

Hello,

KDT 웹 개발자 양성 프로젝트

5기!

with





Module



CommonJS

방식

CommonJS 방식



- Node.js 에서 사용되는 모듈 방식 입니다!
- 키워드로는 `require` / `exports` 를 사용합니다.
- `Require` 로 모듈을 불러 올 때, 코드 어느 곳에서도 불러 올 수 있습니다!

CommonJS 방식



- 전체 모듈로써 내보내고, 전체를 하나의 Obj 로 받아서 사용하는 방법

```
const animals = ['dog', 'cat'];

function showAnimals() {
  animals.map((el) => console.log(el));
}

module.exports = {
  animals,
  showAnimals,
};
```

Animal.js

```
const animals = require('./animals');

console.log(animals);
animals.showAnimals();
```

Module.js

CommonJS 방식



- 내보내고 싶은 것(변수, 함수, 클래스 등등)에 exports 를 붙여서 내보내고, 각각을 따로 선언해서 가져 오는 방식

```
const animals = ['dog', 'cat'];  
  
exports.animals = animals;  
  
exports.showAnimals = function showAnimals() {  
  animals.map((el) => console.log(el));  
};
```

Animal.js

```
const { animals, showAnimals } = require('./animals');  
  
console.log(animals);  
showAnimals();
```

Module.js

CommonJS 방식



- 하나의 객체(or 클래스)에 전부를 넣어놓고 그 객체 자체를 내보내는 방식!

```
const animals = {  
  animals: ["dog", "cat"],  
  showAnimals() {  
    this.animals.map((el) => console.log(el));  
  },  
};
```

```
module.exports = animals;
```

Animal.js

```
const { animals, showAnimals } = require('./animals');
```

```
console.log(animals);  
showAnimals();
```

Module.js



ES6 방식

ES6 방식 - export



- 선언부에 Export 사용하기

```
export const animals = ['dog', 'cat'];  
  
export function showAnimals() {  
  animals.map((el) => console.log(el));  
}
```

Animal.js

```
import { animals, showAnimals } from './animals.js';  
  
console.log(animals);  
showAnimals();
```

Module.js

ES6 방식 - export



- 마지막으로 Export 사용하기

```
const animals = ['dog', 'cat'];

function showAnimals() {
  animals.map((el) => console.log(el));
}

export { animals, showAnimals };
```

Animal.js

```
import { animals, showAnimals } from './animals.js';

console.log(animals);
showAnimals();
```

Module.js

ES6 방식 - import



- 가져올 것들이 많으면 * as 를 사용

```
import * as animals from './animals.js';
```

```
console.log(animals);  
animals.showAnimals();
```

Module.js

- 단, 보통의 경우는 가져올 것을 확실히 명시하는 편이 좋습니다!
 - 메모리 효율 및 처리 속도가 올라갑니다!

ES6 방식 - 모듈 이름 바꾸기(import)



- 모듈 이름을 바꾸고 싶으면 **모듈이름 as 새로운모듈이름** 으로 변경이 가능
(import, export 동시 적용 가능)

```
import { animals as ani, showAnimals as show } from './animals.js';  
  
console.log(ani);  
show();
```

```
export const animals = ['dog', 'cat'];  
  
export function showAnimals() {  
  animals.map((el) => console.log(el));  
}
```

ES6 방식 - 모듈 이름 바꾸기(export)



- 모듈 이름을 바꾸고 싶으면 **모듈이름 as 새로운모듈이름** 으로 변경이 가능
(import, export 동시 적용 가능)

```
import { ani, show } from './animals.js';  
  
console.log(ani);  
show();
```

```
const animals = ['dog', 'cat'];  
  
function showAnimals() {  
  animals.map((el) => console.log(el));  
}  
  
export { animals as ani, showAnimals as show };
```

ES6 방식 – export default



- Export default 로 보내내진 모듈을 Import 할 때에는 {} 를 사용하지 않고 불러옵니다!

```
export default class Animal {  
  constructor() {  
    this.animals = ['dog', 'cat'];  
  }  
  
  showAnimals() {  
    this.animals.map((el) => console.log(el));  
  }  
}
```

```
import Animal from './animals.js';  
  
const ani = new Animal();  
console.log(ani.animals);  
ani.showAnimals();
```



CommonJS 와 ES6 의 차이점



CommonJS 와 ES6 의 차이점

- CommonJS 는 Node.js 에서 사용되고 require / exports 사용
- ES6 는 브라우저에서 사용되고 import / export 사용
- ES6를 Node.js 에서 사용하고 싶으면 package.json 에 `"type": "module"` 추가 필요
- Require 는 코드 어느 지점에서나 사용 가능 / Import 는 코드 최상단에서만 사용 가능
- 하나의 파일에서 둘 다 사용은 불가능!
- 일반적으로 ES6 문법이 필요한 모듈만 불러오는 구조를 가지기에 성능이 우수, 메모리 절약 → 그런데 CommonJS 도 해당 문법이 추가 되었음!



Routing

처리

```
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => {
  res.render('users', { USER_ARR, userCounts: USER_ARR.length });
});

router.get('/id/:id', (req, res) => {
  const userData = USER[req.params.id];
  if (userData) {
    res.send(userData);
  } else {
    res.send('ID를 못찾겠어요');
  }
});

module.exports = router;
```

routes/users.js

express.Router()를
사용하여 하나의
Router 모듈을 생성

라우터에 각각의 미들웨어
기능들을 작성하여
모듈에 기능을 추가!

라우터에 작성 된 모든 기능들을 하나의
모듈로써 외부로 내보내기!



```
const express = require('express');  
  
const app = express();  
const PORT = 4000;  
  
const userRouter = require('./routes/users');  
app.use('/users', userRouter);
```

router.js

routes/users.js 에서 기능을 정의해서
내보낸 모듈을 userRouter 라는
이름의 변수를 선언하여 담아주기!

http://localhost:4000/users 로 들어오는
모든 요청은 이제 userRouter 에 들어있는
코드가 처리하도록 설정



Error 핸들링!



Error 던지기!

```
router.get('/id/:id', (req, res) => {  
  const userData = USER.find((user) => user.id === req.params.id);  
  if (userData) {  
    res.send(userData);  
  } else {  
    const err = new Error('해당 ID를 가진 회원이 없습니다!');  
    err.statusCode = 404;  
    throw err;  
  }  
});
```

- 이런 방식으로 err 가 발생한 지점에서 new Error 를 통해 err 객체를 만들어서 throw 로 전달하면 됩니다!



App.js 에서 에러 핸들링!

- App.js 의 미들 웨어중 마지막 미들웨어에 Throw 된 err 를 받는 미들웨어를 추가해 줍시다!

```
app.use((err, req, res, next) => {  
  console.log(err.stack);  
  res.status(err.statusCode);  
  res.send(err.message);  
});
```

- 해당 미들웨어는 err 인자와 res 를 같이 써야 하므로 app.use() 메소드의 매개변수를 전부 사용해 줘야 합니다. 3개만 쓰면 첫번째는 req, 두번째는 res, 세번째는 next 로 인식합니다 → 그리고 무엇보다 err 를 못받습니다!



Form 으로
데이터 전송하기!



```
<form action="/users/add" method="POST">
  <div>
    <label>아이디</label>
    <input type="text" name="id" />
  </div>
  <div>
    <label>이름</label>
    <input type="text" name="name" />
  </div>
  <div>
    <label>이메일</label>
    <input type="email" name="email" />
  </div>
  <button type="submit">Submit</button>
</form>
```

A visual representation of the HTML form on a bright orange background. It contains three white text input fields stacked vertically. The first field is preceded by the Korean text '아이디' (ID), the second by '이름' (Name), and the third by '이메일' (Email). Below the third field is a white button with the text 'Submit'. The entire form is set against a green horizontal bar at the bottom.

Body-parser 사용



```
const bodyParser = require('body-parser');  
  
app.use(bodyParser.json());  
app.use(bodyParser.urlencoded({ extended: false }));
```

- Json() 은 json 형태로 데이터를 전달한다는 의미 입니다!
- Urlencoded(url-encoded) 옵션은 url 처럼 데이터를 변환하면
localhost:4000/posts?title=title&content=content 해당 데이터를
json 형태 { "title": "title, "content": "content" } 라고 전달 합니다.

Body-parser 은 이제 기본으로 제공 됩니다!



- 하도 많이 써서 이제는 express 에 기본 기능으로 추가가 되었습니다.

```
app.use(express.json());  
app.use(express.urlencoded({ extended: false }));
```

- 단, express 4.16 이상에서만 먹힙니다!

```
"dependencies": {  
  "body-parser": "^1.20.0",  
  "express": "^4.18.1",  
  "yarn": "^1.22.19"  
},
```

Body-parser 의 동작에 대해서 알아 봅시다



- 먼저 body-parser 를 쓰지 않은 상태에서 req.body 값을 찍어서 확인해 봅시다!

```
// app.use(express.json());  
// app.use(express.urlencoded({ extended:  
false }));
```

```
router.post('/add', (req, res) => {  
  console.log(req.body);  
  if (req.query.id && req.query.name && req.query.age) {  
    const newUser = {
```

```
[nodemon] starting node route  
4000 번에서 서버 실행  
undefined  
□
```



```
router.post('/add', (req, res) => {
  if (Object.keys(req.query).length >= 1) {
    if (req.query.id && req.query.name && req.query.email) {
      const newUser = {
        id: req.query.id,
        name: req.query.name,
        email: req.query.email,
      };
      USER.push(newUser);
      res.send('회원 등록 완료');
    } else {
      const err = new Error('쿼리 입력이 잘못 되었습니다!');
      err.statusCode = 404;
      throw err;
    }
  } else if (req.body) {
    if (req.body.id && req.body.name && req.body.email) {
      const newUser = {
        id: req.body.id,
        name: req.body.name,
        email: req.body.email,
      };
      USER.push(newUser);
      res.redirect('/users');
    } else {
      const err = new Error('쿼리 입력이 잘못 되었습니다!');
      err.statusCode = 404;
      throw err;
    }
  } else {
    const err = new Error('No data');
    err.statusCode = 404;
    throw err;
  }
});
```

- ID : 11

NAME : 11

EAMIL : 11@11



Front 에서

Back 으로 요청 보내기

```
<script>
  function deleteUser(id) {
    const xhr = new XMLHttpRequest();
    xhr.open('DELETE', `http://localhost:4000/users/${id}`);
    xhr.responseType = 'json';
    xhr.onload = () => {
      console.log(xhr.response);
      location.reload();
    };
    xhr.send();
  }
</script>
```





```
<script>
  function deleteUser(id) {
    console.log(`http://localhost:4000/users/${id}`);
    $.post(`http://localhost:4000/users/${id}`, function (res) {
      console.log(res);
      location.reload();
    }).done(function (data) {
      console.log(data);
    }).fail(function (err) {
      console.log(err);
    })
  }
</script>
```



```
<script>
  function deleteUser(id) {
    fetch(`http://localhost:4000/users/${id}`, {
      method: 'delete',
      headers: {
        'Content-type': 'application/json'
      },
    }).then((res) => {
      console.log(res);
      location.reload();
    })
  }
</script>
```




Fetch 를 이용해서
삭제 기능 구현!



