

Hello,

KDT 웹 개발자 양성 프로젝트

5기!

with





중대장은...

감동했다!!

이름: 이효석(39)

직책: 강사

주요업무: 음주







MySQL Workbench

Workbench 사용!



- 먼저 schema 선택 하기

Query 1

```
SELECT * FROM mydb.answer
```

_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

Administration Schemas

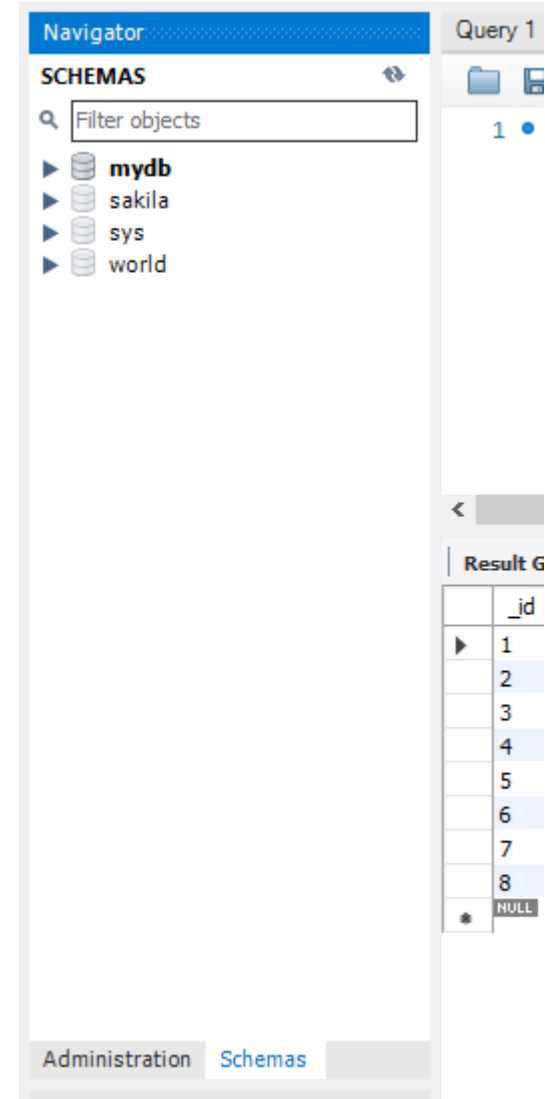
Information

Table: **answer**

Columns:
_id int AI PK

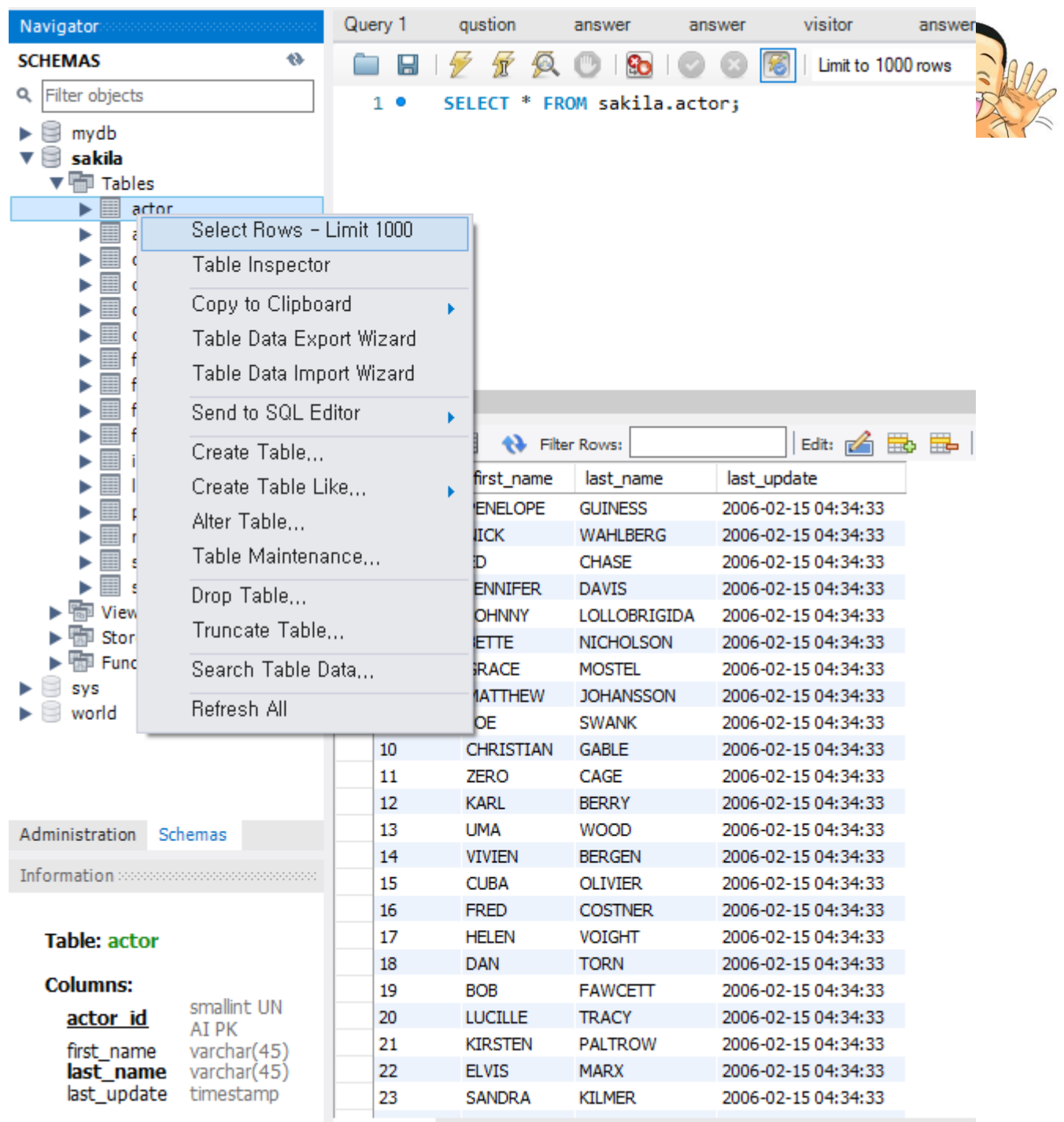
Schema

- 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, last_update. The data is displayed in a table format, showing 23 rows of actor information.

	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	ED	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



정규화



정규화?

- DB 설계에 있어서 중복을 최소화 하기 위해 데이터를 구조화 하는 과정
- 크고 조직화 되지 않은 테이블 → 작고, 잘 조직된 테이블로 변경
- 이렇게 작성을 해야만, 데이터 추가 및 삭제 시에 이상 현상(Abnormal)을 예방 할 수 있습니다!



제 1 정규형(1NF)

- 하나의 컬럼은 반드시 하나의 속성만을 가져야 하는 법칙

회원번호	회원이름	프로그램
101	강호동	스쿼시초급
102	손흥민	헬스
103	김민수	헬스, <u>골프초급</u>

회원번호	회원이름	프로그램
101	강호동	스쿼시초급
102	손흥민	헬스
103	김민수	헬스
103	김민수	골프초급



제 2 정규형(2NF)

- 모든 컬럼에 대한 부분 종속이 없어야 한다 → 현 테이블의 주제(?)와 필요 없는 친구는 빼준다

