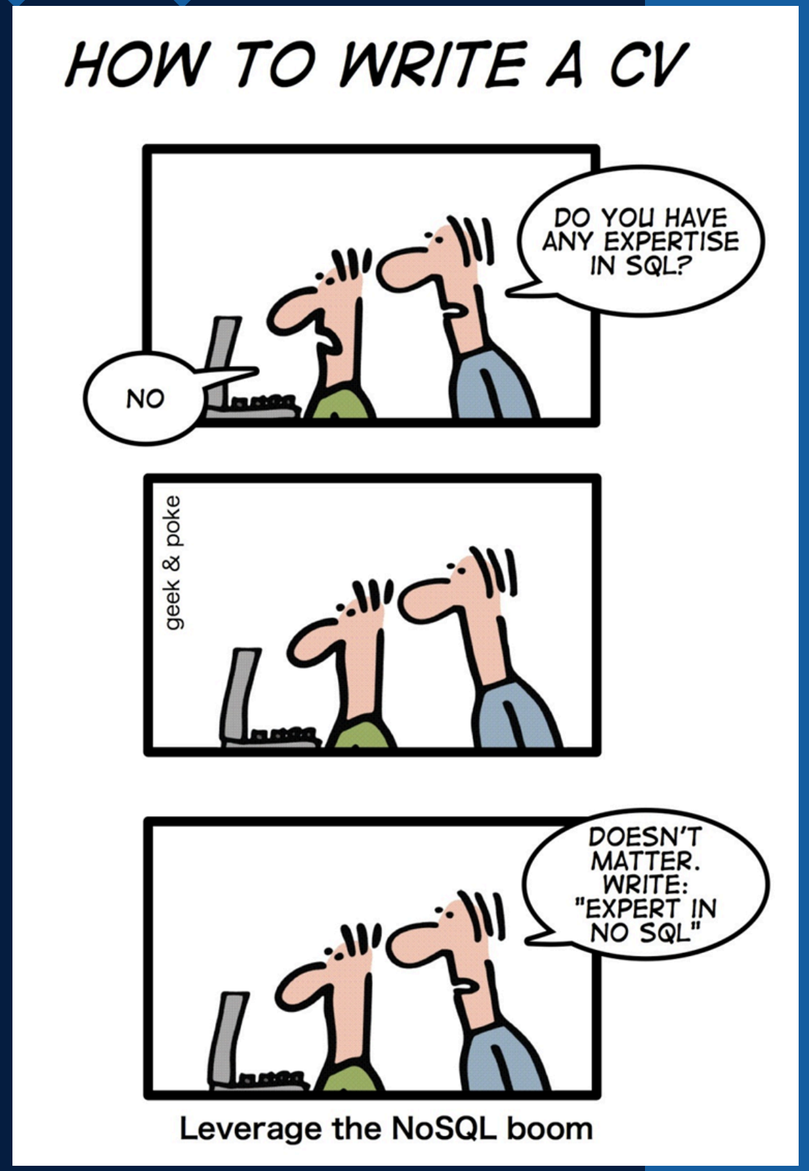
INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

