💡 What is MongoDB?

MongoDB is a **NoSQL database** — meaning it stores data in a flexible, JSON-like format called **BSON** (Binary JSON), rather than traditional rows and columns like SQL databases.

- Think of MongoDB as your **database engine**.
- It gives you commands to **store, query, update, and delete data**.
- You talk to MongoDB using **drivers** like the official MongoDB Node.js driver (mongodb package).

📦 Example (Native MongoDB - using driver only):

```
const { MongoClient } = require('mongodb');

const client = await MongoClient.connect(URL);

const db = client.db('myDB');

await db.collection('users').insertOne({ name: 'Sagar', age: 24 });
```

🧠 What is Mongoose?

Mongoose is an **Object Data Modeling (ODM)** library for MongoDB and Node.js.

It provides a **schema-based solution** to structure your data. It's like a translator that sits between your Node.js code and the MongoDB database.

👉 With Mongoose, you define models using **schemas**, and you get extra features like:

- **Validation**
- **Middleware/hooks**
- **Default values**
- **Virtuals (computed fields)**
- **Relationships (via population)**
- Easier querying & saving

📦 Example (Using Mongoose):

```
const mongoose = require('mongoose');
```

```
const userSchema = new mongoose.Schema({
 name: { type: String, required: true },
 age: Number,
});
const User = mongoose.model('User', userSchema);
await mongoose.connect(URL);
const newUser = new User({ name: 'Sagar', age: 24 });
await newUser.save();
```

App.js(connnect to mongoose)

```
mongoose
 .connect(db_path)
 .then(() => {
  console.log("Database Connected...");
  app.listen(PORT, () => {
   console.log(`Server running at address http://localhost:${PORT}`);
  });
 })
 .catch((err) => {
  console.log("Error Occur while connect to database:", err);
 });
```

🔍 **Key Differences: MongoDB Native Driver vs Mongoose**

Feature	MongoDB Native Driver	Mongoose
Level	Low-level	High-level abstraction
Schema definition	❌ No schema enforced	✅ Schema-based
Validation	Manual	Built-in

Feature	MongoDB Native Driver	Mongoose
Middleware (hooks)	❌	✅ pre, post hooks (like save, remove)
Query builder	Basic	Advanced, chainable
Population (joins)	Manual	Built-in
Ease of use	Low	High (especially for beginners)

✅ **Why use Mongoose?**

- **Structure**: Helps you define clear models and rules (schemas).
- **Cleaner Code**: Abstracts complex MongoDB operations into simpler Mongoose methods.
- **Validation**: Ensures your data meets requirements before saving.
- **Middleware**: Run logic before/after saving, removing, etc.
- **Relationships**: Simulates SQL-like joins using .populate().

🧠 **When NOT to use Mongoose?**

If you need:

- **Full flexibility**, without schema restrictions
- Raw performance with **large-scale, low-level operations**
- Use with **modern alternatives** like Prisma (MongoDB support is growing there)

Then the **MongoDB driver** might be better.

🚀 **In short:**

- **MongoDB** is the database itself.
- **Mongoose** is a tool to **interact with MongoDB** more efficiently, with schemas and extra features.

🔧 Think of MongoDB as a blank notebook, and Mongoose as the organizer that gives you templates, rules, and automation.

16.1 What is MongoDB

1. MongoDB is the product and the company that builds it.
2. The name comes from the work Humongous.
3. NoSQL Document Database: Stores data in flexible, JSON-like documents.
4. Dynamic Schema: Allows fields to vary across documents without predefined schemas.
5. High Performance: Optimized for fast read and write operations.
6. Scalability: Supports horizontal scaling through sharding.
7. High Availability: Provides replication with automatic failover.
8. Rich Query Capabilities: Offers powerful querying, indexing, and aggregation.
9. Geospatial and Text Search: Includes support for location-based and full-text queries.
10. Cross-Platform Compatibility: Works with various operating systems and programming languages.
11. Easy Integration: Integrates smoothly with modern development stacks.


```
{
  "_id": "4f5b5c85-d8d3-4f58-8acf-3f5e5e4e59ea",
  "Items": [
    {
      "ProductId": 1,
      "ProductName": "Elden Ring",
      "Price": "49.97",
      "Quantity": 1
    },
    {
      "ProductId": 2,
      "ProductName": "FIFA 23",
      "Price": "69.97",
      "Quantity": 1
    }
  ]
}
```


