Introduction to MongoDB

"MongoDB is a document-based NoSQL database that offers high performance, scalability, and flexibility for handling large volumes of unstructured data."

What is MongoDB?

MongoDB is a popular open-source NoSQL (Not Only SQL) database system that stores data in flexible, JSON-like documents rather than rigid tables like MySQL.

This makes it ideal for applications where the data structure may evolve over time — such as social media platforms, real-time analytics systems, and content management systems.

Why MongoDB Was Created

A Brief History

MongoDB was created by Dwight Merriman and Eliot Horowitz in 2007 while they were working on a cloud-based advertising platform called *DoubleClick* (later acquired by Google).

They faced limitations with traditional relational databases when trying to scale their application and handle complex, evolving data structures.

So, they decided to build a new kind of database — one that could:

- Scale horizontally across multiple servers
- Handle semi-structured and unstructured data
- Provide better performance for modern web applications

In 2009, MongoDB was released as an open-source project under the name "10gen". Later, the company became MongoDB Inc. , and MongoDB remains one of the most widely used databases today.

Key Features of MongoDB

FEATURE	DESCRIPTION
Flexible Schema	Documents don't require a predefined schema, so fields can vary between documents in the same collection.
High Performance	Optimized for fast read/write operations, especially with large datasets.
Scalability	Easily scales out using sharding (data distribution across multiple machines).
Rich Query Language	Supports powerful queries, indexing, and aggregation pipelines.
Replication & High Availability	Built-in replication ensures data redundancy and failover support.
Geospatial Support	Handles location-based queries efficiently.

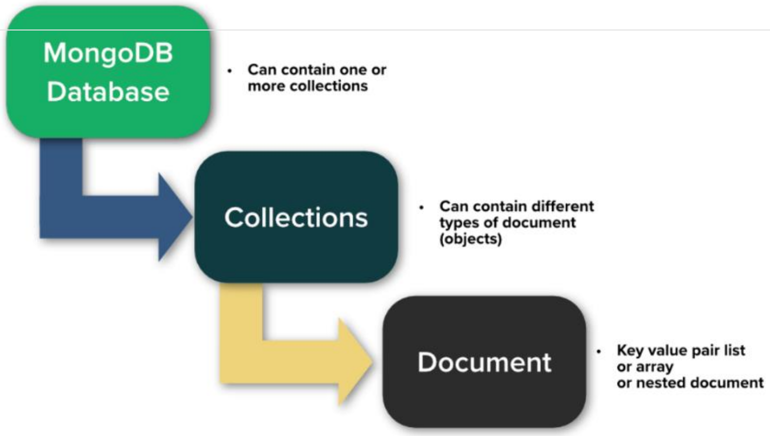
How MongoDB Differs from MySQL

Since you're familiar with MySQL , let's compare MongoDB and MySQL side-by-side to highlight the key differences:

ASPECT	MYSQL (RELATIONAL DB)	MONGODB (NOSQL DOCUMENT DB)
Data Model	Tables with rows and columns	Collections with documents (BSON format)
Schema	Fixed schema required	Flexible schema – each document can be different
Joins	Uses JOINs to relate data across tables	Embedded documents reduce need for joins
Scaling	Vertical scaling (increase power of single server)	Horizontal scaling (add more servers)
Use Case	Financial systems, transactional apps	Big data, real-time apps, content management
Transactions	Strong ACID compliance	Multi-document transactions supported (from v4.0+)

MongoDB Architecture Overview

MongoDB uses a document-based model , which means data is stored in documents (like JSON objects), grouped into collections , which are inside databases .