회원번호	회원이름	프로그램	가격	납부여부
101	강호동	스쿼시초급	5000	0
102	손흥민	헬스	6000	1
103	김민수	헬스	6000	1
103	김민수	골프초급	8000	0

수강등록현황 table				프로그램 table	
회원번호	회원이름	프로그램	납부여부	프로그램	가격
101	강호동	스쿼시초급	0	스쿼시초급	5000
102	손흥민	헬스	1	헬스	6000
103	김민수	헬스	1	골프초급	8000
103	김민수	골프초급	0		



제 3 정규형(3NF)

- 이행 종속성($A = B$, $B = C$ 여서 $A = C$ 인 경우)이 없어야 한다
- 일반 컬럼에만 종속된 컬럼은 다른 테이블로 빼기

프로그램	가격	강사	출신대학
스쿼시	5000	김을용	서울대
헬스	6000	박덕팔	연세대
골프	8000	이상구	고려대
골프중급	9000	이상구	고려대
개인피티	6000	박덕팔	연세대

프로그램	가격	강사	강사	출신대학
스쿼시	5000	김을용	김을용	서울대
헬스	6000	박덕팔	박덕팔	연세대
골프	8000	이상구	이상구	고려대
골프중급	9000	이상구	이상구	고려대
개인피티	6000	박덕팔	박덕팔	고려대



Foreign Key

(외래 키)



기본키(Primary Key)

외래키(Foreign Key)

기본키(Primary Key)

선수번호	이름	팀 코드	포지션	등번호	키
1	김남일	K03	DF	33	177
2	박지성	K07	MF	7	178
3	이영표	K02	MF	22	176

〈선수 테이블〉

팀 코드	팀 명	연고지
K03	스틸러스	포항
K07	드래곤즈	전남
K02	블루윙즈	수원

〈구단 테이블〉

선수번호	이름	팀 코드	포지션	등번호	키	팀 명	연고지
1	김남일	K03	DF	33	177	스틸러스	포항
2	박지성	K07	MF	7	178	드래곤즈	전남
3	이영표	K02	MF	22	176	블루윙즈	수원



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
```

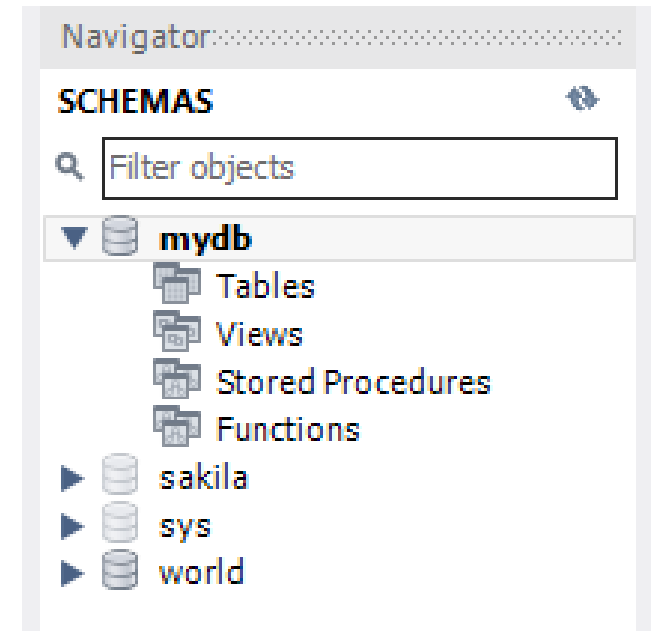




TABLE 생성!

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);
```



