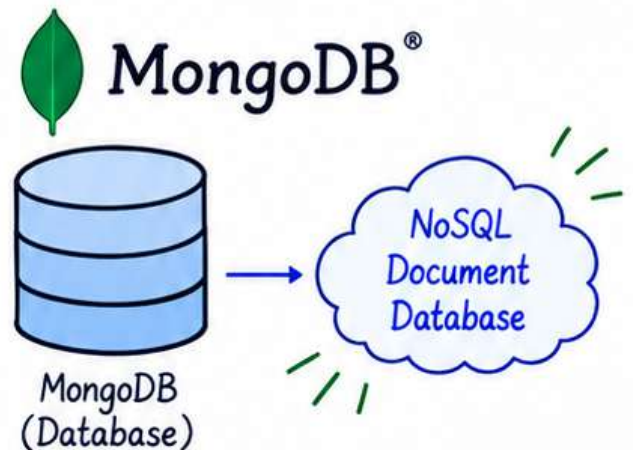


Chapter 14: MongoDB Fundamentals

14.1 Introduction to MongoDB

14.1.1 What is MongoDB?

- * MongoDB is a popular **NoSQL** database that stores data in a flexible, **document-oriented** format.
- * It uses **BSON** (Binary JSON) to store data, which is similar to **JSON**.
- * MongoDB is **open source** and developed by MongoDB Inc.
- * It is designed for **high scalability**, **flexibility** and **performance**.

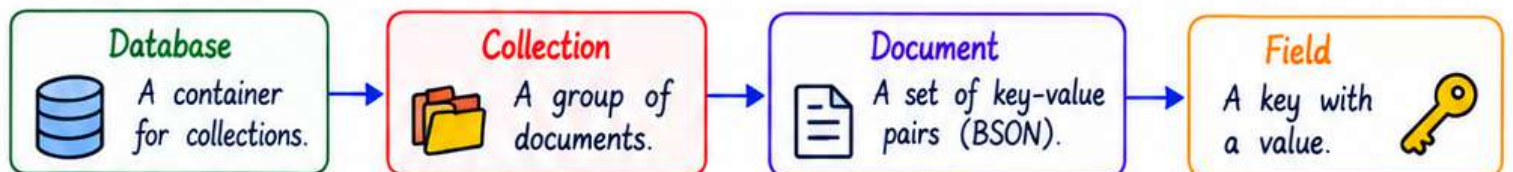


14.1.2 Key Features of MongoDB

- ✓ Document-oriented data model (uses BSON).
- ✓ Flexible schema (no fixed structure).
- ✓ High scalability (horizontal scaling).
- ✓ Fast and efficient queries.
- ✓ Supports rich query language (aggregation, indexing, etc.).
- ✓ High availability and fault tolerance (replica sets).
- ✓ Headless architecture (works well in cloud environments).

14.1.3 How MongoDB Works?

MongoDB stores data in a hierarchical structure called a **database > collection > document**.



14.1.4 Example Document in MongoDB

```
{
  "_id": ObjectId("64f7a1b2c3d4e5f678901234"),
  "name": "Alice",
  "age": 25,
  "email": "alice@example.com",
  "skills": ["JavaScript", "Node.js", "MongoDB"]
}
```

Each document can have different fields, and the structure can vary between documents in the same collection.

Key Takeaway:

- * MongoDB provides a **flexible**, **scalable** and **high-performance** NoSQL database solution for modern applications, storing data in a **document-oriented** way.

14.2 NoSQL Databases

- * NoSQL stands for "Not Only SQL".
- * NoSQL databases are **non-relational** databases that store data in various formats.
- * They are designed to handle **large volumes** of **unstructured**, **semi-structured** and **distributed** data.
- * NoSQL databases provide **high performance**, **scalability** and **flexibility**.
- * They are widely used in **modern web**, **mobile**, **big data** and **real-time** applications.

NoSQL Databases Handle



Unstructured Data



Semi-structured Data



Distributed Data

Examples of NoSQL Databases



MongoDB



Cassandra



Redis



DynamoDB



Neo4j

Types of NoSQL Databases

1. Document Stores



Store data in documents like JSON or BSON.
Example: MongoDB

2. Key-Value Stores



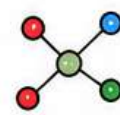
Store data as key-value pairs.
Example: Redis

3. Column-Family Stores



Store data in columns instead of rows.
Example: Cassandra

4. Graph Databases



Store data as nodes and relationships.
Example: Neo4j

NoSQL vs SQL Databases

Feature	NoSQL Databases	SQL Databases
Data Model	Non-relational (Key-Value, Document, Column, Graph)	Relational (Tables)
Schema	Schema-less (Flexible)	Fixed Schema
Scalability	Horizontal Scaling (Scale-out)	Vertical Scaling (Scale-up)
Performance	High Performance	Relatively Lower
Joins	Limited or Not Supported	Supported
Best For	Big Data, Real-time, Unstructured Data	Structured Data, ACID Transactions

Key Points:

- ✓ NoSQL databases are **flexible** and can handle a wide variety of data types.
- ✓ They are designed for **scalability**, **high performance** and **availability**.
- ✓ Choosing between SQL and NoSQL depends on the **type of data** and **application needs**.
- ✓ **MongoDB** is a popular NoSQL document database.



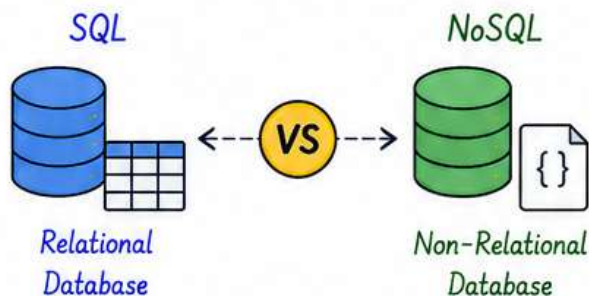
Key Takeaway:



NoSQL databases provide **flexible**, **scalable** and **high-performance** solutions for modern applications that deal with **diverse data**.

14.3 SQL vs NoSQL

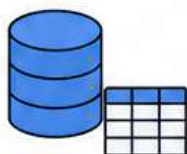
- * SQL (Structured Query Language) databases are **relational** databases that store data in tables with **fixed schema**.
- * NoSQL (Not Only SQL) databases are **non-relational** databases that store data in various formats with **dynamic schema**.
- * Both have their own strengths and are used based on application requirements.



Feature	SQL Databases	NoSQL Databases
1. Data Model	Relational (Tables, Rows, Columns)	Non-relational (Document, Key-Value, Column, Graph)
2. Schema	Fixed Schema	Dynamic / Schema-less
3. Query Language	SQL	Varies (NoSQL Query Language, APIs)
4. Scalability	Vertical Scaling (Scale-up)	Horizontal Scaling (Scale-out)
5. Performance	Good for complex queries	High performance for large scale data
6. Joins	Supports Joins	Limited or No Join Support
7. Transactions	ACID Transactions	BASE (Basically Available, Soft state, Eventually consistent)
8. Data Integrity	High (Strict constraints)	Eventual Consistency
9. Best For	Structured data, complex relationships, ACID compliance	Big data, real-time, flexible schema, high availability
10. Examples	MySQL, PostgreSQL, Oracle, SQL Server	MongoDB, Cassandra, Redis, DynamoDB, Neo4j

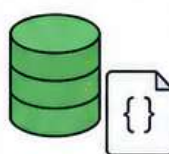
When to Use?

