The Beginner's Handbook to MongoDB

Welcome to your go-to guide for getting started with MongoDB. This handbook is designed for beginners, breaking down complex concepts into simple, understandable steps with practical examples.

1. Introduction to MongoDB

What is MongoDB?

MongoDB is a popular, powerful, and modern database that is widely used for building applications. Unlike traditional databases that store data in tables with rows and columns, MongoDB stores data in a flexible, JSON-like format.

Features and Advantages

- **NoSQL Database:** MongoDB is a "NoSQL" (often interpreted as "Not Only SQL") database. This means it doesn't use the rigid table structure of traditional databases, offering more flexibility.
- **Document-Based:** It stores data in "documents," which are similar to JSON objects. This structure is very intuitive for developers because it maps directly to objects in code.
    - *Example Document:*

```
{
  "name": "Alice",
  "age": 30,
  "hobbies": ["reading", "hiking"],
  "address": {
    "street": "123 Main St",
    "city": "Anytown"
  }
}
```

- **Schema-Less (Flexible Schema):** You don't have to define the structure of your data beforehand. Each document in a collection can have a different structure, making it easy to evolve your application over time.

How it Differs from SQL Databases

Feature	SQL (e.g., MySQL, PostgreSQL)	MongoDB (NoSQL)
Data Model	Tables with rows and columns	Collections with documents (JSON-like)
Schema	Rigid, defined in advance	Flexible, can change on the fly
Structure	Structured data	Structured, semi-structured, or unstructured

Feature	SQL (e.g., MySQL, PostgreSQL)	MongoDB (NoSQL)
Scalability	Vertically scalable (increase server power)	Horizontally scalable (add more servers)
Query Language	SQL (Structured Query Language)	MQL (Mongo Query Language) - rich JSON-based queries

2. Setup & Mongo Shell

Installation

Installing MongoDB varies by operating system (Windows, macOS, Linux). The best approach is to follow the official guide for your specific system.

1. **Download:** Go to the MongoDB Community Server Download Page.
2. **Install:** Follow the installation instructions provided.
3. **Set Environment Variables:** After installation, add the path to MongoDB's bin directory (e.g., C:\Program Files\MongoDB\Server\6.0\bin) to your system's PATH environment variable. This allows you to run MongoDB commands from any terminal window.

Introduction to mongosh

mongosh is the modern, feature-rich command-line shell for MongoDB. It allows you to connect to your database, perform administrative tasks, and run queries.

Basic Commands

1. **Start the MongoDB Server:** Before you can connect, you need to make sure the MongoDB server (the mongod process) is running. This usually starts automatically as a service after installation.
2. **Start the Shell & Connect:** Open your terminal or command prompt and type:
   `mongosh`
   This command connects to the default MongoDB server running on your local machine (mongodb://127.0.0.1:27017).

3. Database Basics

What is a Database?

In MongoDB, a database is a physical container for collections. A single MongoDB server can host multiple databases, each with its own set of collections and permissions.

Behavior of Databases in MongoDB

A unique feature of MongoDB is that it **creates a database only when you first store data in it**. Simply using the use command switches to a database, but it won't appear in show dbs until you insert a document into a collection within it.

Commands

- **show dbs**: Lists all the databases on the server.
```
show dbs
# Output:
# admin    40.00 KiB
# config   72.00 KiB
# local    40.00 KiB
```

- **use <dbname>**: Switches to a specified database. If the database doesn't exist, MongoDB will create it when you first insert data.
```
use myNewStore
# Output:
# switched to db myNewStore
```

- **db**: Shows the name of the currently selected database.
```
db
# Output:
# myNewStore
```

4. Collections & Documents

What is a Collection?

A collection is a group of MongoDB documents. It is the equivalent of a table in a relational (SQL) database. A collection exists within a single database and doesn't enforce a schema.

What is a Document?

A document is a set of key-value pairs, stored in a BSON format. Documents are the basic unit of data in MongoDB and are analogous to a row in a SQL table.

BSON vs. JSON

While documents look like JSON, they are stored in **BSON (Binary JSON)**.