S25CS6.201
Introduction to Software Systems

NO SQL

CRUD Operations - MongoDB

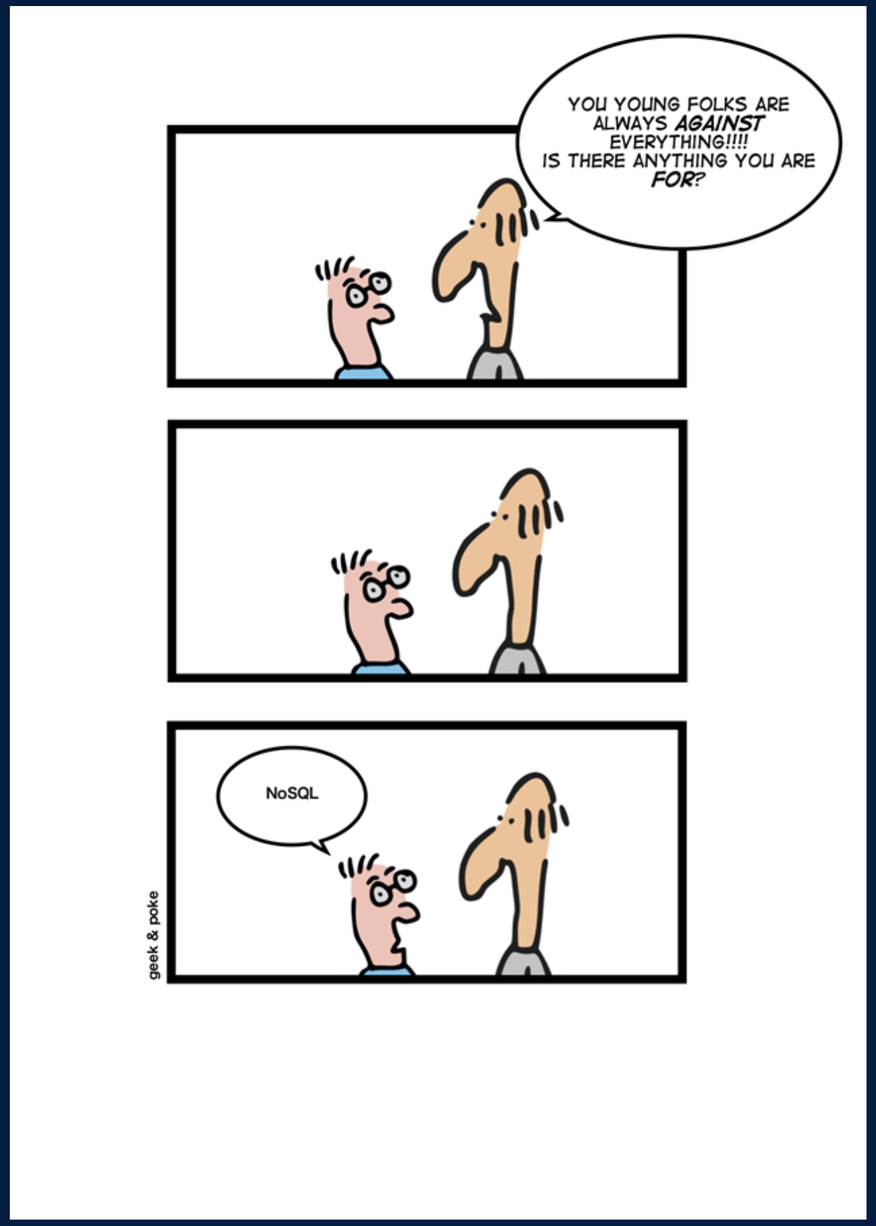
12th February 2025



Relational Databases

We learned relational databases and SQL.

So Why??? we need NoSQL



Limitations of Relational Databases

- SQL databases scale vertically (adding more power to a single server), which becomes expensive and hits hardware limits.
- Requires a predefined schema. Adding/altering columns in large tables is slow and disruptive.
- Designed for structured data (tabular format). Struggles with unstructured (e.g., JSON, XML) or nested data.
- ACID transactions (atomicity, consistency) create bottlenecks for high-velocity data (e.g., real-time analytics).
- Schema changes require migrations, slowing development cycles.

These many issues? No Worries
NoSQL helps in solving (at least reducing) these problems.....

- NoSQL scales **horizontally** (adding more commodity servers/nodes), distributing data across clusters for massive scalability (e.g., handling petabytes of data).
- **Schema-less/flexible schema** allows dynamic addition of fields (e.g., JSON/BSON documents in MongoDB).
- Supports **document stores** (MongoDB), **key-value pairs** (Redis), **graph databases** (Neo4j), and **wide-column stores** (Cassandra).
- Optimized for **high-speed writes/reads** using eventual consistency (BASE - ”Basically Available, Soft State, Eventually Consistent” model).
- **Flexible schema** allows rapid iteration (e.g., adding new fields without downtime).

What is NoSQL Database?

