MongoDB Fundamentals – Detailed Beginner Notes

1. What is MongoDB?

MongoDB is a **NoSQL (Not Only SQL)** database used to store data in a **document-based format** instead of traditional tables.

- Data is stored as **documents** (key–value pairs)
- Documents are stored inside **collections**
- Collections are stored inside a **database**

Data format

MongoDB uses **JSON-like documents** (internally BSON).

Example document:

```
{
  "name": "Tarun",
  "age": 21,
  "branch": "CSE"
}
```

2. Why MongoDB? (Why NoSQL came)

Traditional **RDBMS** databases store data in **tables (rows & columns)** with a fixed structure. This causes problems in modern applications.

Problems with RDBMS:

- Fixed schema (hard to change)
- Poor performance with huge data
- Scaling is difficult
- Not suitable for unstructured / semi-structured data

MongoDB solves this by:

- Allowing flexible structure
- Handling large-scale data easily
- Being fast and scalable

3. Advantages of MongoDB over RDBMS

Advantages:

- Schema-less (flexible fields)
- Easy to scale horizontally
- Faster read/write for big data
- Stores JSON-like data (natural for apps)
- No complex joins

Example flexibility:

```
{ "name": "A" }
{ "name": "B", "age": 22, "skills": ["Java"] }
```

Both are valid in the same collection.

---

4. Disadvantages of MongoDB (compared to RDBMS)

- No strong ACID transactions (limited support)
- Not ideal for banking/financial systems
- No joins like SQL (manual or embedded)
- Data duplication can happen

---

5. MongoDB vs RDBMS (Key Differences)

RDBMS	MongoDB
Tables	Collections
Rows	Documents
Columns	Fields
Fixed schema	Flexible schema
SQL	MongoDB Query Language
CREATE TABLE	Insert document

---

6. MongoDB Server vs MongoDB Shell

mongod (MongoDB Server)

- Actual database engine
- Runs in background
- Stores data on disk

- Listens on port 27017

mongosh (MongoDB Shell)

- Client program
- Used to run commands
- Does NOT store data
- Communicates with mongod

Execution flow:

```
User → mongosh → mongod → Disk
```

Analogy: - mongod = kitchen - mongosh = waiter

---

7. Where MongoDB Data is Stored

MongoDB stores data in **binary files** (not readable manually).

Typical location (Windows):

```
C:\Program Files\MongoDB\Server\<version>\data
```

Files like:

```
collection-0.wt
index-1.wt
```

---

8. Database Creation in MongoDB

MongoDB uses **lazy creation**.

Step 1: Switch database

```
use practice_db
```

- This does NOT create the database
- Just switches context

Step 2: Insert data (this creates DB)

```
db.students.insertOne({ name: "Tarun", age: 21 })
```

Now: - Database is created - Collection is created - Data is stored

---

9. show dbs Command

```
show dbs
```

What it shows:

- Only databases that **contain data**
- Empty databases are NOT shown

Important rule: > A database appears in `show dbs` only after data is inserted.

---

10. Database vs Collection

Database:

- Container for collections
- Example: `practice_db`

Collection:

- Container for documents
- Example: `students`, books

Hierarchy:

```
Database
 └── Collection
      └── Document
```

---

11. Creating Collections

MongoDB does NOT use `CREATE TABLE`.

Collections are created automatically:

```
db.books.insertOne({ title: "MongoDB", price: 399 })
```

Check collections:

```
show collections
```

---

12. Switching Databases

Switch to a database:

```
use practice_db
```

Check current database:

```
db
```

Wrong database typed?

Just switch again:

```
use correct_db
```

---

13. Important Rules to Remember

- No tables in MongoDB
- Database & collections created only on insert
- `use dbname` does not create DB
- `show dbs` shows only non-empty DBs
- Flexible schema is MongoDB's power

---

14. One-Line Summary

> MongoDB is a NoSQL document-based database where databases and collections are created automatically when data is inserted, offering flexibility and scalability compared to traditional RDBMS.

