Day 11: Connecting to MongoDB – Database Integration

# 1. Introduction to MongoDB & Mongoose

## What is NoSQL?

NoSQL stands for **"Not Only SQL,"** referring to a broad category of database systems that do not rely on the traditional relational database model. Unlike relational databases that store data in a structured, table-based format with predefined schemas, NoSQL databases offer **greater flexibility** by allowing data to be stored in various formats such as key-value pairs, wide-column stores, graphs, or documents.

NoSQL databases were designed to address the **scalability, performance, and flexibility challenges** posed by modern applications, particularly those that deal with large amounts of unstructured or semi-structured data. They allow for **horizontal scaling**, meaning that as the data grows, we can distribute it across multiple machines instead of upgrading a single server.

The most common use cases for NoSQL databases include **real-time analytics, content management systems, e-commerce platforms, IoT applications, and social media networks.** These databases are particularly effective in handling dynamic data, where structure might change over time, such as user-generated content, product catalogs, or personalized recommendations.

### NoSQL vs. SQL Databases

|  |  |  |
| --- | --- | --- |
| **Feature** | **SQL Databases (Relational)** | **NoSQL Databases (Non-Relational)** |
| **Data Structure** | Tables (Rows & Columns) | Documents, Key-Value, Graph, or Column-Family |
| **Schema** | Fixed Schema (Predefined Structure) | Dynamic Schema (Flexible Data Model) |
| **Scalability** | Vertical Scaling (Add More CPU, RAM) | Horizontal Scaling (Distribute Across Multiple Servers) |
| **Relationships** | Supports Relationships (Joins, Foreign Keys) | Stores Related Data in Nested Documents (No Joins) |
| **Query Language** | SQL (Structured Query Language) | NoSQL Query Methods (APIs, JSON, etc.) |
| **ACID Compliance** | Fully ACID-Compliant | Often Prioritizes Performance Over ACID |
| **Best Use Cases** | Financial Applications, Inventory Management, Enterprise Systems | Real-Time Analytics, IoT, Social Media, Big Data |

### When to Use SQL vs. NoSQL?

**Use SQL Databases When:**

* Your data is highly structured and requires **strict relationships** (e.g., banking systems, inventory management).
* You need **ACID compliance** to ensure **data consistency and reliability** (e.g., financial transactions).
* You have **complex queries and require JOIN operations** to fetch related data efficiently.
* Your application does not require frequent schema changes.

**Use NoSQL Databases When:**

* Your application requires **high-speed read/write performance** (e.g., real-time analytics, gaming leaderboards).
* You are dealing with **large amounts of unstructured or semi-structured data** (e.g., IoT, logs, sensor data).
* You need **horizontal scaling** to handle **high traffic loads** (e.g., e-commerce, social media platforms).
* Your data model is **dynamic and evolving**, and you do not want to enforce a fixed schema.

Both **SQL and NoSQL databases** serve different purposes, and choosing the right one depends on the **requirements of your application.**

* **SQL databases** are best suited for applications that need **structured data, strong consistency, and complex queries** (e.g., banking, enterprise systems).
* **NoSQL databases** are ideal for applications that require **high-speed access, horizontal scalability, and flexible schema design** (e.g., social networks, real-time applications).

## What is MongoDB?

MongoDB is a **document-oriented NoSQL database** that stores data in a flexible, JSON-like format known as **BSON (Binary JSON).** Instead of organizing data into tables with fixed schemas, MongoDB uses **collections** to group related documents. Each document can have its own structure, meaning that fields can vary from one document to another within the same collection.

At its core, MongoDB operates using two primary components:

1. **Collections:** These function similarly to tables in a relational database but are schema-less, meaning documents within a collection do not need to have the same fields.
2. **Documents:** These are individual records stored in BSON format, which is a binary representation of JSON.

A **MongoDB document** typically consists of **key-value pairs**, and each document has a unique \_id field that serves as its primary identifier.