SQL



- * When data is structured.
- * When you need complex queries and joins.
- * When ACID compliance is important.
- * When relationships between data are complex.

NoSQL



- * When data is unstructured or semi-structured.
- * When you need horizontal scalability.
- * When high performance and availability are critical.
- * When dealing with big data and real-time applications.

Key Points:

- ✓ SQL databases are best for **structured data** and **complex relationships**.
- ✓ NoSQL databases are best for **large scale**, **high performance** and **flexible schema**.
- ✓ SQL ensures **consistency** with ACID transactions.
- ✓ NoSQL ensures **availability** and **partition tolerance** with BASE model.
- ✓ Choose the right database based on application requirements and data nature.



Key Takeaway:

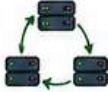
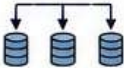


SQL and NoSQL databases are designed for **different use cases**. Understanding their differences helps in choosing the **right database for the right problem**.

14.4 MongoDB Architecture

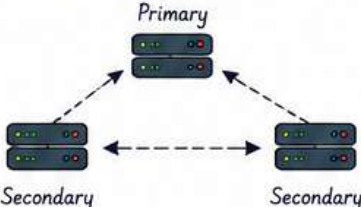
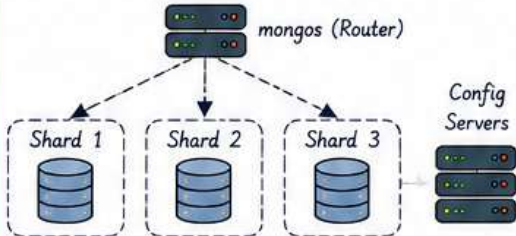
- * MongoDB follows a **distributed architecture** designed for **high performance**, scalability and **high availability**.
- * It is **schema-less** and stores data in **JSON-like** documents.
- * MongoDB architecture is made up of multiple components that work together to store, manage and serve data efficiently.

Components of MongoDB Architecture

- 1. Client Applications**
Applications or drivers connect to MongoDB server to perform read/write operations. 
- 2. MongoDB Server (mongod)**
The core component that handles requests, manages data, executes queries and manages replication. 
- 3. Storage Engine**
Responsible for data storage and retrieval. Default engine is WiredTiger. 
- 4. Replication**
Maintains multiple copies of data across servers for high availability and fault tolerance. 
- 5. Sharding (Horizontal Scaling)**
Distributes data across multiple machines (shards) to handle large scale data and high traffic. 

Types of MongoDB Deployment

- 1. Standalone Deployment**
Single MongoDB server.
Suitable for development and testing.

- 2. Replica Set Deployment**
Multiple servers with one primary and multiple secondary members.
Provides high availability.

- 3. Sharded Cluster Deployment**
Data is distributed across multiple shards for horizontal scaling.


Key Points:

- ✓ MongoDB uses a **distributed architecture** for scalability and high availability.
- ✓ Data is stored in **JSON-like** documents using **collections**.
- ✓ **Replication** ensures data redundancy and fault tolerance.
- ✓ **Sharding** allows MongoDB to handle large datasets and high traffic.

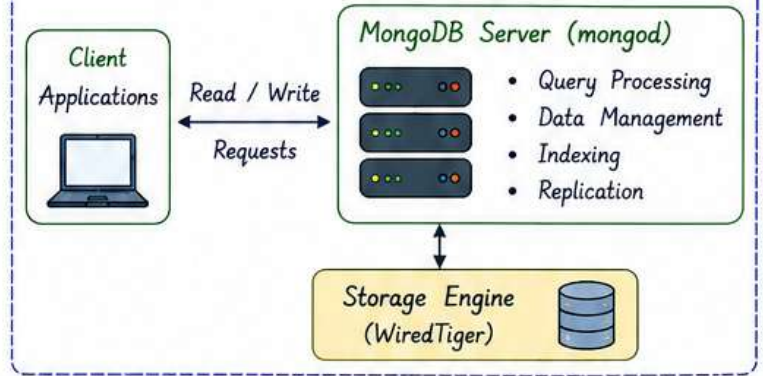


Key Takeaway:



MongoDB architecture is designed to be **flexible**, **scalable** and **highly available**, making it ideal for modern applications and big data.

MongoDB Architecture Overview



How MongoDB Architecture Works?

1. Client sends a request to the MongoDB server. 
2. MongoDB server (mongod) processes the request. 
3. Server uses storage engine to read/write data. 
4. Data is replicated across multiple nodes for availability. 
5. Response is sent back to the client application. 

14.5 MongoDB Installation

What is MongoDB Installation?

- * MongoDB installation is the process of **setting up MongoDB** on your system so that you can start using it to store and manage data.
- * It involves **downloading** the software, **installing** it, and **verifying** that it is running correctly.
- * Once installed, you can use the MongoDB shell (**mongosh**), **Compass (GUI)** or any application to work with data.
- * MongoDB can be installed on **Windows**, **Linux** and **macOS**.

System Requirements

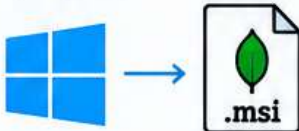
- ✓ 64-bit operating system (Windows, Linux or macOS).
- ✓ Minimum 4 GB RAM (8 GB recommended).
- ✓ Disk space: at least 1 GB.
- ✓ Internet connection (for downloading).
- ✓ Administrative privileges (for installation).



Steps to Install MongoDB

1. Windows Installation

- 1 Visit the official MongoDB website (<https://www.mongodb.com>).
- 2 Download the Windows installer (msi file).
- 3 Run the installer and follow the setup wizard.
- 4 Choose the installation directory and complete the installation.
- 5 Add MongoDB to the system PATH (if required).



2. Linux Installation

- 1 Open a terminal.
- 2 Add the MongoDB repository (for example, Ubuntu).
- 3 Run the package manager commands to install MongoDB.
- 4 Start the MongoDB service.
- 5 Verify the installation.

```
$ sudo apt-get install -y mongodb
$ sudo systemctl start mongod
$ sudo systemctl enable mongod
```



3. macOS Installation

- 1 Install Homebrew (if not installed).
- 2 Run the Homebrew command to install MongoDB.
- 3 Start the MongoDB service.
- 4 Verify the installation.

```
$ brew tap mongodb/brew
$ brew install mongodb-community
$ brew services start mongodb/brew/mongodb-community
```



4. Verify the Installation

- 1 Open a terminal or command prompt.
- 2 Run the MongoDB shell (**mongosh**).
- 3 You should see the MongoDB prompt (>) which means MongoDB is running correctly.

```
$ mongosh
Current Mongosh Log ID: ...
>
```



Key Points:

- ✓ MongoDB installation is simple and can be done in a few steps.
- ✓ Use the official website to download the latest version.
- ✓ After installation, start the MongoDB service and verify using **mongosh**.
- ✓ MongoDB can be installed on **Windows**, **Linux** and **macOS**.



