KWEB Study Week6:
SQL with Node.js

KWEB 2학기 준회원 스터디

Today's Contents

1. Before Study
2. MySQL (MariaDB)
3. SQL with Express
4. Another Example
5. 게시판 Intro
6. 과제

M.G. BAE

J.H. BAEK

2017 Fall Semester

Before Study

- 지난 주차에 우리는 Express의 프로젝트 생성을 도와주는 express-generator 모듈을 사용해보고 모듈이 생성해주는 파일들을 분석해 보았습니다.
- 또한 이전에 다루어 보았던 EJS 템플릿 엔진, Mongo DB에 대한 내용을 되짚어 보고 RDBMS에서 사용되는 SQL에 대해서 간략하게 알아보았습니다.
- 7주차의 목표는 실제 RDBMS인 MySQL (MariaDB)를 설치해 실제 SQL을 다루어 보고 우리의 게시판 웹페이지에 연동 시켜 보는 것입니다.
- 오늘도 지난 스터디 한 번 되짚어보고 시작할게요!

Previous Study - Express Generator

- Express 프로젝트의 규모가 커지게 되면 개발자는 프로젝트 효율적 구조에 대한 고민을 하게 됩니다.
- Express Generator는 이러한 고민에 대한 답으로, 미들웨어 설정, 라우터 소스 파일의 위치, 서버의 실제 실행 등이 고려된 프로젝트 구조를 만들어 줍니다.
- 많은 오픈소스 프레임 워크들은 이런 최선의 구조를 가지고 있고 이러한 내용을 묶어 놓은 것을 Boiler Plate 라고 합니다. 즉, Express Generator는 Express 웹 프레임워크의 Boiler Plate라고 할 수 있습니다.
- Express Generator는 **npm install -g express-generator** 명령어로 사용가능합니다.

Express Generator 참고

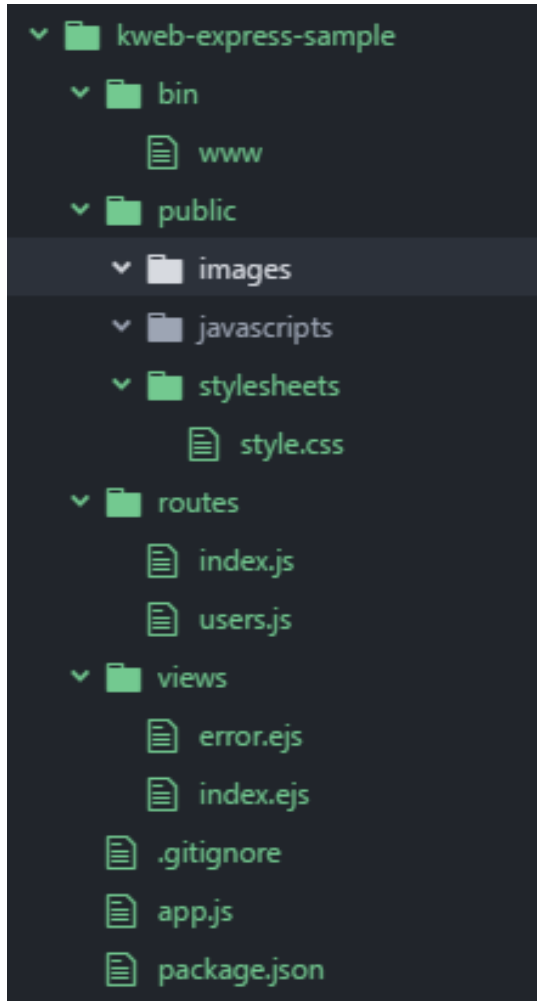
```
$ express  --help

        Usage: express [options] [dir]

        Options:

        --version        express-generator의 버전을 출력합니다.
    -e, --ejs            EJS 엔진을 사용합니다.
        --pug            pug (구 jade) 엔진을 사용합니다.
        --hbs            handlebars 엔진을 사용합니다.
    -H, --hogan          hogan.js 엔진을 사용합니다.
    -v, --view <engine>  뷰 엔진의 사용 여부를 설정합니다. (dust|ejs|hbs|hjs|jade|pug|twig|vash)
    -c, --css <engine>   css 엔진의 사용 여부를 설정합니다. (less|stylus|compass|sass)
        --git            git 과의 연동을 위한 .gitignore 파일을 추가 합니다.
    -f, --force          비어있지 않은 디렉토리에 덮어 씁니다..
    -h, --help           이 도움말을 출력합니다.
```

Previous Study - Express Generator 구조



- /bin/www → Port와 몇가지 로깅이 추가된 구동 파일 입니다. ‘npm start’ 나 ‘npm run start’ 명령어로 실행합니다.
- /public → CSS, JS, Image 같은 정적 파일이 보관됩니다. app.js 에서 static 미들웨어를 통해 사용자에게 제공됩니다.
- /route → URL 라우팅에 관련된 소스가 보관 됩니다. 설정된 express router를 export 하며 app.js 에서 미들웨어로써 사용합니다.
- /views → 템플릿 파일이 보관됩니다. res.render 를 통해서 사용됩니다.
- app.js → 메인 설정 파일입니다. 프로젝트의 모듈들을 로딩하고 조합하는 역할을 합니다.
- package.json → 프로젝트의 의존성이 기록된 파일 입니다. 각종 미들웨어와 템플릿 엔진의 설치 정보가 기록 됩니다.