Here is an example of how MongoDB stores data:

|  |
| --- |
| {  "\_id": ObjectId("64d4e3f60b4a2f6d8a5e7b32"),  "name": "Alice",  "email": "alice@example.com",  "age": 25,  "skills": ["JavaScript", "MongoDB", "Node.js"]  } |

This document belongs to a collection, such as users, and contains **different fields with various data types**, including strings, numbers, and arrays.

Unlike relational databases, where every row in a table must have the same columns, MongoDB allows documents within the same collection to have **different structures.** This flexibility makes it easier to adapt to changing data requirements without modifying existing structures or relationships.

### Why Use MongoDB?

MongoDB has gained immense popularity in modern web development due to its **flexibility, scalability, and ease of use.** Some of the key reasons for choosing MongoDB include:

1. **Flexible Data Model:**
   * Since MongoDB does not enforce a fixed schema, it is ideal for applications where data structures may change over time.
   * You can store different types of data within the same collection without defining a rigid schema in advance.
2. **Works Well with JavaScript:**
   * MongoDB is naturally integrated with JavaScript-based environments like Node.js.
   * It uses a JSON-like syntax for data storage, making it easier for **full-stack JavaScript developers** to work with.
3. **Scalability and Performance:**
   * MongoDB is designed to scale **horizontally** using a process called **sharding**, where data is distributed across multiple servers.
   * It provides **high-speed reads and writes**, making it ideal for applications that handle large volumes of data in real time.
4. **Built-in Support for Replication and High Availability:**
   * MongoDB ensures **data redundancy and fault tolerance** using **Replica Sets.**
   * If one server fails, another takes over automatically, ensuring that applications remain operational.
5. **Easier Querying with a Rich API:**
   * Unlike SQL, which requires complex JOIN operations to retrieve related data, MongoDB allows **embedded documents**, making queries more efficient.
   * It provides powerful built-in methods for **searching, filtering, and aggregating data** without requiring complex SQL joins.
6. **Automatic Indexing and Caching:**
   * MongoDB automatically creates **indexes** on the \_id field and allows developers to define custom indexes to speed up queries.
   * Frequently accessed data is cached in memory, improving performance.

Overall, MongoDB is an excellent choice for **web applications, microservices, real-time analytics, and big data solutions** due to its scalability, speed, and ease of use.

# Setting Up MongoDB Atlas (Cloud Database) (optional)

MongoDB Atlas is a **fully managed cloud database service** that allows developers to create and manage MongoDB databases without worrying about server infrastructure. It provides **automatic backups, security, and scalability**, making it the preferred choice for production applications.

### Step 1: Create a MongoDB Atlas Account

1. **Go to MongoDB Atlas**:
   * Open your web browser and visit [**https://www.mongodb.com/cloud/atlas**](https://www.mongodb.com/cloud/atlas).
2. **Sign Up or Log In**:
   * Click **"Sign Up"** if you don’t have an account.
   * Enter your **email, name, and password** or sign up using Google/GitHub.
   * If you already have an account, simply **log in**.

### Step 2: Create a Free Cluster

A **cluster** in MongoDB Atlas is a **group of MongoDB servers** that work together to **store data, ensure high availability, and provide scalability**.

When you create a cluster in Atlas, you're essentially setting up a **managed MongoDB database** in the cloud. This cluster consists of **multiple nodes (servers)** that help distribute the data and ensure that your database is always available.

**Key Features of a MongoDB Cluster:**

* **Redundancy & High Availability:** Data is replicated across multiple servers to prevent data loss.
* **Scalability:** Can handle large datasets and traffic increases efficiently.
* **Managed Service:** MongoDB Atlas automatically handles maintenance, backups, and monitoring.

For a **free-tier cluster**, MongoDB Atlas provides a **single primary node** with two secondary replicas for basic redundancy.

Once logged in:

1. **Click on "Create a New Project"** (Optional)
   * Give your project a name (e.g., My MongoDB Project).
   * Click **"Next"** and then **"Create Project"**.
2. **Create a Cluster**:
   * Click on **"Build a Cluster"**.
   * Select the **"Shared" (Free) option** under "Cluster Tier."
   * Choose a cloud provider and region:
     + Recommended: **AWS / Google Cloud / Azure** (Choose a location close to your users).
   * Give your cluster a name (default: **Cluster0**, but you can rename it).
   * Click **"Create Cluster"** (This may take a few minutes).