- **JSON (JavaScript Object Notation):** Text-based, human-readable, and great for data interchange. Limited data types (string, number, boolean, array, object).
- **BSON (Binary JSON):** Binary-encoded, making it faster to parse and traverse for machines. Supports more data types than JSON, such as ObjectId, Date, Int32, Int64, and binary data.

Commands

- **show collections**: Lists all collections in the current database.
```
# First, insert some data to create a collection automatically
db.products.insertOne({ name: "Laptop", price: 1200 })
```

```
show collections
# Output:
# products
```

5. Inserting Data

insertOne()

Inserts a single document into a collection.

- **Syntax:** db.collectionName.insertOne(document)
- **Example:**

```
db.products.insertOne({
  name: "Wireless Mouse",
  brand: "Logitech",
  price: 49.99,
  inStock: true
})

# Successful Output:
# {
#   "acknowledged": true,
#   "insertedId": ObjectId("63d8a7b2c5e6f3b2a1b0d9c4")
# }
```

insertMany()

Inserts an array of documents into a collection.

- **Syntax:** db.collectionName.insertMany([document1, document2, ...])
- **Example:**

```
db.products.insertMany([
  { name: "Keyboard", brand: "Corsair", price: 150, inStock: true,
tags: ["gaming", "mechanical"] },
  { name: "Webcam", brand: "Logitech", price: 80, inStock: false,
tags: ["video", "streaming"] },
  { name: "Monitor", brand: "Dell", price: 300, inStock: true,
tags: ["office", "4k"] }
])

# Successful Output:
# {
#   "acknowledged": true,
#   "insertedIds": [
#     ObjectId("63d8a8c4c5e6f3b2a1b0d9c5"),
#     ObjectId("63d8a8c4c5e6f3b2a1b0d9c6"),
#     ObjectId("63d8a8c4c5e6f3b2a1b0d9c7")
#   ]
```

```
# }
```

6. Querying Data

find() and findOne()

- **findOne()**: Returns the first document that matches a query.
- **find()**: Returns a cursor to all documents that match a query. If the query is empty ({}), it returns all documents in the collection.
- **Syntax:** db.collectionName.find(query)
- **Example 1: Find all products**

```
db.products.find({})

# Output (may be formatted differently in your shell):
# [
#   { "_id": ..., "name": "Laptop", "price": 1200 },
#   { "_id": ..., "name": "Wireless Mouse", "brand": "Logitech",
"price": 49.99, "inStock": true },
#   ... (and so on)
# ]
```

- **Example 2: Find all products from the brand "Logitech"**

```
db.products.find({ brand: "Logitech" })

# Output:
# [
#   { "_id": ..., "name": "Wireless Mouse", "brand": "Logitech",
"price": 49.99, "inStock": true },
#   { "_id": ..., "name": "Webcam", "brand": "Logitech", "price":
80, "inStock": false, "tags": [...] }
# ]
```

- **Example 3: Find one product**

```
db.products.findOne({ brand: "Dell" })

# Output:
# { "_id": ..., "name": "Monitor", "brand": "Dell", "price": 300,
"inStock": true, "tags": [...] }
```

7. Query Operators

Comparison Operators

Operator	Description	Example
$eq	Equal to	db.products.find({ price: { $eq:

Operator	Description	Example
		300 } })
$ne	Not equal to	db.products.find({ brand: { $ne: "Logitech" } })
$gt	Greater than	db.products.find({ price: { $gt: 100 } })
$lt	Less than	db.products.find({ price: { $lt: 100 } })
$gte	Greater than or equal to	db.products.find({ price: { $gte: 300 } })
$lte	Less than or equal to	db.products.find({ price: { $lte: 80 } })

- **Practical Example (Greater than): Find all products costing more than $200.**

```
db.products.find({ price: { $gt: 200 } })

# Output:
# [
#   { "_id": ..., "name": "Laptop", "price": 1200 },
#   { "_id": ..., "name": "Monitor", "brand": "Dell", "price":
300, "inStock": true, ... }
# ]
```

Logical Operators

Operator	Description	Example
$and	All conditions must be true	db.products.find({ $and: [{ brand: "Logitech" }, { inStock: true }] })
$or	At least one condition must be true	db.products.find({ $or: [{ price: { $lt: 50 } }, { inStock: false }] })
$not	Inverts the condition	db.products.find({ price: { $not: { $gt: 100 } } }) (price not > 100)
$nor	All conditions must be false	db.products.find({ $nor: [{ price: { $lt: 100 } }, { brand: "Dell" }] })