Previous Study - EJS

- express에서 기본으로 사용되는 템플릿 엔진 입니다.

- 템플릿 엔진은 문법에 맞추어 작성된 템플릿과 데이터를 결합하여 새로운 텍스트 데이터를 생성합니다. Express에서는 EJS를 이용하여 html 스타일의 데이터를 생성합니다.

- Example
  - 간단한 출력 : <%= %> → ex) <h1>this is HTML</h1> => <h1>this is HTML</h1>
  - 그대로 출력 : <%- %> → ex) <h1>this is HTML</h1> => <h1>this is HTML</h1>
  - 반복, 조건 : <% {statement} %> → ex) <% if (true) { %> Condition is TRUE !! <% } %>

Previous Study - MongoDB

- 대표적인 NoSql 데이터 베이스
- JSON 형태로 문서를 저장
- NodeJS에서는 Mongo 패키지를 사용하여 접근

```
const book = new Book('제 마음도 괜찮아질까요?', '강현식 ,서늘한
const books = [
  new Book('집 살래 월세 살래', '이재범(핑크팬더)', 15, 17000,
  new Book('잘 자, 굴삭기 벤!', '되르테 흔', 10, 12000, '116051
];

book_collection.insertOne(book);
book_collection.insert(books);
```

```
book_collection.deleteMany({
  ISBN: '1187383279'
}, () => {
});
```

```
book_collection.find({
  ISBN: '116051111X'
}).toArray((err, data) => {
  console.log(data);
});
```

```
book_collection.updateMany({
  ISBN: '116051111X'
}, {
  $set: {
    price: 1000000000,
    remaining: 9999999999
  }
}, () => {
})
```

MySQL (MariaDB)

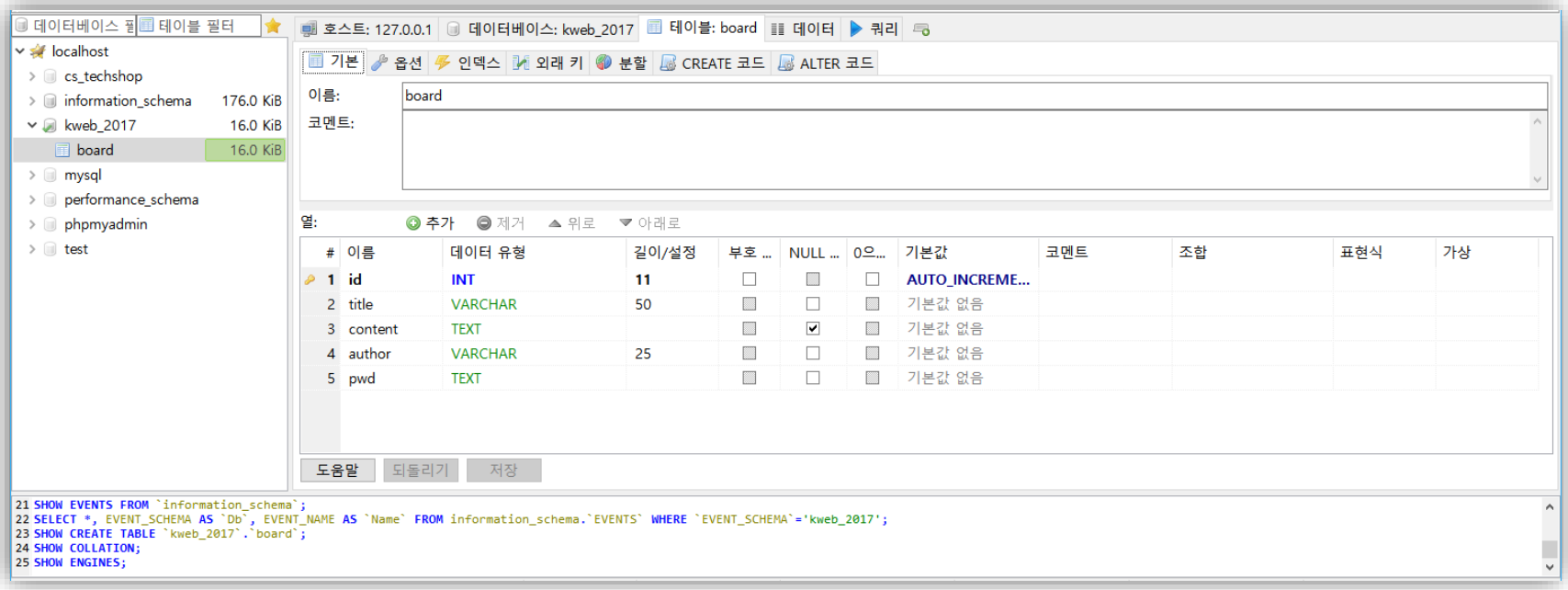
- MySQL은 세계에서 가장 많이 쓰이는 오픈 소스의 RDBMS 입니다.
  - MariaDB는 오라클 소유의 현재 불확실한 MySQL의 라이선스 상태에 반발하여 만들어졌으며, 우리가 쓰는 수준에선 사실상 차이가 없습니다.
- 실제로도 실무에 가장 많이 사용하는 DB 입니다.
- MySQL (MariaDB)은 SQL문을 사용하며, 사용하기 쉽고 쓰기 빠릅니다.
- 우리는 MySQL (MariaDB)를 앞으로의 실습에서 DB로써 사용할 것입니다.

