EXPRESS + MYSQL

Web Programming

TODAY'S TOPICS

- Installation MySQL driver for Node.js
- Establishing connections
- Performing queries
  - Escaping values
  - Preparing queries
  - Getting inserted id
  - Getting number of affected rows vs. changed rows
- Transactions
- Error handling

INSTALLATION

Once you have MySQL up and running on your computer, you can access it by using Node.js.

- To access a MySQL database with Node.js, you need a MySQL driver.
- We are going to use “mysql2” package (https://www.npmjs.com/package/mysql2) which can be downloaded from “npm”
  - Go to your project folder
  - Type command: npm install mysql2

ESTABLISHING CONNECTIONS

Connection options

- **host**: The hostname of the database you are connecting to. (Default: localhost)
- **port**: The port number to connect to. (Default: 3306)
- **user**: The MySQL user to authenticate as.
- **password**: The password of that MySQL user.
- **database**: Name of the database to use for this connection.

```
1   const mysql = require('mysql2/promise');
2
3   const conn = mysql.createConnection({
4     host: 'localhost',
5     user: 'root',
6     password: '...',
7     database: 'webpro',
8   })
```

PERFORMING QUERIES

The most basic way to perform a query is to call the .query() method on a Connection object

```
try{
    const [rows, fields] = await conn.query('SELECT * FROM blogs')
    console.log('rows:', rows)
}catch(err){
    console.log(err)
}
```

ESCAPING QUERY VALUES

Alternatively, you can use ? characters as placeholders for values you would like to have escaped like this:

```
try{
    const [rows, fields] = await conn.query(
        'INSERT INTO `blogs`(title, content, status, pinned) VALUES(?, ?, ?, ?);',
        ['My new blog', 'TEST CONTENT', '01', 0]
    )
    console.log('rows:', rows)
}catch(err){
    console.log(err)
}
```

ESCAPING QUERY VALUES (2)

Multiple placeholders are mapped to values in the same order as passed.

```
try{
    const [rows, fields] = await conn.query(
        'UPDATE `blogs` SET title = ?, content = ? WHERE id = ?;',
        ['My new blog 3', 'TEST CONTENT 2', 13]
    )
    console.log('rows:', rows)
}catch(err){
    console.log(err)
}
```

GETTING THE ID OF AN INSERTED ROW

If you are inserting a row into a table with an auto increment primary key, you can retrieve the insert id like this:

```
try{
    const [rows, fields] = await conn.query(
        'INSERT INTO `blogs`(title, content, status, pinned) VALUES(?, ?, ?, ?);',
        ['My new blog', 'TEST CONTENT', '01', 0]
    )
    console.log('insertId:', rows.insertId)
}catch(err){
    console.log(err)
}
```

GETTING THE NUMBER OF AFFECTED ROWS

You can get the number of affected rows from an insert, update or delete statement.

```
try{
    const [rows, fields] = await conn.query(
        'INSERT INTO `blogs`(title, content, status, pinned) VALUES(?, ?, ?, ?);',
        ['My new blog', 'TEST CONTENT', '01', 0]
    )
    console.log('affectedRows:', rows.affectedRows)
}catch(err){
    console.log(err)
}
```

GETTING THE NUMBER OF CHANGED ROWS

You can get the number of changed rows from an update statement.

- "changedRows" differs from "affectedRows" in that it does not count updated rows whose values were not changed.

```
try{
    const [rows, fields] = await conn.query(
        'UPDATE `blogs` SET title = ?, content = ? WHERE id = ?;',
        ['My new blog 3', 'TEST CONTENT 2', 13]
    )
    console.log('changedRows:', rows.changedRows)
}catch(err){
    console.log(err)
}
```

TRANSACTIONS

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

```
const conn = await pool.getConnection()
// Begin transaction
await conn.beginTransaction();

try{

  let results = await conn.query(
    "INSERT INTO blogs(title, content, status) VALUES(?, ?, ?);",
    [title, content, status]
  )

  const blogId = results[0].insertId;

  results = await conn.query(
    "INSERT INTO images(blog_id, file_path) VALUES(?, ?);",
    [blogId, file.path])

  await conn.commit()
  res.send("success!");
}catch(err){
  await conn.rollback();
  next(err);
}finally{
  console.log('finally')
  conn.release();
}
```