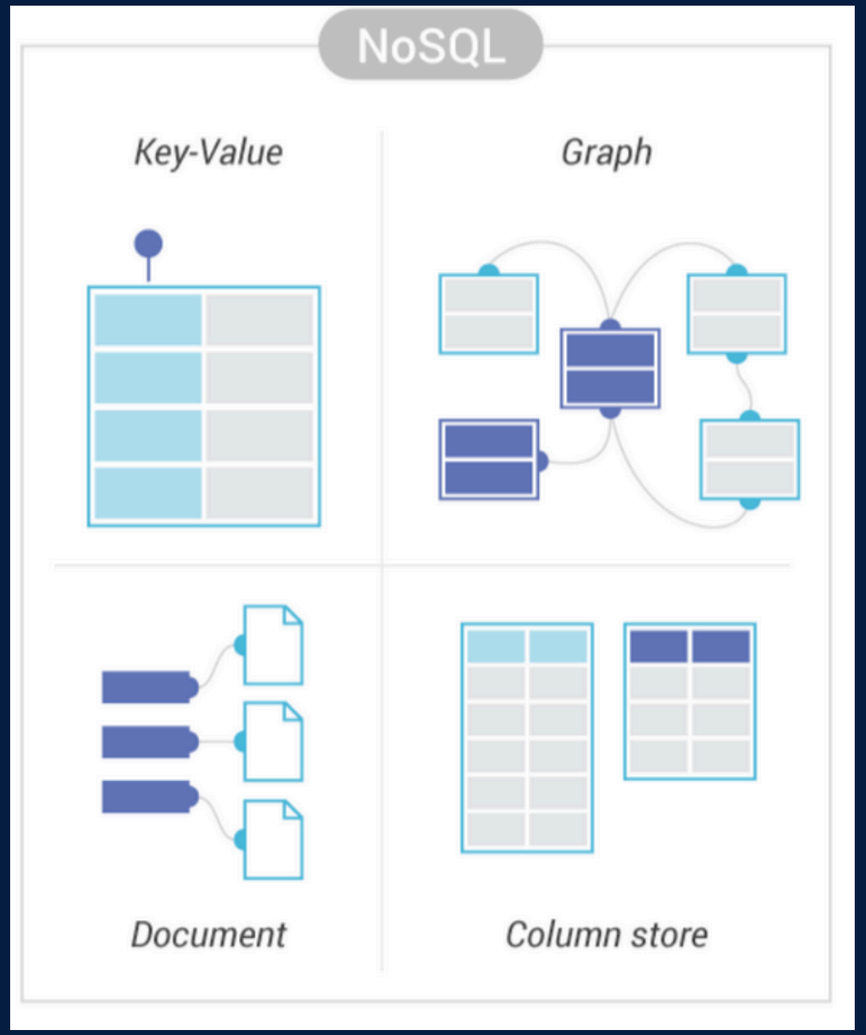
A NOSQL database (Not Only SQL) is a type of database management system that provides a mechanism for storing, retrieving, and managing data that does not follow the traditional relational database model.

Unlike relational databases, NoSQL databases are designed to handle unstructured, semi-structured, or structured data, providing greater flexibility and scalability for certain types of applications.

NoSQL Database Types

NoSQL databases can be broadly categorized into four main types, each with its own unique data model and characteristics. These categories are based on the way data is organized and stored within the database.

1. **Document-oriented database** - pair each key with a complex data structure known as a document.
2. **Key-Value stores** - are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value.
3. **Column-oriented database** - Optimized for queries over large datasets, and store columns of data together, instead of rows
4. **Graph-based database** - are used to store information about networks of data, such as social connections.



Document-oriented database

Document: BSON/JSON structure (e.g., { name: "Alice", age: 25 }).
Collection: Group of documents (like SQL tables).
Database: Container for collections.
Key Terms: _id (unique identifier), embedded fields.

Example:

```
{
  _id: ObjectId("5099803df3f4948bd2f983a0"),
  title: "MongoDB Basics",
  author: "Dileep",
  tags: ["NoSQL", "Database"]
}
```

Relational Database	Document Database
Database	Database
Table	Collection
Row	Document (JSON, BSON)
Column	Field
Index	Index
Join	Embedded Document
Partition	Shard

CRUD Operations Overview

Create:

insertOne(), insertMany().