RDB 설치

- MySQL이나 MariaDB나 원하시는 거 설치하시면 됩니다.
- Google에 검색해서 설치해주세요!
- (잘 안되면 앞에 스터디장 괴롭히시면 됩니당)

HeidiSQL 설치

- SQL을 사용하는 DB들을 쉽게 관리하기 위해 사용하는 프로그램입니다.
  - 설치 URL: https://www.heidisql.com/download.php



- (아쉽게도 윈도우만 지원해서 Mac 유저분들은 앞의 스터디장에게 문의하세요.)

Basic SQL - DATABASE

- Column 값들의 집합은 Row(data set) / Row의 집합은 Table / Table의 집합은 Database

- 즉, MySQL에서의 Database는 테이블의 집합을 뜻합니다. 주로 한 어플리케이션은 하나의 데이터베이스에 대한 접근 권한을 가지게 됩니다.

- 데이터 베이스의 생성 → CREATE DATABASE [데이터 베이스명];
  - Ex) CREATE DATABASE KWEB_BOARD;

- 데이터 베이스 사용 → USE [데이터 베이스명];
  - Ex) USE KWEB_BOARD;

Basic SQL - GRANT

- 보통 하나의 어플리케이션이 하나의 DB에 접근할 권한을 가진다고 했습니다. 따라서 root 권한으로 DB를 생성하고 나면 알맞은 username과 credential에 권한을 부여 해주어야 합니다.

- Case 1.특정 유저에게 특정 데이터 베이스의 모든 권한을 부여
  - GRANT ALL PRIVILEGES ON [데이터베이스].* TO '[유저이름]'@'[호스트명]' IDENTIFIED BY '[비밀번호]';

- Case 2. kweb 유저가 localhost에서 비밀번호 'kweb_pwd' 로 접근시에 kweb_board 데이터 베이스의 모든 권한을 허용
  - GRANT ALL PRIVILEGES ON kweb_board.* TO 'kweb'@'localhost' identified by 'kweb_pwd';

- Case 3. kweb 유저가 어디서든지 에서 비밀번호 'kweb_pwd_super' 로 접근시에 모든 데이터 베이스의 모든 권한을 허용
  - GRANT ALL PRIVILEGES ON *.* TO 'kweb'@'%' identified by 'kweb_pwd_super';

Basic SQL - table column types

- Table은 Column이 모인 Row의 집합이라고 하였습니다.
- RDBMS는 Column의 type에 대해서 strict 하고 때문에 테이블은 각 Column에 대한 Type의 정의를 가지고 있어야 합니다.
- 이러한 정의는 처음 테이블을 생성할때 명시하게 되며 다음 슬라이드의 표와 같은 타입들을 가집니다.

Basic SQL - table column types

- 붉게 표시된 타입들이 주로 사용되는 타입들이며 VARCHAR 타입은 반드시 길이를 명시해야 합니다.

문자	숫자	시간	데이터
CHAR (n < 256)	TINYINT (n < 8)	DATE	BINARY
VARCHAR (n < 66535)	SMALLINT (n < 16)	TIME	BYTE
TINYTEXT (n < 256)	MEDIUMINT (n < 24)	DATETIME	TINYBLOB
TEXT (n < 66536)	INT (n < 32, default 11)	TIMESTAMP	BLOB
MEDIUMTEXT (n < 16777216)	BIGINT	YEAR	MEDIUMBLOB
LONGTEXT (n < 4294967296)	FLOAT		LONGBLOB
	DECIMAL		
	DOUBLE		

