MongoDB Daily Practice Guide (Beginner → Comfortable)

You can **copy–paste and practice each command** in MongoDB Shell or MongoDB Compass or insert using cmd prompt.

Step 0: Database

```
use practiceDB
```

This creates (or switches to) a database called `practiceDB`.

Step 1: Create 3 Collections

We will work with **3 collections**:

1. `users`
2. `products`
3. `orders`

Step 2: Insert Sample Data

1️⃣ Users Collection

```
db.users.insertMany([
  { name: "Arun", age: 25, city: "Chennai", role: "user" },
  { name: "Bala", age: 30, city: "Bangalore", role: "admin" },
  { name: "Charan", age: 22, city: "Chennai", role: "user" },
  { name: "Divya", age: 28, city: "Delhi", role: "user" },
  { name: "Esha", age: 35, city: "Mumbai", role: "manager" }
])
```

What this does:

- Creates multiple user documents
- Each user has name, age, city, role

2️⃣ Products Collection

```
db.products.insertMany([
  { name: "Laptop", price: 60000, category: "Electronics", stock: 10 },
  { name: "Mobile", price: 30000, category: "Electronics", stock: 25 },
  { name: "Table", price: 8000, category: "Furniture", stock: 5 },
  { name: "Chair", price: 4000, category: "Furniture", stock: 20 }
])
```

3️⃣ Orders Collection

```
db.orders.insertMany([
  { user: "Arun", product: "Laptop", amount: 60000, status: "completed" },
  { user: "Arun", product: "Mobile", amount: 30000, status: "pending" },
  { user: "Divya", product: "Chair", amount: 4000, status: "completed" },
  { user: "Esha", product: "Table", amount: 8000, status: "completed" }
])
```

Step 3: find() – Read Data

Get all users

```
db.users.find()
```

Meaning:

- Returns **all documents** from users collection

Find users from Chennai

```
db.users.find({ city: "Chennai" })
```

Meaning:

- Filters data based on condition

Step 4: projection – Select Fields

Show only name and city

```
db.users.find(
  { city: "Chennai" },
  { name: 1, city: 1, _id: 0 }
)
```

Meaning:

- `1` → include field
- `0` → exclude field
- `_id` is excluded manually

Step 5: sort()

Sort users by age (young → old)

```
db.users.find().sort({ age: 1 })
```

Sort users by age (old → young)

```
db.users.find().sort({ age: -1 })
```

Step 6: limit()

Get only first 2 users

```
db.users.find().limit(2)
```

Use case:

- Dashboards
- Pagination

Step 7: skip()

Skip first 2 users

```
db.users.find().skip(2)
```

Pagination example

```
db.users.find().skip(2).limit(2)
```

Step 8: countDocuments()

Count all users

```
db.users.countDocuments()
```

Count users from Chennai

```
db.users.countDocuments({ city: "Chennai" })
```

Step 9: distinct()

Get unique cities

```
db.users.distinct("city")
```

Get unique roles

```
db.users.distinct("role")
```

Step 10: Comparison Operators (Very Important)

Comparison operators are used to **compare values** in queries.

🔹 List of Comparison Operators

Operator	Meaning	Simple Explanation
`$eq`	Equal	Matches values exactly equal to given value
`$ne`	Not Equal	Matches values NOT equal to given value
`$gt`	Greater Than	Matches values greater than given number
`$gte`	Greater Than or Equal	Matches values >= given number
`$lt`	Less Than	Matches values less than given number
`$lte`	Less Than or Equal	Matches values <= given number
`$in`	In Array	Matches any value in given list
`$nin`	Not In Array	Excludes values in given list

🗓️ Examples + Real-time Scenarios

1️⃣ `$eq` – Equal

```
db.users.find({ role: { $eq: "admin" } })
```

Real-time use:

- Find all admin users

2️⃣ `$ne` – Not Equal

```
db.users.find({ city: { $ne: "Chennai" } })
```

Real-time use:

- Find users NOT from Chennai

3️⃣ `$gt` – Greater Than

```
db.users.find({ age: { $gt: 25 } })
```

Real-time use:

- Find users older than 25

4️⃣ `$gte` – Greater Than or Equal

```
db.products.find({ price: { $gte: 30000 } })
```

Real-time use:

- Products costing 30k or more

5️⃣ `$lt` – Less Than

```
db.products.find({ stock: { $lt: 10 } })
```

Real-time use:

- Low stock alert products

6️⃣ `$lte` – Less Than or Equal

```
db.users.find({ age: { $lte: 25 } })
```

Real-time use:

- Young users (≤ 25)

7️⃣ `$in` – Match Multiple Values

```
db.users.find({ city: { $in: ["Chennai", "Delhi"] } })
```

Real-time use:

- Users from selected cities

8️⃣ `$nin` – Exclude Multiple Values

```
db.users.find({ role: { $nin: ["admin", "manager"] } })
```

Real-time use:

- Normal users only

Step 11: Logical Operators (How Conditions Combine)

Logical operators combine **multiple conditions**.