Next Topics (when ready)

- insertMany()
- find() with conditions
- updateOne(), deleteOne()
- MongoDB vs SQL queries

🟢 **MongoDB in VS Code – SIMPLE RUN NOTES**

🔹 **ONE-TIME SETUP**

1. Install **MongoDB Community Server**
2. Install **MongoDB for VS Code** extension
3. MongoDB service running (mongod)
4. Connect using:

mongodb://localhost:27017

Dropping a collection and database:

In case of any mistakes while creating the database or collection, you can drop it.

To drop a collection, replace "collection_name" in the below code with the actual name of the collection to be dropped. The drop() method returns true if the collection specified is successfully dropped.

Syntax:

```
1. db.collection_name.drop()
```

Query:

```
1. db.product_catalog.drop()
```

To drop the database currently in use, execute the below query.

Syntax:

```
1. db.dropDatabase()
```

This query always drops the database currently in use. On successful deletion, the below message is displayed in the shell.

```
1. { ok: 1, dropped: 'databaseName' }
```

The syntax will remain the same irrespective of the name of the database. We do not specify the database name.

🔹 DAILY PRACTICE STEPS (FOLLOW THIS ORDER)

1️⃣ Open your practice folder

mongodb-practice

2️⃣ Create a file

File name **must be**:

something.mongodb.js

Example:

01_crud.mongodb.js

3️⃣ Always start code like this

```
use("practice_db");
```

4️⃣ Write MongoDB commands

Insert

```
db.students.insertOne({
  name: "Tarun",
  stream: "Big Data",
  year: 2026
});
```

Read (IMPORTANT)

```
db.students.find().toArray();
```

5️⃣ **Run the code**

- Click ▶ **Run** (top-right)
  OR
- Right-click → **Run in MongoDB Playground**

6️⃣ **See output**

Output appears in **Playground Result (right side)**

🔹 **RULES (REMEMBER THIS ⭐)**

❌ **Don't do**

```
db.students.find();   // ❌ error in VS Code
```

✅ **Always do**

```
db.students.find().toArray();  // ✅
```

🔹 **UPDATE / DELETE (REFERENCE)**

Update

```
db.students.updateOne(
 { name: "Tarun" },
 { $set: { year: 2027 } }
);
```

Delete

```
db.students.deleteOne({ name: "Tarun" });
```

🔹 **CHECK DATA**

```
db.students.find({ year: 2026 }).toArray();
```

To get collection names

✅ **CORRECT WAY in VS Code Playground**

✔ **Use this instead:**

```
db.getCollectionNames();
```

🔹 **WHEN TO USE WHAT**

Tool	**Use**
Compass	Visual check

Tool	**Use**
VS Code	Practice & save code
mongosh	Quick tests

🧠 GOLDEN LINE (REMEMBER)

VS Code = code storage
Playground = code runner
MongoDB = offline by default

🧠 VERY IMPORTANT (remember this table)

Command	**Where it works**
show dbs	mongosh only
show collections	mongosh only
db.getCollectionNames()	VS Code ✓
db.collection.find()	mongosh
db.collection.find().toArray()	VS Code ✓

🧠 REMEMBER THIS TABLE (SAVE IT)

Command	**Where it works**
db.col.find().pretty()	mongosh only
db.col.find().toArray()	VS Code Playground
show collections	mongosh only
db.getCollectionNames()	VS Code Playground

📘 MongoDB LEARNING ORDER (WE WILL FOLLOW THIS)

1️⃣ Insert ONE document
2️⃣ Insert MULTIPLE documents
3️⃣ Read data (find)
4️⃣ Read with CONDITIONS (WHERE)
5️⃣ Update data
6️⃣ Delete data

1️⃣ DOCUMENT (lowest level – actual data)

What is a Document?

A **document** is **one single record** made of **key–value pairs**.

👉 This is the **smallest unit** in MongoDB.

Example document:

```
{
 "_id": ObjectId("696cd0e1f8f0bb0e388b87cb"),
 "title": "MongoDB Basics",
 "price": 300,
 "author": "TG"
}
```

✓ One document

✓ Like **one row** in SQL

2️⃣ COLLECTION (group of documents)

What is a Collection?

A **collection** is a **container that holds multiple documents** of similar purpose.

👉 Like a **table** in SQL (but flexible).

