Summary of Section 07

▨ Status	Not started

📝 Summary - Introduction of Introduction

Database

- Database is an organized collection of structured information or data, stored in the computer or on cloud.
- SQL Database - SQL stands for Structured Query Language. SQL database is like well organized filing cabinets. They store data in tables using multiple columns and rows. Example - MySQL, SQLite, Oracle etc.
- NoSQL Database - NoSQL databases store data in more flexible formats like **JSON documents or key-value pairs.** Example - mongoDB, Redis, Cassandra etc.
- We use SQL if we want our data should structured and organized in table, also if we need to run complex queries for analytics or if data consistency is must then we use SQL database.
- We use NoSQL if we want our data is **Unstructured or Semi-Structured,** expect rapid data growth, we need flexibility in our data model or our application is Real-time or big data then we use NoSQL database.

Connecting MongoDB with Node Application

```
mongoose
  .connect("mongodb://localhost:27017/mongo-demo")
  .then(() => console.log("MongoDB connected successfully!!"))
  .catch((err) => console.log("MongoDB connection failed!!", err));
```

Define Schema

- We use schema for defining the basic structure of our collection. This help us to do final data validation, so we can stop useless data to enter in the database.

```
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, unique: true, lowercase: true },
  phone: { type: Number },
  password: { type: String, required: true },
  hobbies: { type: [String] },
  isVerified: { type: Boolean, default: false },
});
```

Create Model by using Schema

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

- This “User” is the singular name of our collection. So this “User” becomes “users” in database.

Create & Save a new data

```
// Create a new data in collection
const newUser = new User({
  name: "Harley",
  email: "harley@gmail.com",
  phone: 2351552,
  password: "harley123",
  hobbies: ["coding", "gyming", "trekking"],
});

// Store that new data in our collection
const storedData = await newUser.save();
```

Query the data

```
// Get all users data
const users = await User.find()

// Get uses who follow this condition
const users = await User.find({ name: "Harley", isVerified: true })

// Get uses who follow this condition but only return these fields which are i
n select method
const users = await User.find({ name: "Harley", isVerified: true }).select("na
me hobbies")

// Get uses who follow this condition but only return these fields except in s
elect method
const users = await User.find({ name: "Harley", isVerified: true }).select("-p
assword -isVerified")

// Get user with this fields but only return first 10 data
const users = await User.find().select("name hobbies").limit(10)

// Get user with this fields but skip first 10 data
const users = await User.find().select("name hobbies").skip(10)

// Get user with this fields but sort them in name decending order
const users = await User.find().select("-password -isVerified").sort({ name:
-1 });
```

Comparison Operators

- Comparison operators are used to compare values in the database with the values we specify in the query.
    - `$eq` - equals to
    - `$ne` - not equals to
    - `$gt` - greater than
    - `$gte` - greater than or equal to

- `$lt` - less than
- `$lte` - less than or equal to
- `$in` - matches any of the values in list like age should be, 18, 22, 25 like that we will pass these values in array
- `$nin` - does not match any values in an array which is opposite of $in

```
// Example of using comparison operators
const users = await User.find({ age: { $gte: 18 } });
const users = await User.find({ age: { $lt: 18 } });
const users = await User.find({ age: { $in: [18, 20, 30] } });
const users = await User.find({ age: { $nin: [18, 20, 30] } });
```

Logical Operators

- Logical operators allow us to combine multiple conditions in our query. They help you to ask more complex questions from the database.
- `$or` : Matches documents where **at least one** condition is true.
- `$and` : Matches documents where **all** conditions are true.
- `$nor` : Matches documents where **none** of the conditions are true.
- `$not` : **Inverts** the result of a condition (true becomes false and vice versa).

```
// Example of using logical operators
const users = await User.find({ $or: [{ age: 30 }, { name: "Harley" }] });
const users = await User.find({ $and: [{ age: 30 }, { name: "Harley" }] });
const users = await User.find({ $nor: [{ age: 30 }, { name: "Harley" }] });
const users = await User.find({ age: { $not: { $eq: 30 } } });
```

Regular Expression

- A regular expression is a way to define search pattern for strings.

```
// Start with H
const users = await User.find({ name: /^H/ })
```

```
// End with u
const users = await User.find({ name: /u$/ })

// End with gmail.com
const users = await User.find({ email: /gmail\.com$/ });

// Contain word harley
const users = await User.find({ email: /harley/ });

// Case insensitive
const users = await User.find({ email: /harley/i });

// Not part of word like "xdeveloper.” only developer.
const users = await User.find({ email: /\bdeveloper\b/i });
```

Count and estimate document count

- These two methods are used to get number of documents.

```
// Get exact number of documents in collection (used for small collections)
const users = await User.countDocuments();

// Get approax number of documents in collection (used for big collections)
const users = await User.estimatedDocumentCount();
```

Pagination & Infinite Query

- When we have large number of data and we want to send them to frontend then we can’t send all data in single request. We have to break them into small groups like pages.

```
const currentPage = 2;
const perPageData = 4;

const users = await User.find()
```

```
  .skip((currentPage - 1) * perPageData)
  .limit(perPageData);
```

Updating data in database

```
// Update one(first) document who follow this condition
const result = await User.updateOne(
  { name: "Code Bless You" },
  { $set: { email: "updated@gmail.com" } }
);

// Update one(first) document who follow this condition but return updated
data
const result = await User.findOneAndUpdate(
  { _id: "67fcde699a9227dd325414fa" },
  { $set: { email: "xyz@gmail.com" } },
  { new: true, runValidators: true }
);

// Update document whose id is this and it return updated data
const result = await User.findByIdAndUpdate(
  "67fcde699a9227dd325414fa",
  { $set: { email: "xyz@gmail.com" } },
  { new: true, runValidators: true }
);

// Update multiple documents who follow this condition
const result = await User.updateMany(
  { age: 20 },
  { $set: { email: "harley@gmail.com" } }
);
```

Deleting data from database

```
// For deleting single document
const result = await User.deleteOne({ _id: "67fcde699a9227dd325414fa"
});

// For deleting single document but it return that deleting document
const result = await User.findOneAndDelete({ _id: "67fcde699a9227dd325
414fa" });

// For deleting document with id and return that deleting document
const result = await User.findByIdAndDelete("67fcde699a9227dd325414f
a");

// For deleting multiple documents which data specify this coditions
const result = await User.deleteMany({ age: { $gt: 15 }, isVerified: false });
```

The ultimate Node JS Course ~ Code Bless You❤️