17.9 Using Mongoose for Favourite

4.

```
homeSchema.pre('findOneAndDelete', async function(next) {
  const homeId = this.getQuery()["_id"];
  await Favourite.deleteMany({ homeId: homeId });
  next();
});
```

17.10 Fetching Relations

```
exports.getFavourites = (req, res, next) => {
  Favourite.find()
    .populate("homeId")
    .then((favourites) => {
      const favouriteHomes = favourites.map((favourite) => favourite.homeId);
      res.render("store/favourites", {
        homes: favouriteHomes,
        pageTitle: "Favourites",
      });
  });
};
```

Mongoose Syntax & Theory Cheat Sheet

What is Mongoose?

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a straightforward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks, and more.

Difference Between MongoDB and Mongoose

- MongoDB: NoSQL database for storing documents in a flexible, JSON-like format (BSON).
- Mongoose: ODM library that provides schema structure, validation, and powerful utilities on top of the MongoDB native driver.

Mongoose Setup & Schema Definition

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/mydb');

const userSchema = new mongoose.Schema({
  name: String,
  age: Number,
  email: { type: String, required: true }
});
const User = mongoose.model('User', userSchema);
```

Create (Insert)

```
const user = new User({ name: 'John', age: 30, email: 'john@example.com' });
user.save().then(result => console.log(result));
```

Read (Find)

```
User.find().then(users => console.log(users));
User.findById('id_here').then(user => console.log(user));
```

Mongoose Syntax & Theory Cheat Sheet

Update

```
User.updateOne({ _id: 'id_here' }, { $set: { name: 'Jane' } })
  .then(result => console.log(result));
```

Delete

```
User.deleteOne({ _id: 'id_here' })
  .then(result => console.log(result));
```

Extra Useful Mongoose Methods

- findOne(filter): Finds the first match.
- findByIdAndUpdate(id, update, options): Updates and returns the updated document.
- findByIdAndDelete(id): Deletes and returns the deleted document.
- Model.countDocuments(): Returns count of matching documents.
- Model.exists(filter): Checks if a document matching filter exists.

Essential Mongoose Syntax for CRUD Operations

1. Connect to MongoDB

```
mongoose.connect('mongodb://localhost:27017/dbname')
  .then(() => console.log("Connected"))
  .catch(err => console.error(err));
```

2. Define Schema

```
const userSchema = new mongoose.Schema({
  name: String,
  age: Number,
  email: { type: String, required: true, unique: true },
  createdAt: { type: Date, default: Date.now }
});
```

3. Create Model

```
const User = mongoose.model('User', userSchema);
```

4. Create Document

```
const newUser = new User({ name: 'Sagar', age: 22, email: 'sagar@example.com' });
newUser.save().then(user => console.log(user));
```

5. Read Documents

```
User.find().then(users => console.log(users));
User.findOne({ name: 'Sagar' }).then(user => console.log(user));
User.findById('id').then(user => console.log(user));
```

6. Update Document

```
User.updateOne({ _id: 'id' }, { $set: { age: 23 } });
User.findByIdAndUpdate('id', { age: 25 }, { new: true });
```

7. Delete Document

```
User.deleteOne({ _id: 'id' });
User.findByIdAndDelete('id');
```

8. Query Operators

```
User.find({ age: { $gte: 18, $lte: 30 } });
User.find({ name: /sagar/i });
```

9. Schema Relationships

```
author: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }
```

10. Middleware (Hooks)

```
userSchema.pre('save', function(next) {
  console.log('Before saving...');
  next();
});
```