MongoDB Database
• Can contain one or more collections
Collections
• Can contain different types of document (objects)
Document
• Key value pair list or array or nested document

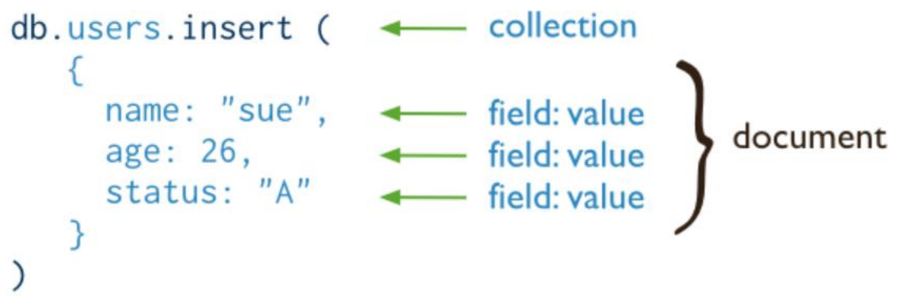
Visual Representation

```
MongoDB Server
│
└── Database: school_db
      │
      ├── Collection: students
      │     └── Document 1: { name: "Alice", age: 20, grades: [85, 90] }
      │     └── Document 2: { name: "Bob", age: 22, major: "CS" }
      │
      └── Collection: teachers
            └── Document 1: { name: "Mr. Smith", subject: "Math", experience: 10
}
```

Document	: A single unit of data, similar to a row in a table.
Collection	: A group of documents, similar to a table.
Database	: A container for collections, just like in MySQL.

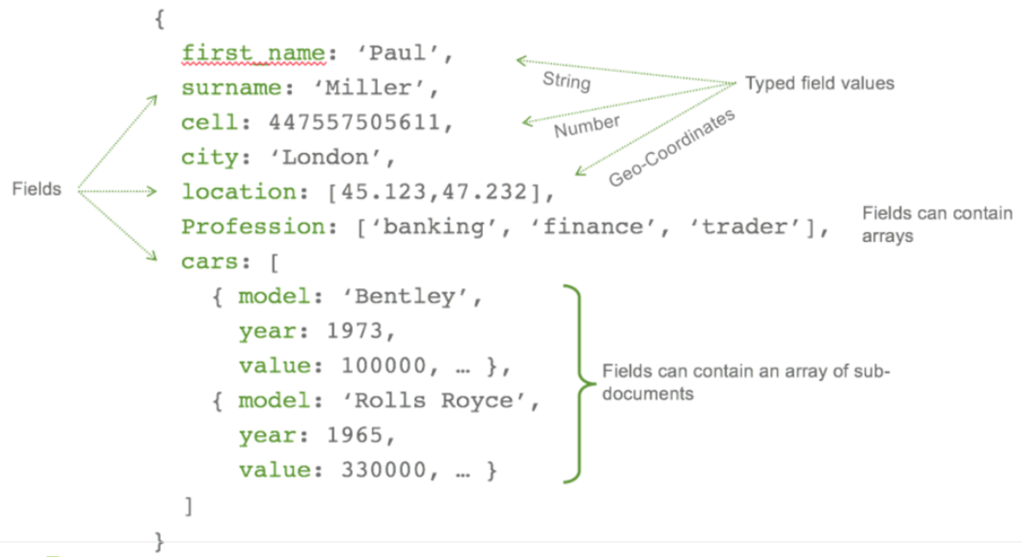
The **document**, which is comprised of field/value pairs, is at the heart of the MongoDB data structure. Most interactions with MongoDB occur at the document level.

A **field** can contain a single value, multiple fields, or multiple elements.



```
db.users.insert (
   {
      name: "sue",
      age: 26,
      status: "A"
   }
)
```

A value made up of multiple fields is referred to as an **embedded document** and is assigned the Object data type (see field cars in the screenshot). When rendered in the JSON format, an embedded document is enclosed in curly braces and adheres to the same structure as the outer (main) document.



```
{
  first_name: 'Paul',
  surname: 'Miller',
  cell: 447557505611,
  city: 'London',
  location: [45.123,47.232],
  Profession: ['banking', 'finance', 'trader'],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  ]
}
```

Fields
String
Number
Geo-Coordinates
Typed field values
Fields can contain arrays
Fields can contain an array of sub-documents