Ejs 를 활용하면 됩니다!

- USER[i].id 값을 전달하면 되므로

```
<a href="" onclick="deleteUser('<%= USER[i].id %>');">삭제</a>
```

- 위와 같이 deleteUser 함수의 매개 변수로 USER[i].id 값을 전달 합니다!

Promise 버전



```
<script>
  function deleteUser(id) {
    fetch(`http://localhost:4000/users/${id}`, {
      method: 'delete',
      headers: {
        'Content-type': 'application/json'
      },
    }).then((res) => {
      console.log(res);
      location.reload();
    })
  }
</script>
```

Async/Await 버전



```
<script>
  async function deleteUser(id) {
    const res = await fetch(`http://localhost:4000/users/delete/${id}`, {
      method: 'delete',
      headers: {
        'Content-type': 'application/json'
      }
    });

    if (res) location.reload();
  }
</script>
```



지금 시작합니다



Express 기본!

- 기본 세팅



Express 기본 구조 만들기!

- 프로젝트 용 폴더 생성

```
✓ EXPRESS
  .gitattributes
```

- Npm init -y

- 명령어를 통해서 node package manager 기본 세팅 완료 하기!

```
✓ EXPRESS
  .gitattributes
  {} package.json      U
```



깃 이그노어 추가하기!

- Package 파일들일 설치 될 node_modules 폴더는 미리 .gitignore 에 추가하기!



- Npm init -y
 - 명령어를 통해서 node package manager 기본 세팅 완료 하기!

필요 모듈 설치 하기!



- 배포 시 필요한 패키지 / npm install -save 옵션(npm i -S)
 - Express
 - cors
 - Ejs

```
lhs@DESKTOP-86MUCGC MINGW64 /d/git/express (main)
$ npm i -S express cors ejs

added 75 packages, and audited 76 packages in 4s

9 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

필요 모듈 설치 하기!



- 개발 시에만 필요한 패키지 / `npm install --save-dev` 옵션(`npm i -D`)
 - Prettier
 - Eslint / `eslint-config-airbnb-base` / `eslint-plugin-import`

```
lhs@DESKTOP-86MUCGC MINGW64 /d/git/express (main)
$ npm i -D prettier eslint eslint-config-airbnb-base eslint-plugin-import

added 158 packages, and audited 234 packages in 10s

73 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Prettier / Eslint 설정 파일 가져오기!



- 기존에 세팅 해놓은 파일을 가지고 와서 바로 적용 합시다!
- 프로젝트 폴더 최 상위에
- .pritterrc / .eslintrc.js 파일 복사 붙여 넣기
- .vscode 폴더도 복사 붙여 넣기!
- 설치 커맨드를 저장한 파일을 만들어도 편합니다~!

≡ set-command.txt

```
1  npm i -S express cors ejs
2  npm i -D prettier eslint eslint-config-airbnb-base eslint-plugin-import
3  |
```



Express 기본!

- 서버 만들기!

서버 실행 코드 작성!



- App.js 파일을 생성하고 기본 서버를 생성하는 코드를 작성해 봅시다!

```
const express = require('express');  
const cors = require('cors');  
  
const app = express();  
const PORT = 4000;  
  
app.use(cors());  
  
app.listen(PORT, () => {  
  console.log(`서버는 ${PORT}번 포트에서 실행 중입니다!`);  
});
```

필요 패키지를 불러오기!

Express 를 실행해서 app 에 넣기!
포트 번호 설정

서버 실행 코드 작성!



- App.js 파일을 생성하고 기본 서버를 생성하는 코드를 작성해 봅시다!

```
const express = require('express');
const cors = require('cors');

const app = express();
const PORT = 4000;

app.use(cors());

app.listen(PORT, () => {
  console.log(`서버는 ${PORT}번 포트에서 실행 중입니다!`);
});
```

서버에 필요한 기능들을 설정
Cors 패키지를 사용하겠다 설정하기

익스프레스 listen 메소드를 사용
서버 실행하기!

서버 실행 하기!



- nodemon app.js 로 서버 실행하기!

```
lhs@DESKTOP-86MUCGC MINGW64 /d/git/express (main)
$ nodemon app.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
```

```
New version of nodemon available!
Current Version: 2.0.20
Latest Version: 2.0.21
```

서버는 4000번 포트에서 실행 중입니다!



주소 : <http://localhost:4000>



<http://localhost:4000>

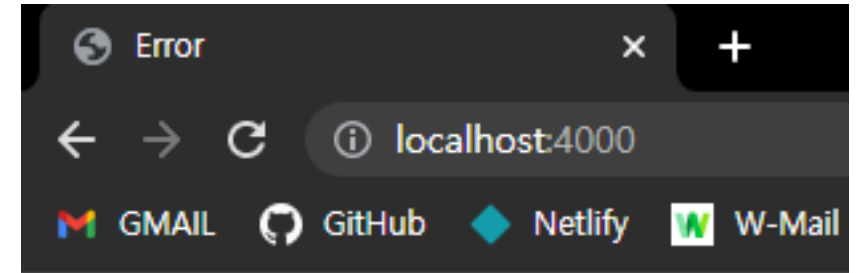
server



주소 : <http://localhost:4000>

server

<http://localhost:4000>



서버에 주소 요청을 받아줄 미들웨어 설정!



- 힘들게 찾아온 요청을 받아 줄 미들웨어를 만들어 줍시다!

```
const express = require('express');  
const cors = require('cors');
```

```
const app = express();  
const PORT = 4000;
```

```
app.use(cors());
```

```
app.get('/', (req, res) => {  
  res.send('어서와 Express 는 처음이지!~');  
});
```

```
app.listen(PORT, () => {  
  console.log(`서버는 ${PORT}번 포트에서 실행 중입니다!`);  
});
```

미들웨어

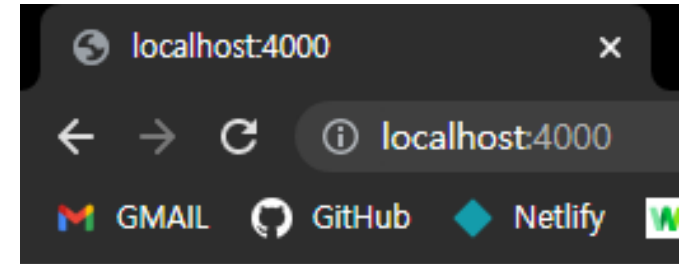


주소 : <http://localhost:4000>

server

<http://localhost:4000/>

<http://localhost:4000/>



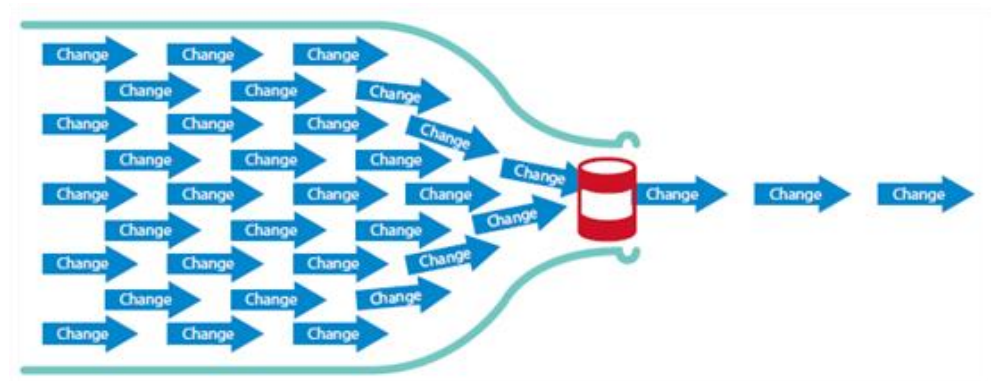
어서 와 Express 는 처음이지!?



주소별로 업무를 나누자! 라우팅!



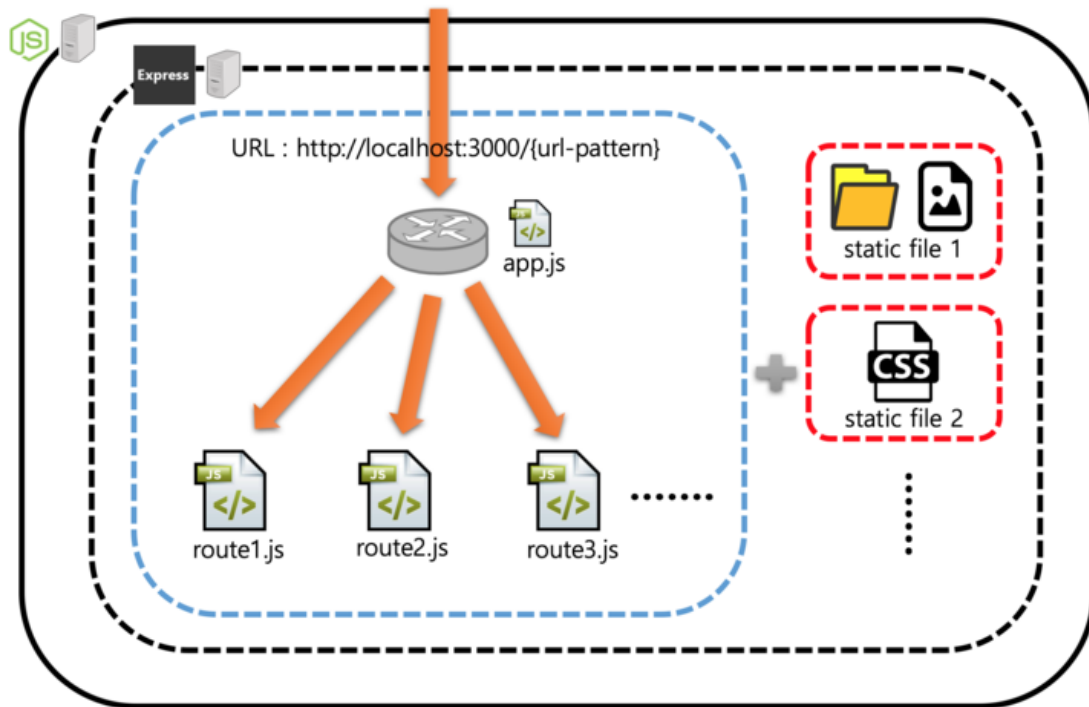
- 모든 요청을 하나의 주소에서 받으면 어떤 요청인지 구분이 어렵겠죠?
- 그리고 메인 서버 코드도 복잡하게 됩니다!



