
The below command is used to insert a single document:

Syntax:

```
db.collection_name.insertOne()
```

©Simplilearn. All rights reserved.

Inserting Documents

Example:

```
db.products.insertOne({
 "productTitle": "1TB Sandisk SSD",
 "productCode": "PRD-147-4588",
 "price": 5000,
 "rating": 4
})
```

Output:

```
Atlas atlas-7erw61-shard-0 [primary] myFirstDatabase> db.products.insertOne({ "productTitle": "1TB Sandisk SSD", "productCode": "PRD-
147-4588", "price": 5000, "rating": 4 })
{
  acknowledged: true,
  insertedId: ObjectId("61b9ae6c45a2bb43e8c0eb5c")
}
```

©Simplilearn. All rights reserved.

Inserting Documents

Below is the syntax for inserting multiple documents:

Syntax:

```
db.collection_name.insertMany()
```

©Simplilearn. All rights reserved.

Inserting Documents

Below is the example for inserting multiple documents:

Example:

```
db.products.insertMany([
 { "productTitle": "1TB Sandisk SSD", "productCode": "PRD-147-4588",
            "price": 5000, "rating": 4 },
 { "productTitle": "256GB WD Green SSD", "productCode": "PRD-146-
4544", "price": 9000, "rating": 3 },
 { "productTitle": "512GB Sandisk Pendrive", "productCode": "PRD-522-
7411", "price": 1600, "rating": 4 }
])
```

©Simplilearn. All rights reserved.

Inserting Documents

Below is the output of the example for inserting multiple documents:

```
Atlas atlas-7erw61-shard-0 [primary] myFirstDatabase> db.products.insertMany([{ "productTitle": "1TB Sandisk
 SSD", "productCode": "PRD-147-4588", "price": 5000, "rating": 4 },{ "productTitle": "256GB WD Green SSD", "
productCode": "PRD-146-4544", "price": 9000, "rating": 3 },{ "productTitle": "512GB Sandisk Pendrive", "prod
uctCode": "PRD-522-7411", "price": 1600, "rating": 4 }])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("61b9af5f45a2bb43e8c0eb5d"),
    '1': ObjectId("61b9af5f45a2bb43e8c0eb5e"),
    '2': ObjectId("61b9af5f45a2bb43e8c0eb5f")
  }
}
```

©Simplilearn. All rights reserved.

Inserting Documents Using MongoDB CRUD Operations

Duration: 25 Min.

Problem Statement:

You have been asked to insert documents in MongoDB using curd operations by creating a project and generating getters and setters.

Outcome:

By completing this demo document, learners will gain proficiency in MongoDB CRUD operations, project creation, and the generation of getters and setters. They will emerge equipped with the skills to seamlessly insert documents into MongoDB collections, effectively manage project structures, and streamline data access through the implementation of getter and setter methods.

Note: Refer to the demo document for detailed steps: 05_Inserting_Documents_Using_MongoDB_CRUD_Operations

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a Maven project
2. Add the dependencies and a print statement
3. Create a Customer and DBOperations class
4. Create a collection
5. Connect the database to the application
6. Create one new class
7. Generate constructor
8. Create a document
9. Generate Getters and Setters

©Simplilearn. All rights reserved.

Reading Documents in MongoDB CRUD Operations

Duration: 10 Min.

Problem Statement:

You have been asked to read documents from MongoDB by executing the application and fetching a document based on the e-mail.

Outcome:

By executing the application and fetching a document based on the e-mail from MongoDB, learners will gain proficiency in reading and retrieving data from a NoSQL database. This hands-on experience will enhance their understanding of database querying and improve their skills in working with MongoDB collections.

Note: Refer to the demo document for detailed steps: 06_Reading_Documents_in_MongoDB_CRUD_Operations

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a Maven project
2. Add the dependencies and a print statement
3. Create a Customer and DBOperations class
4. Create a collection
5. Connect the database to the application
6. Create one new class
7. Generate constructor
8. Create a document
9. Generate Getters and Setters

©Simplilearn. All rights reserved.

Updating Documents

©Simplilearn. All rights reserved.

Updating Documents

For updating documents, the user will have to execute the update command.