Mongo Data types

Data Type	Description	Example
String	Text values	"name": "Alice"
Number (Int)	Whole numbers (32-bit or 64-bit)	"age": 25
Double	Decimal numbers	"price": 19.99
Boolean	true or false	"isStudent": true
Array	List of values	"skills": ["JS", "Mongo", "HTML"]
Object	Embedded document (sub-document)	"address": { "city": "Nairobi" }
Null	A field with no value	"middleName": null
Date	Date/time in ISO format	"createdAt": new Date()
ObjectId	Unique ID auto-generated by MongoDB	"_id": ObjectId("...")
Binary Data	Used for storing files/data like images	Not common in beginner use cases
Timestamp	Internal Mongo timestamps (rarely user-facing)	"created": Timestamp()

📈 When Should You Use MongoDB?

MongoDB shines in scenarios where:

- The data doesn't fit neatly into tables.
- The schema changes frequently.
- You expect high traffic or massive amounts of data.
- You need flexible querying and real-time analytics.
- You're building modern web or mobile apps with dynamic user inputs.

✅ Common Use Cases:

- Social networking apps
- Real-time chat systems
- Content management systems (CMS)
- IoT (Internet of Things) data storage
- E-commerce product catalogs

⚖️ MongoDB vs MySQL: A Student Example

Let’s say you're storing student information.

✅ In MySQL:

You'd have a `students` table with fixed columns like `id`, `name`, `age`, `major`.

```
CREATE TABLE students (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  age INT,
  major VARCHAR(50)
);
```

All rows must follow this structure.

✅ In MongoDB:

Each student can have different fields.

```
{
  "_id": ObjectId("..."),
  "name": "Alice",
  "age": 20,
  "grades": [85, 90],
  "courses": ["Math", "History"]
}
```

Another student might look like:

```
{
  "name": "Bob",
  "age": 22,
  "major": "Computer Science"
}
```

💡 There's no need to define every field upfront. You can add or remove fields per document.

Why Learn MongoDB?

As a student, learning MongoDB opens up many opportunities because:

- It's widely used in industry and startups.
- It integrates well with modern development stacks like MERN (MongoDB, Express, React, Node.js).
- It supports cloud-native applications and microservices.
- It allows faster prototyping due to its flexible schema design.
- It's great for projects involving big data, AI, and real-time processing.

Other NoSql DB

mongoDB
Apache CASSANDRA
APACHE HBASE
amazon DynamoDB
neo4j
RAVENDB
CouchDB relax
redis
HYPERTABLE INC
OrientDB
Google Cloud Bigtable

Summary

- MongoDB is a NoSQL, document-based database that stores data in flexible JSON-like documents.
- Unlike MySQL, it doesn't require a fixed schema, making it ideal for evolving data models.
- It supports horizontal scaling, high availability, and rich query capabilities.
- MongoDB is widely used in modern applications and complements technologies like Node.js and React.
- For students transitioning from MySQL, MongoDB offers a fresh perspective on how data can be structured and accessed.

Next Steps

Now that you understand what MongoDB is and why it exists, you'll explore:

- Installing MongoDB
- Working with MongoDB Compass (GUI tool)
- Inserting and querying documents
- Using MongoDB with JavaScript (Node.js)

What You'll Need

TOOL	DESCRIPTION
MongoDB Community Server	The core database system.
MongoDB Compass	A graphical user interface (GUI) to interact with MongoDB without writing code.
Terminal / Command Prompt	To run MongoDB services and shell commands.

Students familiar with MySQL can think of MongoDB Compass as something like *MySQL Workbench*.

📥 Step 1: Download MongoDB

1. Go to the Official Website

Visit: https://www.mongodb.com/try/download/community

Choose the correct version based on your operating system (Windows, macOS, or Linux).