주소별로 업무를 나누자! 라우팅!



- 주소 요청에 따라 각각 담당하는 파일을 나눈다면!?
- 병목 현상 → 해결 / 코드 유지 보수 → 편해짐



주소별로 업무를 나누자! 라우팅!



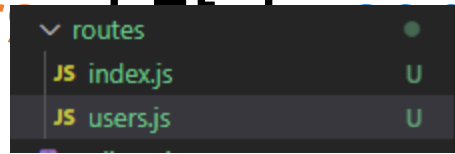
- 특정 요청은 특정 주소로만 들어도록 처리 + 해당 요청은 하나의 파일에 몰아서 처리하도록 설정!
- Routes 폴더 생성하기

```
> routes
```

주소별로 업무를 나누자! 라우팅!



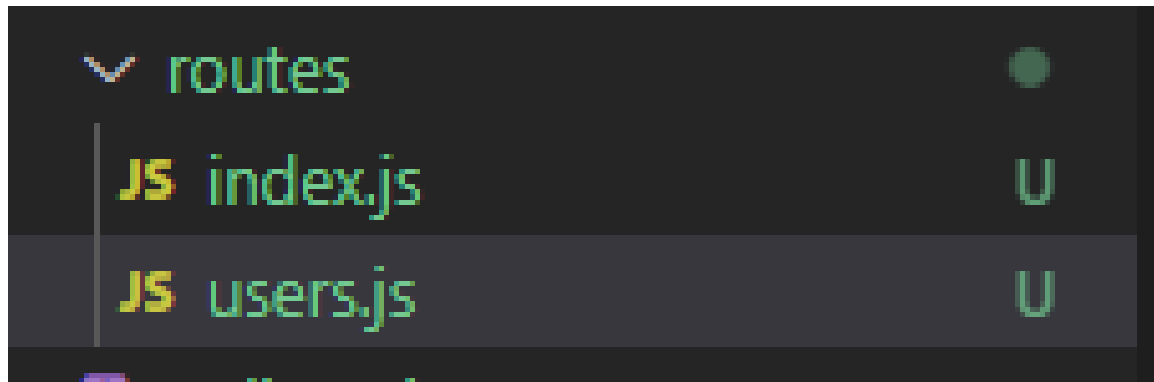
- 서버의 가장 기본 주소에 대한 처리도 mainRouter 를 만들어서 처리! → 서버 코드를 깔끔하게 유지하기 위함!
- 회원 관련 요청은 전부 `/users` `users.js` 파일을 통해 처리하도록 설정!



주소별로 업무를 나누자! 라우팅!



- Routes 폴더에 메인 라우터를 위한 index.js 파일과 회원 관련 기능을 위한 users.js 파일 생성



메인 라우터 작업!



- 가장 기본이 되는 요청 (<http://localhost:4000/>) 을 처리하는 index.js 작업

```
const express = require('express');  
const router = express.Router();  
  
router.get('/', (req, res) => {  
  res.send('여기는 메인 라우터 입니다!');  
});  
  
module.exports = router;
```

Express Router 를 호출하고
Router 변수에 넣어주기!

<http://localhost:4000/>
요청에 대한 처리!

외부(app.js 서버 코드)에서 사용이
가능하도록 모듈로 빼주기!

메인 라우터를 서버에 적용!



- mainRouter 역할을 할, index.js 모듈을 불러오고 서버에 추가해 줍시다!

```
const express = require('express');
const cors = require('cors');

const app = express();
const PORT = 4000;

app.use(cors());

const mainRouter = require('./routes');

app.use('/', mainRouter);

app.listen(PORT, () => {
  console.log(`서버는 ${PORT}번 포트에서 실행 중입니다!`);
});
```

Index.js 파일을 모듈로 불러오기
Index.js 는 생략 가능!

서버 주소인
http://localhost:4000 이후에
들어오는 주소가 '/' 로 들어오면

메인 라우터로 요청을 보내도록
설정해주기!

유저 라우터 작업!



- 회원 관련 작업 요청(<http://localhost:4000/users>)을 처리하는 유저 라우터 작업!

```
const express = require('express');  
  
const router = express.Router();  
  
router.get('/', (req, res) => {  
  res.send('여기는 유저 라우터 입니다!');  
});  
  
module.exports = router;
```

<http://localhost:4000/users>
요청에 대한 처리!

유저 라우터를 서버에 적용!



- userRouter 역할을 할, users.js 모듈을 불러오고 서버에 추가해 줍시다!

```
const mainRouter = require('./routes');  
const userRouter = require('./routes/users');  
  
app.use('/', mainRouter);  
app.use('/users', userRouter);
```

users.js 파일을 모듈로 불러오기
Index.js 는 생략 가능!

서버 주소인
http://localhost:4000 이후에
들어오는 주소가 '/users' 로 들어오면

유저 라우터로 요청을 보내도록
설정해주기!



server

<http://localhost:4000/>

<http://localhost:4000/users>

<http://localhost:4000/>



server

<http://localhost:4000/>

<http://localhost:4000/users>

`http://localhost:4000/users`



server

`http://localhost:4000/`

`http://localhost:4000/users`

<http://localhost:4000/tetz>



뷰 엔진 적용 하기!



- 백엔드 서버에서 데이터를 바로 받아서 그릴 수 있는 뷰 엔진을 적용

```
const express = require('express');
const cors = require('cors');
const app = express();
const PORT = 4000;

app.use(cors());
app.set('view engine', 'ejs');

const mainRouter = require('./routes');
const userRouter = require('./routes/users');
app.use('/', mainRouter);
app.use('/users', userRouter);

app.listen(PORT, () => {
  console.log(`서버는 ${PORT}번 포트에서 실행 중입니다!`);
});
```

뷰엔진을 ejs 로 세팅한다는 코드 추가

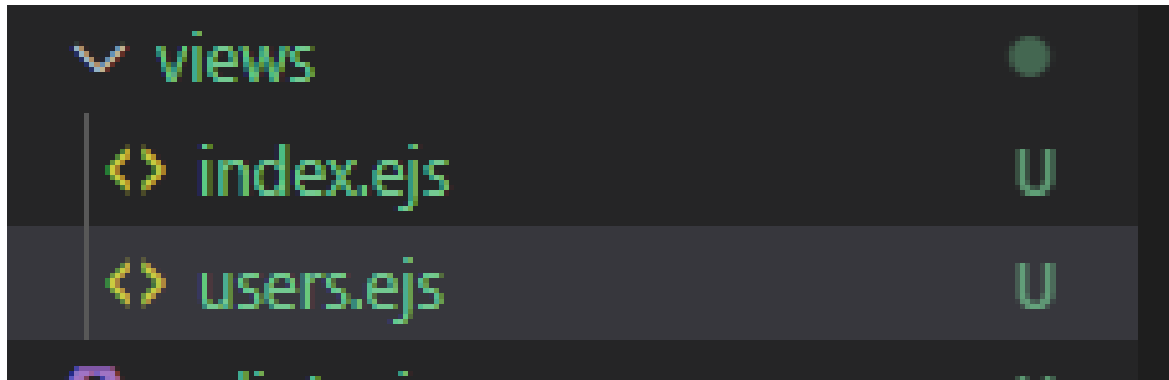
해당 코드를 추가하면 Express 가
알아서 프로젝트 폴더의 views 폴더를
인식하게 됩니다!

그리고 ejs 파일을 찾아가죠!

뷰 폴더 생성 & 뷰 파일 설정!



- View 파일이 저장되는 views 폴더 생성하기
- 메인 페이지를 위한 index.ejs 파일과 회원 서비스를 위한 users.ejs 파일 생성 하기!



뷰 파일 코드 작성!



- 간단하게 H1 태그로만 구성해 봅시다!

```
<!DOCTYPE html>
<html lang="en">

<head>
</head>

<body>
  <h1>메인 페이지 입니다!</h1>
</body>

</html>
```

```
<!DOCTYPE html>
<html lang="en">

<head>
</head>

<body>
  <h1>회원 페이지 입니다!</h1>
</body>

</html>
```



미들웨어에 뷰 파일 연결하기!

- 뷰 파일을 그리는 것은 `res.render` 를 사용 합니다!
- 먼저 메인 페이지 부터 연결해 보시죠!

```
const express = require('express');  
  
const router = express.Router();  
  
router.get('/', (req, res) => {  
  res.render('index');  
});  
  
module.exports = router;
```

Routes/index.js

`Res.render('뷰파일명')`
을 통해서 해당 요청이 들어오면
뷰 파일에 연결이 되도록 설정하기!

<http://localhost:4000/>

ROUTER

server

Document

x

+



localhost:4000



GMAIL



GitHub



Netlify



W-Mail



메인 페이지 입니다!

<http://localhost:4000/users>

뷰 파일에 데이터 전달하기!



- 뷰 파일에 백엔드 데이터를 전달하려면, `res.render` 메소드의 두 번째 인자로 객체를 전달하고 해당 객체에 데이터를 넣어 주면 됩니다!

```
const express = require('express');  
const router = express.Router();  
  
router.get('/', (req, res) => {  
  res.render('index', { msg: '이 데이터는 백엔드가 보냈어요!' });  
});  
  
module.exports = router;
```

Routes/index.js

`Res.render('뷰파일명', {데이터})`
을 통해서 백엔드의 데이터를
뷰파일에 전달 가능!

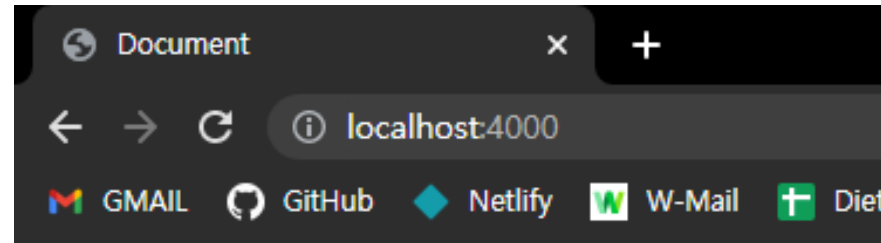
뷰 파일에서 백엔드 데이터 사용하기!



- 뷰 파일에서는 ejs 문법 `<% %>` or `<%= %>` 을 사용하여 데이터 출력이 가능합니다!

```
res.render('index', { msg: '이 데이터는 백엔드가 보냈어요!' });
```

```
<body>  
  <h1>메인 페이지 입니다!</h1>  
  <p><%= msg %></p>  
</body>
```



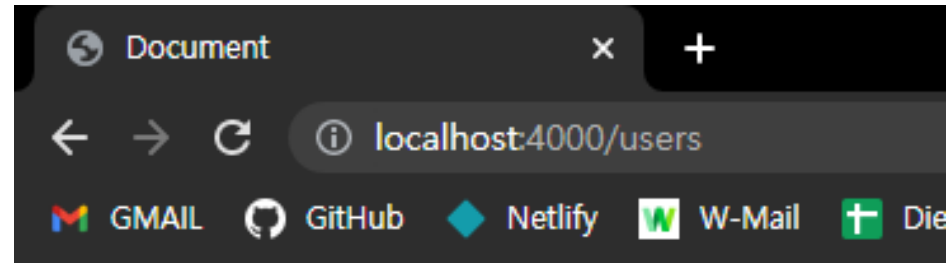
메인 페이지 입니다!