### Step 3: Configure Database Access

Once the cluster is created, you need to **set up database access**:

1. **Go to "Database Access"** in the left sidebar.
2. Click **"Add New Database User"**.
3. Choose **"Password"** as the authentication method.
4. Create a **username** and **password** (Save this for later use).
5. Click **"Add User"**.

### Step 4: Configure Network Access

To connect your application, you must **allow IP access**:

1. **Go to "Network Access"** in the left sidebar.
2. Click **"Add IP Address"**.
3. Select **"Allow Access from Anywhere"** (for development).
   * If deploying to production, add **specific IP addresses** for security.
4. Click **"Confirm"**.

### Step 5: Get the Connection String

1. **Go to "Database" → "Clusters"** → Click **"Connect"**.
2. Select **"Connect your application"**.
3. Copy the **connection string** (it will look like this):

|  |
| --- |
| mongodb+srv://your-username:your-password@cluster0.mongodb.net/myFirstDatabase?retryWrites=true&w=majority |

1. Replace your-username and your-password with the credentials you set earlier.
2. Use this **connection string** in your application.

### Step 6: Test Connection (Optional - Using Compass)

To test the connection locally, use **MongoDB Compass**:

MongoDB Compass is a **graphical user interface (GUI) tool** provided by MongoDB to help developers and database administrators **visually interact with their MongoDB databases**.

Instead of using the command line (mongo shell), Compass provides an **easy-to-use interface** to explore, query, and manage your data efficiently.

1. Download **MongoDB Compass**: <https://www.mongodb.com/try/download/compass>.
2. Open Compass and paste the **connection string**.
3. Click **"Connect"** to access the database.

By setting up MongoDB Atlas, you now have a **fully managed, cloud-hosted database** that can be accessed from anywhere. This eliminates the need for manual database management and provides an easy way to store and retrieve data in modern applications.

# Do I Need MongoDB Atlas for My CRUD API?

**No, MongoDB Atlas is not required for your CRUD API.** You have **two options** when working with MongoDB for your API:

### Option 1: Using MongoDB Atlas (Cloud Database)

Best if:

* You don’t want to set up and manage a local database.
* Your API needs to be accessible from anywhere.
* You’re deploying your app to a cloud platform like Vercel, Netlify, or Heroku.

### Option 2: Running MongoDB Locally

Best if:

* You want to test your API on your own machine without internet dependency.
* You prefer full control over your database configuration.
* Your API will run on a **local development environment** (like a personal laptop).

# Installing and Setting Up MongoDB

Follow these steps to install MongoDB on Windows and use mongosh to interact with the database.

## Step 1: Download MongoDB Community Edition

1. Go to the official MongoDB download page:  
   <https://www.mongodb.com/try/download/community>
2. Select **Windows** as the operating system.
3. Choose the **MSI package** (default option).
4. Click **Download** and wait for the file to be saved.

## Step 2: Install MongoDB

1. Locate the downloaded .msi file and double-click to start the installation.
2. In the setup wizard, select **"Complete"** installation.
3. On the **Service Configuration** screen, ensure the following options are checked:
   * **Run MongoDB as a service** (this allows MongoDB to start automatically).
   * Keep the default service name (**MongoDB**).
4. Click **Install** and wait for the process to finish.
5. Click **Finish** to close the installer.

## Step 3: Verify Installation

**Add MongoDB to Your System PATH**

Mongod is installed but not recognized, you need to add it to the system PATH.

### On Windows

1. Search for **"Environment Variables"** in the Start menu.
2. Under **System Variables**, find Path and click **Edit**.
3. Click **New** and add:

|  |
| --- |
| C:\Program Files\MongoDB\Server\8.0\bin |

1. Click **OK** and restart your terminal.

To check,

1. Open **Command Prompt (cmd)** and check if MongoDB is installed correctly by running:

|  |
| --- |
| mongod --version |

If installed successfully, it will display the version number.