Read:

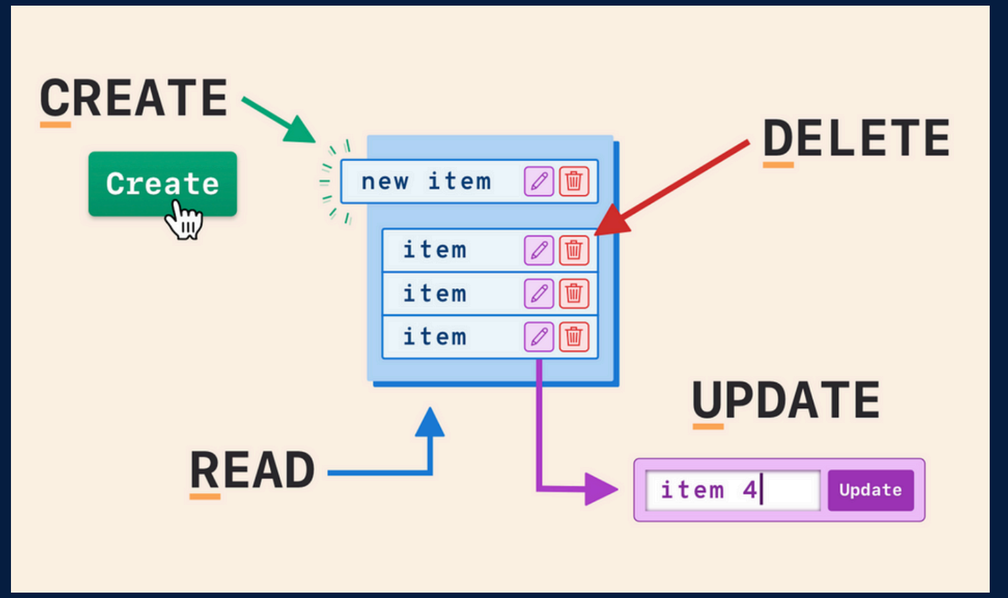
find(), findOne(), query operators ($gt, $in).

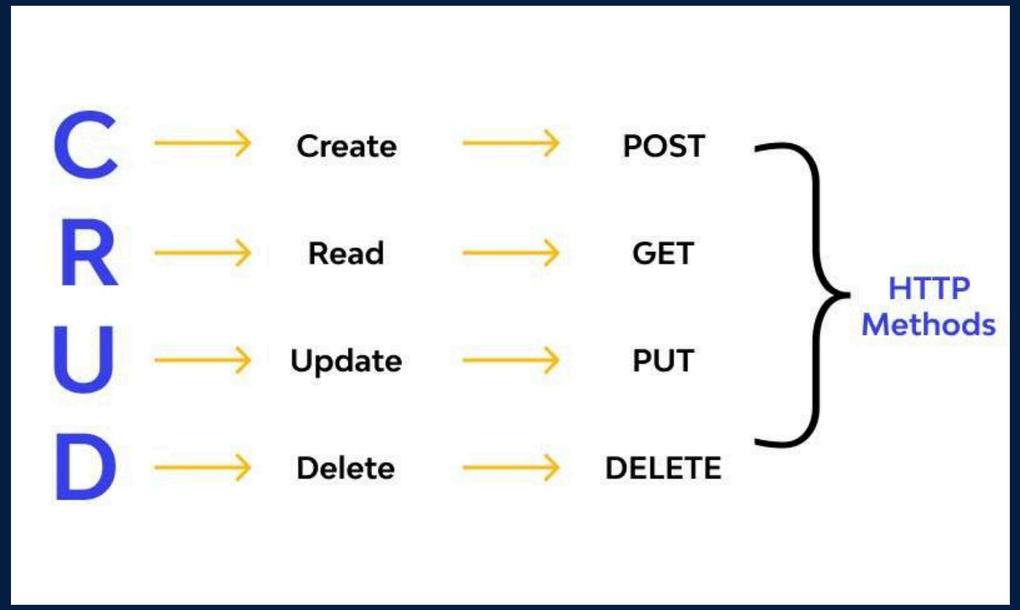
Update:

updateOne(), updateMany(), $set, $unset.

Delete:

deleteOne(), deleteMany().





