Clarusway

Backend
Workshop - Teamwork
3

Workshop

Subject: NOSQL & Expressjs

Learning Goals

- Migrating from RDBS to NOSQL databases.
- Ability to perform CRUD operations in NOSQL.
- Increase our general knowledge about backend and Expressjs

Introduction

Express.js and MongoDB are technologies frequently used together to develop web applications. Express.js handles server-side operations while MongoDB database is used to store and manage data. By combining these two technologies, you can create dynamic and data-driven web applications.

The cooperation between the two mainly occurs in the following ways.

- Database Access: Establishes a connection between Express.js and MongoDB for web applications to perform operations such as inserting, updating, querying, and deleting data into MongoDB.

- API Development: Express.js can be used to create APIs to serve MongoDB data. For example, a RESTful API...
- Database Operations: You can use MongoDB for Express.js to make it easier to manage users in your web application to create their accounts, log in, and update their information.

This study aims to reinforce the preliminary knowledge required to perform the above tasks.

Pre-requirements

We will use the VSCode and at the same time, we need to install MongoDB and Mongosh on our computer.

Lets start

1. How many types of SQL languages are there, which one is for executing which type of commands??

Answer:

There are four types of database languages:

- Data Definition Language (DDL) e.g., CREATE, ALTER, DROP, TRUNCATE, RENAME, etc. All these commands are used for create and modify the structure of objects in a database
- Data Manipulation Language (DML) e.g., SELECT, UPDATE, INSERT, DELETE, etc. These commands are used for the manipulation of already updated data that's why they are the part of Data Manipulation Language.
- DATA Control Language (DCL) e.g., GRANT and REVOKE. These commands are used for giving and removing the user access on the database. So, they are the part of Data Control Language.
- Transaction Control Language (TCL) e.g., COMMIT, ROLLBACK, and SAVEPOINT. These are the commands used for managing transactions in the database. TCL is used for managing the changes made by DML.

2. What does NOSQL stand for?

Answer:

For some “NoSQL” stands for “non SQL”, for some “not only SQL.” Either way, most agree that NoSQL databases are databases that store data different format than relational tables.

3. Why Express.js use mongoDB, what canbe reasons?

Answer:

Using Express as a back-end framework is a popular MongoDB stack design. Express is lightweight and approachable for JSON and REST API operations. MongoDB Atlas is a scalable and flexible document database as a service and makes a perfect companion to Express in many stacks like MERN, MEAN, and MEVN.

There are several reasons why MongoDB is commonly used with Node.js:

- JSON-based data model: MongoDB uses a document-based data model that is similar to JSON, which makes it a good fit for use with Node.js, since both use JavaScript Object Notation (JSON).
- Scalability: MongoDB is designed to be scalable, making it well-suited for use with Node.js applications that require high scalability and availability.
- Flexibility: MongoDB allows for flexible data modeling and schema-less design, making it easier to work with evolving data structures in Node.js applications.
- Performance: MongoDB is designed to provide high performance for read and write operations, which can improve the speed of Node.js applications.
- Community support: Both MongoDB and Node.js have large and active communities, which can provide support and resources for developers who are using both technologies.

Overall, the combination of MongoDB and Node.js can provide developers with a powerful and flexible platform for building scalable and high-performance applications

4. How can we represent the data in the following tables in MongoDb?

Table name "post"

post_id	title
1	amazing duo Express.js and MongoDb
2	MVC architecture

Table name "comment"

comment_id	post_id	author	content
1	1	Qadir	I agree with you
2	1	Victor	Their communities are very good
3	2	Rafe	Django use MVT
4	1	Cedric	Good post

Answer:

```
{
    "_id" : NumberLong(1),
    "title" : "amazing duo Express.js and MongoDb",
    "comments" : [
```

```
        {
            "author" : "Qadir",
            "content" : "I agree with you "
        },
        {
            "author" : "Victor",
            "content" : "Their communities are very good"
        }
        {
            "author" : "Cedric",
            "content" : "Good post"
        }
    ]
}
{
    "_id" : NumberLong(2),
    "title" : "MVC architecture",
    "comments" : [
        {
            "author" : "Rafe",
            "content" : "Django use MVT!"
        }
    ]
}
```

5. Sometimes analogies are used to understand the subject. Let's do that too. Can you match the MVC structure with the story below? Which one is the controller, which one is the model, which one is the view?

- Assume that you visit a restaurant. You won't cook food in the kitchen, even though you can do so at home. Instead, you visit the establishment and wait for the waiter to arrive.
- The waiter now approaches you, and you place your dinner order. The server has only written down the specifics of your food order; he is unaware of who you are or what you desire.
- The waiter then makes his way to the kitchen. The server does not prepare your dish in the kitchen.
- Your food is prepared by the cook. Your order and your table number are given to the waiter.
- Cook, then have food prepared for you. The dish is prepared by him using ingredients. Let's say you select the vegetable sandwich option. He then uses the refrigerator to get the bread, tomato, potato, capsicum, onion, bit, and cheese that he needs.
- The chef gives the waiter the last of the meal. Moving this meal outside the kitchen is now the waiter's responsibility.
- The server is now aware of the foods you've ordered and how to serve them.