Key Takeaway:

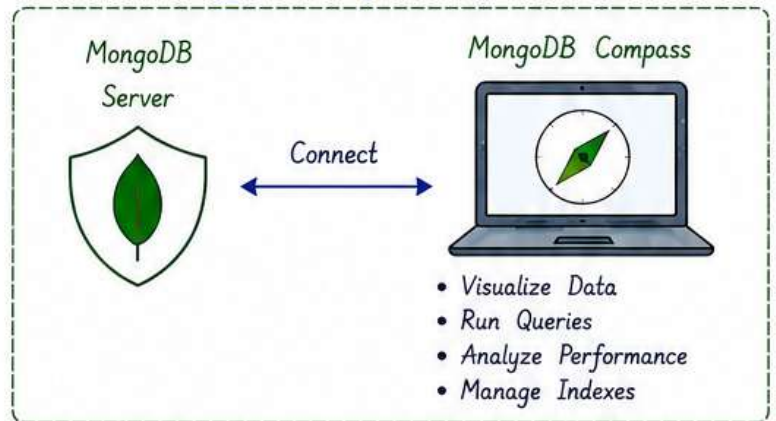


MongoDB installation is the first step to start working with MongoDB, and once installed, you can easily **store**, **manage** and **query** your data.

14.6 MongoDB Compass

- * MongoDB Compass is the **official GUI** tool for MongoDB.
- * It allows us to **visualize, explore and manage** our MongoDB databases easily.
- * It provides an intuitive interface to perform **CRUD operations**, run queries, view performance and manage data.
- * Compass **simplifies** database development and administration.

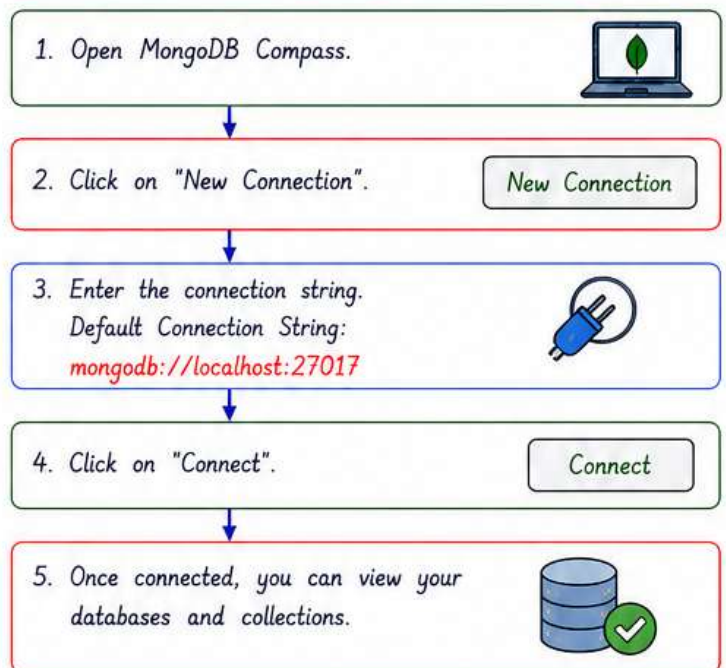
MongoDB Compass Overview



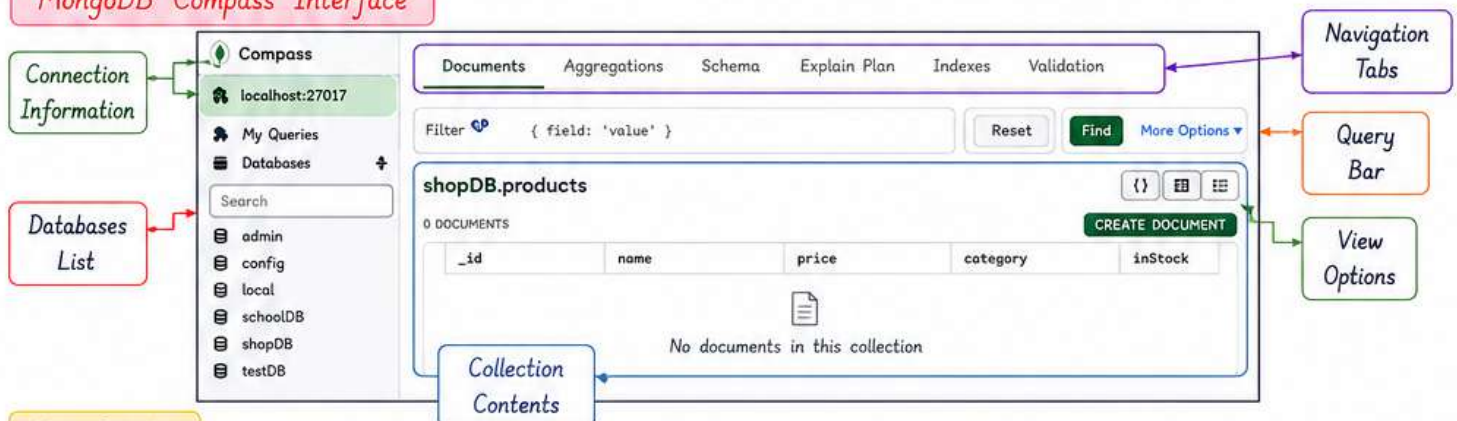
Features of MongoDB Compass

	Visualize Data Browse and explore data in collections in a table or JSON view.
	Run Queries Write and execute queries using the Query Builder or Aggregation Editor.
	Analyze Performance View performance statistics for queries and database operations.
	Manage Indexes Create, view and manage indexes easily.
	Schema Analysis Analyze schema patterns and data structure.
	Easier Connection Easily connect to local or remote MongoDB instances.

How to Connect Using MongoDB Compass?



MongoDB Compass Interface



Key Points:

- ✓ MongoDB Compass provides a user-friendly GUI.
- ✓ It helps to visualize and manage data easily.
- ✓ Supports CRUD operations, queries and aggregations.
- ✓ Helps in schema analysis and performance monitoring.
- ✓ Ideal for developers, DBAs and data analysts.