이 데이터는 백엔드가 보냈어요!

실습, 회원 페이지 연결하기!



- <http://localhost:4000/users> 요청이 들어오면 views 폴더의 users.ejs 파일이 렌더링 되도록 서버를 구성해 주세요!
- 렌더링 할 때, user 라는 데이터에 본인의 이름을 보내고 해당 데이터를 ejs 문법을 사용하여 출력하여 주세요!



회원 페이지 입니다!

tetz 회원님 반갑습니다!

스태틱 폴더 설정 하기!



- 프론트에서 백엔드 서버에 마구 접근을 하면 보안 적으로 위험에 노출
- 그래서 접근이 가능한 폴더는 STATIC 을 사용해서 따로 설정을 해주셔야 합니다!



스태틱 폴더 설정 하기!



- `express.static('지정할 폴더명')` 을 통해 지정 가능!
- `App.use` 를 사용하여 스태틱 폴더 사용을 서버에 알려주기!

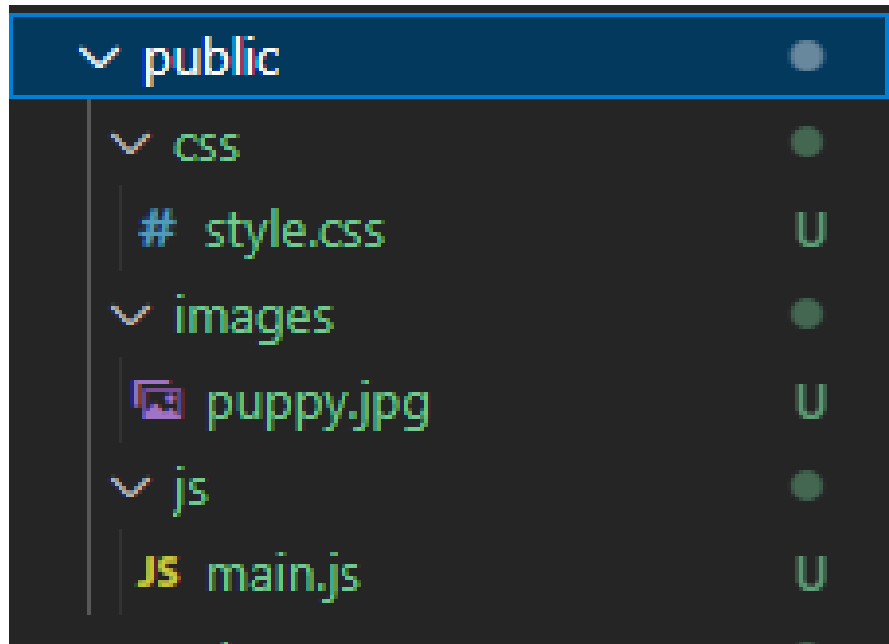
```
app.use(cors());  
app.set('view engine', 'ejs');  
app.use(express.static('public'));
```

- 스태틱 폴더는 프로젝트 폴더 최상위에 존재해야 합니다!

프론트에서 스태틱 폴더 사용하기!



- 퍼블릭 폴더에 css / js / images 폴더 세팅 후 불러와서 테스트!
- 이제 프론트에서 / 라는 주소는 백엔드의 /public 폴더를 뜻합니다!





```
<!DOCTYPE html>
<html lang="en">

<head>
  <link rel="stylesheet" href="/css/style.css">
  <script defer src="/js/main.js"></script>
</head>

<body>
  <h1>메인 페이지 입니다!</h1>
  <p><%= msg %></p>
  
</body>

</html>
```

localhost:4000 내용:

!

확인

메인 페이지 입니다!

이 데이터는 백엔드가 보냈어요!



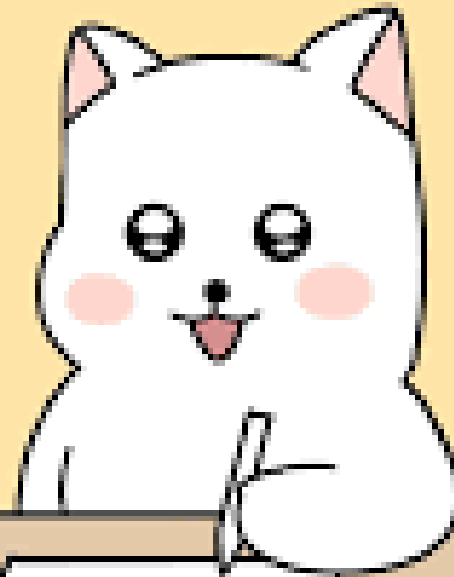


대학생 풍습이



시험 문제는
기초만 알면 됩니다-

교수의 기초와
학생의 기초의
차이를 알 수 있었다





게시판
만들기!!

View part!



- 프론트 코드는 제가 작업한 파일을 바탕으로 사용하겠습니다!
- <https://github.com/xenosign/backend2/blob/main/views/board.ejs>
- https://github.com/xenosign/backend2/blob/main/views/board_write.ejs
- https://github.com/xenosign/backend2/blob/main/views/board_modify.ejs

✓ views	
<> board_modify.ejs	1, U
<> board_write.ejs	1, U
<> board.ejs	1, U



Board.js 로 라우팅!

- 먼저 board.js 파일을 만들고 express 모듈 추가 등의 기본 코드만을 추가하고 모듈화 작업

```
const express = require('express');  
  
const router = express.Router();  
  
module.exports = router;
```

- App.js 에서 해당 모듈 불러오고 주소 라우팅 설정

```
const boardRouter = require('./routes/board');  
app.use('/board', boardRouter);
```




Board.js 기본 기능 정의

- 글 전체 목록 보여주기
 - GET /board/ → 글 전체 목록 보여주기
- 글 쓰기
 - GET /board/write → 수정 모드로 이동
 - POST /board/ → 글 추가 기능 수행
- 글 수정
 - GET /board/modify → 글 수정 모드로 이동
 - POST /board/title/:title → 파라미터로 들어온 title 값과 동일한 글을 수정
- 글 삭제
 - DELETE /board/title/:title → 파라미터로 들어온 title 값과 동일한 글을 삭제



Board.js 기본 코드 작업

- 데이터로 사용할 배열 선언

```
const ARTICLE = [  
  {  
    title: 'title',  
    content:  
      'Lorem ipsum, dolor sit amet consectetur adipisicing elit. Quia delectus iusto  
fugiat autem cupiditate adipisci quas, in consectetur repudiandae, soluta, suscipit  
debitis veniam nobis aspernatur blanditiis ex ipsum tempore impedit.',  
  },  
  {  
    title: 'title2',  
    content:  
      'Lorem ipsum, dolor sit amet consectetur adipisicing elit. Quia delectus iusto  
fugiat autem cupiditate adipisci quas, in consectetur repudiandae, soluta, suscipit  
debitis veniam nobis aspernatur blanditiis ex ipsum tempore impedit.',  
  },  
];
```

Board.js 기본 코드 작업



- 요청 주소 미들 웨어 만들기



```
router.get('/', (req, res) => {  
  // 글 전체 목록이 보이는 페이지  
});  
  
router.get('/write', (req, res) => {  
  // 글 쓰기 모드로 이동  
});  
  
router.post('/write', (req, res) => {  
  // 글 추가  
});  
  
router.get('/modify/:title', (req, res) => {  
  // 글 수정 모드로 이동  
});  
  
router.post('/modify/:title', (req, res) => {  
  // 글 수정  
});  
router.delete('/delete/:title', (req, res) => {  
  // 글 삭제  
});
```



전체 목록 보여주기



전체 목록 보여주기

- 글 전체 목록을 데이터에 담아서 board.ejs 파일 그려주기
- `res.render(뷰파일명, 데이터);` 사용
- Ejs 코드에서 사용한 변수명과 일치하는지 체크 필요

```
router.get('/', (req, res) => {  
  const articleCounts = ARTICLE.length;  
  res.render('board', { ARTICLE, articleCounts });  
});
```



```
div class="board_write">
  <span>현재 등록 글 : &nbsp; <%= articleCounts %></span>
  <a class="btn red" href="/board/write">글쓰기</a>
</div>
<div class="board_body">
  <ul class="board">
    <% if (articleCounts > 0) { %>
      <% for(let i=0; i < articleCounts; i++) { %>
        <li>
          <div class="title">
            <%= ARTICLE[i].title %>
```



글 쓰기 모드로 이동



글 쓰기 모드로 이동

- Board.ejs 뷰에서 글쓰기 버튼을 클릭 → </board/write> 라는 주소로 이동
- 해당 요청이 들어오면 글 쓰기 페이지 board_write.ejs 를 그려주기

```
<div class="board_write">  
  <span>현재 등록 글 : &nbsp;   <%= articleCounts %></span>  
  <a class="btn red" href="/board/write">글쓰기</a>  
</div>
```

- 데이터 전달 필요 X

```
router.get('/write', (req, res) => {  
  res.render('board_write');  
});
```



글 추가 기능



글 추가 기능(프론트)

- 글쓰기 모드에서 입력 받은 title, content 를 새로운 글로 추가하기
- 아무것도 안 쓸 경우의 예외를 처리하기 위해 필수 입력 지정(required)
- 주소는 </board/write>, 메소드는 POST 로 전달

```
<form action="/board/write" method="POST" class="board_form">
  <div class="form_title">
    <h3>제목</h3>
    <input type="text" name="title" required />
  </div>
  <div class="form_content">
    <h3>내용</h3>
    <textarea name="content" id="content" cols="30" rows="10" required></textarea>
  </div>
  <button type="submit">글 작성하기</button>
</form>
```



글 추가 기능(백)

- 주소는 `/board/write` 로 들어왔으므로 `/write` 처리
- 메소드는 post 이므로 `router.post('/write', (req, res) => {})` 로 처리
- 프론트에서 예외 처리를 하였지만, 데이터가 누락되는 경우를 대비



실습, 추가 기능 백엔드 코드 작성하기!

- Req.bdy로 들어온 title / content 를 백엔드의 ARTICLE 배열에 추가하는 코드를 만들어 봅시다!
- 추가가 완료 되면 /board 로 리다이렉트 해줘서 추가 된 글이 바로 보이도록 설정해 주세요!
- Req.body 의 값이 잘 들어오지 않았으면 Err 를 발생 시키면 됩니다!