LETS' CREATE A NEW BLOG

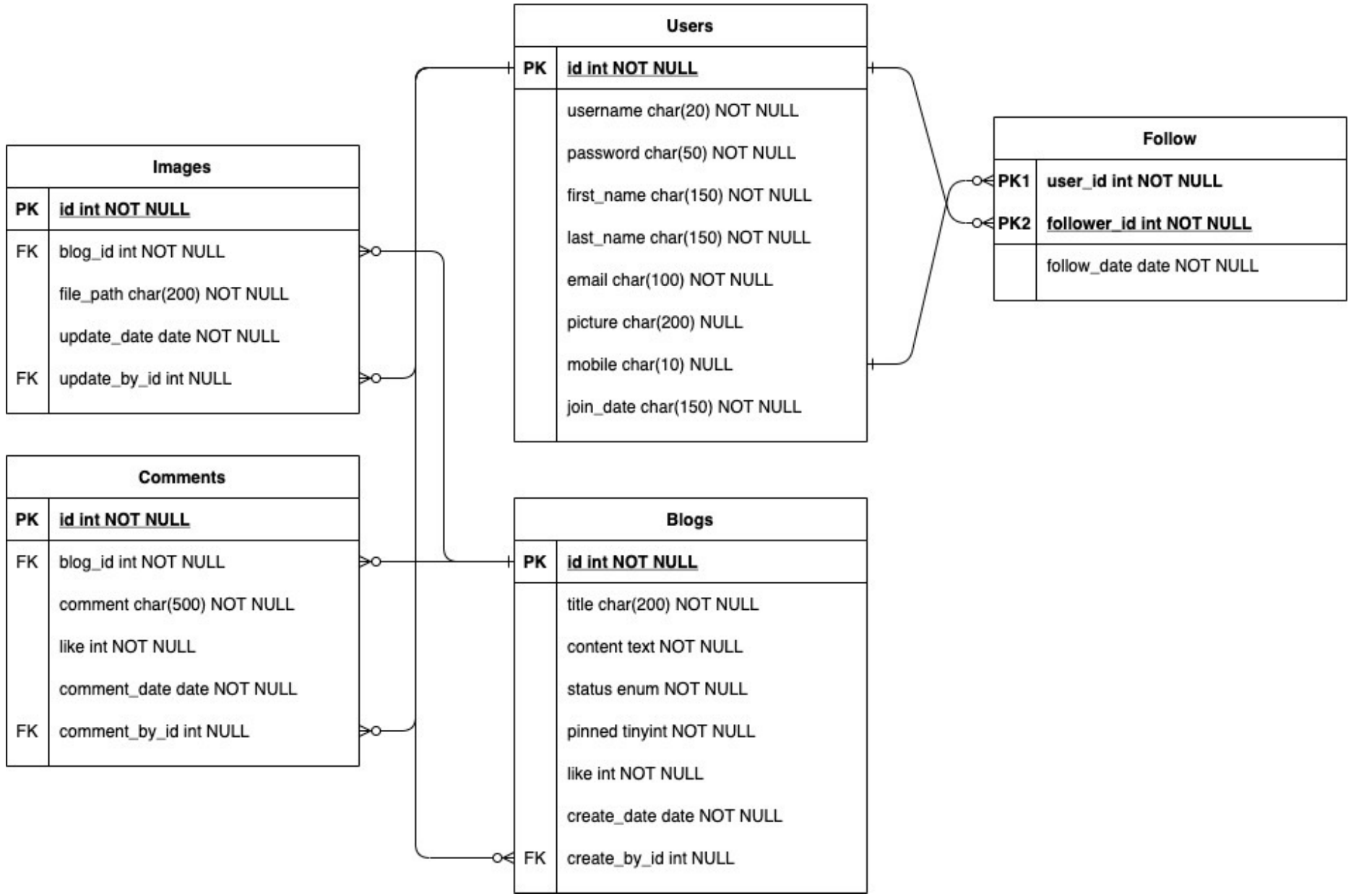
Steps:

1. Install "multer": > npm install multer
2. Create a controller
3. Get data from the POST request
4. Insert into blogs table
5. Insert successfully uploaded file paths in to images table

If something go wrong rollback everything!

YOUBLOG ER DIAGRAM

Let’s migrate the database and do some queries


Users
PK id int NOT NULL
username char(20) NOT NULL
password char(50) NOT NULL
first_name char(150) NOT NULL
last_name char(150) NOT NULL
email char(100) NOT NULL
picture char(200) NULL
mobile char(10) NULL
join_date char(150) NOT NULL
Follow
PK1 user_id int NOT NULL
PK2 follower_id int NOT NULL
follow_date date NOT NULL
Images
PK id int NOT NULL
FK blog_id int NOT NULL
file_path char(200) NOT NULL
update_date date NOT NULL
FK update_by_id int NULL
Comments
PK id int NOT NULL
FK blog_id int NOT NULL
comment char(500) NOT NULL
like int NOT NULL
comment_date date NOT NULL
FK comment_by_id int NULL
Blogs
PK id int NOT NULL
title char(200) NOT NULL
content text NOT NULL
status enum NOT NULL
pinned tinyint NOT NULL
like int NOT NULL
create_date date NOT NULL
FK create_by_id int NULL

INTRO TO RESTFUL API

Web Programming

WHAT IS REST?

RESTful API

GET PUT POST DELETE

REST is acronym for **RE**presentational **S**tate **T**ransfer. It is architectural style for **distributed hypermedia systems** and was first presented by Roy Fielding in 2000 in his famous dissertation.

REST is essentially a guideline!

RESOURCE

The key abstraction of information in REST is a **resource**.

- Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person), and so on.
- REST uses a **resource identifier** to identify the particular resource involved in an interaction between components.
- The data format of a representation is known as a media type. The media type identifies a specification that defines how a representation is to be processed.

RESOURCE METHODS

Another important thing associated with REST is **resource methods** to be used to perform the desired transition.

- **GET – retrieving resource**
- **PUT/PATCH – updating**
- **POST – creating**
- **DELETE - deleting**

CRUD

LET'S DESIGN OUR BLOG APP RESTFUL

- GET "/blogs"
- POST "/blogs"
- PUT "/blogs/:id"
- GET "/blogs/:id"
- DELETE "/blogs/:id"

- GET "/comments/:blog_id"
- POST "/comments/:blog_id"
- PUT "/comments/:blog_id/:id"
- GET "/comments/:blog_id/:id"
- DELETE "/comments/:blog_id/:id"