Key Takeaway:

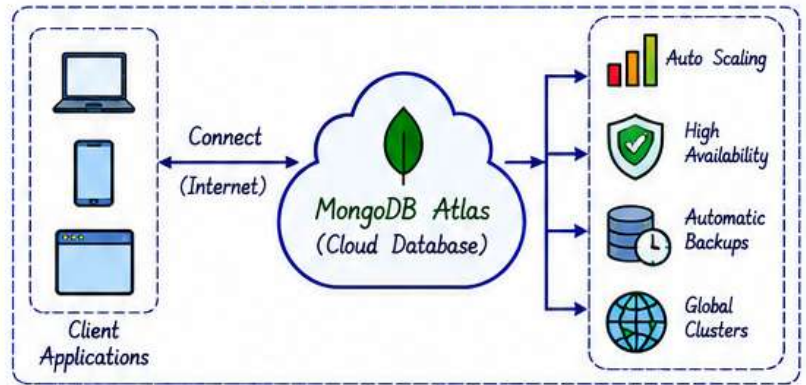


MongoDB Compass makes it **easy to connect, explore and manage** your MongoDB databases with a powerful and intuitive graphical interface.

14.7 MongoDB Atlas

- * MongoDB Atlas is the **official cloud database service** provided by MongoDB.
- * It is a **fully managed Database-as-a-Service (DBaaS)** platform.
- * Atlas helps us to **deploy, manage and scale** MongoDB databases in the cloud without setting up servers manually.
- * We can **access** MongoDB using the **internet** from anywhere in the world.

MongoDB Atlas Architecture



Features of MongoDB Atlas

	Fully Managed MongoDB handles installation, updates and maintenance.
	Global Availability Deploy databases in multiple cloud regions across the world.
	Auto Scaling Automatically scales storage and performance based on usage.
	High Security Provides encryption, access control and network security.
	Automatic Backups Continuous backups and point-in-time recovery.
	Monitoring & Alerts Real-time metrics and performance monitoring dashboard.

How to Use MongoDB Atlas?

1. Go to MongoDB Atlas website.
<https://www.mongodb.com/atlas>
2. Create a **free account** (or sign in).
3. Create a new **project**.
4. Build a **cluster** (choose free tier or paid plan).
5. Configure **network access** (allow your IP or use 0.0.0.0/0 for testing).
6. Create a **database user**.
7. Get the **connection string** and connect using MongoDB Compass or your application.

Atlas Plans

Plan	Cost	Features
Free Tier	Free	• Shared cluster • 512 MB storage • Good for learning & small projects
Dedicated Tier	Paid	• More storage and performance • Production applications
Multi-Region	Paid	• Global clusters • High availability and auto scaling

Use Cases

- ✓ Web and mobile application databases
- ✓ Startups and product development
- ✓ Real-time applications (chat, IoT, e-commerce)
- ✓ Enterprise applications with high availability
- ✓ Learning and prototyping MongoDB with free tier



MongoDB Atlas

Build • Scale • Go Global

Key Points:

- ✓ MongoDB Atlas is a **cloud-based, fully managed** service.
- ✓ No need to install or maintain servers.
- ✓ Provides **auto scaling, backups** and **high security**.
- ✓ Offers a **free tier** for learning and testing.
- ✓ Can be accessed from **anywhere** using the internet.
- ✓ Easily integrates with MongoDB Compass and applications.



Key Takeaway:

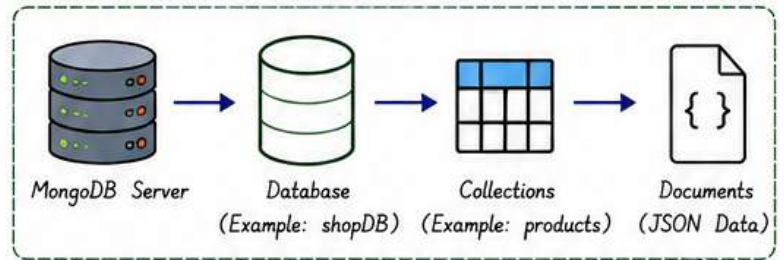


MongoDB Atlas makes it **simple** to create, deploy and manage MongoDB databases in the cloud, helping developers build **scalable** and **reliable** applications without worrying about infrastructure.

14.8 Databases

- * A database is a **structured collection of data** that can be easily accessed, managed and updated.
- * MongoDB stores data in **databases**. Each database contains **collections**, and each collection contains **documents**.
- * In MongoDB, a database is a container for collections and related data.
- * We can create multiple databases in the same MongoDB server instance.

MongoDB Database Structure



Database Example

In our application, we can have a database named **shopDB**.

It may contain the following collections:

- products
- users
- orders



Example in Shell

```
> show dbs
admin      40.00 KB
config     12.00 KB
local      72.00 KB

> use shopDB
switched to db shopDB

> db
shopDB

> db.createCollection("products")
{ "ok" : 1 }

> show collections
products

> db.dropDatabase()
{ "dropped" : "shopDB", "ok" : 1 }
```



Database Operations

Operation	Description
Show Databases	List all available databases.
Use Database	Switch to a specific database.
Create Database	Created automatically when data is stored for the first time.
Drop Database	Delete a database and all its data.

Working with Databases (Mongo Shell)

Command	Description
show dbs	Show all databases.
use <database_name>	Switch to the specified database.
db	Show the current database.
db.createCollection("name")	Create a collection in the current database.
db.dropDatabase()	Drop the current database.

Types of Databases in MongoDB

1. User Databases

- Created by developers for applications.
- Example: shopDB, schoolDB



2. Admin Database

- Contains administrative information.
- Used for authentication and user management.



3. Local Database

- Stores replication information.
- Used internally by MongoDB.



Key Points:

- ✓ A database is a container for collections.
- ✓ Collections store documents (JSON data).
- ✓ Databases are created automatically when data is stored.
- ✓ We can create multiple databases in the same server.
- ✓ Use appropriate database names to organize data.
- ✓ Use **show dbs** command to list all databases.



Key Takeaway:



Databases help us **organize** and **manage** related data efficiently in MongoDB. Understanding databases is the first step to building any MongoDB application.

14.9 Collections

- * A collection is a **group of MongoDB documents**.
- * Collections are similar to **tables** in relational databases.
- * Collections do not enforce a fixed schema. Documents in the same collection can have **different fields**.
- * Collections are created **automatically** when we insert the first document.
- * We can store a large number of documents in a collection.

Collection Operations

Operation	Description
Create Collection	Creates a new collection.
Insert Document	Adds a document to a collection.
Find Documents	Retrieves documents from a collection.
Update Document	Modifies existing documents.
Delete Document	Removes documents from a collection.
Drop Collection	Deletes a collection and all its documents.

Working with Collections (Mongo Shell)

Command	Description
<code>show collections</code>	List all collections in the database.
<code>db.createCollection("name")</code>	Create a new collection.
<code>db.collectionName.insertOne(doc)</code>	Insert a single document.
<code>db.collectionName.insertMany(docs)</code>	Insert multiple documents.
<code>db.collectionName.find()</code>	Retrieve all documents.
<code>db.collectionName.find(query)</code>	Retrieve documents that match query.
<code>db.collectionName.updateOne()</code>	Update a document.
<code>db.collectionName.deleteOne()</code>	Delete a document.
<code>db.collectionName.drop()</code>	Drop the collection.