DATA 수정 및 삭제



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

[illegible]



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

[illegible]



Table 수정 및 삭제하기



ALTER TABLE - 테이블 변경

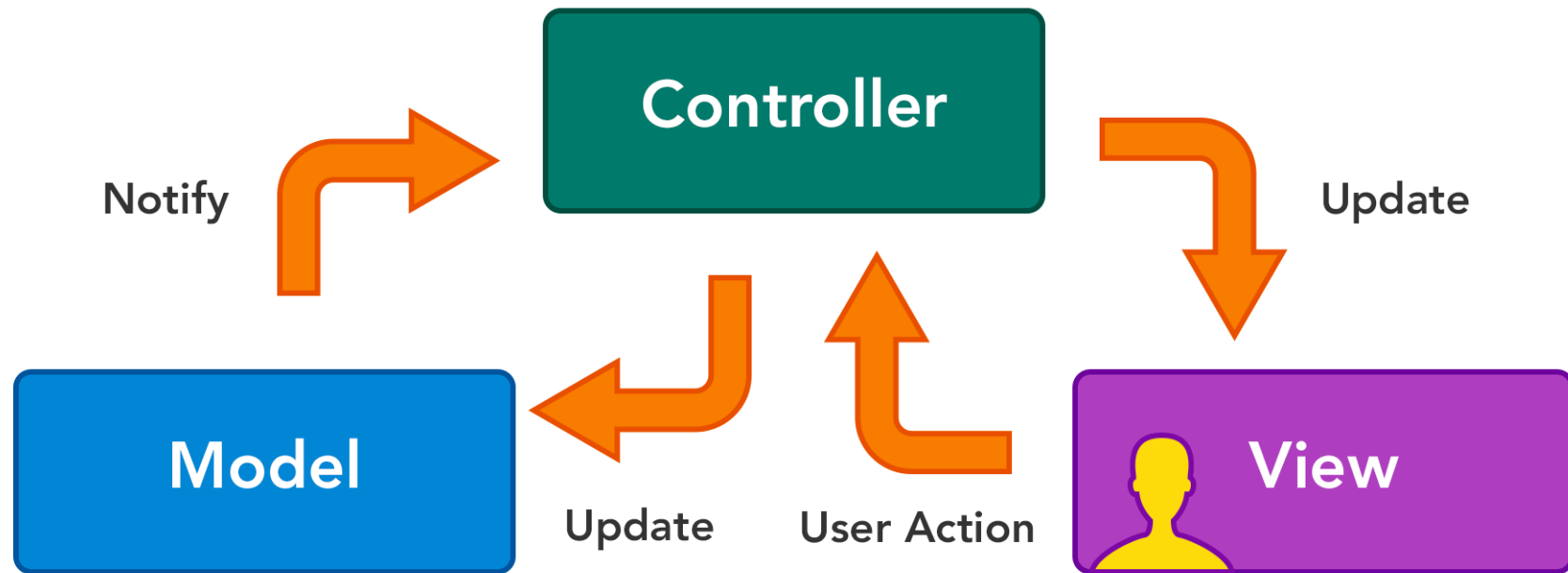
```
-- 테이블명 변경
ALTER TABLE people RENAME TO friends,
-- 컬럼 자료형 변경
CHANGE COLUMN person_id person_id TINYINT,
-- 컬럼명 변경
CHANGE COLUMN person_name person_nickname VARCHAR(10),
-- 컬럼 삭제
DROP COLUMN birthday,
-- 컬럼 추가
ADD COLUMN is_married TINYINT AFTER age;
```

DROP TABLE - 테이블 삭제

```
DROP TABLE friends;
```



MVC 패턴





DB 통신용 서버 구축하기



DB 통신을 하는 Back 서버를 구축

- 이제 DB 통신을 하는 Backend 서버를 구축하고 해당 서버로 부터 데이터를 받아 봅시다!
- 백엔드 폴더에 DB를 컨트롤 하는 controllers 폴더 생성

```
> controllers  
> node_modules  
> public  
> routes  
> views  
⊙ .eslintrc.js  
◆ .gitattributes
```



MySQL 서버 통신용 모듈 작성

- dbConnect.js 라는 DB 통신용 모듈을 작성해 봅시다!

