MongoDB

HANDBOOK

CODEWITHHARRY

MongoDB Handbook

1. Introduction

In this MongoDB handbook, we'll cover **everything you need to use MongoDB confidently** as fast as possible. We'll use an **e-commerce website** as our primary example with three main collections:

- `products` : store products information
- `orders` : store order details
- `contacts` : messages from the contact form

You'll learn all essential MongoDB operations using the latest MongoDB features.

2. MongoDB Basics

What is MongoDB?

MongoDB is a **NoSQL document-oriented database**. It stores data in **JSON-like documents** (actually **BSON**, a binary form of JSON).

Key Components

- `mongod` – the main **MongoDB server daemon**. It handles connections, database storage, and queries. You must run this before using MongoDB locally.

- `mongosh` – the **MongoDB Shell**. It's the command-line interface for interacting with MongoDB. You use it to run commands, queries, and scripts.

- **MongoDB Compass** – a **GUI tool** for visualizing data, running queries, and analyzing performance. Perfect for beginners who prefer a graphical view.

3. Installing MongoDB

Installing MongoDB can be easily done by visiting the official website. The community edition is free and suitable for most use cases.

4. Basic Mongo Shell Commands

```
show dbs                 // List all databases
use ecommerce            // Create or switch to a database
show collections         // List all collections in the current DB
db.dropDatabase()        // Drop current database
```

5. Creating Collections and Inserting Data

We'll create three collections:

- `products`
- `orders`
- `contacts`

Insert Sample Products

Let's populate our database using `insertMany()`.

```
use ecommerce

db.products.insertMany([
  {
    name: "Wireless Mouse",
    price: 799,
    category: "Electronics",
```

```
    stock: 120,

    ratings: 4.5,

    tags: ["computer", "accessory", "wireless"],

    createdAt: new Date()

  },
  {

    name: "Mechanical Keyboard",

    price: 2499,

    category: "Electronics",

    stock: 80,

    ratings: 4.8,

    tags: ["keyboard", "mechanical"],

    createdAt: new Date()

  },
  {

    name: "Gaming Laptop",

    price: 85999,

    category: "Computers",

    stock: 30,

    ratings: 4.6,

    tags: ["gaming", "laptop"],

    createdAt: new Date()

  }

])
```

Insert Orders

```
db.orders.insertMany([
  {

    orderId: "ORD001",

    user: "John Doe",

    products: [

      { name: "Wireless Mouse", quantity: 1, price: 799 },

      { name: "Mechanical Keyboard", quantity: 1, price: 2499 }

    ],

    total: 3298,

    status: "Delivered",

    createdAt: new Date()

  },
```

```
  {
    orderId: "ORD002",
    user: "Jane Smith",
    products: [
      { name: "Gaming Laptop", quantity: 1, price: 85999 }
    ],
    total: 85999,
    status: "Pending",
    createdAt: new Date()
  }
])
```

Insert Contact Messages

```
db.contacts.insertMany([
  { name: "Alice", message: "Loved your website!", phone: "9876543210", createdAt:
new Date() },
  { name: "Bob", message: "Do you have discounts on laptops?", phone: "9123456789",
createdAt: new Date() },
  { name: "Carol", message: "I want to cancel my order.", phone: "9988776655",
createdAt: new Date() }
])
```

6. Reading Data (Find Queries)

Find All Documents

```
db.products.find()
```

Pretty Print

```
db.products.find().pretty()
```

Filter by Field

```
db.products.find({ category: "Electronics" })
```

Using Comparison Operators

```
db.products.find({ price: { $gt: 1000 } })        // greater than 1000
db.products.find({ price: { $gte: 1000, $lte: 50000 } })
```

Logical Operators

```
db.products.find({ $or: [{ category: "Electronics" }, { stock: { $lt: 50 } }] })
```

Projection (Select Specific Fields)

```
db.products.find({}, { name: 1, price: 1, _id: 0 })
```

Sorting and Limiting

```
db.products.find().sort({ price: -1 }).limit(2)
```

7. Updating Documents

Update One

```
db.products.updateOne(
  { name: "Wireless Mouse" },
  { $set: { price: 899 } }
)
```

Update Many

```
db.products.updateMany(
  { category: "Electronics" },
  { $inc: { stock: 10 } }
)
```

Using `$push` to Add to Arrays

```
db.products.updateOne(
  { name: "Wireless Mouse" },
  { $push: { tags: "new" } }
)
```

8. Deleting Documents

Delete One

```
db.contacts.deleteOne({ name: "Alice" })
```

Delete Many

```
db.orders.deleteMany({ status: "Delivered" })
```

9. Indexing and Performance

Create an Index

```
db.products.createIndex({ name: 1 })
```

View All Indexes

```
db.products.getIndexes()
```

Explain Query Performance

```
db.products.find({ price: { $gt: 5000 } }).explain("executionStats")
```

---

10. Aggregation Framework

Basic Example

Total revenue from all orders:

```
db.orders.aggregate([
  { $group: { _id: null, totalRevenue: { $sum: "$total" } } }
])
```

Group by Status

```
db.orders.aggregate([
  { $group: { _id: "$status", totalOrders: { $sum: 1 } } }
])
```

Lookup (Join Orders with Products)

```
db.orders.aggregate([
  {
    $lookup: {
      from: "products",
      localField: "products.name",
      foreignField: "name",
      as: "productDetails"
```

```
    }
  }
])
```

11. MongoDB Atlas (Cloud)

Steps to Use Atlas

1. Go to https://cloud.mongodb.com
2. Create a free cluster (Serverless recommended)
3. Whitelist your IP and create a DB user
4. Follow the connection string instructions to connect via `mongosh` or your application

12. Aggregation Pipeline

Aggregation pipelines in MongoDB are used to **analyze and transform data**. They work in **stages**: each stage performs an operation, and the output of one stage becomes the input of the next.