Create Operations

Insert a single document:

```
db.books.insertOne({
  title: "The Lord of the Rings",
  author: "J.R.R. Tolkien",
  year: 1954
});
```

Insert multiple documents:

```
db.books.insertMany([{...}, {...}]);
```

Read Operations

Basic Query:

```
// Find all documents in the 'users' collection
db.users.find();
```

```
db.books.find({ author: "J.R.R. Tolkien" });
```

With Operators:

```
db.books.find({ year: { $gt: 2000 } }); // Books published after 2000
```

```
db.users.find({
  $and: [
    { age: { $gte: 18 } },
    { role: "customer" }
  ]
})
```

```
db.orders.find({ status: { $in: ["shipped", "delivered"] } })
```

```
// Find orders NOT in 'pending' status
db.orders.find({
  status: { $ne: "pending" }      // $ne = Not equal
});
```

Projection:

```
db.books.find({}, { title: 1, author: 1 }); // Return only title & author
```

Sorting and Pagination:

```
// Sort users by 'createdAt' in descending order (newest first)
db.users.find().sort({ createdAt: -1 });

// Pagination: Skip first 20 results, return next 10
db.products.find()
  .skip(20)    // Skip first 20 documents
  .limit(10);  // Return 10 documents
```

Update Operations

Update a field:

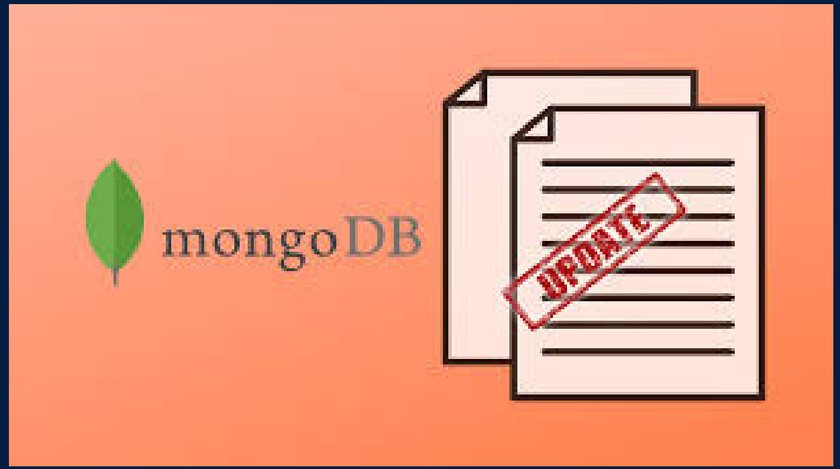
```
db.books.updateOne(
  { title: "The Hobbit" },
  { $set: { year: 1954 } }
);
```

Add/Remove fields: ($unset to remove)

```
db.books.updateMany(
  { genre: "Fantasy" },
  { $set: { publisher: "Allen & Unwin" } }
);
```

Warning:

Be careful on accidental updates without filters.



Delete Operations

Delete a document:

```
db.books.deleteOne({ title: "The Great Gatsby" });
```

Delete multiple documents:

```
db.books.deleteMany({ year: { $lt: 1900 } }); // Delete books before 1900
```

Always double-check filters before deleting!

Nested Object

```
// Insert a user with a nested 'address' object
db.users.insertOne({
  name: "Alice",
  age: 30,
  address: {      // Nested object
    street: "123 Main St",
    city: "New York",
    country: "USA"
  }
});
```

```
// Insert an order with nested items in an array
db.orders.insertOne({
  orderId: "ORD123",
  customer: "Bob",
  items: [        // Array of nested objects
    { product: "Laptop", price: 999, quantity: 1 },
    { product: "Mouse", price: 25, quantity: 2 }
  ],
  total: 1049
});
```

```
// Insert a blog post with deeply nested comments/replies
db.posts.insertOne({
  title: "MongoDB Guide",
  author: "Charlie",
  comments: [     // Array of nested objects with their ow
    {
      user: "Dave",
      text: "Great post!",
      replies: [  // Nested array inside a nested object
        { user: "Eve", text: "Agreed!" },
        { user: "Frank", text: "Thanks!" }
      ]
    },
    {
      user: "Grace",
      text: "Very helpful."
    }
  ]
});
```