Basic SQL - CREATE TABLE

- 테이블을 생성할때는 아래의 명령어를 사용해 생성합니다.
  - CREATE TABLE [테이블 명] ([[컬럼 정의],], [[키 정의],]) [[기타 테이블 속성]];

```
CREATE TABLE kweb_board (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    subject VARCHAR(255),
    content TEXT
) CHARSET UTF8 COLLATE utf8_general_ci;

CREATE TABLE kweb_board (
    id INT NOT NULL AUTO_INCREMENT,
    subject VARCHAR(255),
    content TEXT,
    PRIMARY KEY (id)
) CHARSET UTF8 COLLATE utf8_general_ci;
```

Basic SQL - DROP TABLE

- 테이블을 지울 때에는 DROP TABLE [테이블명];
- 단순히 데이터만을 지울 때에는 TRUNCATE [테이블명];
- 주의! DELETE FROM [테이블명]; 을 실행 시에는 데이터 셋은 모두 지워지지만 auto_increment 데이터는 사라지지 않습니다. 이를 지우기 위해서는 TRUNCATE 키워드를 사용해야 합니다. (뒤에 DELETE 문에서 떠올려 봅시다.)

Basic SQL - INSERT INTO

- CRUD에서의 Create를 담당하는 쿼리 입니다.
  - 형식: INSERT INTO [테이블명] ([컬럼명1],[컬럼명2],…) VALUES ([값1],[값2],…);

- 앞서 만든 kweb_board 테이블에 데이터를 넣는 SQL문 예시입니다.

```
INSERT INTO kweb_board VALUES
  (1, 'Subject 1', 'Content 1'),
  (2, 'Subject 2', 'Content 2');

INSERT INTO kweb_board(subject, content)
VALUES
  ('Subject 1', 'Content 1'),
  ('Subject 2', 'Content 2');
```

Basic SQL - SELECT

- CRUD에서의 Read를 담당하는 쿼리 입니다.
  - 형식: SELECT (* | [[컬럼명1], [컬럼명2],…]) FROM [테이블명] (WHERE [조건]);

- Case 1. 앞서 만든 kweb_board 에서 id가 1인 데이터의 제목 컬럼만을 원한다면

```
SELECT subject FROM kweb_board WHERE id=1;
```

- Case 2. 앞서 만든 kweb_board 에서 내용에 2를 포함한 데이터의 모든것을 원한다면

```
SELECT * FROM kweb_board WHERE content LIKE '%2%';
```

Basic SQL - UPDATE

- CRUD에서의 Update를 담당하는 쿼리 입니다.
  - 형식: UPDATE FROM [테이블명] SET [[컬럼1]=[새로운 데이터1],…] (WHERE [조건]);

- 앞서 만든 kweb_board 에서 id가 1인 데이터의 제목을 Subject 3로 수정하는 SQL문 예시입니다.

```
UPDATE FROM kweb_board SET subject='Subject 3' WHERE id=1;
```

Basic SQL - DELETE

- CRUD에서의 Delete를 담당하는 쿼리 입니다.
  - 형식: DELETE FROM [테이블명] (WHERE [조건]);

- 앞서 만든 kweb_board 에서 id가 1인 데이터를 지우는 SQL문 예시입니다.

```
DELETE FROM kweb_board WHERE id=1;
```

SQL with Express

- 지금까지 멘붕하느라 수고하셨고 실습으로 정리해봅시다.
- 지난 주 과제 파일을 수정해 우리의 DB와 연동시켜봅시다.
- 알맞게 routing 해오셨을거라 믿어 의심치 않습니다. 크게 4가지만 있으면 됩니다.
  - 게시물 List가 보이는 Something
  - 게시물 안의 내용을 확인할 수 있는 Something
  - 게시물을 수정할 수 있는 Something
  - 게시물을 삭제할 수 있는 Something

실습 전! 짚고가기

```
router.get('/', function(req, res, next) {
    res.render('board', { rows: rows });
});

router.get('/view', function(req, res, next) {
    res.render('board', { rows: rows });
})

router.get('/write', function(req, res, next) {
    res.render('board', { rows: rows });
});

router.get('/update', function(req, res, next) {
    res.render('board', { rows: rows });
});

router.post('/write', function(req, res, next) {
    //Implement
});

router.post('/update', function(req, res, next) {
    //Implement
});

router.post('/delete', function(req, res, next) {
    //Implement
});

module.exports = router;
```

- 이번 실습에서 쓸 기본 Routing 틀입니다. 본인 과제에 맞게 변형하셔서 알아들으시면 됩니다.
  - GET / → 게시물 List 출력하는 페이지 렌더링
  - GET /view → 게시물 내용 출력하는 페이지 렌더링
  - GET /write → 새로운 게시물 올리는 페이지 렌더링
  - GET /update → 기존의 게시물 수정하는 페이지 렌더링
  - POST /write → 새로운 게시물의 내용을 받아 DB에 추가
  - POST /update → 기존 게시물의 내용을 수정
  - POST /delete → 기존 게시물 삭제