글 수정 모드로 이동



글 수정 모드로 이동

- 뷰에서 수정 버튼을 클릭 → </board/modify/:title> 라는 주소로 이동
- title 파라미터는 ejs 에서 직접 입력하여 전달

```
<a class="btn orange" href="board/modify/<%= ARTICLE[i].title %>">수정</a>
```

- 데이터 전달 필요 O, ARTICLE 배열에서 제목이 같은 글을 찾아 전달

```
router.get('/modify/:title', (req, res) => {  
  const arrIndex = ARTICLE.findIndex(  
    (article) => article.title === req.params.title  
  );  
  const selectedArticle = ARTICLE[arrIndex];  
  res.render('board_modify', { selectedArticle });  
});
```



글 수정 모드 뷰 페이지

- 전달 받은 데이터(selectedArticle)를 `<input>` `<textarea>` 값으로 전달
- 데이터 전송은 수정할 글의 title 을 파라미터로 담아서
`board/modify/:title` 로 전달 / 메소드는 **POST**

```
<form action="/board/modify/<%=selectedArticle.title%>" method="POST" class="board_form">
  <div class="form_title">
    <h3>제목</h3>
    <input type="text" name="title" value="<%= selectedArticle.title %>" required />
  </div>
  <div class="form_content">
    <h3>내용</h3>
    <textarea name="content" id="content" cols="30" rows="10" required>
      <%= selectedArticle.content %></textarea>
    </div>
  <button type="submit">글 수정하기</button>
</form>
```




PUT!?

- 순수 HTML은 데이터를 전송할 때 **GET, POST** 만 지원했었습니다
- 그래서 method 속성에 **PUT, DELETE** 를 입력해도 정상적으로 전달 X
- 이럴 때 **PUT, DELETE** 로 전달하고 싶다면?

```
<form action="#" method="post">  
  <input type="hidden" name="_method" value="put" />  
</form>
```

- 따라서 보통 수정, 삭제도 POST 로 구현하고 주소 또는 데이터로 구분하는 경우가 많아요 → 그래서 POST를 쓸 겁니다! 😊
- 그리고 form 태그에 한해서만 **POST, PUT, DELETE** 전송이 가능합니다!



글 수정 기능



글 수정 기능(백)

- 주소는 `/board/modify/:title` 로 요청이 들어 옵니다!
- 메소드는 post 이므로 `router.post('/modify/:title', (req, res) => {})` 로 처리
- 프론트에서 예외 처리를 하였지만, 데이터가 누락되는 경우를 대비



```
router.post('/modify/:title', (req, res) => {
  if (req.body.title && req.body.content) {
    const arrIndex = ARTICLE.findIndex(
      (_article) => _article.title === req.params.title
    );
    if (arrIndex !== -1) {
      ARTICLE[arrIndex].title = req.body.title;
      ARTICLE[arrIndex].content = req.body.content;
      res.redirect('/board');
    } else {
      const err = new Error('해당 제목의 글이 없습니다.');
```

err.statusCode = 404;

throw err;

}

} else {

const err = new Error('요청 쿼리 이상');

err.statusCode = 404;

throw err;

}

});



글 삭제 기능



글 삭제 기능(프론트)

- 삭제는 **DELETE** 요청을 해야 합니다! → 그런데 지금 우리는 A 태그에서만 요청을 보낼 수 있습니다!
- 따라서, 이전에 배운 JS의 fetch 를 사용합니다!

```
<a class="btn blue" href="#" onclick="deleteArticle('<%= ARTICLE[i].title %>')">삭제</a>
```

실습, 삭제 기능 프론트 & 백엔드 코드 작성



- 프론트에서 Fetch 를 이용해서 글 삭제 요청을 보내는 코드와, 백엔드에서 이를 받아서 실제로 글을 삭제하는 코드를 작성해 주세요!



글 삭제 기능(프론트)

- Fetch 를 통해 삭제 요청을 보냅니다
- 그리고 요청이 완료 되면 페이지를 다시 읽어 들어서 삭제 사항이 반영 되도록 합니다.

```
function deleteArticle(title) {  
  fetch(`board/title/${title}`, {  
    method: 'delete',  
    headers: {  
      'Content-type': 'application/json'  
    },  
  }).then((res) => {  
    location.href = '/board';  
  })  
}
```




글 삭제 기능(백)

- 요청이 `/board/delete/:title` 이고 **DELETE** 메소드 이므로 해당 주소를 받는 미들웨어를 만들어 줍니다.

```
router.delete('delete/:title', (req, res) => {});
```

- 다른 메소드에서는 `res.redirect` 를 통해 페이지를 다시 로드 했지만 **DELETE** 메소드에서 `redirect` 를 하면 **DELETE** 메소드를 따라가기 때문에 해당 주소에 대한 처리가 안되어 있어서 **404 NOT FOUND 에러**가 발생합니다! → 따라서 프론트에서 페이지를 리로딩 하여 GET 방식으로 페이지를 요청한 것 입니다!



```
router.delete('/delete/:title', (req, res) => {
  const arrIndex = ARTICLE.findIndex(
    (article) => article.title === req.params.title
  );
  if (arrIndex !== -1) {
    ARTICLE.splice(arrIndex, 1);
    res.send('삭제 완료!');
  } else {
    const err = new Error('해당 제목을 가진 글이 없습니다.');
    err.statusCode = 404;
    throw err;
  }
});
```





DataBase



데이터 들의 집합



DBMS

(DataBase Management System)



DBMS

- 데이터 베이스를 관리하고 운영하는 SW
- 다양한 형태, 서비스가 존재 합니다!





SQL

(Structured Query Language)

SQL



- 구조가 있는 질문용 언어라는 뜻
- SELECT, INSERT, UPDATE, DELETE 같은 언어를 통해 데이터 베이스의 데이터를 다루는 언어
- 다수의 DBMS가 SQL 방식을 따르기 때문에 SQL 을 배우면, 많은 DBMS를 비교적 빠르게 습득이 가능
- MySQL, SQLite, ORACLE 등이 SQL 구문을 사용



DB의 종류



관계형(SQL)

VS

비관계형(NoSQL)



관계형 DB



관계형, Relational DBMS(RDBMS)

- RDBMS 는 SQL 을 사용하는 DB 입니다
- 키와 값의 관계를 테이블화 시킨 원칙을 토대로 DB 를 구성합니다

아이디	이름	연락처
flower	화사	010-1111-1111
finetree	솔라	010-2222-2222
moon	문별	010-3333-3333
whee	휘인	010-4444-4444



관계형, Relational DBMS(RDBMS)

- 즉, RDBMS 는 테이블로 구성이 됩니다.
- 먼저 테이블이 구성되고 테이블의 구조에 맞추어 데이터가 들어가기 때문에 DB를 구성하기 전에 스키마라 불리는 DB의 구조, 관계, 제약 사항에 대한 정의가 필요합니다
- 사실 예전엔 컴퓨터가 느리고 용량이 작아서 이런 구조가 아니면 속도 측면과 용량면에서 답이 없었습니다 ☺



관계형, Relational DBMS(RDBMS)

- 장점

- 구조화가 명확하게 되어 있어서 예외가 없음
- 데이터 입, 출력 속도가 매우 빠릅니다
- 신뢰성이 매우 높음

- 단점

- DB의 구조 변경이 매우 어려움 → 빅데이터 등에는 사용이 어려움(새로운 키가 추가 되면 전체 스키마 변경이 필요)



비관계형 DB

비관계형, Non Relational DBMS(NoSQL)



- SQL 을 사용하지 않는 모든 DB를 통칭합니다
 - ex) 한국어 vs 비한국어(= 영어, 프랑스어, 스페인어 등등)
- 대표적으로 문서형, 그래프형, 키밸류형, 와이드 컬럼형 등이 있습니다!
- 약간 컴퓨터 언어와 일맥 상통하는 부분으로 특정 목적에 맞는 DB가 존재 합니다 (ex. Html → 웹 개발 / 파이썬 → 인공지능 / Swift → 앱)

비관계형, Non Relational DBMS(NoSQL)



- 장점

- 보통 대용량 데이터 처리에 효율적
- DB의 구조 변경이 쉽고, 확장성이 뛰어남
- 복잡한 데이터 구조의 표현이 가능

- 단점

- 데이터 자체가 크면 전체 데이터를 일부 읽어서 처리해야 하므로 데이터가 크면 속도가 저하되는 문제 발생

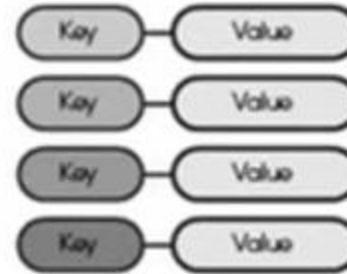
Document



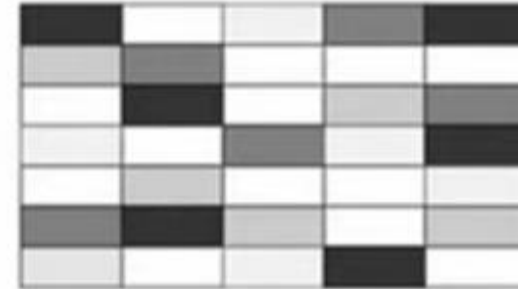
Graph



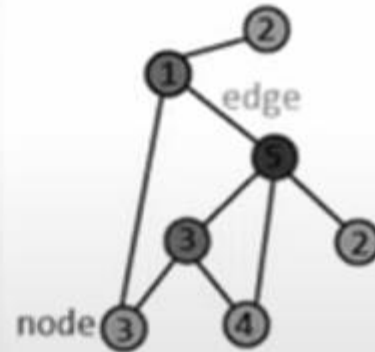
Key-Value



Wide-Column



```
{
  "user": {
    "id": "143",
    "name": "improgrammer",
    "city": "New York"
  }
}
```