2. Select Package Type

- Choose MSI Installer (for Windows)
- Choose .tgz or .pkg for macOS
- Choose .tar.gz or package manager for Linux

⚙ Step 2: Install MongoDB

On Windows:

1. Run the downloaded `.msi` file.
2. Follow the setup wizard.
3. When asked where to install, choose a simple path like `C:\mongodb`.
4. Do NOT install MongoDB as a service unless you're advanced.
5. Complete the installation.

On macOS:

1. Extract the `.tgz` file or install via Homebrew:

```
brew tap mongodb/brew && brew install mongodb-community@6.0
```

2. Create a data directory:

```
mkdir -p /data/db
```

On Linux:

Use your package manager or extract the tarball:

```
sudo tar -zxvf mongodb-linux-x86_64-ubuntu2004-*.tgz
```

Move it to `/usr/local` and set up environment variables.

Step 3: Start MongoDB Server

Open a terminal or command prompt and type:

```
mongod
```

This starts the MongoDB database server.

If you see messages like:

```
waiting for connections on port 27017
```

✅ Great! MongoDB is running successfully.

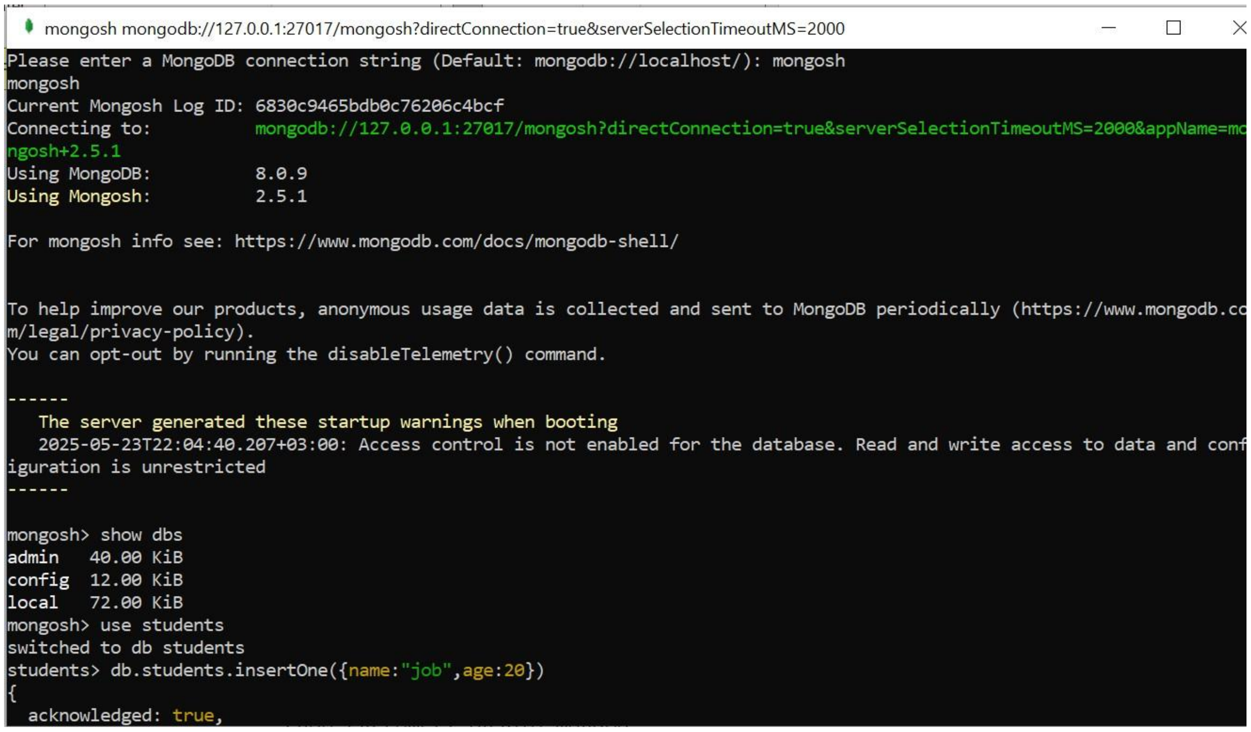
Port `27017` is the default MongoDB port, similar to port `3306` in MySQL.