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

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '비밀번호',
  port: '3306',
  database: 'mydb',
});
```

```
connection.connect();
```

```
module.exports = connection;
```

controllers/dbConnect.js



DB 통신용

컨트롤러 작성



```
const connection = require('./dbConnect');

const db = {
  getUsers: (cb) => {
    connection.query('SELECT * FROM mydb.user;', (err, data) => {
      if (err) throw err;
      console.log(data);
      cb(data);
    });
  },
};
```

```
module.exports = db;
```

contorollers/dataController.js



DB 통신용 라우터 생성



DB 통신 라우터 생성

- Routes 폴더에 data.js 생성

```
const express = require('express');
const db = require('../controllers/userController');

const router = express.Router();

router.get('/', (req, res) => {
  db.getUsers((data) => {
    res.send(data);
  });
});

module.exports = router;
```

routes/data.js



메인 서버에 라우터 등록

```
const mainRouter = require('./routes');  
const userRouter = require('./routes/users');  
const boardRouter = require('./routes/board');  
const dbRouter = require('./routes/db');
```

```
app.use('/', mainRouter);  
app.use('/users', userRouter);  
app.use('/board', boardRouter);  
app.use('/db', dbRouter);
```



POSTMAN 으로

테스트!

localhost:4000/data



GET



localhost:4000/data

```
1  [
2  {
3    "ID_PK": 1,
4    "NAME": "이호",
5    "EMAIL": "tetz@spreatics",
6    "PASSWORD": "1324",
7    "ADDRESS": "서울 서대문구",
8    "AGE": 39,
9    "MEMBERSHIP": 0,
10   "REGISTER_DATE": "2022-11-23T08:03:19.000Z",
11   "UPDATE_DATE": "2022-11-23T08:09:42.000Z"
12 },
13 {
14   "ID_PK": 5,
15   "NAME": "이호석",
16   "EMAIL": "tett@spreatics",
17   "PASSWORD": "1324",
18   "ADDRESS": "서울 서대문구",
19   "AGE": 38,
20   "MEMBERSHIP": 0,
21   "REGISTER_DATE": "2022-11-23T08:05:23.000Z",
22   "UPDATE_DATE": "2022-11-23T08:05:23.000Z"
23 }
```



지금 시작합니다



게시판에 DB 적용하기



게시판 서비스를 위한 TABLE 생성

```
1 • ○ CREATE TABLE board (  
2     `ID_PK` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,  
3     `TITLE` VARCHAR(100) NOT NULL,  
4     `CONTENT` VARCHAR(300) NOT NULL,  
5     `REGISTER_DATE` DATETIME DEFAULT CURRENT_TIMESTAMP,  
6     `UPDATE_DATE` DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP  
7 ) ;
```




테스트를 위한 기본 데이터 삽입

- 9 • `INSERT INTO board (TITLE, CONTENT) VALUES ('제목1', '테스트 컨텐츠 입니다!');`
- 10 • `INSERT INTO board (TITLE, CONTENT) VALUES ('제목1', '테스트 컨텐츠 입니다!');`

Result Grid					
		Filter Rows:		Edit:	
				Export/Import:	
	ID_PK	TITLE	CONTENT	REGISTER_DATE	UPDATE_DATE
▶	1	제목1	테스트 컨텐츠 입니다!	2022-11-24 14:27:39	2022-11-24 14:27:39
	2	제목2	테스트 컨텐츠 입니다!	2022-11-24 14:27:41	2022-11-24 14:27:41
✱	NULL	NULL	NULL	NULL	NULL



DB 버전

게시판 작업 시작!



게시판 용 컨트롤러 작성

- Controllers 폴더에 boardController.js 작성

```
const connection = require('./dbConnect');

const boardDB = {
  getAllArticles: (cb) => {
    connection.query('SELECT * FROM mydb.board;', (err, data) => {
      if (err) throw err;
      console.log(data);
      cb(data);
    });
  },
};
```

```
module.exports = boardDB;
```

Controllers/boardController.js



dbBoard.js 라우터 만들기

- routes 폴더에 dbBoard.js 라우터 작성

```
const express = require('express');
const boardDB = require('../controllers/boardController');

const router = express.Router();

router.get('/getAll', (req, res) => {
  boardDB.getAllArticles((data) => {
    res.send(data);
  });
});