MySQL (MariaDB) with Express

- 우리는 콘솔에서 RDBMS에 접근해 보았습니다.
- But! Node.js에서 programmatic 하게 RDBMS에 접근하기 위해서는 다른 방법이 필요합니다.
- 여러 언어에서 이러한 범주의 방법을 connector 라고 하며 node에서는 mysql:3306 프로토콜을 사용하는 RDBMS Connector 로써 mysql 모듈을 제공합니다.
- mysql 모듈은 npm install mysql -save 를 통해서 사용할 수 있습니다. (MariaDB 쓰셔도 똑같이 하시면 됩니다 사실)

MySQL with Node.js (createConnection)

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

// 미리 접속정보를 설정합니다.
const conn = mysql.createConnection({
  host: '127.0.0.1',
  user: 'root',
  password: 'tyekdns@2',
  // database: 'test'
});
// 실제로 설정된 DB에 접근합니다. 성공/실패 여부를 콜백에 전달합니다.
// 콜벡이 싱행되기 전까지 연결 여부는 Pending이기 때문에 쿼링을 하면 애러가 발생합니다.
// 콜벡에서 err이 날아온다면 적절한 애러처리를, 제대로된 연결이 성립되었다면 후속 작업을 진행하면 됩니다.
conn.connect((err) => {
  if (err) {
    console.log(err.sqlMessage);
  }

  UseConnection();
});
```

createConnection vs. createPool

- MySQL DB와의 연결을 성립하는 방법은 ① createConnection을 이용하여 단일 커넥션을 생성하는 것과 ② createPool을 이용하여 커넥션 풀을 만드는 2가지가 있습니다.

- ①번 방식은 DB와의 연결을 만들고 종료하면서 사용하고 ②번 방식은 DB와의 연결을 Connection Pool에 저장시키고 필요할 때 가져다 쓰고 Pool에 반환한다.

- Connectgion Pool이란?
  - 데이터베이스와 연결된 커넥션을 미리 만들어서 풀(pool) 속에 저장해 두고 있다가 필요할 때 커넥션을 풀에서 쓰고 다시 풀에 반환하는 기법
  - 연결을 여러개 만들어 놓고 사용하기 때문에 많은 사용자가 접속하는 어플리케이션(웹 또는 기타)일 때 단일 연결일때 보다 속도 향상을 가져온다고 합니다.
  - 이번 실습 땐 ②번 방식을 사용해봅시다.

DB with Express - /db/*.*

- Express Generator가 생성한 프로젝트에 db라는 디렉토리를 추가해봅시다.
- 디렉토리 db에는 db_info.js와 db_con.js 2개의 파일이 들어있습니다.
- db_info.js는 우리가 연결할 db의 정보가 들어있습니다. Host 위치, 계정 ID, PW 사용할 datebase 등 여러 정보를 담고 있습니다.
- db_con.js는 우리가 실제 db를 connect할 정보를 담고 있습니다.

DB with Express - /db/db_info.js

```
module.exports = (function () {
    return {
        local: {
            connectionLimit : 10,
            host     : 'localhost',
            user     : 'root',
            password : '',
            database : 'kweb_2017',
            debug    :  false,
            multipleStatements : true
        }
    }
})();
```

- local에 대한 정보를 담고 있습니다.
  - connectionLimit: 한번에 연결될 수 있는 connection 개수 제한 (연결 품질을 위해!)
  - host: host 주소
  - user: user ID
  - password: user PW
  - database: 사용할 DB
  - 나머진 그냥 적으세요.
- 여러분들의 DB info에 맞게 쓰시면 됩니다.

DB with Express - /db/db_con.js

```
var mysql = require('mysql');
var config = require('../db/db_info').local;

module.exports = function () {
    return {
        init: function () {
            return mysql.createPool({
                connectionLimit : config.connectionLimit,
                host     : config.host,
                user     : config.user,
                password : config.password,
                database : config.database,
                debug    :  config.debug,
                multipleStatements : config.multipleStatements
            })
        },