## Installing MongoDB Shell (mongosh)

MongoDB stopped bundling mongosh with the main server installation. You need to download and install it manually:

1. **Download mongosh:**
   * Go to the [MongoDB Shell (mongosh) Download Page](https://www.mongodb.com/try/download/shell).
   * Select **Windows** and download the .msi installer.
2. **Install mongosh:**
   * Run the .msi installer and follow the on-screen instructions.
   * Make sure to install it in the recommended default location.
3. **Verify the Installation:**
   * Open **Command Prompt** and type

|  |
| --- |
| mongosh –version |

If successful, this will display the mongosh version.

1. **(Optional) Add mongosh to Environment Variables:**
   * If mongosh is still not recognized, add its installation directory (C:\Program Files\MongoDB\mongosh\bin) to your **PATH** environment variable.
   * Then, restart the Command Prompt and try again.

## Step 4: Start the MongoDB Server

### As a Background Process:

Manage MongoDB as a Windows service, running in the background and configured to start automatically with Windows. This method is more suitable for production environments.

1. Open **Command Prompt** as Administrator.
2. Run the following command to start the MongoDB service:
3. It will already be running as we have set it up to run as a background service. To start it again, stop it first.

|  |
| --- |
| net start MongoDB |

1. If you need to stop MongoDB:

|  |
| --- |
| net stop MongoDB |

## Step 5: Open MongoDB Shell (mongosh)

Since MongoDB is already running as a background service, you can directly start the mongo shell without starting the server manually.

1. Open **Command Prompt** and type:

|  |
| --- |
| mongosh |

1. If MongoDB is running, you will see an interactive prompt:

|  |
| --- |
| test> |

This means you are connected to the default test database.

## Start the MongoDB Server As a Foreground Process:

MongoDB runs as a background service, but you can manually start it if needed:

|  |
| --- |
| mongod --dbpath “C:\data\db” |

Starts the MongoDB server as a foreground process. The Command Prompt window must remain open, and closing it will stop the server.

**Default Data Directory**: MongoDB uses C:\data\db as the default directory to store data. Ensure this directory exists, or create it manually. Open File Explorer and navigate to the C: drive. Create the folders data\db.

--dbpath "C:\data\db"

This option specifies the location where MongoDB should store its database files. If you are using the default location itself you can skit this part. Otherwise you need to include it if you are choosing a different path.

As such, we can omit the path in the command:

|  |
| --- |
| mongod |

This starts the MongoDB server on the default port 27017.

## Difference Between mongod and mongosh

1. **mongod (MongoDB Daemon)**
   * This **starts the MongoDB server** (background process).
   * --dbpath is needed to specify where the database files are stored.
   * Must be running before you can connect to MongoDB.
2. **mongosh (MongoDB Shell)**
   * This is the **MongoDB interactive shell** (client) used to run queries.
   * It connects to an already running mongod instance.
   * No need for --dbpath; just start it with: mongosh
   * If MongoDB is running on a different port or host, use:

# 2. MongoDB Basics in mongosh

Once MongoDB is installed and running, you can use mongosh (MongoDB Shell) to interact with the database. Below are essential commands to help students understand how to create and manage databases, collections, and documents.

## 1. Starting MongoDB Shell (mongosh)

To open the MongoDB shell, run:

|  |
| --- |
| mongosh |

If MongoDB is running correctly, you should see the shell prompt:

test>

The test> prompt means MongoDB is connected to the default test database.

## 2. Working with Databases

### Show all databases

|  |
| --- |
| show dbs |

Initially, this may show only the default databases:

|  |
| --- |
| admin 0.000GB  config 0.000GB  local 0.000GB |

New databases appear only after inserting data.

### Create or switch to a database

Unlike SQL databases, MongoDB does not require an explicit "create database" command. Simply use:

|  |
| --- |
| use myDatabase |

If myDatabase does not exist, MongoDB will create it when you insert data.

### Check the current database

|  |
| --- |
| db |

This returns the active database name.