Linux commands

```
curl -fsSL https://pgp.mongodb.com/server-7.0.asc | sudo gpg --dearmor
-o /usr/share/keyrings/mongodb-server-7.0.gpg

echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-
server-7.0.gpg ] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-
org/7.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-
7.0.list


sudo apt update

sudo apt install -y mongodb-org

sudo systemctl enable mongod
sudo systemctl start mongod

sudo systemctl status mongod
```

mongosh "mongodb+srv://employee:SlqdA4X71a50aszj@cluster0.zhracze.mongodb.net/userdb"

Windows



```
mongosh mongodb://127.0.0.1:27017/mongosh?directConnection=true&serverSelectionTimeoutMS=2000
Please enter a MongoDB connection string (Default: mongodb://localhost/): mongosh
mongosh
Current Mongosh Log ID: 6830c9465bdb0c76206c4bcf
Connecting to:          mongodb://127.0.0.1:27017/mongosh?directConnection=true&serverSelectionTimeoutMS=2000&appName=mo
ngosh+2.5.1
Using MongoDB:          8.0.9
Using Mongosh:          2.5.1

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


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.co
m/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

------
   The server generated these startup warnings when booting
   2025-05-23T22:04:40.207+03:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------

mongosh> show dbs
admin   40.00 KiB
config  12.00 KiB
local   72.00 KiB
mongosh> use students
switched to db students
students> db.students.insertOne({name:"job",age:20})
{
  acknowledged: true,
```

MongoDB on Linux

Starting MongoDB on Linux

Start MongoDB service:

```
sudo systemctl start mongod
```

Check status of MongoDB service:

```
sudo systemctl status mongod
```

Restart MongoDB (if needed):

```
sudo systemctl restart mongod
```

Accessing MongoDB Shell

Open MongoDB shell:

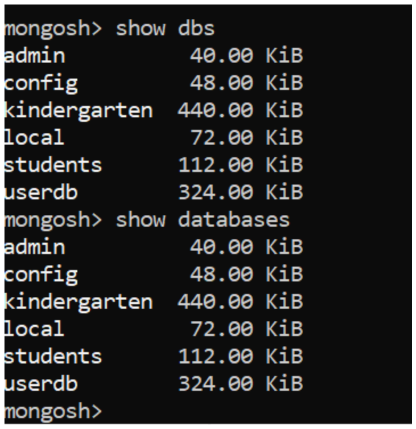
```
mongosh
```

This opens the default `test` database.

Working with Databases

Show all databases:

```
show dbs
// or
show databases
```

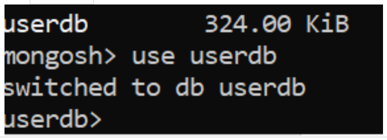


```
mongosh> show dbs
admin           40.00 KiB
config          48.00 KiB
kindergarten   440.00 KiB
local           72.00 KiB
students       112.00 KiB
userdb         324.00 KiB
mongosh> show databases
admin           40.00 KiB
config          48.00 KiB
kindergarten   440.00 KiB
local           72.00 KiB
students       112.00 KiB
userdb         324.00 KiB
mongosh>
```

Switch to or create a database:

```
use userdb
```

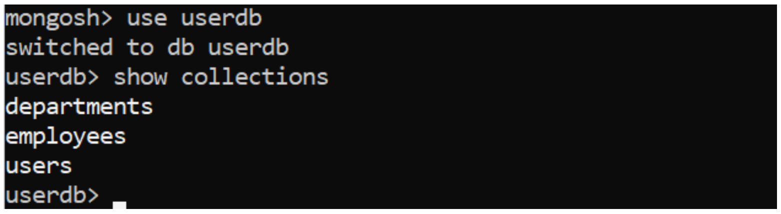
If `mydb` doesn't exist, it will be created when you insert data.



```
userdb          324.00 KiB
mongosh> use userdb
switched to db userdb
userdb>
```