        test_open: function (con) {
            con.connect(function (err) {
                if (err) {
                    console.error('mysql connection error :' + err);
                } else {
                    console.info('mysql is connected successfully.');
                }
            })
        }
    }
};
```

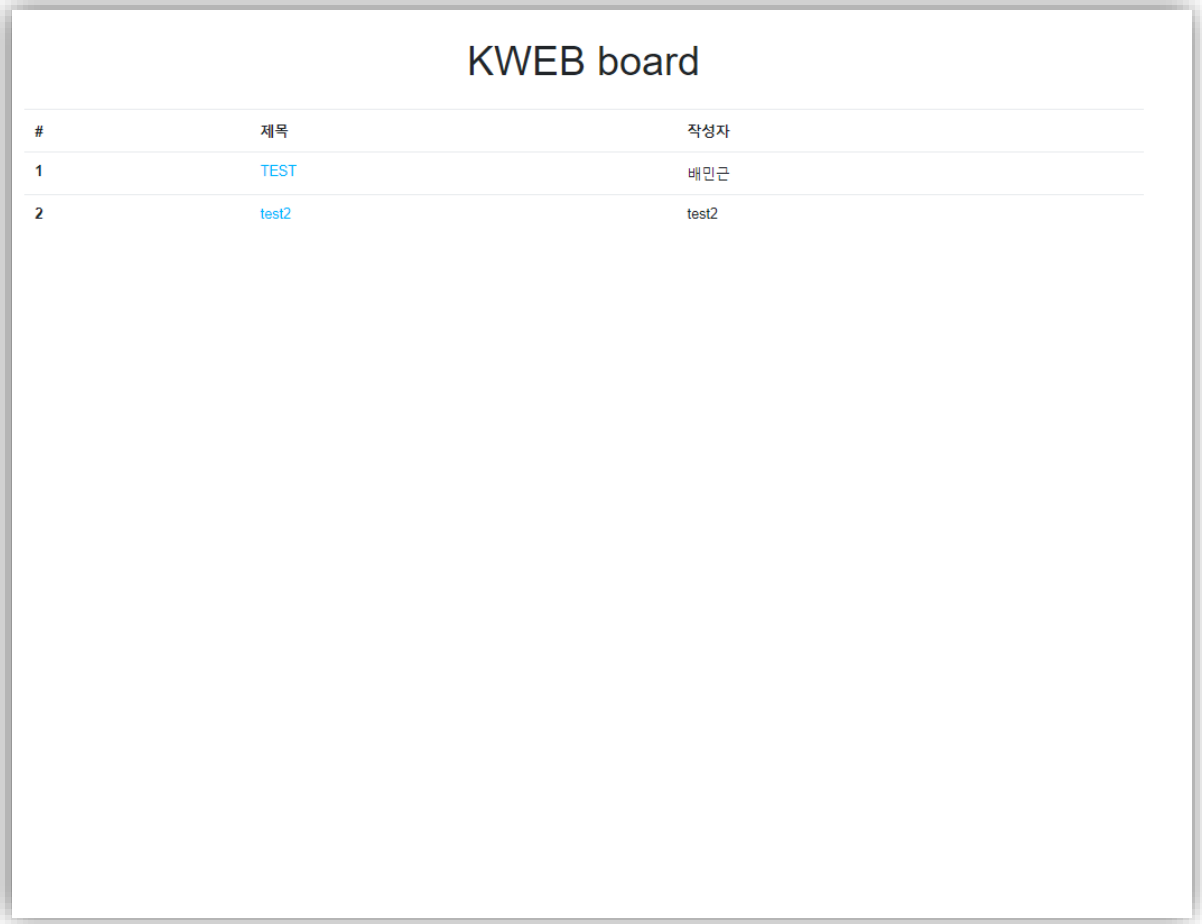
- 이 파일로 우리가 DB와의 연결을 위해 사용할 connection Pool을 만듭니다.
- 이 파일의 설정을 가져와서 router 파일들에서 사용할 것입니다.

MySQL with Node.js - query 사용

- conn.query(query, handler); / conn.query(query, data, handler);
- query 부분에는 문자열 쿼리가 들어가며 optional한 인자인 data에는 배열로 query 문자열을 채우기 위한 데이터가 전달됩니다. (data가 없다면 앞처럼 비겠죠?)
- query가 var sql = “SELECT * FROM board WHERE id = ?” 라면?
  - conn.query(sql,[1],handler);로 쓰면 “SELECT * FROM board WHERE id = 1”이 실행됩니다.
- mysql 패키지의 쿼리 포맷팅은 아래의 URL을 참고하시면 됩니다.
  - https://github.com/mysqljs/mysql#escaping-query-values

실습 1 - 게시판 표시

- GET / 안의 내용을 수정하여 오른쪽과 같이 게시물을 표시해봅시다.
- SELECT 문을 적극 활용하여 봅시다.
- 오른쪽은 Bootstrap 을 이용하여 화면을 렌더링했습니다.



실습 1 - 게시판 표시

- GET / 에 해당하는 router를 다음과 같이 수정해봅시다.
- SQL을 실행하여 DB로부터 게시물 데이터들을 가져와봅시다.
- 그리고! ejs로 뿌려봅시다! 앞 슬라이드의 표처럼 렌더링해봅시다.

```
router.get('/', function(req, res, next) {
    pool.getConnection(function (err,conn) {
        if(err) {
            if(conn) {
                conn.release();
            }
            callback(err,null);
            return;
        }
        var sql = "SELECT * FROM board";
        var exec = conn.query(sql,[] ,function(err, rows) {
            conn.release();

            if (err) throw err;
            res.render('board', { rows: rows });
        });
    });
});
```

실습 1 - 게시물 표시

- 이번에는 GET /view 에 해당하는 router를 수정할 것입니다.
- 게시판 표시와의 차이는 우리가 선택한 1개의 게시물만 가져오는 것입니다.
- 왼쪽과 같이 /view/:id 로 쓰시면 /view/[무언가] 에 관한 routing입니다.
  - req.params.id로 URL을 가져올 수 있습니다.