module.exports = router;
```

routes/dbBoard.js

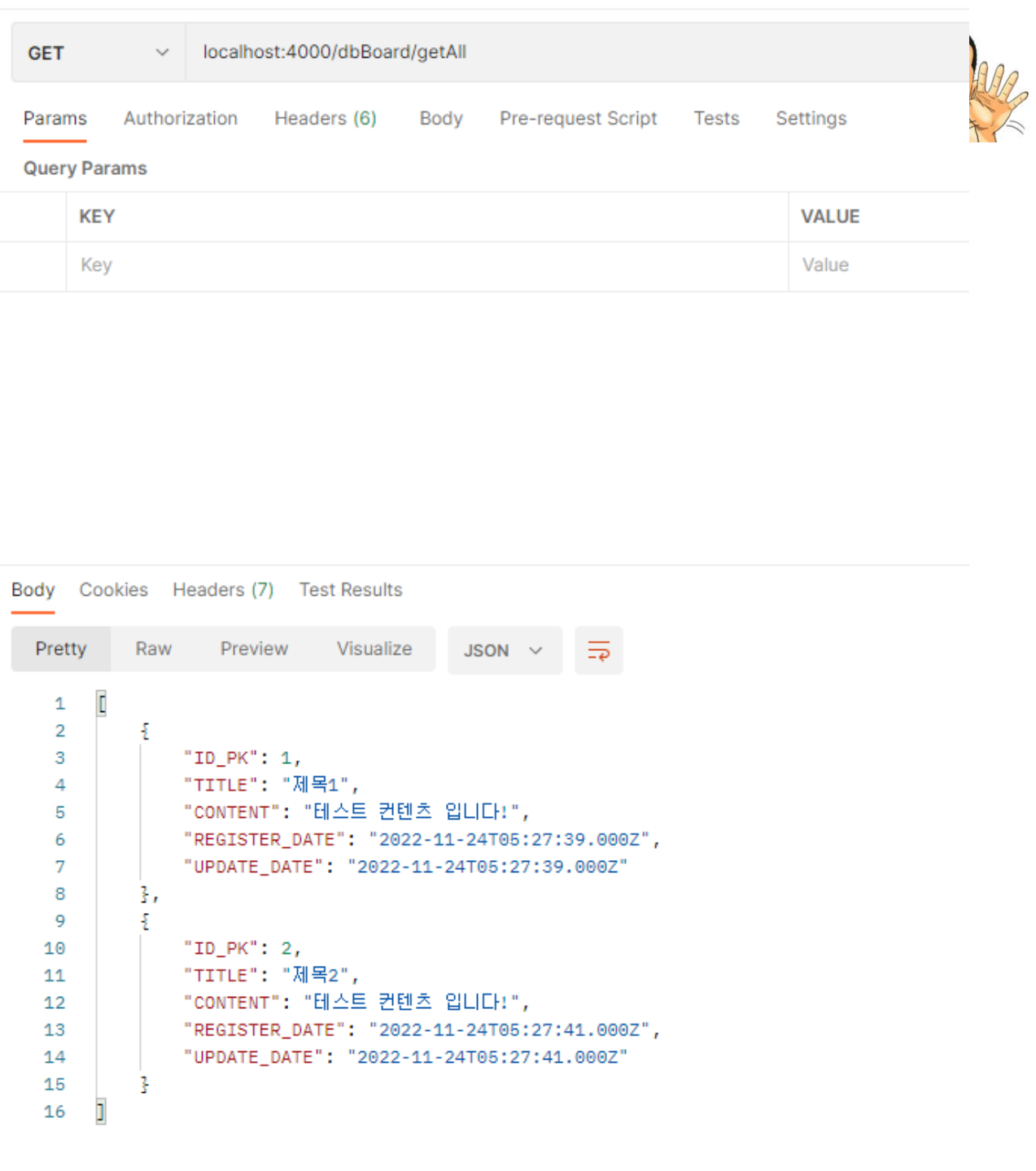


메인 서버에 라우터 등록

```
const mainRouter = require('./routes');
const userRouter = require('./routes/users');
const boardRouter = require('./routes/board');
const dbRouter = require('./routes/db');
const dbBoardRouter = require('./routes/dbBoard');

app.use('/', mainRouter);
app.use('/users', userRouter);
app.use('/board', boardRouter);
app.use('/db', dbRouter);
app.use('/dbBoard', dbBoardRouter);
```

포스트 맨으로 테스트!



GET localhost:4000/dbBoard/getAll

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "ID_PK": 1,
4     "TITLE": "제목1",
5     "CONTENT": "테스트 컨텐츠 입니다!",
6     "REGISTER_DATE": "2022-11-24T05:27:39.000Z",
7     "UPDATE_DATE": "2022-11-24T05:27:39.000Z"
8   },
9   {
10    "ID_PK": 2,
11    "TITLE": "제목2",
12    "CONTENT": "테스트 컨텐츠 입니다!",
13    "REGISTER_DATE": "2022-11-24T05:27:41.000Z",
14    "UPDATE_DATE": "2022-11-24T05:27:41.000Z"
15  }
16 }
```





DB에서 데이터를
받아서 EJS에 출력



기본 페이지 작업!

- MySQL 로 부터 데이터를 받아서 EJS 에 전달하는 라우터 작성

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

routes/dbBoard.js



Tetz Board

현재 등록 글 : 2

글쓰기

수정

삭제

수정

삭제





DB로 부터 받아오는 데이터 찍어보기!

- DB에서 데이터가 잘 들어오는지 확인해 봅시다!

```
router.get('/', (req, res) => {  
  boardDB.getAllArticles((data) => {  
    console.log(data);  
    const ARTICLE = data;  
    const articleCounts = ARTICLE.length;  
    res.render('db_board', { ARTICLE, articleCounts });  
  });  
});
```

routes/dbBoard.js



DB로 부터 받아오는 데이터 찍어보기!

- 아하! 키가 컬럼 명을 그대로 받아오기 때문에 대문자로 변경이 되었군요!

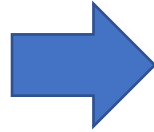
```
[
  RowDataPacket {
    ID_PK: 1,
    TITLE: '제목1',
    CONTENT: '테스트 콘텐츠 입니다!',
    REGISTER_DATE: 2022-11-24T05:27:39.000Z,
    UPDATE_DATE: 2022-11-24T05:27:39.000Z
  },
  RowDataPacket {
    ID_PK: 2,
    TITLE: '제목2',
    CONTENT: '테스트 콘텐츠 입니다!',
    REGISTER_DATE: 2022-11-24T05:27:41.000Z,
    UPDATE_DATE: 2022-11-24T05:27:41.000Z
  }
]
```



EJS 파일 변경

- EJS 파일을 변경!

```
<div class="title">
  <%= ARTICLE[i].title %>
</div>
<div class="content">
  <%= ARTICLE[i].content %>
</div>
```



```
<div class="title">
  <%= ARTICLE[i].TITLE %>
</div>
<div class="content">
  <%= ARTICLE[i].CONTENT %>
</div>
```

Tetz Board



현재 등록 글 : 2

글쓰기

제목1

테스트 콘텐츠 입니다!