Show collections in current database:

```
show collections
```



```
mongosh> use userdb
switched to db userdb
userdb> show collections
departments
employees
users
userdb>
```

Drop a collection:

```
db.collectionName.drop()
```

Rename a collection:

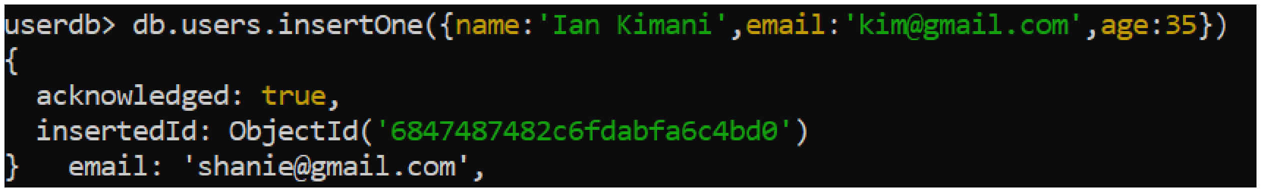
```
db.oldName.renameCollection("newName")
```

CRUD Operations – Create, Read, Update, Delete

✓ 1. Create (Insert Documents)

Insert one document:

```
db.collectionName.insertOne({ name: "Alice", age: 25 })
```



```
userdb> db.users.insertOne({name:'Ian Kimani',email:'kim@gmail.com',age:35})
{
  acknowledged: true,
  insertedId: ObjectId('6847487482c6fdabfa6c4bd0')
}   email: 'shanie@gmail.com',
```

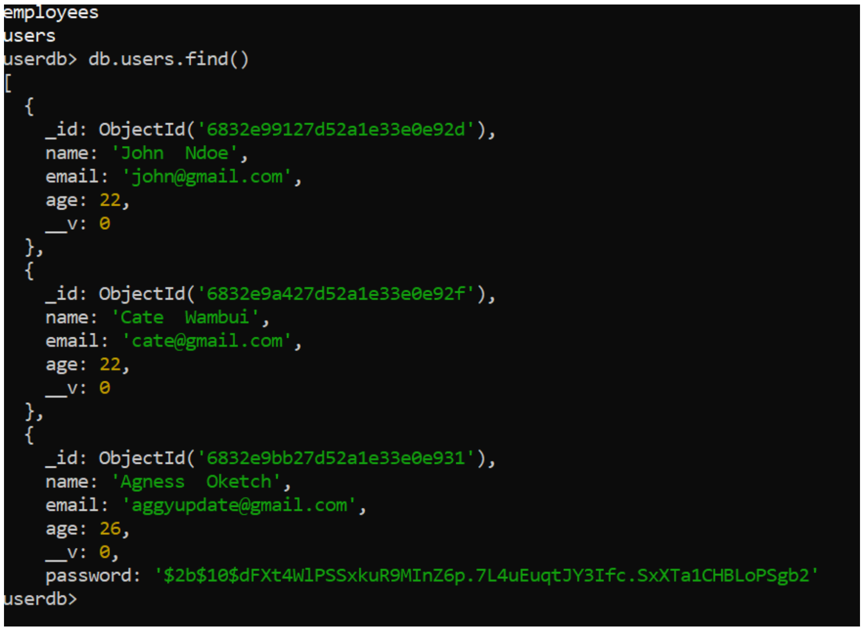
Insert multiple documents:

```
db.collectionName.insertMany([
  { name: "Bob", age: 30 },
  { name: "Charlie", age: 35 }
])
```