```
router.get('/view/:id', function(req, res, next) {
    pool.getConnection(function (err,conn) {
        if(err) {
            if(conn) {
                conn.release();
            }
            callback(err,null);
            return;
        }
        var id = req.params.id;
        var sql = "SELECT * FROM board WHERE id = ?";
        var exec = conn.query(sql,[id] ,function(err, row)
            conn.release();

            if (err) throw err;
            res.render('view', { row: row[0] });
        });
    });
});
```

실습 1 - 게시물 업로드

- 이번엔 게시물을 업로드를 구현해봅시다. 업로드인만큼 업로드 페이지 + 업로드하는 Logic 모두 구현해야 합니다.
- GET /write와 POST /write를 구현하시면 됩니다.
- GET /write는 오른쪽과 같이 그냥 write하는 페이지만 렌더링 하시면 됩니다.
  - 페이지에서 입력은 Form태그 쓰셔야 합니다!

```
<!DOCTYPE html>
<html>
<head>
    <title>KWEB board</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css" inte
    <link rel='stylesheet' href='/stylesheets/style.css' />
</head>
<body>
<div class="container">
    <form method="post" action="/board/write">
        <div class="form-group">
            <label for="title">제목</label>
            <input type="text" class="form-control" id="title" name="title" placeholder="제목을 입력하세요.">
        </div>
        <div class="form-group">
            <label for="content">내용</label><br>
            <textarea class="form-control" id="content" name="content" placeholder="내용을 입력하세요."></textarea>
        </div>
        <div class="form-group">
            <label for="author">작성자</label>
            <input class="form-control" id="author" name="author" placeholder="작성자를 입력하세요.">
        </div>
        <div class="form-group">
            <label for="pwd">게시물 확인 비밀번호</label>
            <input class="form-control" id="pwd" name="pwd" placeholder="비밀번호를 입력하세요.">
        </div>
        <button type="submit" class="btn btn-primary">업로드</button>
    </form>
</div>
<br>
<br>
<div class="container">
    <a href="/board"><button type="button" class="btn btn-primary">뒤로</button></a>
</div>
</body>
</html>
```

실습 1 - 게시물 업로드

- POST /write 를 구현해봅시다.
- form 태그에서 넘겨받은 내용들은 req.body.[무언가] 로 받습니다.
- SQL 문을 이용하여 DB에 넣어봅시다.

```
router.post('/write', function(req, res, next) {
    pool.getConnection(function (err,conn) {
        if(err) {
            if(conn) {
                conn.release();
            }
            callback(err,null);
            return;
        }
        var title = req.body.title;
        var content = req.body.content;
        var author = req.body.author;
        var pwd = req.body.pwd;

        var sql = "INSERT INTO board (title,content,author,pwd) VALUES (?,?,?,?)";
        var exec = conn.query(sql,[title,content,author,pwd] ,function(err, rows) {
            conn.release();

            if (err) throw err;
            res.redirect("/board");
        });
    });
});
```

실습 1 - Review

- CRUD 중 Create와 Read에 대해 구현해보았습니다.
- 난이도는 적당했나요..? ㅎㅎㅎㅎㅎ
- (Update와 Delete는 숙제입니다 ㅎ)
- 예제 코드는 아래 URL 참고하세요!
  - https://github.com/baemingun/kweb_week7

Anti-Pattern

- 지금까지는 Express Generator가 생성한 패턴에 따라 Coding을 진행했습니다.
- 근데 사실 좋은 Pattern이긴 하지만 개발 과정에 따라 프로젝트 구조가 적합하지 않을 수도 있습니다.
- 한 번 Express Generator가 생성한 패턴이 아닌 다른 구조로 프로젝트를 작성해봅시다.

DAO

- DAO는 Database Access Object의 약자입니다.
- 응용프로그램 내부에서 DB에 접근해야 하는 부분은 예측할 수 없습니다.
- 만약 DB에 접근해야 할때마다 새로운 연결을 수립하고 CRUD를 수행한다면 새로운 연결을 수립하면서 오버헤드가 발생하거나 DB 구조의 변경이 있을 때마다 영향을 받는 코드를 모두 수정해 주어야 할것 입니다.
- 따라서 DB에 접근하는 코드는 역할별로 분류하여 한곳에 모아놓고 사용하는 것이 바람직합니다. 이 역할을 담당하는 객체를 데이터 베이스 접근객체 aka DAO라고 합니다.

DTO

- DTO는 Data Transfer Object의 약자로 DAO화 다른 서비스 객체간의 데이터 전송을 위해 사용되는 객체입니다.

- 간단하게는 key, value형태의 Object에서 약간의 로직이 추가된 Javascript Class가 이 역할을 수행합니다.

실습 2 - BoardService