- **Practical Example (AND): Find Logitech products that are in stock.**

```
db.products.find({ brand: "Logitech", inStock: true }) // Implicit
AND
// OR
db.products.find({ $and: [{ brand: "Logitech" }, { inStock: true
}] })

# Output:
# [{ "_id": ..., "name": "Wireless Mouse", "brand": "Logitech",
"price": 49.99, "inStock": true }]
```

Nesting Queries (Dot Notation)

To query a field inside an embedded document, use dot notation.

- **Example:** Let's add a document with nested data.
```
db.users.insertOne({
  name: "Charlie",
  contact: {
    email: "charlie@example.com",
    phone: "555-0101"
  }
})
```

- Now, find the user by their email:
```
db.users.find({ "contact.email": "charlie@example.com" })
```

8. Updating Data

updateOne()

Updates the first document that matches the query.

- **Syntax:** db.collectionName.updateOne(filter, update, options)
- **Example:** Use $set to change the price of the "Webcam".
```
db.products.updateOne(
  { name: "Webcam" }, // Filter: which document to update
  { $set: { price: 75.50, inStock: true } } // Update: what
changes to make
)
```

updateMany()

Updates all documents that match the query.

- **Syntax:** db.collectionName.updateMany(filter, update, options)
- **Example:** Add a "clearance" tag to all items under $100. Use $push\ to add an item to an array.
```
db.products.updateMany(
  { price: { $lt: 100 } },
  { $push: { tags: "clearance" } }
)
```

Common Update Operators

Operator	Description
$set	Sets the value of a field.

Operator	Description
$unset	Removes a field from a document.
$inc	Increments the value of a field by a specified amount.
$push	Adds an element to an array.
$pull	Removes all instances of a value from an array.

9. Deleting Data

deleteOne()

Deletes the first document that matches the filter.

- **Example:** Delete the "Webcam".
  ```
  db.products.deleteOne({ name: "Webcam" })
  ```

deleteMany()

Deletes all documents that match the filter.

- **Example:** Delete all products from the brand "Corsair".
  ```
  db.products.deleteMany({ brand: "Corsair" })
  ```

Dropping a Collection

Deletes an entire collection, including all its documents and indexes.

- **Syntax:** db.collectionName.drop()
- **Example:**
  ```
  db.products.drop()
  ```

Dropping a Database

Deletes the currently selected database. **Use with caution!**

- **Syntax:** db.dropDatabase()
- **Example:**
  ```
  use myNewStore     // Make sure you are in the correct database
  db.dropDatabase()
  ```

10. Wrap-Up / Summary

Quick Command Reference

Command	Usage
show dbs	Lists all databases.
use <db>	Switches to a database.
show collections	Lists all collections in the current DB.

Command	Usage
db.coll.insertOne()	Inserts one document.
db.coll.insertMany()	Inserts many documents.
db.coll.find()	Finds documents based on a query.
db.coll.findOne()	Finds the first document matching a query.
db.coll.updateOne()	Updates the first document matching a query.
db.coll.updateMany()	Updates all documents matching a query.
db.coll.deleteOne()	Deletes the first document matching a query.
db.coll.deleteMany()	Deletes all documents matching a query.
db.coll.drop()	Deletes a collection.
db.dropDatabase()	Deletes the current database.

Best Practices for Beginners

1. **Choose Meaningful Field Names:** Keep your field names descriptive and consistent.
2. **Embed vs. Reference:** If data is closely related and read together (like comments on a blog post), embedding it in the same document is efficient. If data is large or accessed independently, reference it using its _id.
3. **Use the Right Data Types:** Store dates as Date objects, not strings. Use numbers for numerical data. This makes querying much more powerful.
4. **Indexes Improve Performance:** For fields that you query often, create an index. Indexes dramatically speed up read operations.
5. **Be Careful with drop:** The drop() and dropDatabase() commands are permanent and cannot be undone. Always double-check which database/collection you are in before running them.

Happy coding!