Important Points about Collections

- ✓ Collections are schema-less (flexible structure).
- ✓ Documents in a collection can have different fields.
- ✓ Collections are created automatically when we insert data.
- ✓ Collection names must be unique within a database.
- ✓ Use meaningful names to organize your data.



Key Points:

- ✓ Collections store groups of MongoDB documents.
- ✓ They are equivalent to tables in SQL databases.
- ✓ Collections are schema-less and highly flexible.
- ✓ Collections are created automatically when needed.
- ✓ Use proper collection names for better organization.

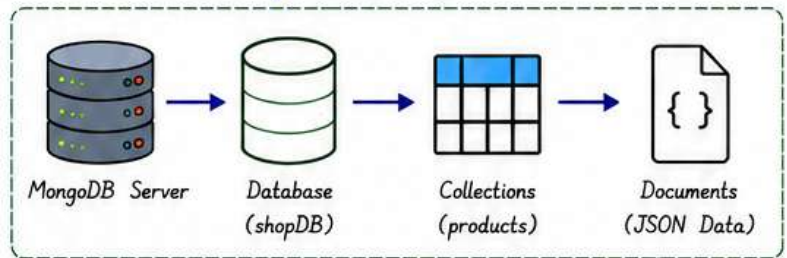


Key Takeaway:



Collections are the main building blocks in MongoDB. They help us store and manage data in a **flexible** and **efficient** way.

MongoDB Structure



Collection Example

In our database **shopDB**, a collection named **products** may contain the following documents:

```
{
  "_id": 1,
  "name": "Laptop",
  "price": 55000,
  "brand": "Dell"
}

{
  "_id": 2,
  "name": "Phone",
  "price": 25000,
  "brand": "Samsung"
}

{
  "_id": 3,
  "name": "Tablet",
  "price": 18000,
  "brand": "Lenovo"
}
```



Collection: **products**

Example in Shell

```
> use shopDB
switched to db shopDB
> db.createCollection("products")
{ "ok" : 1 }
> db.products.insertOne(
  { "name" : "Laptop", "price" : 55000, "brand" : "Dell" }
)
{ "acknowledged" : true, "insertedId" : ObjectId("...") }
> db.products.find(
  { "_id" : ObjectId("..."), "name" : "Laptop",
    "price" : 55000, "brand" : "Dell" }
)
> db.products.updateOne(
  { "name" : "Laptop" },
  { $set: { "price" : 52000 } }
)
> db.products.deleteOne( { "name" : "Tablet" } )
> db.products.drop()
{ "ok" : 1 }
```



Collection Naming Rules

- * Collection name cannot be an empty string.
- * Cannot contain \0 (null character).
- * Cannot start with "system." (reserved prefix).
- * Should not contain "\$" in the name.
- * Use lowercase letters and underscores (_).



14.10 Documents

- * A document is a **basic unit of data** in MongoDB.
- * Documents are stored in **collections**.
They are similar to rows in a relational table.
- * Documents are written in **BSON** (Binary JSON) format, which is a binary representation of JSON.
- * Each document can have different fields.
Fields in documents can be of different data types.
- * Documents in the same collection **do not need** to have the same structure.

Document Characteristics

Feature	Description
Schema-less	Documents in a collection can have different fields.
Flexible	Fields can be added or removed at any time.
Nested Structure	Documents can contain embedded documents and arrays.
Unique Identifier	Every document has a unique <code>_id</code> field. It is automatically generated.
BSON Format	Documents are stored in BSON (Binary JSON).

Working with Documents (Mongo Shell)

Command	Description
<code>db.collection.insertOne(doc)</code>	Insert a single document.
<code>db.collection.insertMany(docs)</code>	Insert multiple documents.
<code>db.collection.find()</code>	Retrieve all documents.
<code>db.collection.find(query)</code>	Retrieve documents that match query.
<code>db.collection.findOne(query)</code>	Retrieve a single document.
<code>db.collection.updateOne(filter, update)</code>	Update a single document.
<code>db.collection.updateMany(filter, update)</code>	Update multiple documents.
<code>db.collection.deleteOne(filter)</code>	Delete a single document.
<code>db.collection.deleteMany(filter)</code>	Delete multiple documents.

Data Types in Documents

Data Type	Example
String	"Laptop"
Number (Int/Double)	55000, 4.5
Boolean	true, false
Array	["electronics", "mobile"]
Object (Embedded Document)	{ "address": { "city": "Hyderabad" } }
Date	ISODate("2024-05-20T10:00:00Z")
ObjectId	ObjectId("64f7c1e9c2b9d48a1f9e1c01")
Null	null

Key Points:

- ✓ A document is a record in MongoDB.
- ✓ Stored in collections.
- ✓ Written in BSON (Binary JSON) format.

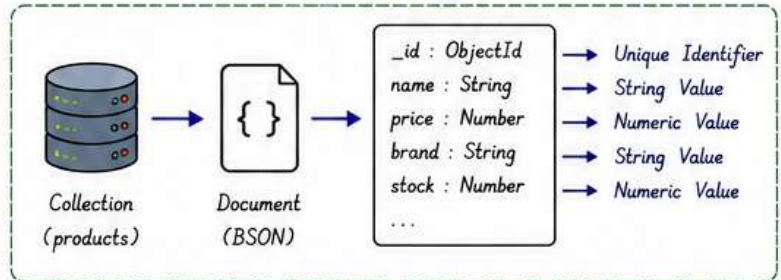
Key Takeaway:



Documents make MongoDB **flexible** and **powerful**.

They allow us to store **structured**, **semi-structured** and **unstructured** data easily.

Document Structure



Document Example

Example of documents in a collection named **products**:

```
{
  "_id": ObjectId("64f...01"),
  "name": "Laptop",
  "price": 55000,
  "brand": "Dell",
  "stock": 10,
  "tags": ["electronics", "computer"],
  "rating": 4.5
}
```

```
{
  "_id": ObjectId("64f...02"),
  "name": "Phone",
  "price": 25000,
  "brand": "Samsung",
  "stock": 20,
  "tags": ["electronics", "mobile"],
  "rating": 4.2
}
```

```
{
  "_id": ObjectId("64f...03"),
  "name": "Tablet",
  "price": 18000,
  "brand": "Lenovo",
  "stock": 15,
  "tags": ["electronics", "tablet"],
  "rating": 4.0
}
```

Example in Shell

```
> use shopDB
switched to db shopDB
> db.products.insertOne(
  { "name": "Laptop", "price": 55000, "brand": "Dell", "stock": 10 }
)
{ "acknowledged": true, "insertedId": ObjectId("...") }
> db.products.find()
{ "_id": ObjectId("...01"), "name": "Laptop", "price": 55000, "brand": "Dell", "stock": 10, "rating": 4.5 }
{ "_id": ObjectId("...02"), "name": "Phone", "price": 25000, "brand": "Samsung", "stock": 20, "rating": 4.2 }
{ "_id": ObjectId("...03"), "name": "Tablet", "price": 18000, "brand": "Lenovo", "stock": 15, "rating": 4.0 }
> db.products.findOne( { "name": "Phone" } )
{ "_id": ObjectId("...02"), "name": "Phone", "price": 25000, "brand": "Samsung", "stock": 20, "rating": 4.2 }
> db.products.updateOne( { "name": "Laptop" }, { $set: { "price": 52000 } } )
{ "acknowledged": true, "matchedCount": 1, "modifiedCount": 1 }
> db.products.deleteOne( { "name": "Tablet" } )
{ "acknowledged": true, "deletedCount": 1 }
```

Important Points

- * Documents are the heart of MongoDB.
- * Every document must have a **unique _id field**.
- * MongoDB is schema-less, so documents in a collection can have different fields.
- * Nested documents and arrays allow storing complex data.
- * Documents are stored in **BSON** format for efficiency.