1	Fruit	A Foo	B Baz	
2	City	E DC	D PLA	G FLD
3	State	A NZ	C CL	

mongoDB



ArangoDB

CouchDB



AllegroGraph



neo4j



redis



Ignite



MEMCACHED



riak



cassandra



APACHE
HBASE





SCYLLA



https://youtu.be/Q_9cFgzZr8Q



개발자	오라클
발표일	1995년 5월 23일
라이선스	GPL v2 (커뮤니티 에디션) 상용 라이선스 (표준, 엔터프라이즈, 클러스터)
링크	 



https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all

SQL Statement:

```
SELECT * FROM Customers;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Click "**Run SQL**" to execute the SQL statement above.

W3Schools has created an SQL database in your browser.

The menu to the right displays the database, and will reflect any changes.

Feel free to experiment with any SQL statement.

You can restore the database at any time.

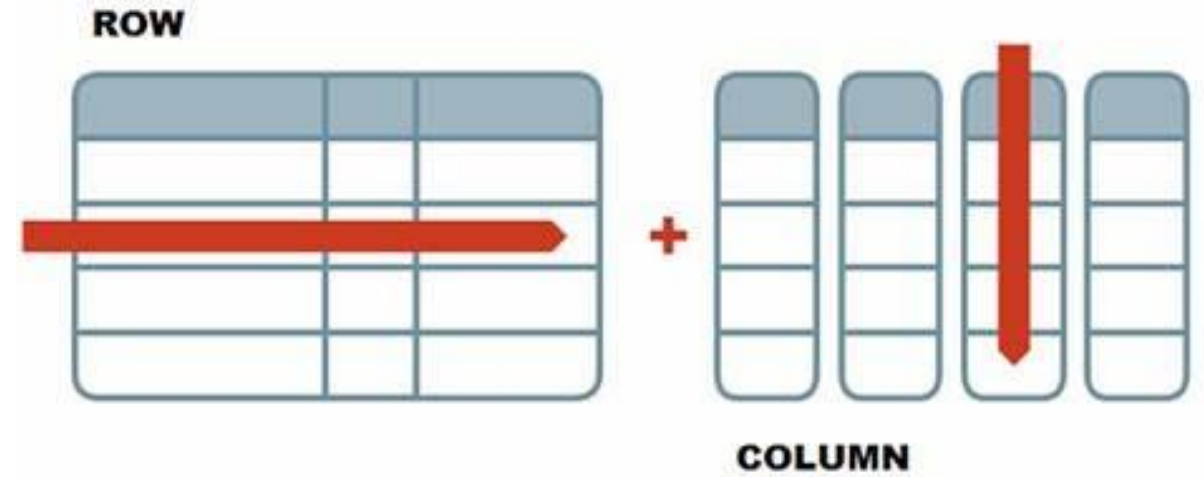




Diagram illustrating a table structure with labels and arrows:

- 행(row)**: Points to the rows of the table.
- 열 이름**: Points to the header row.
- 열(column)**: Points to the columns of the table.

아이디	이름	연락처
flower	화사	010-1111-1111
finetree	솔라	010-2222-2222
moon	문별	010-3333-3333
whee	휘인	010-4444-4444





SELECT



SELECT, DB 에서 원하는 데이터 가져오기

- 갓춰진 DB 에서 데이터를 뽑아 올 때 사용하는 구문 입니다!
- 일단
- `SELECT * FROM Customer;`

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain
9	Bon app'	Laurence Lebihans	12, rue des Bouchers	Marseille	13008	France
10	Bottom-Dollar Marketse	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen	T2F 8M4	Canada
11	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London	EC2 5NT	UK
12	Cactus Comidas para llevar	Patricio Simpson	Cerrito 333	Buenos Aires	1010	Argentina
13	Centro comercial Moctezuma	Francisco Chang	Sierras de Granada 9993	México D.F.	05022	Mexico
14	Chop-suey Chinese	Yang Wang	Hauptstr. 29	Bern	3012	Switzerland
15	Comércio Mineiro	Pedro Afonso	Av. dos Lusíadas, 23	São Paulo	05432-043	Brazil



원하는 컬럼의 값만 가지고 올 때?

- 모든 컬럼 값이 필요 한게 아니라면!?
- `SELECT 컬럼명 FROM table`
- `SELECT City, Country FROM Customers`

SQL Statement:



```
SELECT City, Country FROM Customers;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 91

City	Country
Berlin	Germany
México D.F.	Mexico
México D.F.	Mexico
London	UK
Luleå	Sweden
Mannheim	Germany
Strasbourg	France

DB에는 영향 없이 원하는 데이터 추가 할 때!



- 필요한 값을 임의로 추가해서 가져오고 싶을 땐?
- `SELECT` 컬럼명, 원하는 데이터 `FROM` table
- `SELECT` City, 1, '원하는 문자열', NULL `FROM` Customers

SQL Statement:

SELECT City, 1, '원하는 문자열', NULL FROM Customers

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 91

1	City	'원하는 문자열'	NULL
1	Berlin	원하는 문자열	null
1	México D.F.	원하는 문자열	null
1	México D.F.	원하는 문자열	null
1	London	원하는 문자열	null
1	Luleå	원하는 문자열	null
1	Mannheim	원하는 문자열	null
1	Strasbourg	원하는 문자열	null

원하는 조건을 충족하는 Row 만 가져 올 때!



- 원하는 조건을 충족 하는 값을 찾고 싶을 땐? **WHERE**
- **SELECT * FROM table WHERE 조건**
- **SELECT * FROM OrderDetails WHERE Quantity > 5;**

SQL Statement:

SELECT * FROM OrderDetails WHERE Quantity > 5;

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 476

OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
4	10249	14	9
5	10249	51	40
6	10250	41	10
7	10250	51	35
8	10250	65	15
9	10251	22	6



Row 를 정렬해서 가져 올 때!

- 선택한 값들을 정렬해서 가지고 올 때는? ORDER BY
- SELECT * FROM Customers ORDER BY ContactName;
- SELECT * FROM Customers ORDER BY ContactName DESC;
- SELECT * FROM Customers ORDER BY CustomerName ASC, ContactName DESC;

SQL Statement:

SELECT * FROM Customers ORDER BY ContactName;

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 91

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
69	Romero y tomillo	Alejandra Camino	Gran Vía, 1	Madrid	28001	Spain
52	Morgenstern Gesundkost	Alexander Feuer	Heerstr. 22	Leipzig	04179	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
81	Tradição Hipermercados	Anabela Domingues	Av. Inês de Castro, 414	São Paulo	05634-030	Brazil
31	Gourmet Lanchonetes	André Fonseca	Av. Brasil, 442	Campinas	04876-786	Brazil
19	Eastern Connection	Ann Devon	35 King George	London	WX3 6FW	UK
41	La maison d'Asie	Annette Roulet	1 rue Alsace-Lorraine	Toulouse	31000	France
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
21	Familia Arquibaldo	Aria Cruz	Rua Orós, 92	São Paulo	05442-030	Brazil
75	Split Rail Beer & Ale	Art Braunschweiger	P.O. Box 555	Lander	82520	USA

SQL Statement:

SELECT * FROM Customers ORDER BY ContactName DESC;

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 91

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
54	Océano Atlántico Ltda.	Yvonne Moncada	Ing. Gustavo Moncada 8585 Piso 20-A	Buenos Aires	1010	Argentina
42	Laughing Bacchus Wine Cellars	Yoshi Tannamuri	1900 Oak St.	Vancouver	V3F 2K1	Canada
36	Hungry Coyote Import Store	Yoshi Latimer	City Center Plaza 516 Main St.	Elgin	97827	USA



Row 의 개수를 지정 하고 싶을 때!

- 가지고 오는 ROW의 수를 지정하고 싶을 때? **LIMIT**
- **SELECT * FROM Customers LIMIT 10;**
- **SELECT * FROM Customers LIMIT 30, 10;**
 - 원하는 Row 순번, 자르는 개수

SQL Statement:

```
SELECT * FROM Customers LIMIT 10;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 10

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólide Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain
9	Bon app'	Laurence Lebihans	12, rue des Bouchers	Marseille	13008	France
10	Bottom-Dollar Marketse	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen	T2F 8M4	Canada

Column 명을 변경해서 가지고 오고 싶을 때?



- Column 명을 변경해서 가지고 올 때? **AS**
- **SELECT** CustomerId **AS** id, CustomerName **AS** name **FROM**
Customers;



SQL Statement:

```
SELECT CustomerId AS id, CustomerName AS name FROM Customers;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 91

id	name
1	Alfreds Futterkiste
2	Ana Trujillo Emparedados y helados
3	Antonio Moreno Taquería
4	Around the Horn
5	Berglunds snabbköp
6	Blauer See Delikatessen



테이블 합치기!

JOIN

여러 테이블의 값을 하나로 합치고 싶다면!?



- 여러 테이블을 하나로 붙이고 싶을 땐? JOIN
- `SELECT * FROM Products P JOIN Categories C ON P.CategoryID = P.CategoryID;`

SQL Statement:

SELECT * FROM Products P JOIN Categories C ON P.CategoryID = P.CategoryID;

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 616

ProductID	ProductName	SupplierID	CategoryID	Unit	Price	CategoryName	Description
1	Chais	1	1	10 boxes x 20 bags	18	Beverages	Soft drinks, coffees, teas, beers, and ales
1	Chais	1	2	10 boxes x 20 bags	18	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
1	Chais	1	3	10 boxes x 20 bags	18	Confections	Desserts, candies, and sweet breads
1	Chais	1	4	10 boxes x 20 bags	18	Dairy Products	Cheeses
1	Chais	1	5	10 boxes x 20 bags	18	Grains/Cereals	Breads, crackers, pasta, and cereal
1	Chais	1	6	10 boxes x 20 bags	18	Meat/Poultry	Prepared meats
1	Chais	1	7	10 boxes x 20 bags	18	Produce	Dried fruit and bean curd
1	Chais	1	8	10 boxes x 20 bags	18	Seafood	Seaweed and fish
2	Chang	1	1	24 - 12 oz bottles	19	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Chang	1	2	24 - 12 oz bottles	18	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings





Local 에서
연습 시작!

MySQL 설치하기!



- Windows

- <https://velog.io/@joajoa/MySQL-%EB%8B%A4%EC%9A%B4%EB%A1%9C%EB%93%9C-%EB%B0%8F-%EC%84%A4%EC%B9%98-%EB%B0%A9%EB%B2%95>

- Mac

- <https://velog.io/@kms9887/mysql-%EB%8B%A4%EC%9A%B4%EB%A1%9C%EB%93%9C-workbench>

이제 Local 에 Mysql DB 를 구축해 봅시다!



- 터미널 실행!

- Mysql -V

```
C:\Users\wtetz>mysql -V  
mysql Ver 8.0.28 for Win64 on x86_64 (MySQL Community Server - GPL)
```

- 이게 안 뜨면 시스템 환경 변수 설정
 - <https://dog-developers.tistory.com/21>



이제 Local 에 Mysql DB 를 구축해 봅시다!

- `mysql -u root -p`
- 설정한 비밀번호를 치고 들어가기

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 53
Server version: 8.0.28 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _
```


이제 Local 에 Mysql DB 를 구축해 봅시다!



- show databases;

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mydb      |
| mysql     |
| performance_schema |
| sakila    |
| sys       |
| world     |
+-----+
7 rows in set (0.00 sec)
```

이제 Local 에 Mysql DB 를 구축해 봅시다!



- use sakila;
- show tables;

```
mysql> use sakila;
Database changed
mysql> show tables;
+-----+
| Tables_in_sakila |
+-----+
| actor              |
| actor_info         |
| address            |
| category           |
| city               |
| country            |
| customer           |
| customer_list      |
| film               |
| film_actor         |
| film_category      |
| film_list          |
| film_text          |
| inventory          |
| language           |
| nicer_but_slower_film_list |
| payment            |
| rental             |
| sales_by_film_category |
| sales_by_store     |
+-----+
```



여기서 부터는 자유롭게 Query 를 사용!