2. Read (Query Documents)

Get all documents in a collection:

```
db.collectionName.find()
```



```
employees
users
userdb> db.users.find()
[
  {
    _id: ObjectId('6832e99127d52a1e33e0e92d'),
    name: 'John  Ndoe',
    email: 'john@gmail.com',
    age: 22,
    __v: 0
  },
  {
    _id: ObjectId('6832e9a427d52a1e33e0e92f'),
    name: 'Cate  Wambui',
    email: 'cate@gmail.com',
    age: 22,
    __v: 0
  },
  {
    _id: ObjectId('6832e9bb27d52a1e33e0e931'),
    name: 'Agness  Oketch',
    email: 'aggyupdate@gmail.com',
    age: 26,
    __v: 0,
    password: '$2b$10$dFXt4WlPSSxkuR9MInZ6p.7L4uEuqtJY3Ifc.SxXTa1CHBLoPSgb2'
userdb>
```

Find specific documents:

```
db.collectionName.find({ age: 25 })
```

Feature	`find()`	`findOne()`
Return Type	**Cursor** (array-like)	**Single document** (or `null`)
Returns	**All matching documents**	**The first matching document** only
Usage	When you expect **multiple results**	When you need **just one result**
Performance	Slightly more resource-intensive	Slightly faster if you only need one result

What is Projection?

1. Projection

Controls which fields are returned in the query result.

Syntax:

```
db.collection.find(<filter>, <projection>)
```

Example:

```
db.posts.find({}, { title: 1, date: 1 })
```

- `{}` = match all documents (no filter)
- `{ title: 1, date: 1 }` = include only `title` and `date` fields

Rules:

RULE #	DESCRIPTION
1	Use `1` to include a field
2	Use `0` to exclude a field
3	Cannot mix inclusion and exclusion in same projection (except `_id`)
4	`_id` is included by default, unless explicitly excluded

Exclude `_id`:

```
db.posts.find({}, { title: 1, date: 1, _id: 0 })
```

Limiting Results

Limit number of documents:

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

Sorting Results

Sort by field (ascending/descending):

```
db.products.find().sort({ price: -1 })   // Descending
db.products.find().sort({ name: 1 })     // Ascending
```

Counting Documents

Count matching documents:

```
db.customers.countDocuments()
db.collectionName.countDocuments({ age: 25 })
db.orders.countDocuments({ customer_id: ObjectId("64a1...") })
```

Distinct Values

Get unique values of a field:

```
db.collectionName.distinct("age")

db.products.distinct("category")
```

Comparison & Logical Operators

Greater than (`$gt`):

```
db.products.find({ price: { $gt: 1000 } })
```

Multiple conditions (AND by default):

```
db.orders.find({ total: { $gt: 1000 }, total: { $lt: 3000 } })
```

OR condition:

```
db.customers.find({
  $or: [
    { "address.city": "Nairobi" },
    { name: /Bob/ }
  ]
})
```

7. Regex Search

Case-insensitive search:

```
db.customers.find({ name: /alice/i })
```

Combining Features

Chain filters, projections, sort, limit:

```
db.products.find(
  { price: { $gt: 500 } },          // Filter
  { name: 1, price: 1 }             // Projection
)
  .sort({ price: -1 })              // Sort
  .limit(2)                         // Limit
```

Feature	What It Does
`projection`	Choose which fields to return
`limit()`	Limit number of results
`sort()`	Order the results
`skip()`	Skip documents
`countDocuments()`	Count matched documents
`distinct()`	Return unique field values
`$gt, $lt, $or`	Advanced filtering
`/regex/`	Pattern search

3. Update Documents

Update one document:

```
db.collectionName.updateOne(
  { name: "Alice" },
  { $set: { age: 26 } }
)
```

Update many documents:

```
db.collectionName.updateMany(
  { age: { $lt: 30 } },
  { $set: { status: "active" } }
)
```

❌ 4. Delete Documents

🗑 *Delete one document:*

```
db.collectionName.deleteOne({ name: "Charlie" })
```

🚮 *Delete multiple documents:*

```
db.collectionName.deleteMany({ age: { $gt: 30 } })
```