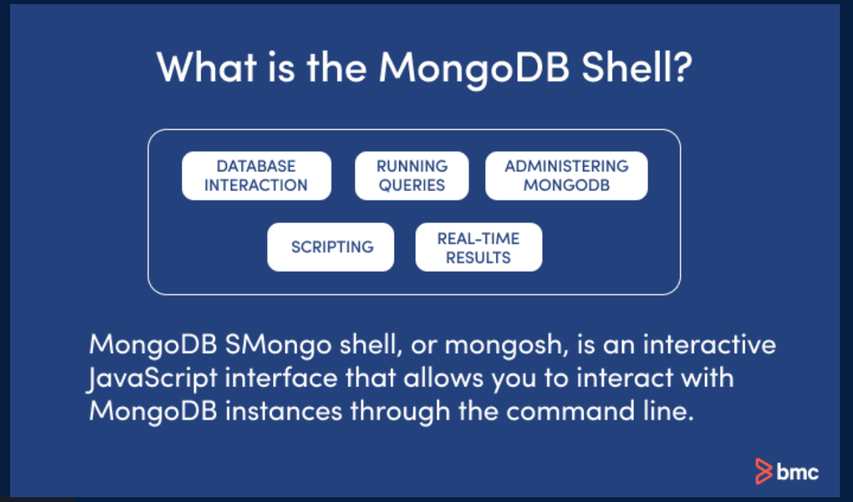
Ways to Access MongoDB

MongoDB Shell (mongosh)

Command-line interface for interacting with MongoDB.

Syntax:

```
mongosh "mongodb+srv://<username>:<password>@cluster0.mongodb.net/<database>"
```



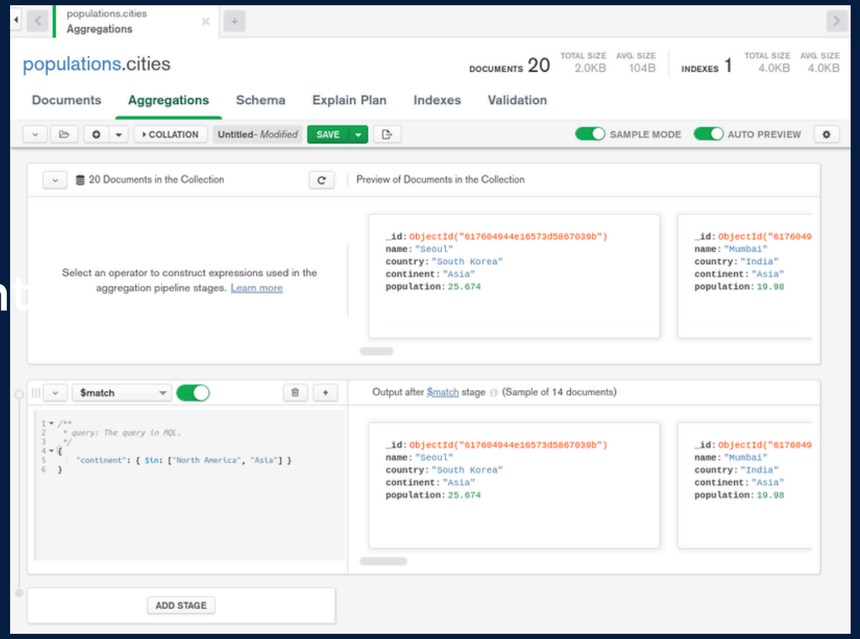
MongoDB Compass (GUI)

Graphical User Interface for visual database management.