```
> const BoardService = require('./services/board.service');
BoardService init OK
undefined
> BoardService
{ createArticle: [Function: createArticle],
  updateArticle: [Function: updateArticle],
  findAll: [Function: findAll],
  findById: [Function: findById],
  deleteById: [Function: deleteById] }
```

```
> BoardService.findAll()
[ { id: 1, subject: 'dummy 001', content: 'dummy content 001' },
  { id: 2, subject: 'dummy 002', content: 'dummy content 002' } ]
```

```
> BoardService.createArticle('TEST', 'TEST')
{ id: 3, subject: 'TEST', content: 'TEST' }
```

```
BoardService.deleteById(3)
ndefined
BoardService.findAll()
{ id: 1, subject: 'dummy 001', content: 'dummy content 001' },
{ id: 2, subject: 'dummy 002', content: 'dummy content 002' } ]
```

```
> BoardService.updateArticle(3, 'TEST MODI', 'TEST MODI CON')
undefined
> BoardService.findAll()
[ { id: 1, subject: 'dummy 001', content: 'dummy content 001' },
  { id: 2, subject: 'dummy 002', content: 'dummy content 002' },
  { id: 3, subject: 'TEST MODI', content: 'TEST MODI CON' } ]
>
```

실습 2 - BoardService

- 보드 서비스는 5개의 함수를 가진 객체 입니다. 각각의 함수는 직접적인 DB연결 없이 CRUD를 구현하고 있습니다.
  - findAll : 모든 글을 반환
  - findById: id가 같은 글을 반환, 없으면 null
  - createArticle : subject와 content를 받아 글을 생성
  - updateArticle: id, subject, content를 받아 업데이트 수행, 없는 id 전달시 애러 발생
  - deleteArticle: id를 받아 삭제 수행, 없는 id 전달시 애러 발생

- 오늘 우리는 findAll, findById, createArticle 함수를 mysql과 연동해 볼것 입니다.

실습 2 - BoardService

- 실습 프로젝트 다운로드
  - git clone https://github.com/bjh970913/KWEB-BOARD-DAO.git

- 위 프로젝트에는 mysql과의 연동에 맞추어 포팅된 KWEB-Board-dao가 있습니다.
  - npm install -g nodemon 로 실행합니다.

- Nodemon test.js 를 실행하면 DAO의 기능을 테스트 하는 코드가 실행 됩니다.

- 여러분은 앞으로의 실습을 통해서 mysql 을 통해서 DB 연결, 테이블 생성, 테이블 초기화, 데이터 입력, 데이터 조회, 데이터 삭제를 가능케 하는 코드를 작성하게 될것입니다.

실습 2 - BoardService

- 다음은 Github에 기록된 board service가 CRUD의 CRU 까지의 목표를 달성하게 하는 기록입니다. 직접 따라해보고 모르겠는 부분은 스터디장에게 질문해 주세요

- Init 함수에서 Mysql 연결을 생성하도록 변경
  - https://github.com/bjh970913/KWEB-BOARD-DAO/commit/1c8cc5f138df156b9f00850e264beef99df280de

- Init 함수에서 테이블을 만들도록 변경
  - https://github.com/bjh970913/KWEB-BOARD-DAO/commit/dfdb1d9c71e25e66cc00bf42103bbd873de06f2e

- 테이블 초기화 함수를 수정
  - https://github.com/bjh970913/KWEB-BOARD-DAO/commit/e53541e4b9f62a525616a24d55f1be597c5e5adb

실습 2 - BoardService

- Article DTO와 전체 조회 함수를 수정
  - https://github.com/bjh970913/KWEB-BOARD-DAO/commit/7198c02ec3ccc679ecb93de37997ff892e048a81

- id로 조회 함수와 생성 함수를 수정
  - https://github.com/bjh970913/KWEB-BOARD-DAO/commit/f3e55abfdc21c787d761d293feda86dc3226e046

- 글 수정 함수를 수정
  - https://github.com/bjh970913/KWEB-BOARD-DAO/commit/55a3232fecb46b22477373f5447cf7fbfb2e32da

과제 (~ 12/8 23:59)

- 7주차는 Node.js와 Express.js 에 MySQL (MariaDB)를 연동해보았습니다.
- 이번주 과제는 실습 1의 Update, Delete 기능을 완성하는 것입니다.
- 과제는 지난 주와 같이 Github에 올려 링크로 카톡으로 제출해주세요~
- 제출기한: 12월 8일 오후 11시 59분 59초까지
- 과제 제출 E-mail
  - 월 7시: 15 배민근 (baemingun@naver.com)
  - 화 7시: 16 백지훈 (bjh970913@gmail.com)

It’s all today!