Chapter 6: MongoDB with Mongoose

Table of Contents

What is MongoDB? How is it different from SQL?

Definition: MongoDB is a popular open-source NoSQL database that stores data in flexible, JSON-like documents. Unlike traditional SQL databases (like MySQL or PostgreSQL), which use tables and rows, MongoDB uses collections and documents.

Real-World Example: Imagine a filing cabinet. In SQL, every file (row) must have the same fields (columns) in the same order. In MongoDB, each file (document) can have different fields, and you can add new types of information at any time.

Key Differences:

- **Schema:** SQL databases have a fixed schema; MongoDB is schema-less (flexible structure).
- **Data Format:** SQL uses tables/rows; MongoDB uses collections/documents (JSON-like).
- **Scalability:** MongoDB is designed for horizontal scaling (easy to add more servers).

Fun Fact: MongoDB's name comes from "humongous," reflecting its ability to handle huge amounts of data.

Installing MongoDB locally or using MongoDB Atlas

Definition: You can run MongoDB on your own computer (locally) or use a cloud service like MongoDB Atlas.

Local Installation:

1. Go to mongodb.com/try/download/community
2. Download and install MongoDB Community Edition for your OS
3. Start the MongoDB server (usually with `mongod` command)

MongoDB Atlas (Cloud):

1. Go to mongodb.com/cloud/atlas

2. Create a free account
3. Set up a new cluster (cloud database)
4. Get your connection string to use in your apps

Real-World Example: Running MongoDB locally is like having a mini-fridge at home. Using Atlas is like renting a fridge in a shared kitchen that's always online and managed for you.

Basic MongoDB commands (CRUD) using Mongo shell

Definition: CRUD stands for Create, Read, Update, Delete—the four basic operations for managing data.

Mongo Shell Examples:

```
// Create (Insert)
db.users.insertOne({ name: "Alice", age: 25 })

// Read (Find)
db.users.find({ age: { $gt: 20 } })

// Update
db.users.updateOne({ name: "Alice" }, { $set: { age: 26 } })

// Delete
db.users.deleteOne({ name: "Alice" })
```

Real-World Example: Think of a contact list on your phone: adding a contact (Create), searching for a contact (Read), editing a contact (Update), and deleting a contact (Delete).

Fun Fact: MongoDB queries use JavaScript-like syntax, making it easy for JavaScript developers to learn.

What is Mongoose? Why use it?

Definition: Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a way to define schemas, models, and validation for your data.

Why use Mongoose?

- Enforces structure (schemas) on your documents
- Makes data validation easy
- Provides helpful methods for querying and updating data

Real-World Example: Mongoose is like a helpful librarian who makes sure every book (document) in the library (database) is organized and follows certain rules.

Mongoose Setup & Connection with Node

Definition: To use Mongoose, you install it in your Node.js project and connect it to your MongoDB database.

Setup Steps:

1. Install Mongoose:

```
npm install mongoose
```

2. Connect to MongoDB:

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/mydb', { useNewUrlParser: true,
useUnifiedTopology: true })
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.error('Connection error', err));
```

Real-World Example: Connecting Mongoose to MongoDB is like plugging your laptop into Wi-Fi so you can access the internet (database).

Defining Schemas and Models

Definition: A schema defines the structure of your documents. A model is a wrapper for the schema, providing an interface to interact with the database.

Example:

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

Real-World Example: A schema is like a blueprint for a house, and a model is the actual house you build and live in.

CRUD Operations using Mongoose

Definition: Mongoose provides methods to Create, Read, Update, and Delete documents in MongoDB.

Examples:

```
// Create
dbUser = new User({ name: 'Bob', age: 30, email: 'bob@example.com' });
dbUser.save();
```

```
// Read
User.find({ age: { $gt: 20 } }).then(users => console.log(users));

// Update
User.updateOne({ name: 'Bob' }, { $set: { age: 31 } });

// Delete
User.deleteOne({ name: 'Bob' });
```

Real-World Example: Managing users in a web app: registering (Create), viewing profiles (Read), editing info (Update), deleting accounts (Delete).

Mongoose validations

Definition: Mongoose allows you to define validation rules in your schema to ensure data is correct before saving.

Example:

```
const productSchema = new mongoose.Schema({
  name: { type: String, required: true },
  price: { type: Number, min: 0 }
});
```

Real-World Example: Validation is like a security guard checking that everyone entering a club is on the guest list and meets the age requirement.

Fun Fact: You can create custom validation functions in Mongoose for advanced checks.

Relationships in MongoDB (References & Population)

Definition: Relationships let you connect documents in different collections, similar to foreign keys in SQL.

Example:

```
const postSchema = new mongoose.Schema({
  title: String,
  author: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }
});
```

Population: Mongoose can automatically replace the referenced ObjectId with the actual document using the `populate` method.

Real-World Example: A blog post (Post) references its author (User). Population is like looking up the author's full profile when viewing the post.

Indexes & performance optimization basics

Definition: Indexes are special data structures that improve the speed of data retrieval operations in MongoDB.

Example:

```
userSchema.index({ email: 1 }); // Create an index on the email field
```

Real-World Example: An index is like the index in a book—it helps you quickly find the page you need without reading the whole book.

Fun Fact: MongoDB automatically creates an index on the `_id` field of every document.

Practice Questions

1. What is MongoDB and how does it differ from SQL databases?
2. How can you install MongoDB locally and what is MongoDB Atlas?
3. Write basic CRUD commands for MongoDB using the shell.
4. What is Mongoose and why is it useful in Node.js projects?
5. How do you connect Mongoose to a MongoDB database?
6. What is the difference between a schema and a model in Mongoose?
7. Show an example of a Mongoose schema with validation.
8. How do you perform CRUD operations using Mongoose?
9. What are references and population in MongoDB/Mongoose?
10. Why are indexes important in MongoDB and how do you create one?

Ready for the next chapter? Real-life Project Building (Hands-On)!