🔹 List of Logical Operators

Operator	Meaning	Simple Explanation
`$and`	AND	All conditions must be true
`$or`	OR	Any one condition must be true
`$not`	NOT	Negates a condition
`$nor`	NOR	All conditions must be false

🧳 Examples + Real-time Scenarios

1️⃣ `$and` – All Conditions Must Match

```
db.users.find({
  age: { $gt: 25 },
  city: "Chennai"
})
```

Real-time use:

- Users older than 25 AND from Chennai

2️⃣ `$or` – Any One Condition

```
db.users.find({
  $or: [
    { city: "Chennai" },
    { city: "Delhi" }
  ]
})
```

Real-time use:

- Users from Chennai OR Delhi

3️⃣ `$not` – Reverse Condition

```
db.users.find({ age: { $not: { $gt: 30 } } })
```

Real-time use:

- Users age NOT greater than 30

4️⃣ `$nor` – None Should Match

```
db.users.find({
  $nor: [
    { city: "Mumbai" },
    { role: "admin" }
  ]
})
```

Real-time use:

- Users NOT from Mumbai AND NOT admin

Step 12: Advanced find() – Combination Queries

These are **real backend queries** used daily.

1️⃣ Find users from Chennai older than 23

```
db.users.find({
  city: "Chennai",
  age: { $gt: 23 }
})
```

2️⃣ Find users from Chennai or Delhi AND age > 25

```
db.users.find({
  $and: [
    { age: { $gt: 25 } },
    { $or: [
        { city: "Chennai" },
        { city: "Delhi" }
      ]
    }
```

```
  ]
})
```

Real-time use:

- Filter users for targeted notifications

3️⃣ Find products with low stock and high price

```
db.products.find({
  stock: { $lt: 10 },
  price: { $gt: 20000 }
})
```

Real-time use:

- Inventory risk analysis

Step 11: Logical Operators

Users from Chennai OR Delhi

```
db.users.find({
  $or: [
    { city: "Chennai" },
    { city: "Delhi" }
  ]
})
```

Users age > 25 AND role = user

```
db.users.find({
  age: { $gt: 25 },
  role: "user"
})
```

Step 12: update()

Update one user

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

Update many users

```
db.users.updateMany(
  { city: "Chennai" },
  { $set: { active: true } }
)
```

Step 13: delete()

Delete one document

```
db.users.deleteOne({ name: "Charan" })
```

Delete many documents

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

Final Practice Goal ✅

After practicing this document, you should be comfortable with:

- Creating collections
- Inserting data
- Reading data with filters
- Sorting, limiting, skipping
- Counting documents
- Getting distinct values
- Using comparison & logical operators
- Updating & deleting data

Real MongoDB Interview Questions (With Simple Answers)

These are **real questions** asked in Node.js / Backend interviews.

1️⃣ What is the difference between `find()` and `findOne()`?

Answer:

- `find()` returns a cursor (multiple documents)
- `findOne()` returns only the **first matching document**

2️⃣ What is projection in MongoDB?

Answer: Projection is used to **select only required fields** from documents to reduce data transfer.

3️⃣ Difference between `$gt` and `$gte`?

Answer:

- `$gt` → greater than
- `$gte` → greater than or equal to

4️⃣ How do you implement pagination in MongoDB?

Answer: Using `skip()` and `limit()` together.

```
db.users.find().skip(10).limit(10)
```

5️⃣ Difference between `$or` and `$and`?

Answer:

- `$or` → any one condition must match
- `$and` → all conditions must match

6️⃣ What is `countDocuments()` used for?

Answer: To count the number of documents matching a condition.

7️⃣ What does `distinct()` do?

Answer: Returns **unique values** of a field from a collection.

8️⃣ Difference between `updateOne()` and `updateMany()`?

Answer:

- `updateOne()` updates only first matched document
- `updateMany()` updates all matched documents

9️⃣ Can we combine multiple conditions in `find()`?

Answer: Yes, using logical operators like `$and`, `$or`, `$nor`.

🔟 Is MongoDB schema-less?

Answer: Yes, but in real applications we **follow a fixed structure** using validations or Mongoose schemas.

Daily Practice Tasks (Do These to Become Confident)

Try to solve these **without looking at answers first**.

🟢 Easy Tasks

1. Find all users whose age is less than 30
2. Find all products with price greater than 10,000
3. Get only names of users from Delhi
4. Count how many users are from Chennai
5. Get all unique cities from users

🟡 Medium Tasks

1. Find users from Chennai or Bangalore
2. Find users whose age is between 25 and 35
3. Sort products by price (high to low)
4. Find products with stock less than 10
5. Skip first 1 user and show next 2 users

🔴 Advanced Tasks (Real Backend Level)

1. Find users from Chennai or Delhi AND age > 25

2. Find products that are Electronics and price > 40,000
3. Find orders that are completed
4. Update all users from Mumbai to active = false
5. Delete orders with pending status

How to Know You Are Ready ✅

You are MongoDB-ready if:

- You can read a requirement and write a query
- You can explain WHY you used `$and` or `$or`
- You can predict query output before running it