Example: books collection

It contains MANY documents:

```
{ "title": "MongoDB Basics", "price": 300 }
{ "title": "Java", "price": 450, "author": "X" }
{ "title": "Python" }
```

✓ All are documents

✓ All inside books collection

✓ Fields can be different

3️⃣ DATABASE (top level)

What is a Database?

A **database** is a **container for collections**.

👉 Like a **folder** that contains tables.

Example: practice_db

Inside it:

```
practice_db
 ├── students
 ├── books
 └── users
```

✓ Database contains collections
✓ Collections contain documents

4️⃣ VISUAL HIERARCHY (REMEMBER THIS)

```
Database
 └── Collection
      └── Document
```

OR with your example:

```
practice_db
 └── books
      └── { title: "MongoDB Basics", price: 300 }
```

5️⃣ SQL vs MongoDB (to lock understanding)

SQL	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field

6️⃣ REAL-LIFE ANALOGY (VERY IMPORTANT)

Think of a College:

Real Life	MongoDB
College	Database
Classroom	Collection
Student	Document
Student details	Fields

Example:

```
{
  "name": "Tarun",
  "roll": 21,
  "branch": "CSE"
}
```

7️⃣ Why MongoDB is flexible here

In SQL:

- All students must have same columns

In MongoDB:

- One student may have extra fields

```
{ "name": "A" }
{ "name": "B", "skills": ["Java"] }
```

✓ Same collection

✓ No problem

8️⃣ COMMON CONFUSION (clear now)

❌ "Each collection gets one _id"

✓ **Each document gets its own _id**

❌ "Database stores key–value pairs"

✓ **Document stores key–value pairs**

9️⃣ ONE-SENTENCE DEFINITIONS (EXAM READY)

- **Document**: A single record stored as key–value pairs
- **Collection**: A group of documents
- **Database**: A container for collections

🔍 QUICK CHECK (answer in your head)

- _id belongs to → **Document**
- books is a → **Collection**
- practice_db is a → **Database**

🟢 STEP 1: INSERT ONE DOCUMENT

First: go to correct database

In mongosh, type:

```
use practice_db
```

Check:

```
db
```

Output should be:

```
practice_db
```

🔹 insertOne() – create data

Syntax:

db.collectionName.insertOne({ key: value })

Example (TYPE THIS):

```
db.students.insertOne({
  name: "Tarun",
  age: 21,
  branch: "CSE"
})
```

Output:

```
{
  acknowledged: true,
  insertedId: ObjectId("...")
}
```

✓ This means data is stored

✓ Collection students created (if not existed)

🔹 Verify insertion

```
db.students.find()
```

You'll see:

```
{ _id: ObjectId("..."), name: "Tarun", age: 21, branch: "CSE" }
```

VERY IMPORTANT CONCEPTS HERE 🧠

1️⃣ What is _id?

- Automatically added by MongoDB
- Unique identifier (primary key equivalent)

2️⃣ Did we create a table?

❌ NO

✓ Collection created automatically

3️⃣ Did we define schema?

❌ NO

✓ MongoDB is schema-less

🔹 SQL vs MongoDB (STEP 1)

SQL	MongoDB
INSERT INTO students VALUES (...)	db.students.insertOne({...})
Primary key	_id
Table must exist	Collection auto-created

🟢 STEP 2: INSERT MULTIPLE DOCUMENTS (insertMany)

Why do we need insertMany()?

- To insert **multiple documents at once**
- Faster and cleaner than multiple insertOne()