수정

삭제

제목2

테스트 콘텐츠 입니다!

수정

삭제





글 쓰기 기능

구현



글 쓰기용 EJS 파일 생성!

- 기존 board_write.ejs 파일을 재사용 하기! → dbBoard_write.ejs
- Form 태그의 action 주소를 dbBoard 용으로 변경!

```
<form action="/dbBoard/write" method="POST" class="board_form">
  <div class="form_title">
    <h3>제목</h3>
    <input type="text" name="title" required />
```



글 쓰기 모드로 이동하는 주소 라우팅

- GET 방식 dbBoard/write 라는 주소 요청이 들어오면 글 쓰기 페이지로 이동하는 라우터 만들기!

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

routes/dbBoard.js



Form 태그 데이터 받기



Action 으로 보낸 주소 라우터 만들기

- 해당 주소 값으로 form 태그 데이터가 잘 들어오는지 확인

```
router.post('/write', (req, res) => {  
  console.log(req.body);  
});
```

routes/dbBoard.js

서버는 4000번에서 실행 중입니다!

```
[Object: null prototype] { title: '11', content: '11' }
```

새로운 글을 DB에 등록하는 컨트롤러 만들기!



- 데이터는 잘 들어오므로 해당 데이터를 DB에 등록하는 컨트롤러 작업!

```
writeArticle: (newArticle, cb) => {  
  connection.query(  
    `INSERT INTO mydb.board (TITLE, CONTENT) VALUES ('${newArticle.title}', '${newArticle.content}')`,  
    (err, data) => {  
      if (err) throw err;  
      cb(data);  
    }  
  );  
},
```

Controllers/boardContoroller.js

컨트롤러를 사용하도록 라우터 수정



- 기존 DB 통신 결과는 DATA 가 들어오는지 여부를 보면 되었지만, 글 쓰기 결과는 다른 데이터를 반환 합니다!
- 여기서는 affectedRows 값을 통해 성공 여부를 확인

```
OkPacket {  
  fieldCount: 0,  
  affectedRows: 1,  
  insertId: 18,  
  serverStatus: 2,  
  warningCount: 0,  
  message: '',  
  protocol41: true,  
  changedRows: 0  
}
```



컨트롤러를 사용하도록 라우터 수정

```
router.post('/write', (req, res) => {  
  if (req.body.title && req.body.content) {  
    boardDB.writeArticle(req.body, (data) => {  
      console.log(data);  
      if (data.affectedRows >= 1) {  
        res.redirect('/dbBoard');  
      } else {  
        const err = new Error('글 쓰기 실패');  
        throw err;  
      }  
    });  
  } else {  
    const err = new Error('글 제목 또는 내용이 없습니다!');  
    throw err;  
  }  
});
```

routes/dbBoard.js



제목2

테스트 콘텐츠 입니다!

수정

삭제

aa

aa

수정

삭제

CC

cc

수정

삭제



글 수정하기 기능

구현



글 수정 용 EJS 파일 생성!

- 기존 board_modify.ejs 파일을 재사용! → dbBoard_modify.ejs
- Form 태그의 action 주소를 dbBoard 용으로 변경!
- title 과 content 도 대문자로 변경
- 그리고 제목이 겹쳐도 이제는 유니크한 값인 ID_PK 값을 이용하여 글을 수정할 겁니다! → 즉 같은 제목의 글도 구분이 가능!



```
<form action="/dbBoard/modify/<%= selectedArticle.ID_PK %>" 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>
```



글 수정 모드로 이동하는 주소 라우팅

- GET 방식 dbBoard/modify 라는 주소 요청이 들어오면 글 쓰기 페이지로 이동하는 라우터 만들기!

```
router.get('/modify/:id', (req, res) => {  
  res.render('dbBoard_modify');  
});
```

routes/dbBoard.js



글 수정 모드로 이동하는 주소 라우팅

- 하지만 아직은 작동을 안 할 겁니다!
- 특정 글의 내용을 전달 해야 하는데, 아직 DB 내에서 특정 게시글을 찾아 주는 컨트롤러 기능이 없습니다!
- 그럼 만들러 가시죠!

ID_PK 값으로 특정 글을 찾는 컨트롤러 만들기



```
getArticle: (id, cb) => {  
  connection.query(  
    `SELECT * FROM mydb.board WHERE ID_PK = ${id};`,  
    (err, data) => {  
      if (err) throw err;  
      cb(data);  
    }  
  );  
},
```

Controllers/boardContoroller.js



글 수정 모드로 이동 라우터 변경!

```
router.get('/modify/:id', (req, res) => {  
  boardDB.getArticle(req.params.id, (data) => {  
    if (data.length > 0) {  
      res.render('dbBoard_modify', { selectedArticle: data[0] });  
    }  
  });  
});  
});
```

routes/dbBoard.js



게시글 목록 페이지에서 수정 모드로 이동

```
<div class="foot">
  <a class="btn orange" href="/dbBoard/modify/<%= ARTICLE[i].ID_PK %>">수정</a>
  <a class="btn blue" href="#" onclick="deleteArticle('<%= ARTICLE[i].TITLE %>')">삭제</a>
</div>
```

- 게시글은 이제 ID_PK 값으로 찾으므로 전달하는 값을 제목에서 ID_PK로 수정!
- 데이터도 title, content 소문자에서 대문자로 변경!