Syntax:

```
db.collection_name.updateOne(
 { <filter-criteria> },
 { <new-values> },
 { <update-options> }
)
```

©Simplilearn. All rights reserved.

Updating Documents

Below is the example for updating documents:

Example:

```
db.products.updateOne(
{
 "_id": ObjectId("61b9ae6c45a2bb43e8c0eb5c")
},
{
 $set: {
 "productTitle": "2TB Sandisk SSD",
 "productCode": "PRD-147-4588",
 "price": 6000,
 "rating": 5
          }
})
```

©Simplilearn. All rights reserved.

Updating Documents

Below is the output of the example:

```
Atlas atlas-7erw61-shard-0 [primary] myFirstDatabase> db.products.updateOne({ "_id": ObjectId("61b9ae6c45a2bb4
3e8c0eb5c") }, { $set: { "productTitle": "2TB Sandisk SSD", "productCode": "PRD-147-4588", "price": 6000, "rat
ing": 5 } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

©Simplilearn. All rights reserved.

Updating Documents

To update multiple documents, use this syntax:

Syntax:

```
db.collection_name.updateMany(
  { <filter-criteria> },
  { <new-values> },
  { <update-options> }
)
```

©Simplilearn. All rights reserved.

Updating Documents

Below is the example for updating documents:

Example:

```
db.products.updateMany(
  {},
  { $inc: { "price": 550 } }
)
```

©Simplilearn. All rights reserved.

Updating Documents

Below is the output of the example:

```
Atlas atlas-7erw61-shard-0 [primary] myFirstDatabase> db.products.updateMany({}, { $inc: { "price": 550 } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
```

```
Atlas atlas-7erw61-shard-0 [primary] myFirstDatabase> db.products.find()
[
  {
    _id: ObjectId("61b9ae6c45a2bb43e8c0eb5c"),
    productTitle: '2TB Sandisk SSD',
    productCode: 'PRD-147-4588',
    price: 6000,
    rating: 5
  },
  {
    _id: ObjectId("61b9af5f45a2bb43e8c0eb5d"),
    productTitle: '1TB Sandisk SSD',
    productCode: 'PRD-147-4588',
    price: 5000,
    rating: 4
  },
  {
    _id: ObjectId("61b9af5f45a2bb43e8c0eb5e"),
    productTitle: '256GB WD Green SSD',
    productCode: 'PRD-146-4544',
    price: 9000,
    rating: 3
  },
  {
    _id: ObjectId("61b9af5f45a2bb43e8c0eb5f"),
    productTitle: '512GB Sandisk Pendrive',
    productCode: 'PRD-522-7411',
    price: 1600,
    rating: 4
  }
]
```

BEFORE UPDATION

```
Atlas atlas-7erw61-shard-0 [primary] myFirstDatabase> db.products.find()
[
  {
    _id: ObjectId("61b9ae6c45a2bb43e8c0eb5c"),
    productTitle: '2TB Sandisk SSD',
    productCode: 'PRD-147-4588',
    price: 6550,
    rating: 5
  },
  {
    _id: ObjectId("61b9af5f45a2bb43e8c0eb5d"),
    productTitle: '1TB Sandisk SSD',
    productCode: 'PRD-147-4588',
    price: 5550,
    rating: 4
  },
  {
    _id: ObjectId("61b9af5f45a2bb43e8c0eb5e"),
    productTitle: '256GB WD Green SSD',
    productCode: 'PRD-146-4544',
    price: 9550,
    rating: 3
  },
  {
    _id: ObjectId("61b9af5f45a2bb43e8c0eb5f"),
    productTitle: '512GB Sandisk Pendrive',
    productCode: 'PRD-522-7411',
    price: 2150,
    rating: 4
  }
]
```

AFTER UPDATION

©Simplilearn. All rights reserved.

Updating Documents Using MongoDB CRUD Operations

Duration: 25 Min.

Problem Statement:

You have been assigned a task to work with the CRUD operations and update the documents for existing and non-existing customers.

Outcome:

By completing this task, learners will gain proficiency in executing CRUD operations, enabling them to effectively manage and manipulate customer data. They will develop the skills to update documents for both existing and non-existing customers, enhancing their understanding of database management and application development.

Note: Refer to the demo document for detailed steps: 07_Updating_Documents_Using_MongoDB_CRUD Operations

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed:

1. Update customer operations
2. Add try-catch block
3. Execute another operation for an update
4. Create a new function
5. Create more functions
6. Write a new method

©Simplilearn. All rights reserved.

Querying Documents

©Simplilearn. All rights reserved.

Querying Documents

To query documents in MongoDB, you typically use the **find()** method on a collection.

Syntax:

```
db.collection_name.find(
  { <filter-criteria> },    // Optional: Specifies the selection criteria
  { <projection> }          // Optional: Specifies the fields to include or exclude
)
```

This method specifies the filter criteria, projection (fields to include or exclude), sort order, and other options.

©Simplilearn. All rights reserved.

Querying Documents

Consider the **users** collection with the following documents:

Example:

```
[
  { "_id": 1, "name": "John Doe", "age": 30, "email": "john@example.com" },
  { "_id": 2, "name": "Jane Doe", "age": 25, "email": "jane@example.com" },
  { "_id": 3, "name": "Alice", "age": 28, "email": "alice@example.com" },
  { "_id": 4, "name": "Bob", "age": 35, "email": "bob@example.com" }
]
```

Execute the following command to find all users but only include their names and email fields:

Example:

```
db.users.find({}, { name: 1, email: 1, _id: 0 })
```

©Simplilearn. All rights reserved.

Querying Documents

The output of the query will be:

```
[
  { "name": "John Doe", "email": "john@example.com" },
  { "name": "Jane Doe", "email": "jane@example.com" },
  { "name": "Alice", "email": "alice@example.com" },
  { "name": "Bob", "email": "bob@example.com" }
]
```

©Simplilearn. All rights reserved.

Query Operators

©Simplilearn. All rights reserved.

Query Operators

Query operators help to locate data within the database. Below are the two types of query operators:

01 Comparison query operators

02 Logical query operators

©Simplilearn. All rights reserved.

Query Operators

Comparison query operators include:

$eq	Matches with the value of the field, which is equal to the specified value
$ne	Matches with the value of the field, which is not equal to the specified value
$lt	Matches with the value of the field, which is less than the specified value
$lte	Matches with the value of the field, which is less than or equal to the specified value

©Simplilearn. All rights reserved.

Query Operators

Comparison query operators include:

$gt	Matches with the value of the field, which is greater than the specified value
$gte	Matches with the value of the field, which is greater than or equal to the specified value
$in	Matches with any value which is specified in an array
$nin	Matches with none of the values, which are specified in an array

©Simplilearn. All rights reserved.

Query Operators

Logical query operators include:

$and	It is a join clause with logical AND, which returns all the documents which match both clauses.
$nor	It is a join clause with logical NOR, which returns all the documents which are failed to match both clauses.
$not	It is a query operator that inverts the effect of a query expression, which returns documents that are not equal to the specified value.
$or	It is a join clause with logical OR that returns all the documents which match either of the clauses.

©Simplilearn. All rights reserved.

simplilearn

Query Operators

Example for **$eq** operator:

Example:

```
db.queryProducts.find({
“rating”: { $eq: 1 } })
```

Output:

```
[
 {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-422',
        name: 'iPhone',
        price: 450$,
        rating: 1,
        inStock: true,
        colors: [ 'black', 'silver', 'white' ],
        category: 'A',
        details: { weight: 7, size: 5 }
 }
]
```

©Simplilearn. All rights reserved.

Query Operators

Example for **$ne** operator:

Example:

```
db.queryProducts.find({
“rating”: { $ne: 2 } })
```

Output:

```
[
 {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-422',
        name: 'iPhone',
        price: 450$,
        rating: 1,
        inStock: true,
        colors: [ 'black', 'silver', 'white' ],
        category: 'A',
        details: { weight: 7, size: 5 }
 }
]
```

©Simplilearn. All rights reserved.

Query Operators

Example for **$lt** operator:

Example:

```
db.queryProducts.find({
“price”: { $lt: 450 } })
```

Output:

```
[
{
        _id: ObjectId("61bc33e045a2bb43e8c0eb61"),
        code: 'PRD-421',
        name: 'Samsung Phone',
        price: 300$,
        rating: 2,
        inStock: true,
        colors: [ 'red', 'blue', 'black' ],
        category: 'B',
        details: { weight: 10, size: 5 }
 }
]
```

©Simplilearn. All rights reserved.

Query Operators

Example for **$lte** operator:

Example:

```
db.queryProducts.find({ “price”: { $lte: 450 } })
```

©Simplilearn. All rights reserved.

Query Operators

Output:

```
[
{
        _id: ObjectId("61bc33e045a2bb43e8c0eb61"),
        code: 'PRD-421',
        name: 'Samsung Phone',
        price: 300$,
        rating: 2,
        inStock: true,
        colors: [ 'red', 'blue', 'black' ],
        category: 'B',
        details: { weight: 10, size: 5 }
 },
 {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-422',
        name: 'iPhone',
        price: 450$,
        rating: 1,
        inStock: true,
        colors: [ 'black', 'silver', 'white' ],
        category: 'A',
        details: { weight: 7, size: 5 }
 }
]
```

©Simplilearn. All rights reserved.

Query Operators

Example for **$gt** operator:

Example:

```
db.queryProducts.find({
"price": { $gt: 300 } })
```

Output:

```
[
 {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-422',
        name: 'iPhone',
        price: 450$,
        rating: 1,
        inStock: true,
        colors: [ 'black', 'silver', 'white' ],
        category: 'A',
        details: { weight: 7, size: 5 }
 }
]
```

©Simplilearn. All rights reserved.

Query Operators

Example for **$gte** operator:

```
db.queryProducts.find({ “price”: { $gte: 300 } })
```

©Simplilearn. All rights reserved.

Query Operators

Output:

```
[
{
        _id: ObjectId("61bc33e045a2bb43e8c0eb61"),
        code: 'PRD-421',
        name: 'Samsung Phone',
        price: 300$,
        rating: 2,
        inStock: true,
        colors: [ 'red', 'blue', 'black' ],
        category: 'B',
        details: { weight: 10, size: 5 }
 },
 {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-422',
        name: 'iPhone',
        price: 450$,
        rating: 1,
        inStock: true,
        colors: [ 'black', 'silver', 'white' ],
        category: 'A',
        details: { weight: 7, size: 5 }
 }
]
```

©Simplilearn. All rights reserved.

Query Operators

Example for **$in** operator:

```
db.queryProducts.find({ “colors”: {$in: [“black”, “white”] } } )
```

©Simplilearn. All rights reserved.

Query Operators

Output:

```
[
 {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-422',
        name: 'iPhone',
        price: 450$,
        rating: 1,
        inStock: true,
        colors: [ 'black', 'silver', 'white' ],
        category: 'A',
        details: { weight: 7, size: 5 }
 }
 ]
```

©Simplilearn. All rights reserved.

Query Operators

Example for **$nin** operator:

```
db.queryProducts.find({ “colors”: { $nin: [“black”, “white”] } })
```

©Simplilearn. All rights reserved.

Query Operators

Output:

```
[
{
        _id: ObjectId("61bc33e045a2bb43e8c0eb61"),
        code: 'PRD-421',
        name: 'Samsung Phone',
        price: 300$,
        rating: 2,
        inStock: true,
        colors: [ 'red', 'blue', 'black' ],
        category: 'B',
        details: { weight: 10, size: 5 }
 },
 {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-422',
        name: 'iPhone',
        price: 450$,
        rating: 1,
        inStock: true,
        colors: [ 'black', 'silver', 'white' ],
        category: 'A',
        details: { weight: 7, size: 5 }
 }
]
```

©Simplilearn. All rights reserved.

Query Operators

The user needs to use the **$and** keyword to query documents based on the AND condition.

Example:

```
db.queryProducts.find({
 $and: [
 { “rating”: 1 },
 { “category”: “A” }
 ]
})
```

Output:

```
[
    {
    _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
    code: 'PRD-422',
    name: 'iPhone',
    price: 450$,
    rating: 1,
    inStock: true,
    colors: [ 'black', 'silver', 'white' ],
    category: 'A',
    details: { weight: 7, size: 5 }
 }
]
```

©Simplilearn. All rights reserved.

Query Operators

The user needs to use the **$or** keyword to query documents based on the OR condition.

Example:

```
db.queryProducts.find({
 $or: [
 { “category”: “A” },
 { “price”: { $gt: 350 } }
 ]
})
```

Output:

```
[ {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-422',
        name: 'iPhone',
        price: 450$,
        rating: 1,
        inStock: true,
        colors: [ 'black', 'silver', 'white' ],
        category: 'A',
        details: { weight: 7, size: 5 }
 }
]
```

©Simplilearn. All rights reserved.

Query Operators

The user must use the **$nor** keyword to query documents based on the NOR condition.

Example:

```
db.queryProducts.find({
 $nor: [
 { “category”: “A” },
 { “price”: { $gt: 450 } }
 ]
})
```

Output:

```
[
{
        _id: ObjectId("61bc33e045a2bb43e8c0eb61"),
        code: 'PRD-421',
        name: 'Samsung Phone',
        price: 300$,
        rating: 2,
        inStock: true,
        colors: [ 'red', 'blue', 'black' ],
        category: 'B',
        details: { weight: 10, size: 5 }
 },
]
```

©Simplilearn. All rights reserved.

Query Operators

To query documents based on the **NOT** condition, the user needs to use the **$not** keyword.

Example:

```
db.queryProducts.find({
 $not: [
 { “category”: “B” },
 { “price”: { $gt: 300 } }
 ]
})
```

Output:

```
[ {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-422',
        name: 'iPhone',
        price: 450$,
        rating: 1,
        inStock: true,
        colors: [ 'black', 'silver', 'white' ],
        category: 'A',
        details: { weight: 7, size: 5 }
 }
]
```

©Simplilearn. All rights reserved.

Query Operators

Below is the output of the example:

```
[ {
        _id: ObjectId("61bc33e045a2bb43e8c0eb61"),
        code: 'PRD-421',
        name: 'Samsung Phone',
        price: 300$,
        rating: 2,
        inStock: true,
        colors: [ 'red', 'blue', 'black' ],
        category: 'B',
        details: { weight: 10, size: 5 }
 }, {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-422',
        name: 'iPhone',
        price: 450$,
        rating: 1,
        inStock: true,
        colors: [ 'black', 'silver', 'white' ],
        category: 'A',
        details: { weight: 7, size: 5 }
 } ]
```

©Simplilearn. All rights reserved.

Query Operators

Below is an example of the query using both AND and OR operators:

```
db.queryProducts.find({
 $and: [
 { “rating”: 1 }
 ],
 $or: [
 { “category”: “B” },
 { “price”: { $lte: 450 } }
 ]
})
```

©Simplilearn. All rights reserved.

Query Operators

Objects within an object are termed nested objects.

Users can query a complete nested document or a nested field inside the document.

©Simplilearn. All rights reserved.

Query Operators

Below shows the querying of nested documents:

Example:

```
db.queryProducts.find({
"details": { "weight": 7,
"size": 5 } })
```

Output:

```
[
  {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-422',
        name: 'iPhone',
        price: 450$,
        rating: 1,
        inStock: true,
        colors: [ 'black', 'silver', 'white' ],
        category: 'A',
        details: { weight: 7, size: 5 }
  }
]
```

©Simplilearn. All rights reserved.

Query Operators

Below shows the query on the specific field:

Example:

```
db.queryProducts.find({ "details.size": 5 })
```

©Simplilearn. All rights reserved.

Query Operators

Output:

```
[ {
        _id: ObjectId("61bc33e045a2bb43e8c0eb61"),
        code: 'PRD-421',
        name: 'Samsung Phone',
        price: 300$,
        rating: 2,
        inStock: true,
        colors: [ 'red', 'blue', 'black' ],
        category: 'B',
        details: { weight: 10, size: 5 }
 }, {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-422',
        name: 'iPhone',
        price: 450$,
        rating: 1,
        inStock: true,
        colors: [ 'black', 'silver', 'white' ],
        category: 'A',
        details: { weight: 7, size: 5 }
 } ]
```

©Simplilearn. All rights reserved.

Query Operators

Nested fields can also be queried using the **$and** operator.

Example:

```
db.queryProducts.find({
 $and: [
 { "details.weight": { $gt: 7 }
},
 { "details.size": 5 }
 ]
})
```

Output:

```
[
{
        _id: ObjectId("61bc33e045a2bb43e8c0eb61"),
        code: 'PRD-421',
        name: 'Samsung Phone',
        price: 300$,
        rating: 2,
        inStock: true,
        colors: [ 'red', 'blue', 'black' ],
        category: 'B',
        details: { weight: 10, size: 5 }
 }
]
```

©Simplilearn. All rights reserved.

Query Operators

Below shows the query on arrays:

```
db.queryProducts.find({ "colors": “black” })
```

©Simplilearn. All rights reserved.

Query Operators

Below is the output of the example:

```
[
{
        _id: ObjectId("61bc33e045a2bb43e8c0eb61"),
        code: 'PRD-421',
        name: 'Samsung Phone',
        price: 300$,
        rating: 2,
        inStock: true,
        colors: [ 'red', 'blue', 'black' ],
        category: 'B',
        details: { weight: 10, size: 5 }
 },
 {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-422',
        name: 'iPhone',
        price: 450$,
        rating: 1,
        inStock: true,
        colors: [ 'black', 'silver', 'white' ],
        category: 'A',
        details: { weight: 7, size: 5 }
 }
]
```

©Simplilearn. All rights reserved.

Query Operators

Below is the example to fetch limited documents:

```
db.queryProducts.find().limit(3)
```

©Simplilearn. All rights reserved.

Query Operators

Below is the output of the example:

```
[ {
        _id: ObjectId("61bc33e045a2bb43e8c0eb61"),
        code: 'PRD-41',
        name: 'Pendrive’, price: 488, rating: 4,
        inStock: true,
        colors: [ 'red', 'black', 'violet' ],
        category: 'A',
        details: { weight: 45, materialType: 'cotton', size: 12 }
 }, {
        _id: ObjectId("61bc33e045a2bb43e8c0eb62"),
        code: 'PRD-42',
        name: 'T-Shirt’, price: 450, rating: 5,
        inStock: true,
        colors: [ 'black', 'orange', 'white' ],
        category: 'B',
        details: { weight: 48, materialType: 'silk', size: 24 }
 }, {
        _id: ObjectId("61bc33e045a2bb43e8c0eb63"),
        code: 'PRD-43',
        name: 'Shirt’, price: 796,    rating: 4,
        inStock: true,
        colors: [ 'yellow', 'white', 'green' ],
        category: 'B',
        details: { weight: 85, materialType: 'silk', size: 26 }
 } ]
```

©Simplilearn. All rights reserved.

Deleting Documents

©Simplilearn. All rights reserved.

Deleting Documents

To delete a single document, use the following syntax:

```
db.collection_name.deleteOne(
 { <filter-criteria> }
)
```

©Simplilearn. All rights reserved.

Deleting Documents

Example:

```
db.products.deleteOne(
{
 "_id": ObjectId("61b9c35e45a2bb43e8c0eb60")
})
```

Output:

```
Atlas atlas-7erw61-shard-0 [primary] myFirstDatabase> db.products.insertOne({ "productTitle": "1TB Sandisk SSD
", "productCode": "PRD-147-4588", "price": 5000, "rating": 4 },{ "productTitle": "256GB WD Green SSD", "produc
tCode": "PRD-146-4544", "price": 9000, "rating": 3 })
{
  acknowledged: true,
  insertedId: ObjectId("61b9c35e45a2bb43e8c0eb60")
}
Atlas atlas-7erw61-shard-0 [primary] myFirstDatabase> db.products.deleteOne({ "_id": ObjectId('61b9c35e45a2bb4
3e8c0eb60') })
{ acknowledged: true, deletedCount: 1 }
```

©Simplilearn. All rights reserved.

Deleting Documents

To delete multiple documents, use this syntax:

```
db.collection_name.deleteMany(
 { <filter-criteria> }
)
```

©Simplilearn. All rights reserved.

Deleting Documents

Example:

```
db.products.deleteMany(
 { “price”: { $lt: 2500 } }
)
```

©Simplilearn. All rights reserved.

Deleting Documents

Output:

```
Atlas atlas-7erw61-shard-0 [primary] myFirstDatabase> db.products.deleteMany({ "price": { $lt: 2500 } })
{ acknowledged: true, deletedCount: 1 }
```

```
Atlas atlas-7erw61-shard-0 [primary] myFirstDatabase> db.products.find()
[
  {
    _id: ObjectId("61b9ae6c45a2bb43e8c0eb5c"),
    productTitle: '2TB Sandisk SSD',
    productCode: 'PRD-147-4588',
    price: 6550,
    rating: 5
  },
  {
    _id: ObjectId("61b9af5f45a2bb43e8c0eb5d"),
    productTitle: '1TB Sandisk SSD',
    productCode: 'PRD-147-4588',
    price: 5550,
    rating: 4
  },
  {
    _id: ObjectId("61b9af5f45a2bb43e8c0eb5e"),
    productTitle: '256GB WD Green SSD',
    productCode: 'PRD-146-4544',
    price: 9550,
    rating: 3
  },
  {
    _id: ObjectId("61b9af5f45a2bb43e8c0eb5f"),
    productTitle: '512GB Sandisk Pendrive',
    productCode: 'PRD-522-7411',
    price: 2150,
    rating: 4
  }
]
```

BEFORE DELETION

```
Atlas atlas-7erw61-shard-0 [primary] myFirstDatabase> db.products.find()
[
  {
    _id: ObjectId("61b9ae6c45a2bb43e8c0eb5c"),
    productTitle: '2TB Sandisk SSD',
    productCode: 'PRD-147-4588',
    price: 6550,
    rating: 5
  },
  {
    _id: ObjectId("61b9af5f45a2bb43e8c0eb5d"),
    productTitle: '1TB Sandisk SSD',
    productCode: 'PRD-147-4588',
    price: 5550,
    rating: 4
  },
  {
    _id: ObjectId("61b9af5f45a2bb43e8c0eb5e"),
    productTitle: '256GB WD Green SSD',
    productCode: 'PRD-146-4544',
    price: 9550,
    rating: 3
  }
]
```

AFTER DELETION

©Simplilearn. All rights reserved.

Deleting Documents Using MongoDB CRUD Operations

Duration: 15 Min.

Problem Statement:

You have been assigned a task to delete documents from the MongoDB collection using the delete operation with the try-catch block.

Outcome:

By completing this task, you will gain proficiency in utilizing MongoDB's delete operation within a try-catch block. This will enhance your understanding of error handling and data manipulation in MongoDB collections, empowering you to confidently manage and maintain database integrity in real-world scenarios.

Note: Refer to the demo document for detailed steps: 08_Deleting_Documents_Using_MongoDB_CRUD_Operations

ASSISTED PRACTICE

©Simplilearn. All rights reserved.

Assisted Practice: Guidelines

Steps to be followed :

1. Create methods to delete customers
2. Handle exceptions

©Simplilearn. All rights reserved.

Bulk Write Operation

©Simplilearn. All rights reserved.

Bulk Write Operation

MongoDB supports the ability to perform multiple write operations in one command to clients, known as bulk write operations. It includes:

©Simplilearn. All rights reserved.

Bulk Write Operation

Example:

```
db.queryProducts.bulkWrite([
 {
 insertOne: {
 “document” : { <document-definition> }
 }
 },
 {
 updateOne: {
 “filter”: { “code”: “PRD-54” },
 “Update”: { $set: { “price”: 1000 } }
 }
 },
 {
 deleteOne: {
 “filter”: { "rating": 0 }
 }
 }
])
```

Output:

```
{
 acknowledged: true,
 insertedCount: 1,
 insertedIds: { '0':
ObjectId("61bc4dc745a2bb43e8c0eb70") },
 matchedCount: 1,
 modifiedCount: 1,
 deletedCount: 1,
 upsertedCount: 0,
 upsertedIds: {}
}
```

©Simplilearn. All rights reserved.

Bulk Write Operation

Ordered bulk write operations are executed in a serial order.

If any of the operations fails during the execution of the write operations, MongoDB will stop executing the remaining write operations in the list.

©Simplilearn. All rights reserved.

simplilearn

Bulk Write Operation

Unordered bulk write operations are executed in random order, and their behavior is not guaranteed.

If any write operations fails in unordered write operations, then MongoDB will continue executing the remaining write operations in the list.

©Simplilearn. All rights reserved.

Indexes od MongoDB

©Simplilearn. All rights reserved.

Indexes

Indexes in MongoDB enhance the database's performance and improve the efficient execution of queries.

MongoDB scans each document of the collection to select the one that matches the query statement.

©Simplilearn. All rights reserved.

Indexes

When an index is used for a query statement, it will display the same number of documents as the given index.

Indexes are a data structure that stores a required size of the collection's data set in a manageable form.

©Simplilearn. All rights reserved.

Indexes

An index in MongoDB stores the values of one or multiple fields in a sorted manner based on the value of the field.

An index is created on the field_id, which prevents the client from inserting two documents with the same _id value.

©Simplilearn. All rights reserved.

Indexes

This command will create an ascending index of the field price and return the index name.

Example:

```
db.queryProducts.createIndex(
 { price: 1 }
)
```

©Simplilearn. All rights reserved.

Indexes

The name of the index is the concatenation of the indexed keys and the order manner (ascending or descending).

The following demonstrates how MongoDB constructs index names:

1

2

3

Example:

{ price: 1 } has the name price_1

{ weight: -1, size: 1 } has the name weight_-1_size_1.

©Simplilearn. All rights reserved.

Indexes

A custom name can also be used to create an index that is human-readable instead of using index naming conventions.

The index can be created using the **createIndex()** command with the name option.

©Simplilearn. All rights reserved.

Indexes

Example:

```
db.queryProducts.createIndex(
 { “price”: 1 },
 { “name”: “Product Prices” }
)
```

Output:

```
[
 { v: 2, key: { _id: 1 }, name: '_id_' },          ← Default Index
 { v: 2, key: { weight: -1, size: 1 }, name: 'weight_-1_size_1' },      ← Index with default name
 { v: 2, key: { price: 1 }, name: 'Product Prices' }      ← Index with custom name
]
```

©Simplilearn. All rights reserved.

simplilearn

Indexes

The drop index command will drop the index and return the index name.

Example:

```
db.collection_name.dropIndex(<index-name> or
<index-specification>)
```

©Simplilearn. All rights reserved.

Indexes

Example:

db.queryProducts.createIndex(“weight_-1_size_1”)

Output:

```
{
 nIndexesWas: 3,
 ok: 1,
 '$clusterTime': {
        clusterTime: Timestamp({ t: 1639993405, i: 3 }),
        signature: {
        hash:
Binary(Buffer.from("a4ca341a77cec25bb69827327f3726
9e9f5f6e2c", "hex"), 0),
        keyId: Long("6984063280851451905")
        }
 },
 operationTime: Timestamp({ t: 1639993405, i: 3 })
}
```

©Simplilearn. All rights reserved.

Types of Indexes

©Simplilearn. All rights reserved.

Types of Indexes

MongoDB has two major indexes:

mongoDB

01 Singe field

02 Compound index

©Simplilearn. All rights reserved.

Types of Indexes

MongoDB supports the creation of user-defined ascending or descending indexes on a single field of the BSON document.

Example:

```
db.queryProducts.createIndex(
 { price: 1 }
)
```

©Simplilearn. All rights reserved.

Types of Indexes

MongoDB also supports the multi-field index defined by the user.

Example:

```
db.queryProducts.createIndex(
 { “category”: 1, “code”: 1 }
)
```

©Simplilearn. All rights reserved.

Key Takeaways

- The ad hoc query is a command which takes a value from a variable.
- Indexing helps to enhance operations like search and filters.
- Sharding is a process that splits a large dataset into many datasets.
- SQL is used to manage data in a relational database management system (RDBMS) using domain-specific languages.
- A binary serialization format is used to pile documents and make local procedure calls.

©Simplilearn. All rights reserved.

TECHNOLOGY

Thank You

©Simplilearn. All rights reserved.

simplilearn