Features: Schema analysis, query building, index managemen

If you want to Download:

https://www.mongodb.com/products/tools/compass



Language-specific libraries to integrate MongoDB into applications:

Python: pymongo

```
from pymongo import MongoClient
client = MongoClient("mongodb+srv://<username>:<password>@cluster0.mongodb.net/")
db = client["mydatabase"]
```

Node.js: mongodb npm package

```
const { MongoClient } = require('mongodb');
const client = new MongoClient("mongodb+srv://<username>:<password>@cluster0.mongodb.net/");
```

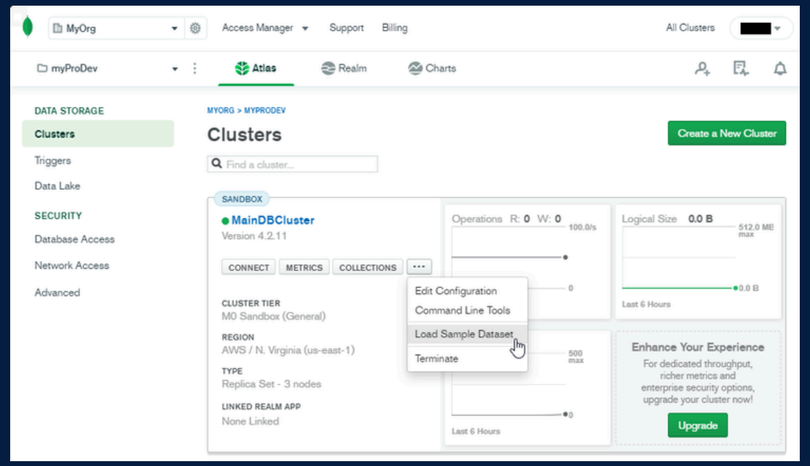
MongoDB Atlas Web UI

Cloud-based interface for managing clusters, users, and security.

Features: Real-time monitoring, backups, alerts.

Website:

https://www.mongodb.com/products/platform/atlas-database



Hoping Everyone setup their Atlas setup,

If Not, these are the steps:

Step 1: Create an Account

- Go to “https://www.mongodb.com/products/platform/atlas-database”
- Create an Atlas Account

Step 2: Create a Project

- Click New Project → Name it (e.g., Lab4-NoSQL).

Step 3: Build a Cluster

- Click Build a Database.
- Choose Shared Cluster (Free Tier).
- Select Cloud Provider & Region (e.g., AWS, Mumbai).
- Click Create Cluster (takes 1-3 minutes).

step 4: (contd..)

Step 4: Set Up Security

- **Database Access:**
  - Go to Database Access → Add New Database User.
  - Choose Password Authentication → Set username (e.g., lab4-user) and password.
  - Assign privileges: Read and write to any database.
- **Network Access:**
  - Go to Network Access → Add IP Address.
  - Whitelist your IP: Click Allow Access from Anywhere (0.0.0.0/0) for testing (not recommended for production).

Step 5: Connect to the Cluster

- Go to Database → Connect.
- Choose a connection method:
  - **MongoDB Shell**: Copy the connection string and run it in terminal
    - if it says no mongosh, go to the mongodb site and install mongo shell

```
mongosh "mongodb+srv://lab4-user:password@cluster0.mongodb.net/"
```

Confused about how to write queries in shell?

1. Basic Commands

- **Connect to a MongoDB instance:**

bash Copy

```bash
mongosh "mongodb://localhost:27017"  # Local connection
mongosh "mongodb://username:password@host:port/dbname"  # Remote with auth
```

- **Show databases:**

javascript Copy

```javascript
show dbs
```

- **Switch/Create a database:**

javascript Copy

```javascript
use mydb  // Creates or switches to `mydb`
```

- **Show collections:**

javascript Copy

```javascript
show collections
```

Importing a Collection (mongoimport)

Import a JSON File

```
mongoimport  --uri="mongodb://localhost:27017"   --db mydatabase     --collection mycollection  --file data.json --jsonArray
```

- `--uri="mongodb://localhost:27017"` → Connects to the local MongoDB server.
- `--db mydatabase` → Specifies the database name.
- `--collection mycollection` → Specifies the collection name.
- `--file data.json` → Specifies the JSON file to import.
- `--jsonArray` → Required if the file contains an array of JSON documents.

Exporting a Collection (mongoexport)

Export as JSON

```
mongoexport --uri="mongodb://localhost:27017" --db mydatabase --collection mycollection --out data.json --jsonArray
```

- `--out data.json` → Specifies the output file.
- `--jsonArray` → Exports data as an array of JSON documents.

Time for the Lab Activity
Happy Coding :)