- 다만 아무래도 CLI 는 불편하기 때문에 GUI 를 사용하는게 좋겠죠?
- 명령어가 익숙하면 상관이 없지만, 그렇지 않다면 아무래도 Workbench 를 사용하는 편이 좋습니다!
- MySQL Workbench 실행



MySQL Workbench

Workbench 사용!



- 먼저 schema 선택 하기

Navigator

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE

- Dashboard
- Performance Reports
- Performance Schema Setup

Administration Schemas

Information

Table: **answer**

Columns:

- _id** int AI PK

Query 1

question answer ans

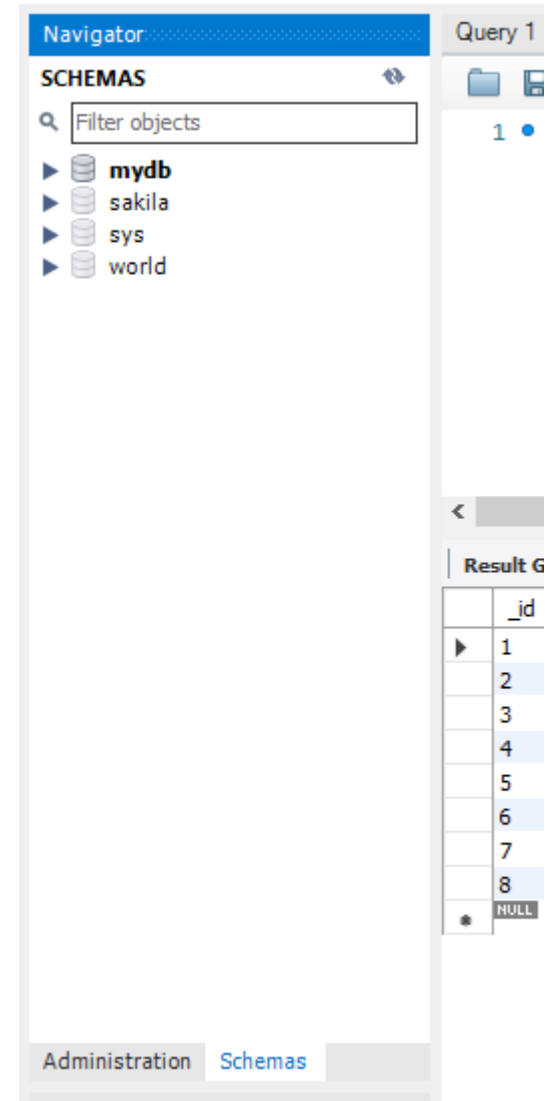
1 • **SELECT * FROM mydb.answer**

Result Grid

	_id	question_id	answer_text
▶	1	0	그런 모임을 왜 이제서
	2	0	1년 전에 알려줬어도
	3	1	원소리여, 그냥 하던
	4	1	오호? 그런게 있어? 들
	5	2	무슨 버그가 발생한 거
	6	2	아... 내일도 야근 각
	7	3	일단 빠르게 개발 완
	8	3	그거 내일 아침에 와
*	NULL	NULL	NULL

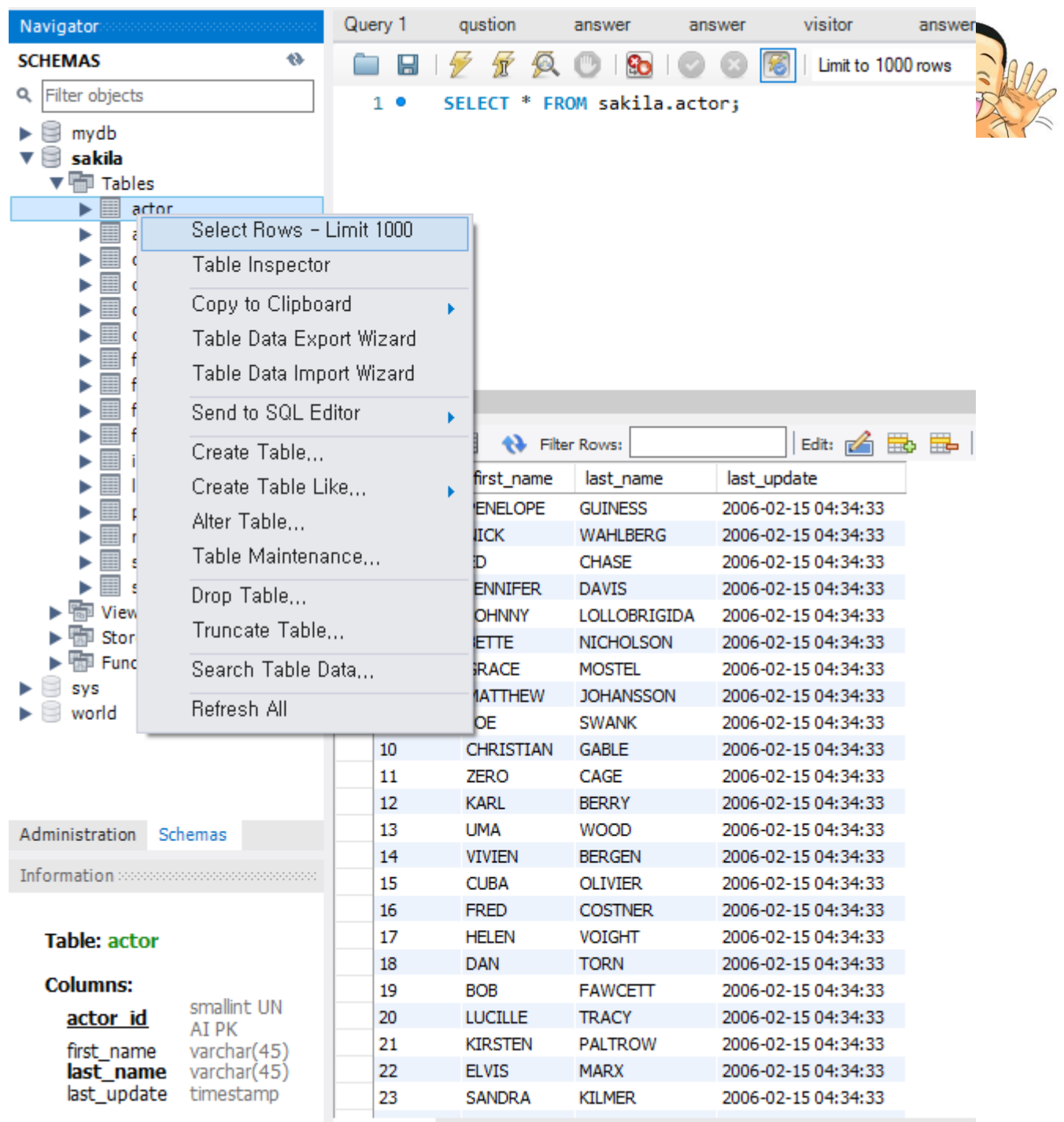
Schema

- 만들어진 DB 를 확인 할 수 있습니다!



Schema

- 각각의 DB 에 있는 테이블과 데이터 확인 가능



The screenshot displays a database management interface. On the left, the 'Navigator' pane shows the 'sakila' database schema with the 'actor' table selected. A context menu is open over the 'actor' table, listing various actions such as 'Select Rows - Limit 1000', 'Table Inspector', 'Copy to Clipboard', 'Table Data Export Wizard', 'Table Data Import Wizard', 'Send to SQL Editor', 'Create Table...', 'Create Table Like...', 'Alter Table...', 'Table Maintenance...', 'Drop Table...', 'Truncate Table...', 'Search Table Data...', and 'Refresh All'. The main pane shows the 'actor' table data, with columns 'first_name', 'last_name', and 'last_update'. The data is displayed in a table format, with rows numbered 1 to 23. The 'last_update' column shows a timestamp of '2006-02-15 04:34:33' for all rows.

	first_name	last_name	last_update
1	ENVELOPE	GUINNESS	2006-02-15 04:34:33
2	WICK	WAHLBERG	2006-02-15 04:34:33
3	D	CHASE	2006-02-15 04:34:33
4	ENNIFER	DAVIS	2006-02-15 04:34:33
5	JOHNNY	LOLLOBRIGIDA	2006-02-15 04:34:33
6	ETTE	NICHOLSON	2006-02-15 04:34:33
7	GRACE	MOSTEL	2006-02-15 04:34:33
8	MATTHEW	JOHANSSON	2006-02-15 04:34:33
9	OE	SWANK	2006-02-15 04:34:33
10	CHRISTIAN	GABLE	2006-02-15 04:34:33
11	ZERO	CAGE	2006-02-15 04:34:33
12	KARL	BERRY	2006-02-15 04:34:33
13	UMA	WOOD	2006-02-15 04:34:33
14	VIVIEN	BERGEN	2006-02-15 04:34:33
15	CUBA	OLIVIER	2006-02-15 04:34:33
16	FRED	COSTNER	2006-02-15 04:34:33
17	HELEN	VOIGHT	2006-02-15 04:34:33
18	DAN	TORN	2006-02-15 04:34:33
19	BOB	FAWCETT	2006-02-15 04:34:33
20	LUCILLE	TRACY	2006-02-15 04:34:33
21	KIRSTEN	PALTROW	2006-02-15 04:34:33
22	ELVIS	MARX	2006-02-15 04:34:33
23	SANDRA	KILMER	2006-02-15 04:34:33



수업을 위한 DB 만들기

DB 생성!



- `CREATE SCHEMA `mydb` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;`

```
Query 1 x
1 CREATE SCHEMA `myDB` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
2
```

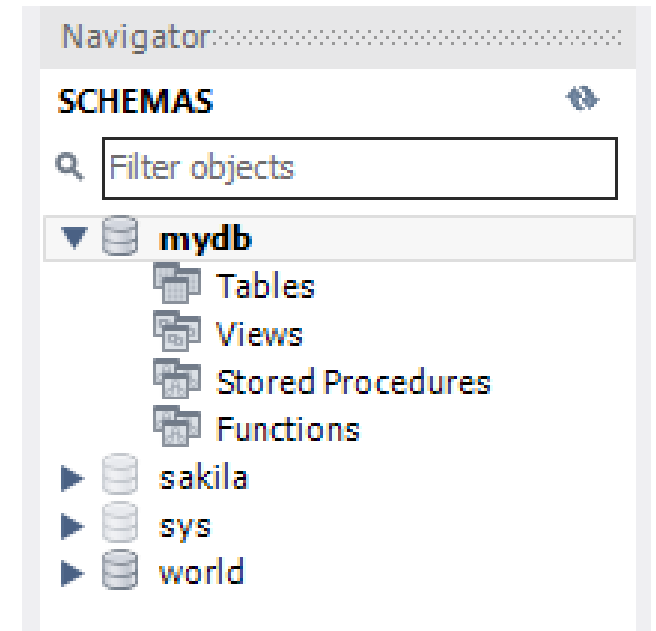




TABLE 생성!