## 3. Working with Collections

### Show collections in the current database

|  |
| --- |
| show collections |

If no collections exist, nothing will be displayed.

### Create a collection (optional)

Collections are automatically created when you insert data, but you can manually create one:

|  |
| --- |
| db.createCollection("students") |

## 4. Working with Documents (CRUD Operations)

MongoDB stores data in **documents**, which are similar to JSON objects.

## Insert Data

### Insert a Single Document

|  |
| --- |
| db.students.insertOne({ name: "John Doe", age: 22, course: "Computer Science" }) |

Output:

|  |
| --- |
| {  acknowledged: true,  insertedId: ObjectId("654d2efb0c8d4d7b6c3d6e2a")  } |

### Insert Multiple Documents

|  |
| --- |
| db.students.insertMany([  { name: "Alice", age: 24, course: "Data Science" },  { name: "Bob", age: 21, course: "Software Engineering" }  ]) |

## Retrieve Data

### Get all documents in a collection

|  |
| --- |
| db.students.find() |

This will return:

|  |
| --- |
| [  { "\_id": ObjectId("654d2efb0c8d4d7b6c3d6e2a"), "name": "John Doe", "age": 22, "course": "Computer Science" },  { "\_id": ObjectId("654d2f0c0c8d4d7b6c3d6e2b"), "name": "Alice", "age": 24, "course": "Data Science" },  { "\_id": ObjectId("654d2f1d0c8d4d7b6c3d6e2c"), "name": "Bob", "age": 21, "course": "Software Engineering" }  ] |

### Retrieve documents with a filter

Find students enrolled in "Data Science":

|  |
| --- |
| db.students.find({ course: "Data Science" }) |

Find a single document:

|  |
| --- |
| db.students.findOne({ name: "Alice" }) |

## Update Data

### Update a single document

Change "John Doe’s" age to 23:

|  |
| --- |
| db.students.updateOne({ name: "John Doe" }, { $set: { age: 23 } }) |

### Update multiple documents

Increase the age of all students by 1:

|  |
| --- |
| db.students.updateMany({}, { $inc: { age: 1 } }) |

## Delete Data

### Delete a single document

|  |
| --- |
| db.students.deleteOne({ name: "Bob" }) |

### Delete multiple documents

|  |
| --- |
| db.students.deleteMany({ age: { $gt: 23 } }) |

### Drop (Delete) a collection

|  |
| --- |
| db.students.drop() |

## 5. Additional Useful Commands

### Count documents in a collection

|  |
| --- |
| db.students.countDocuments() |

### Sorting Results

Sort by age in ascending order:

|  |
| --- |
| db.students.find().sort({ age: 1 }) |

Descending order:

|  |
| --- |
| db.students.find().sort({ age: -1 }) |

### Limit Results

Show only 2 students:

|  |
| --- |
| db.students.find().limit(2) |

## 6. Exiting the MongoDB Shell

To exit mongosh, type:

|  |
| --- |
| exit |

# Problem: Inserting and Querying Data

### Objective:

Create a collection named students in a MongoDB database. Insert the following student records and then perform a few queries to retrieve specific information.

### Data to Insert:

1. { "name": "Alice", "age": 20, "major": "Computer Science" }
2. { "name": "Bob", "age": 22, "major": "Mathematics" }
3. { "name": "Charlie", "age": 21, "major": "Computer Science" }
4. { "name": "David", "age": 23, "major": "Physics" }

### Tasks:

1. Insert all four student records into the students collection.
2. Retrieve all students who are majoring in "Computer Science."
3. Find all students who are 21 years or older.
4. Update Bob's age to 23.
5. Delete the student record for "David."