14.11 BSON

- * BSON stands for **Binary JSON**.
- * It is a **binary** representation of JSON.
- * MongoDB uses BSON to store documents.
- * BSON is **efficient**, **compact** and **fast** to read/write.
- * It supports more data types than JSON.
- * BSON is the internal format used by MongoDB for data storage and transfer.

BSON Data Types

Data Type	BSON Type	Example	Description
Double	0x01	4.5	64-bit floating point
String	0x02	"Laptop"	UTF-8 string
Document	0x03	{ ... }	Embedded document
Array	0x04	[...]	Array of values
Binary Data	0x05	BinData(...)	Binary data
Boolean	0x08	true / false	Boolean value
Date	0x09	ISODate(...)	Date & time
Null	0x0A	null	Null value
ObjectId	0x07	ObjectId(...)	Unique 12-byte ID
Integer (32-bit)	0x10	100	32-bit integer
Int64 (64-bit)	0x12	10000000000	64-bit integer

Working with BSON (Mongo Shell)

Command	Description
<code>db.collection.insertOne(doc)</code>	Insert a document (converted to BSON)
<code>db.collection.find()</code>	Retrieve documents (returned as BSON)
<code>db.collection.findOne()</code>	Retrieve a single document
<code>db.collection.updateOne()</code>	Update a document
<code>db.collection.deleteOne()</code>	Delete a document

BSON vs JSON

Feature	JSON	BSON
Format	Text	Binary
Readability	Human Readable	Not Human Readable
Size	Larger	Smaller
Parsing	Slower	Faster
Data Types	Limited	More (Date, BinData, ObjectId, etc.)
Used By	APIs, Web	MongoDB

Key Points:

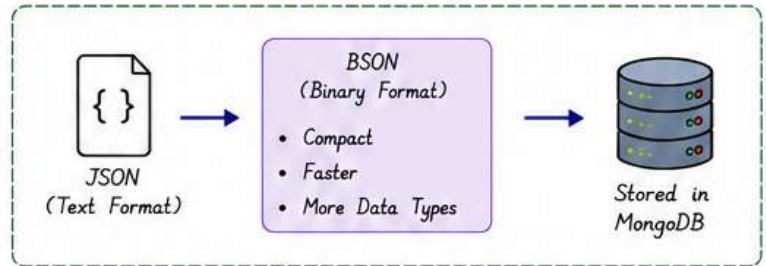
- ✓ BSON = Binary JSON.
- ✓ Used internally by MongoDB.
- ✓ Compact, fast and efficient.

Key Takeaway:



BSON is the backbone of MongoDB.
It helps in **storing**, **querying** and **transferring** data efficiently.

JSON vs BSON



BSON Example

JSON Document (Human Readable)

```
{
  "_id" : 1,
  "name" : "Laptop",
  "price" : 55000,
  "inStock" : true,
  "tags" : [ "electronics", "computer" ],
  "rating" : 4.5
}
```

BSON (Binary)

```
16 00 00 00 10 5f 69 64 00
01 00 00 00 02 6e 61 6d 60
07 00 00 00 4c 61 70 74 6f
70 00 10 70 72 69 63 65 00
90 d7 00 00 08 69 6e 3 74
6f 63 6b 00 01 02 74 61 67
73 00 03 00 00 00 0b 00 00
00 65 6c 65 63 74 72 6f 6e
69 63 73 00 63 6f 6d 70 75
74 65 72 00 01 72 61 74 69
6e 67 00 00 00 00 00 12 40
00
```

Why BSON?

- * **Smaller size** compared to JSON.
- * **Faster** parsing and processing.
- * Supports **additional** data types.
- * Preserves **data type** information.
- * **Efficient** for network transfer and storage.



Example in Shell

```
> use shopDB
switched to db shopDB
> db.products.insertOne(
  { name: "Laptop", price: 55000, inStock: true }
)
{ "acknowledged" : true, "insertedId" : ObjectId("664fc1a0e2b9d48a1f9e1c01") }
> db.products.find().pretty()
{
  "_id" : ObjectId("664fc1a0e2b9d48a1f9e1c01"),
  "name" : "Laptop",
  "price" : 55000,
  "inStock" : true
}
// Data is stored internally in BSON format
```

Important Points

- * BSON is **not just JSON**. It is a binary encoding of JSON.
- * All data in MongoDB is stored in **BSON format**.
- * BSON reduces storage space and improves performance.
- * BSON supports rich data types for real-world applications.

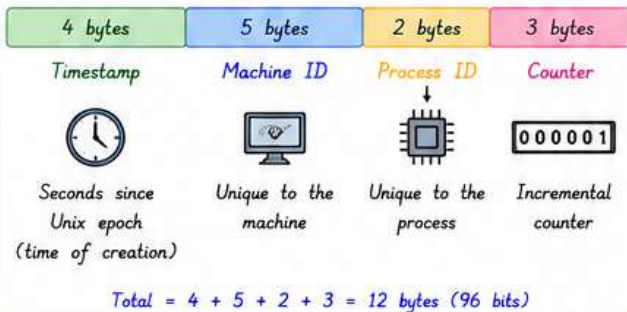


14.12 ObjectId

- * ObjectId is a **12-byte (96-bit)** unique identifier used by MongoDB.
- * It is automatically generated for every document and stored in the `_id` field.
- * ObjectId is **unique** across time and machines.
- * It contains embedded information like **timestamp**, **machine id**, **process id** and **counter**.
- * We can also create ObjectId manually if needed.
- * ObjectId is used for indexing and fast retrieval of documents.