Write Mode

제목

제목2

내용

테스트 콘텐츠 입니다!

글 수정하기

글 수정 페이지에서 전달 된 값으로 DB 수정!



- ID_PK 값을 받아서 해당 ID를 가지는 DB를 UPDATE 해주면 됩니다!
- 컨트롤러 작업!

```
modifyArticle: (id, modifyArticle, cb) => {  
  connection.query(  
    `UPDATE mydb.board SET TITLE = '${modifyArticle.title}', CONTENT =  
    '${modifyArticle.content}' WHERE ID_PK = ${id};`,  
    (err, data) => {  
      if (err) throw err;  
      cb(data);  
    }  
  );  
},
```

Controllers/boardContoroller.js



글을 수정하는 라우터 생성!

```
router.post('/modify/:id', (req, res) => {  
  if (req.body.title && req.body.content) {  
    db.modifyArticle(req.params.id, req.body, (data) => {  
      if (data.affectedRows >= 1) {  
        res.redirect('/dbBoard');  
      } else {  
        const err = new Error('글 수정 실패');  
        throw err;  
      }  
    });  
  } else {  
    const err = new Error('글 제목 또는 내용이 없습니다!');  
    throw err;  
  }  
});
```

routes/dbBoard.js



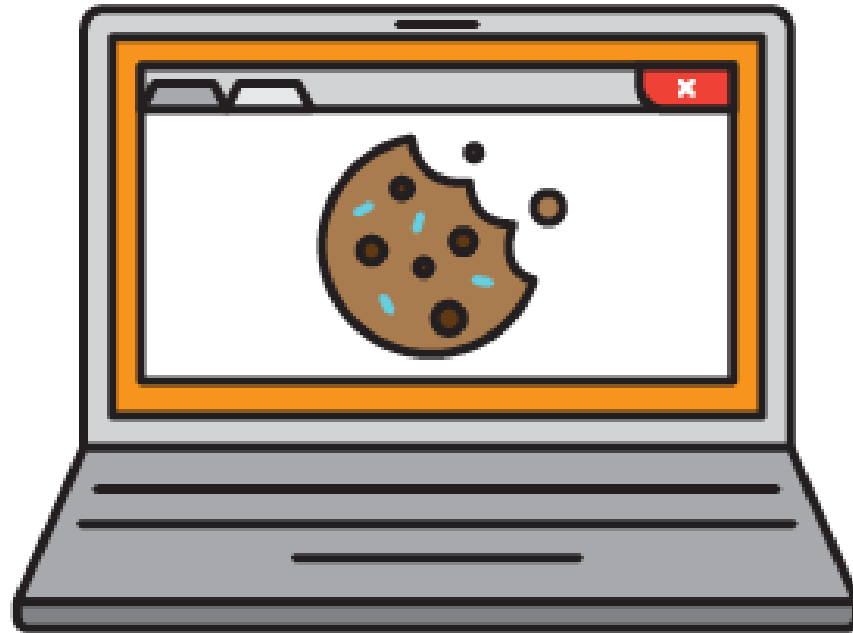


실습, 삭제 기능 구현하기!

- 삭제 버튼을 누르면 삭제 처리하시면 됩니다!
- 기존 처럼 제목을 이용해서 지우는 것이 아니라 ID_PK 를 사용해서 삭제 처리 해주세요! 😊



Cookie!?



WEBSITE
COOKIES



프론트에서 쿠키 사용하기



웹 브라우저

1. HTTP 요청



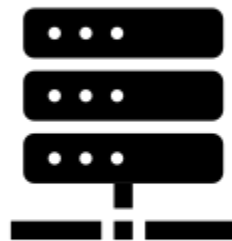
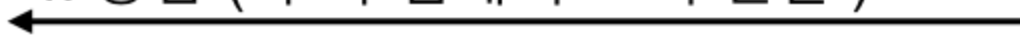
2. 쿠키를 HTTP 헤더(Set-Cookie)에 담아서 응답



3. 쿠키 저장 / HTTP 재요청 (쿠키 포함)



4. 응답 (쿠키 업데이트 시 전달)