Create a sample collection

Let's start with a `sales` collection.

```
use ecommerce;

db.sales.insertMany([
  { _id: 1, item: "Apple",  price: 10, quantity: 5, category: "Fruit" },
  { _id: 2, item: "Banana", price: 5,  quantity: 10, category: "Fruit" },
  { _id: 3, item: "Carrot", price: 8,  quantity: 6, category: "Vegetable" },
  { _id: 4, item: "Tomato", price: 6,  quantity: 8, category: "Vegetable" },
```

```
    { _id: 5, item: "Mango",  price: 15, quantity: 3, category: "Fruit" }
]);
```

This collection has 5 documents. Each document represents a product with its price, quantity, and category.

What is an aggregation pipeline?

Think of it like a factory line:

- Each **stage** takes input (your data)
- Performs some operation (like filtering, sorting, grouping)
- Sends the result to the next stage

Example structure:

```
db.sales.aggregate([
  { /* stage 1 */ },
  { /* stage 2 */ },
  { /* stage 3 */ }
]);
```

Aggregation Pipeline Example 1: `$match` : Filter documents

Let's get only the sales where category is `"Fruit"` :

```
db.sales.aggregate([
  { $match: { category: "Fruit" } }
]);
```

Output:

```
{ _id: 1, item: "Apple",  price: 10, quantity: 5,  category: "Fruit" }
{ _id: 2, item: "Banana", price: 5,  quantity: 10, category: "Fruit" }
{ _id: 5, item: "Mango",  price: 15, quantity: 3,  category: "Fruit" }
```

Aggregation Pipeline Example 2: `$project` : Select specific fields

Let's display only `item` and `price` , and hide `_id` :

```
db.sales.aggregate([
  { $project: { _id: 0, item: 1, price: 1 } }
]);
```

Output:

```
{ item: "Apple", price: 10 }
{ item: "Banana", price: 5 }
{ item: "Carrot", price: 8 }
{ item: "Tomato", price: 6 }
{ item: "Mango", price: 15 }
```

Aggregation Pipeline Example 3: `$group` : Group and calculate totals

Let's calculate total sales (price × quantity) for each **category**:

```
db.sales.aggregate([
  {
    $group: {
      _id: "$category",
      totalSales: { $sum: { $multiply: ["$price", "$quantity"] } }
    }
```

```
  }
]);
```

Output:

```
{ _id: "Fruit", totalSales: 145 }
{ _id: "Vegetable", totalSales: 96 }
```

How this works:

- `$group` groups all documents by `category`
- `$sum` adds up the result of `price × quantity` for each document

Aggregation Pipeline Example 4: `$sort` : Sort results

Sort the total sales in descending order:

```
db.sales.aggregate([
  {
    $group: {
      _id: "$category",
      totalSales: { $sum: { $multiply: ["$price", "$quantity"] } }
    }
  },
  { $sort: { totalSales: -1 } }
]);
```

Output:

```
{ _id: "Fruit", totalSales: 145 }
{ _id: "Vegetable", totalSales: 96 }
```

Aggregation Pipeline Example 5: Combine `$match` + `$group`

Find total sales for **only Fruits**:

```
db.sales.aggregate([
  { $match: { category: "Fruit" } },
  {
    $group: {
      _id: null,
      totalFruitSales: { $sum: { $multiply: ["$price", "$quantity"] } }
    }
  }
]);
```

Output:

```
{ totalFruitSales: 145 }
```

Aggregation Pipeline Summary table

Stage	Description	Example Use
`$match`	Filter documents	`{ category: "Fruit" }`
`$project`	Show/hide fields or create new ones	`item`, `price`
`$group`	Group data and calculate sums, averages	total sales per category
`$sort`	Sort results	Sort by total sales
`$limit`	Limit number of results	Top 3 items

The stages you've learned ($match, $project, $group, $sort, $limit) are just the most common and beginner-friendly ones: often called the core stages.

But MongoDB actually supports dozens of stages (over 25).

---

13. MongoDB Indexes

Indexes in MongoDB are special data structures that improve the speed of read operations on a collection. They work similarly to indexes in books, allowing the database to quickly locate and access the data without scanning every document.

Create Index

```
db.products.createIndex({ name: 1 })  // Ascending index on 'name' field
```

View Indexes

```
db.products.getIndexes()  // List all indexes on 'products' collection
```

Should you always create indexes?

While indexes improve read performance, they can slow down write operations (inserts, updates, deletes) because the index must also be updated. Therefore, it's essential to create indexes thoughtfully based on your application's query patterns.

14. Useful Admin Commands

```
db.stats()                    // Show DB stats
db.serverStatus()             // Server info
db.products.countDocuments()  // Count documents
db.products.renameCollection("items")  // Rename collection
db.products.drop()            // Drop collection
```