Structure of ObjectId (12 Bytes)

ObjectId = Timestamp (4 bytes) + Machine ID (5 bytes) + Process ID (2 bytes) + Counter (3 bytes)



ObjectId Properties

Property	Description
Type	ObjectId
Size	12 bytes (96 bits)
Format	24-character hexadecimal string
Example	ObjectId("64fc1a0e2b9d48a1f9e1c01")
Generated By	Automatically by MongoDB
Uniqueness	Unique across machine, process and time
Used In	_id field of every document

Generating ObjectId

Method	Example
Automatically	Inserted by MongoDB when document is created.
Manually (Shell)	<code>new ObjectId() _</code>
Manually (Driver)	ObjectId() in drivers (e.g., Python, Node.js)

Comparison with Other IDs

Feature	Relational DB (Auto Increment)	MongoDB (ObjectId)
Type	Integer	12-byte (96-bit)
Uniqueness	Unique per table	Globally unique
Information	No info (only sequence)	Contains timestamp, machine, process, counter
Ordering	Sequential	Roughly ordered by time
Generation	At Insert (DB)	At Insert (MongoDB)
Exposure	Predictable	Harder to guess

Key Points:

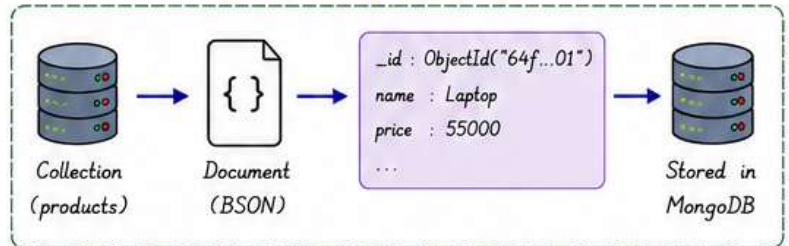
- ✓ ObjectId is a 12-byte unique identifier.
- ✓ Automatically generated for every document.
- ✓ Contains timestamp, machine id, process id and counter.

Key Takeaway:



ObjectId is the default primary key in MongoDB. It provides **uniqueness**, stores **useful information** and helps in **efficient data management**.

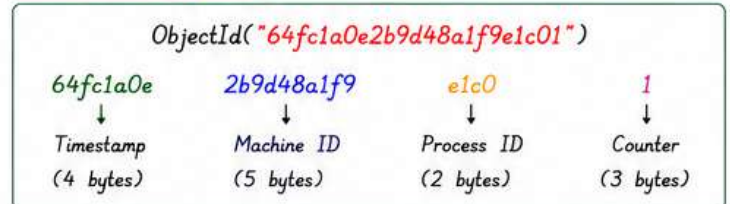
Where is ObjectId Used?



Every document in a collection must have a unique `_id`. If we do not provide one, MongoDB automatically generates an ObjectId and assigns it.



Example of ObjectId



Features of ObjectId

- * Automatically generated by MongoDB.
- * Globally unique (across machines and time).
- * Contains timestamp → helps in sorting by creation time.
- * Efficient for indexing and storage.
- * Cannot be updated once created.



Example in Shell

```
> use shopDB
switched to db shopDB
> db.products.insertOne( { name: "Laptop", price: 55000 } )
{ "acknowledged" : true, "insertedId" : ObjectId("64fc1a0e2b9d48a1f9e1c01") }
> db.products.find().pretty()
{
  "_id" : ObjectId("64fc1a0e2b9d48a1f9e1c01"),
  "name" : "Laptop",
  "price" : 55000
}
> db.products.find().sort({_id:1}) // sort by ObjectId (creation time)
> new ObjectId()
ObjectId("64fc1a0e2b9d48a1f9e1c02") // manually generate
```



Important Points

- * Every document must have an `_id` of type ObjectId.
- * ObjectId values look like random strings but contain useful information.
- * We should not change the `_id` of a document.
- * ObjectIds are best for distributed systems and sharding.



- ✓ Ensures uniqueness across machines and time.
- ✓ Efficient for indexing, sorting and retrieval.



14.13 MongoDB Shell

- * MongoDB Shell is an **interactive JavaScript** interpreter used to communicate with MongoDB.
- * It allows us to run commands, perform **CRUD operations** and manage MongoDB instances.
- * It comes with MongoDB installation.
- * We can also use it to write and run JavaScript code.
- * It is very useful for learning, testing and debugging MongoDB.

Starting MongoDB Shell

Open command prompt / terminal and run:

```
mongo # Connect to local MongoDB server
```

- If MongoDB is running, the shell will open like below.
- Default host is **localhost** and port is **27017**.

```
C:\> mongo
MongoDB shell version v7.0.8
connecting to: mongodb://127.0.0.1:27017/?directConnection=true&
serverSelectionTimeoutMS=2000
MongoDB server version: 7.0.8
>
```

Basic Shell Commands

Command	Description
<code>show dbs</code>	List all databases.
<code>use <database_name></code>	Switch to a specific database.
<code>db</code>	Show the current database.
<code>show collections</code>	List all collections in current database.
<code>db.createCollection("name")</code>	Create a new collection.
<code>db.dropDatabase()</code>	Delete the current database.
<code>db.collection.drop()</code>	Delete a collection.
<code>exit</code>	Exit the MongoDB shell.

CRUD Operations in Shell

Operation	Command	Example
Create (Insert)	<code>db.collection.insertOne(doc)</code>	<code>db.products.insertOne({ name: "Laptop", price: 55000 })</code>
Read (Find)	<code>db.collection.find()</code>	<code>db.products.find()</code>
Update	<code>db.collection.updateOne()</code> or <code>updateMany()</code>	<code>db.products.updateOne({ name: "Laptop", \$set: { price: 52000 } })</code>
Delete	<code>db.collection.deleteOne()</code> or <code>deleteMany()</code>	<code>db.products.deleteOne({ name: "Laptop" })</code>

Key Points:

- ✓ MongoDB Shell is an interactive command-line tool.
- ✓ Used to manage databases, collections and documents.
- ✓ Supports CRUD operations.



- ✓ Comes with MongoDB installation.
- ✓ Very useful for development and testing.

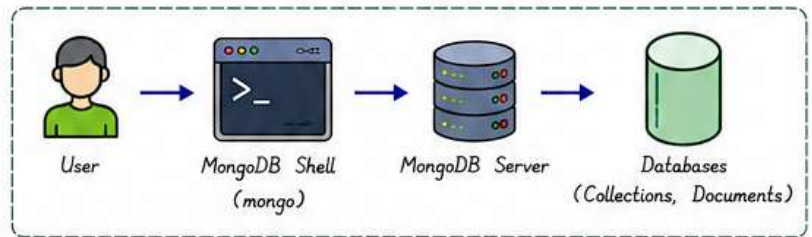


Key Takeaway:



MongoDB Shell is the primary way to **interact with MongoDB**. It helps us manage data efficiently using simple commands.

How MongoDB Shell Works?



MongoDB Shell sends commands to MongoDB server.
The server processes the commands and returns the results.