---

1 Syntax (simple)

```
db.collectionName.insertMany([
 { document1 },
 { document2 },
 { document3 }
])
```

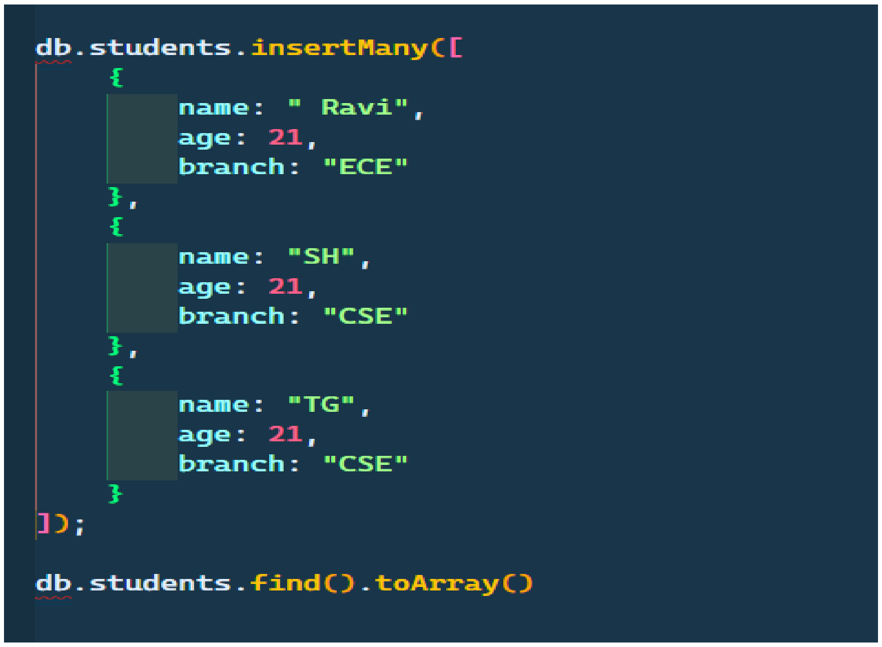
➞ Uses an **array of documents** []

---

2 Example (TYPE THIS EXACTLY)

Make sure you are in correct database:

use practice_db



```
db.students.insertMany([
    {
        name: " Ravi",
        age: 21,
        branch: "ECE"
    },
    {
        name: "SH",
        age: 21,
        branch: "CSE"
    },
    {
        name: "TG",
        age: 21,
        branch: "CSE"
    }
]);

db.students.find().toArray()
```

// This is in vs code playground(for mongosh, at last line db.students.find() is enough)

3 Output you will see

```
{
 acknowledged: true,
 insertedIds: {
  '0': ObjectId("..."),
  '1': ObjectId("..."),
  '2': ObjectId("...")
 }
```

}

Meaning:

- 3 documents inserted
- Each document got **its own _id**

4️⃣ Verify insertion

db.students.find()

You'll now see:

- Your old student(s)
- Plus Ravi, Sita, Aman

5️⃣ IMPORTANT observations 🧠

✓ Different documents, different fields

{ "name": "Aman", "branch": "ME" }

- No age
- Still valid

✓ MongoDB doesn't enforce schema

- No error
- Flexible storage

6️⃣ SQL vs MongoDB (STEP 2)

SQL	**MongoDB**
Multiple INSERT statements	insertMany()
Row-by-row	Bulk insert
Schema enforced	Schema-less

7️⃣ Common mistakes ❌

❌ Forgetting []
❌ Missing commas between documents
❌ Writing objects instead of array

Correct form:

[{ }, { }, { }]

🟢 STEP 3: READ DATA using find()

This is the **SELECT** of MongoDB.

1️⃣ Basic find() (SELECT * equivalent)

Syntax:

db.collectionName.find()

Example:

db.students.find()

👉 This returns **ALL documents** in the students collection.

2️⃣ Understand the output

Example output:

```
{
 "_id": ObjectId("..."),
 "name": "Ravi",
 "age": 20,
 "branch": "ECE"
}
```

Key points:

- Each result is a **document**
- _id is always present
- Order doesn't matter

3️⃣ Make output readable (IMPORTANT)

By default, output may look compact.

Use:

db.students.find().pretty()

✓ Same data
✓ Better readability

4️⃣ find() vs SQL

SQL	MongoDB
SELECT * FROM students;	db.students.find()

5️⃣ Reading from another collection

```
db.books.find()
```

MongoDB always follows:

```
db.<collection>.find()
```

6️⃣ What if collection doesn’t exist?

```
db.unknown.find()
```

👉 Output:

```
(empty)
```

❌ No error
✓ MongoDB is flexible

7️⃣ Important rules 🧠

- find() never modifies data
- It only **reads**
- Safe command