3. 테이블 생성시 제약 넣기

```
CREATE TABLE people (  
  person_id INT AUTO_INCREMENT PRIMARY KEY,  
  person_name VARCHAR(10) NOT NULL,  
  nickname VARCHAR(10) UNIQUE NOT NULL,  
  age TINYINT UNSIGNED,  
  is_married TINYINT DEFAULT 0  
);
```

제약	설명
AUTO_INCREMENT	새 행 생성시마다 자동으로 1씩 증가
PRIMARY KEY	중복 입력 불가, NULL(빈 값) 불가
UNIQUE	중복 입력 불가
NOT NULL	NULL(빈 값) 입력 불가
UNSIGNED	(숫자일시) 양수만 가능
DEFAULT	값 입력이 없을 시 기본값

DB 의 TABLE 생성!



```
Query 1 x
[Icons: Folder, Save, Run, Undo, Redo, Stop, Refresh, Checkmark, Close, Execute] | Limit to 1000 rows | [Icons: Star, Eraser, Zoom In, Zoom Out, Split View, Full Screen]

1 CREATE TABLE user (
2     `ID_PK` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
3     `NAME` VARCHAR(100) NOT NULL,
4     `EMAIL` VARCHAR(100) NOT NULL UNIQUE,
5     `PASSWORD` VARCHAR(100) NOT NULL,
6     `ADDRESS` VARCHAR(100) NOT NULL,
7     `AGE` TINYINT UNSIGNED,
8     `MEMBERSHIP` TINYINT DEFAULT 0,
9     `REGISTER_DATE` DATETIME DEFAULT CURRENT_TIMESTAMP,
10    `UPDATE_DATE` DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
11 );
12
```



DATA 삽입하기



생성한 TABLE 에 DATA 삽입!

```
INSERT INTO user (NAME, EMAIL, PASSWORD, ADDRESS, AGE)  
VALUES ('이효석', 'tetz@spreatics', '1324', '서울 서대문구', 38);
```

[illegible]

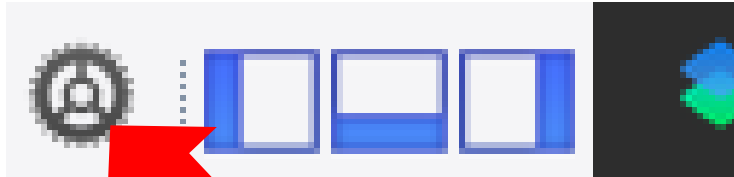


DATA 수정 및 삭제

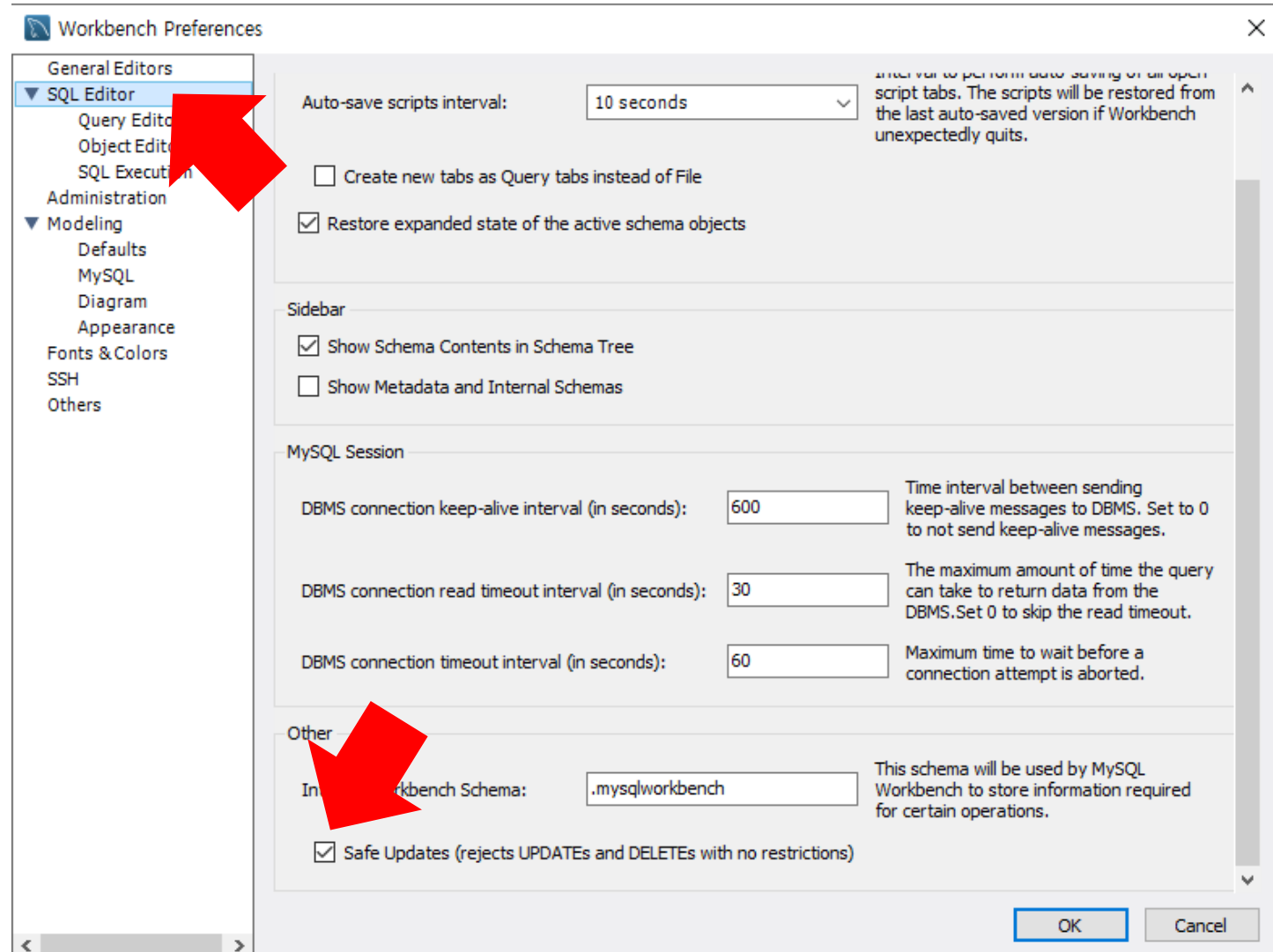


먼저 설정 변경이 필요합니다!

- 환경 설정 > SQL Editor



- 체크 박스를 해제





DATA 삭제



- ```
17 ● DELETE FROM user WHERE ID_PK = 3;
```

[illegible]



# DATA 수정



- ```
18 • UPDATE user SET AGE = AGE + 1 WHERE ID_PK = 1;
19
```

[illegible]



Table 수정 및 삭제하기



ALTER TABLE - 테이블 변경

```
-- 테이블명 변경
ALTER TABLE people RENAME TO friends,
-- 컬럼 자료형 변경
CHANGE COLUMN person_id person_id TINYINT,
-- 컬럼명 변경
CHANGE COLUMN person_name person_nickname VARCHAR(10),
-- 컬럼 삭제
DROP COLUMN birthday,
-- 컬럼 추가
ADD COLUMN is_married TINYINT AFTER age;
```

DROP TABLE - 테이블 삭제

```
DROP TABLE friends;
```



WorkBench 로
수행하기!



TABLE 생성!



Navigator

SCHEMAS

Filter objects

- mydb
 - Tables
 - answer
 - explanation
 - new_table
 - Columns
 - Indexes
 - Foreign Keys
 - Triggers
 - question
 - timestamps
 - visitor
 - Views
 - Stored Procedures
 - Functions
- sakila
- sys
- world

Administration Schemas

Information

Schema: mydb

question answer answer visitor answer actor new_table timestamps new_table - Table x

Table Name: Schema: **mydb**

Charset/Collation: Engine:

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Column Name: Data Type:

Charset/Collation: Default:

Comments:

Storage: ☐ Virtual ☐ Stored

☐ Primary Key ☐ Not Null ☐ Unique

☐ Binary ☐ Unsigned ☐ Zero Fill

☐ Auto Increment ☐ Generated

Columns Indexes Foreign Keys Triggers Partitioning Options

Apply Revert





TABLE 수정!



Navigator

SCHEMAS

Filter objects

mydb

Tables

- answer
- explain
- new_t
- Co
- In
- For
- Tri
- questi
- timest
- visitor

Views

Stored Pro

Functions

sakila

sys

world

Administration

Sch

Information

question

answer

ansi

Table Name

Charset/Collatio

Comments:

Select Rows - Limit 1000

Table Inspector

Copy to Clipboard

Table Data Export Wizard

Table Data Import Wizard

Send to SQL Editor

Create Table...

Create Table Like...

Alter Table...

Table Maintenance...

Drop Table...

Truncate Table...

Search Table Data...

Refresh All



Data 수정 및 삭제



Result Grid									
Filter Rows: <input type="text"/>									
Edit:									
Export/Import:									
Wrap Cell Content: <input type="checkbox"/>									
	ID_PK	NAME	EMAIL	PASSWORD	ADDRESS	AGE	MEMBERSHIP	REGISTER_DATE	UPDATE_DATE
	1	이호	tetz@spreatics	1324	서울 서대문구	39	0	2022-11-23 17:03:19	2022-11-23 17:07:25
	5	이호석	tett@spreatics	1324	서울 서대문구	38	0	2022-11-23 17:05:23	2022-11-23 17:05:23
	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Result Grid									
Filter Rows: <input type="text"/>									
Edit:									
Export/Import:									
Wrap Cell Content: <input type="checkbox"/>									
	ID_PK	NAME	EMAIL	PASSWORD	ADDRESS	AGE	MEMBERSHIP	REGISTER_DATE	UPDATE_DATE
	1	이호	tetz@spreatics	1324	서울 서대문구	39	0	2022-11-23 17:03:19	2022-11-23 17:07:25
	5	이호석	tett@spreatics	1324	서울 서대문구	38	0	2022-11-23 17:05:23	2022-11-23 17:05:23
	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Result Grid

user 7

Apply

Revert



Review SQL Script

Apply SQL Script

Review the SQL Script to be Applied on the Database

```
1 UPDATE `mydb`.`user` SET `NAME` = '이효' WHERE (`ID_PK` = '1');
2
```



Back

Apply

Cancel



Review SQL Script

Apply SQL Script

Applying SQL script to the database

The following tasks will now be executed. Please monitor the execution.
Press Show Logs to see the execution logs.

☒ Execute SQL Statements

SQL script was successfully applied to the database.

Show Logs

Back

Finish

Cancel