Answer:
View= You
Waiter= Controller
Cook= Model
Refrigerator= Data

(In the next workshop, maybe we will look for answers to these questions on the Express.Js project. 🗓️)

Teamwork

Subject:

Learning Goals

- Being able to perform operation on MongoDB.

Introduction

We performed operations on relational databases with SQL. Now let's do a study to perform similar operations on MongoDB, a NOSQL database.

Lets start

Write mongoDB statements corresponding to the SQL statements given below.

1.

```
SELECT *
FROM people
```

```
db.people.find()
```

2.

```
SELECT id,
       user_id,
       status
FROM people
```

```
db.people.find(
    { },
    { user_id: 1, status: 1 }
)
```

3.

```
SELECT user_id, status
FROM people
```

```
db.people.find(
    { },
    { user_id: 1, status: 1, _id: 0 }
)
```

4.

```
SELECT *
FROM people
WHERE status = "A"
```

```
db.people.find(
    { status: "A" }
)
```

5.

```
SELECT user_id, status
FROM people
WHERE status = "A"
```

```
db.people.find(
    { status: "A" },
    { user_id: 1, status: 1, _id: 0 }
)
```

6.

```
SELECT *
FROM people
WHERE status != "A"
```

```
db.people.find(
    { status: { $ne: "A" } }
)
```

7.

```
SELECT *
FROM people
WHERE status = "A"
AND age = 50
```

```
db.people.find(
    { status: "A",
      age: 50 }
)
```

8.

```
SELECT *
FROM people
WHERE status = "A"
OR age = 50
```

```
db.people.find(
    { $or: [ { status: "A" } , { age: 50 } ] }
)
```

9.

```
SELECT *
FROM people
WHERE age > 25
```

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

10.

```
SELECT *
FROM people
WHERE age < 25
```

```
db.people.find(
   { age: { $lt: 25 } }
)
```

11.

```
SELECT *
FROM people
WHERE age > 25
AND   age <= 50
```

```
db.people.find(
   { age: { $gt: 25, $lte: 50 } }
)
```

12.

```
SELECT *
FROM people
WHERE user_id like "%bc%"
```

```
db.people.find( { user_id: /bc/ } )
or
db.people.find( { user_id: { $regex: /^bc/ } } )
```

13.

```
SELECT *
FROM people
WHERE status = "A"
ORDER BY user_id ASC
```

```
db.people.find( { status: "A" } ).sort( { user_id: 1 } )
```

14.

```
SELECT *
FROM people
WHERE status = "A"
ORDER BY user_id DESC
```

```
db.people.find( { status: "A" } ).sort( { user_id: -1 } )
```

15.

```
SELECT COUNT(*)
FROM people
```

```
db.people.count()
or
db.people.find().count()
```

16.

```
SELECT COUNT(user_id)
FROM people
```

```
db.people.count( { user_id: { $exists: true } } )
```

17.

```
SELECT COUNT(*)
FROM people
WHERE age > 30
```

```
db.people.count( { age: { $gt: 30 } } )
or
db.people.find( { age: { $gt: 30 } } ).count()
```

18.

```
SELECT DISTINCT(status)
FROM people
```

```
b.people.aggregate( [ { $group : { _id : "$status" } } ] )
or
db.people.distinct( "status" )
```

19.

```
SELECT *
FROM people
LIMIT 1
```

```
db.people.findOne()
or
db.people.find().limit(1)
```

20.

```
SELECT *
FROM people
LIMIT 5
SKIP 10
```

```
db.people.find().limit(5).skip(10)
```

21.

```
CREATE TABLE people (
    id MEDIUMINT NOT NULL
        AUTO_INCREMENT,
    user_id Varchar(30),
    age Number,
    status char(1),
    PRIMARY KEY (id)
)
```

```
db.people.insertOne( {
    user_id: "abc123",
    age: 55,
    status: "A"
 } )
```

22.

```
ALTER TABLE people
ADD join_date DATETIME
```

```
db.people.updateMany(
    { },
    { $set: { join_date: new Date() } }
)
```

23.

```
DROP TABLE people
```

```
db.people.drop()
```

24.

```
INSERT INTO people(user_id,age,status)
VALUES ("bcd001",45,"A")
```

```
db.people.insertOne(
   { user_id: "bcd001", age: 45, status: "A" }
)
```

25.

```
UPDATE people
SET status = "C"
WHERE age > 25
```

```
db.people.updateMany(
   { age: { $gt: 25 } },
   { $set: { status: "C" } }
)
```

26.

```
UPDATE people
SET age = age + 3
WHERE status = "A"
```

```
db.people.updateMany(
   { status: "A" } ,
   { $inc: { age: 3 } }
)
```

27.

```
DELETE FROM people
WHERE status = "D"
```

```
db.people.deleteMany( { status: "D" } )
```

28.

```
DELETE FROM people
```

```
db.people.deleteMany({})
```

☺ **Thanks for Attending** ✍

Clarusway