Example in MongoDB Shell

```
> show dbs
admin      40.00 KB
config     12.00 KB
local      72.00 KB
# List databases

> use shopDB
switched to db shopDB
# Switch to database

> db
shopDB
# Current database

> db.createCollection("products")
{ "ok" : 1 }
# Create collection

> show collections
products
# List collections

> db.products.insertOne(
  { name: "Laptop", price: 55000 })
{ "acknowledged" : true, "insertedId" : ObjectId("64fc1a0e2b9d48a1f9e1c01") }
# Insert document

> db.products.find()
{ "_id" : ObjectId("64fc1a0e2b9d48a1f9e1c01"), "name" : "Laptop", "price" : 55000 }
# Find documents

> db.products.updateOne(
  { name: "Laptop", $set: { price: 52000 } })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
# Update document

> db.products.deleteOne({ name: "Laptop" })
{ "acknowledged" : true, "deletedCount" : 1 }
# Delete document

> exit
bye
# Exit shell
```

Connecting to Remote Server

To connect to a remote MongoDB server:

```
mongo --host <host> --port <port> -u <username> -p <password>
```

Example:

```
mongo --host 127.0.0.1 --port 27017 -u admin -p admin123
```



Important Points

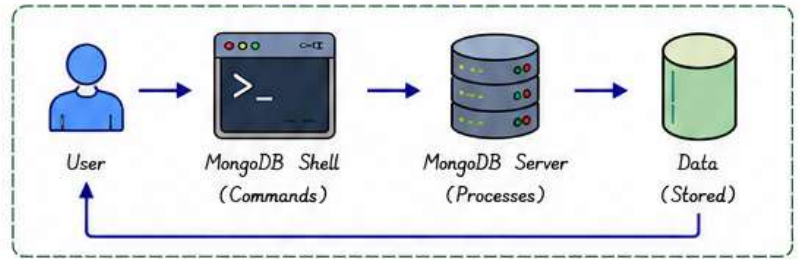
- * MongoDB Shell uses **JavaScript** syntax.
- * All database operations can be performed using shell commands.
- * It is a powerful tool for developers and DBAs.
- * Helpful for learning, testing and automation (scripts).



14.14 CRUD Operations

- * CRUD stands for **Create, Read, Update** and **Delete**.
- * These are the basic operations performed on data in any database.
- * MongoDB provides CRUD operations through shell, drivers and APIs.
- * CRUD operations are performed on databases, collections and documents.
- * They help us to manage data efficiently.

CRUD Cycle in MongoDB



CRUD Operations Overview

Operation	Purpose	MongoDB Command	HTTP Method
Create 	Insert new documents into a collection.	<code>db.collection.insertOne()</code> <code>db.collection.insertMany()</code>	POST
Read 	Retrieve documents from a collection.	<code>db.collection.find()</code> <code>db.collection.findOne()</code>	GET
Update 	Modify existing documents in a collection.	<code>db.collection.updateOne()</code> <code>db.collection.updateMany()</code>	PUT / PATCH
Delete 	Remove documents from a collection.	<code>db.collection.deleteOne()</code> <code>db.collection.deleteMany()</code>	DELETE

CRUD in One Example

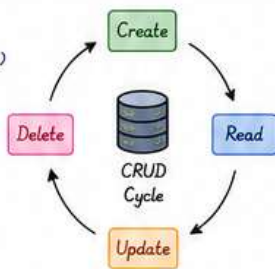
- Create**

```
> db.items.insertOne(  
  { name: "Pen", price: 10, stock: 100 } )
```
- Read**

```
> db.items.find( { name: "Pen" } )
```
- Update**

```
> db.items.updateOne( { name: "Pen" },  
  { $set: { price: 12 } } )
```
- Delete**

```
> db.items.deleteOne( { name: "Pen" } )
```



Important Notes

- * All CRUD operations in MongoDB are performed on collections.
- * Documents are stored in **BSON format**.
- * `_id` (ObjectId) is automatically generated for each document.
- * Use proper filters to update or delete only required documents.
- * Always validate data before performing write operations.



CRUD Summary

Operation	What it Does	Example Command	Result
Create	Adds new document(s)	<code>db.collection.insertOne()</code> / <code>insertMany()</code>	New document inserted
Read	Retrieves document(s)	<code>db.collection.find()</code> / <code>findOne()</code>	Documents returned
Update	Modifies existing document(s)	<code>db.collection.updateOne()</code> / <code>updateMany()</code>	Documents updated
Delete	Removes document(s)	<code>db.collection.deleteOne()</code> / <code>deleteMany()</code>	Documents deleted

Key Points:

- ✓ CRUD = Create, Read, Update, Delete.
- ✓ MongoDB supports CRUD through shell, drivers and APIs.
- ✓ Performed on databases, collections and documents.

Key Takeaway:



CRUD operations are the foundation of data management in MongoDB. Mastering them helps us **store, retrieve, update** and **delete** data effectively.

Examples in MongoDB Shell

1. CREATE - Insert Documents

```
> use shopDB  
switched to db shopDB  
# Insert single document  
> db.products.insertOne(  
  { name: "Laptop", price: 55000, brand: "Dell", stock: 10 }  
)  
{ "acknowledged" : true, "insertedId" : ObjectId("64fc1a0e2b9d48a1f9e1c01") }  
  
# Insert multiple documents  
> db.products.insertMany([  
  { name: "Phone", price: 25000, brand: "Samsung", stock: 20 },  
  { name: "Tablet", price: 18000, brand: "Lenovo", stock: 15 }  
)  
{ "acknowledged" : true, "insertedIds" : [ ObjectId("64fc1a0e2b9d48a1f9e1c02"),  
  ObjectId("64fc1a0e2b9d48a1f9e1c03") ] }
```

2. READ - Retrieve Documents

```
# Get all documents  
> db.products.find()  
{ "_id" : ObjectId("64fc1a0e2b9d48a1f9e1c01"), "name" : "Laptop", "price" : 55000, "brand" : "Dell", "stock" : 10 }  
{ "_id" : ObjectId("64fc1a0e2b9d48a1f9e1c02"), "name" : "Phone", "price" : 25000, "brand" : "Samsung", "stock" : 20 }  
{ "_id" : ObjectId("64fc1a0e2b9d48a1f9e1c03"), "name" : "Tablet", "price" : 18000, "brand" : "Lenovo", "stock" : 15 }  
  
# Find documents with condition  
> db.products.find( { brand: "Dell" } )  
  
# Find single document  
> db.products.findOne( { name: "Laptop" } )
```

3. UPDATE - Modify Documents

```
# Update single document  
> db.products.updateOne(  
  { name: "Laptop" },  
  { $set: { price: 52000, stock: 8 } }  
)  
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }  
  
# Update multiple documents  
> db.products.updateMany(  
  { price: { $lt: 20000 } },  
  { $set: { discount: true } }  
)  
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

4. DELETE - Remove Documents

```
# Delete single document  
> db.products.deleteOne( { name: "Tablet" } )  
{ "acknowledged" : true, "deletedCount" : 1 }  
  
# Delete multiple documents  
> db.products.deleteMany( { stock: { $lt: 5 } } )  
{ "acknowledged" : true, "deletedCount" : 2 }
```

Best Practices

- * Use meaningful collection and field names.
- * Keep documents small and focused.
- * Use indexes for faster read and search.
- * Avoid unnecessary updates and deletes.
- * Backup your data regularly.