# MongoDB in Go

Using MongoDB in Go for CRUD (Create, Read, Update, Delete) operations typically involves using the official MongoDB Go Driver (go.mongodb.org/mongo-driver). Here's a step-by-step guide:

## Step 1: Install MongoDB Go Driver

Run the following command to add the MongoDB driver to your project:

|  |
| --- |
| go get go.mongodb.org/mongo-driver/mongo |

## Step 2: Imports

When you use the driver, you can directly import the necessary packages like this:

|  |
| --- |
| import (  "go.mongodb.org/mongo-driver/bson"  "go.mongodb.org/mongo-driver/bson/primitive"  "go.mongodb.org/mongo-driver/mongo"  "go.mongodb.org/mongo-driver/mongo/options"  ) |

The driver is modular, but the mongo package will handle all dependencies for you. So, the single go get command will suffice.

**1. go.mongodb.org/mongo-driver/bson**

**Purpose:** bson is used to handle BSON (Binary JSON), which is MongoDB's internal data format.

**2. go.mongodb.org/mongo-driver/bson/primitive**

**Purpose:** primitive provides types for BSON values that don't have native Go representations. Common types include:

**3. go.mongodb.org/mongo-driver/mongo**

**Purpose:** This is the main MongoDB driver package. It provides the core functionality for:

**4. go.mongodb.org/mongo-driver/mongo/options**

**Purpose:** options is used to configure various MongoDB client settings. You’ll commonly use it to:

### Why You See All These Imports

Most CRUD operations in MongoDB with Go will require all of these imports because they work together to handle data conversion, connections, and database interactions. So, if you're doing full CRUD operations, it’s typical to see these imports grouped in your code.

## Step 3: Connecting to MongoDB

### 1. MongoDB Configuration Variables

|  |
| --- |
| var mongoUri string = "mongodb://localhost:27017"  var mongoDbName string = "taskmanager\_db"  var mongoCollectionTask string = "tasks"  var taskCollection \*mongo.Collection |

* **mongoUri**: The connection string for MongoDB. mongodb://localhost:27017: Connects to a MongoDB instance running locally on the default port 27017.
* **mongoDbName**: The name of the database to be used (taskmanager\_db).
* **mongoCollectionTask**: The name of the collection where tasks are stored.
* **taskCollection**: A global variable to hold the collection reference. This allows the application to reuse the same collection reference across multiple operations.

### 2. Connecting to MongoDB

|  |
| --- |
| client, err := mongo.Connect(context.TODO(), options.Client().ApplyURI(mongoUri)) |

mongo.Connect:

* This function is used to create and establish a connection to the MongoDB server.
* context.TODO(): A context is passed to manage the connection (and any potential timeouts or cancellations). context.TODO() is a placeholder when no specific context is needed.
* The context allows you to **control how long the connection attempt lasts**.
* If a context with a timeout is used, you can cancel or stop the connection attempt after a certain amount of time.
* options.Client().ApplyURI(mongoUri): Creates a new client configuration and applies the MongoDB connection URI.

### 3. Error Handling for Connection

|  |
| --- |
| if err != nil {  log.Fatal("MongoDB Connection Error:", err)  } |

If the connection to MongoDB fails, the program logs the error and exits. This is important because the application cannot function without a successful database connection.

### 4. Pinging the Database to Verify Connection

|  |
| --- |
| err = client.Ping(context.TODO(), nil)  if err != nil {  log.Fatal("Failed to ping MongoDB:", err)  } |

* client.Ping: This checks if the MongoDB server is reachable and that the connection is functional.
* If the ping fails, the application logs the error and exits.

### 5. Assigning the Collection

|  |
| --- |
| taskCollection = client.Database(mongoDbName).Collection(mongoCollectionTask) |

client.Database(mongoDbName).Collection(mongoCollectionTask):

* This accesses the taskmanager\_db database and retrieves the tasks collection.
* The taskCollection variable now holds a reference to this collection, which can be used for CRUD operations (insert, find, update, delete) on the tasks data.

### 6. Log Connection Success

|  |
| --- |
| log.Println("Connected to MongoDB!") |

Logs a success message to indicate the connection was established.

### Flow Summary

1. The function connectDB is responsible for connecting to MongoDB using the provided URI.
2. It creates a client and pings the server to ensure connectivity.
3. Once the connection is verified, it sets the collection reference (taskCollection) for use in CRUD operations.

## Step 4: Ensure Connection

To ensure connection,