웹 서버



프론트에서 쿠키 사용하기

- Document.cookie 로 바로 만들어서 사용하면 됩니다

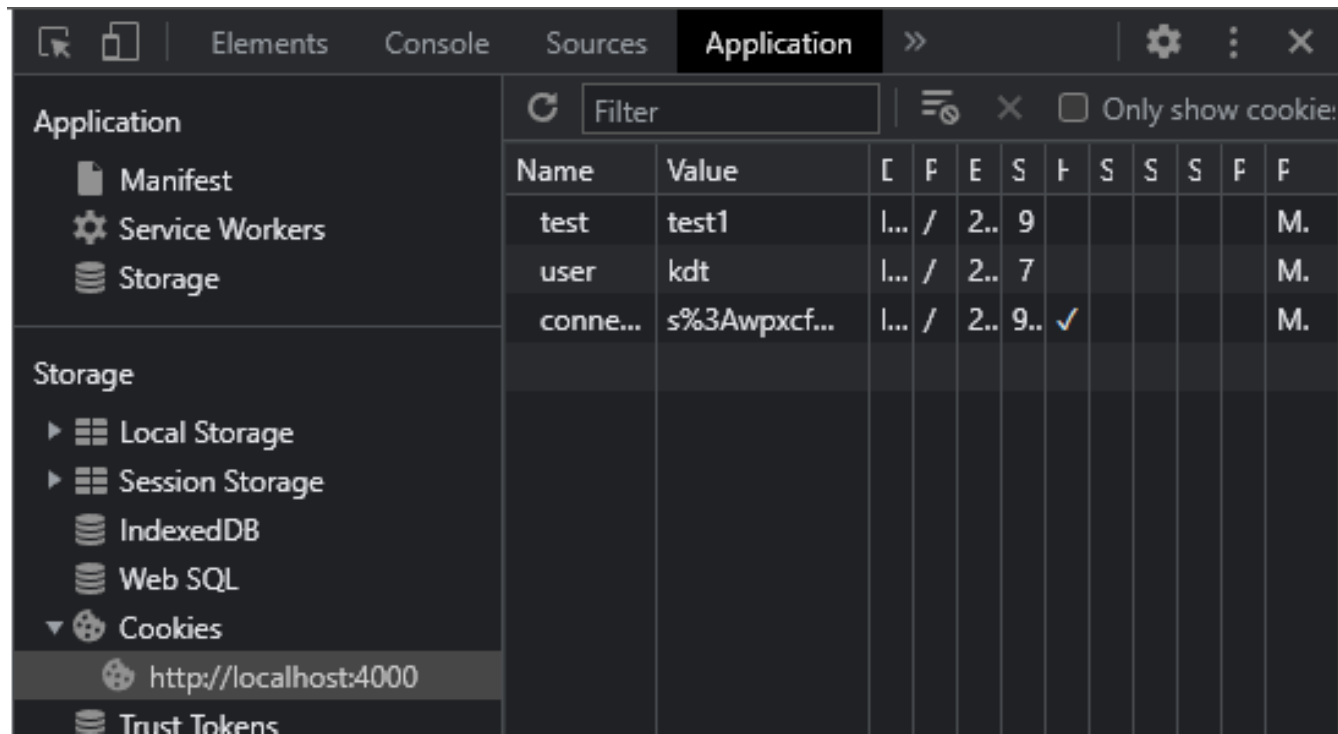
```
<script>  
  console.log(document.cookie);  
  document.cookie = "user=kdt; expires=26 Nov 2023 13:00:00 GMT; path="/";  
  document.cookie = "test=test1; expires=26 Nov 2023 13:00:00 GMT; path="/";  
  console.log(document.cookie);  
</script>
```

- Expires 값과 브라우저의 시간을 비교해서 쿠키가 해당 시간이 되면 자동으로 삭제 처리



브라우저에서 쿠키 확인하기

- 개발자 도구 → Application → Storage → Cookies 에 가면 쿠키 정보 확인이 가능합니다





백엔드에서 쿠키 사용하기



백엔드(express)에서 쿠키 사용하기

- 백엔드에서는 cookie-parser 라는 모듈을 사용합니다!
- 일단 설치 → `npm i cookie-parser -s`
- 메인 서버에 쿠키 모듈 호출 및 미들 웨어 등록!

```
const cookieParser = require('cookie-parser');  
app.use(cookieParser());
```



백엔드(express)에서 쿠키 사용하기

- 쿠키를 만드려면 `res.cookie('쿠키 이름', '데이터', '옵션 객체');` 를 써서 쿠키를 구우면 됩니다!

```
res.cookie('alert', true, {  
  expires: new Date(Date.now() + 1000 * 60),  
  httpOnly: true,  
});
```

- 옵션의 의미
 - Expires: 쿠키가 자동으로 삭제되는 일자를 지정
 - httpOnly: 해당 쿠키는 서버와의 http 통신에서만 읽을 수 있음을 표시, 프론트에서 처럼 JS 로 해당 쿠키를 읽으려 하면 웹브라우저가 이를 차단



실제로 쿠키가 잘 작동하는지 확인하기

- Index.ejs 에서 cookie가 서버에서 잘 전송 되었는지 확인해 봅시다!
- 먼저 프론트에서 쿠키를 발행하는 코드는 주석 처리 + 기존에 발행 된 쿠키도 삭제 처리

```
window.onload = () => {  
  console.log(document.cookie);  
}
```




실제로 쿠키가 잘 작동하는지 확인하기

- 엥? 안되는데요?
- 안되는 이유는 httpOnly 옵션 때문에 http 통신이 발생 했을 때에만 쿠키의 값에 접근이 가능합니다!
- 따라서, 프론트에서 document.cookie 의 값을 읽으려고 하는 것을 브라우저가 차단하기 때문입니다.
- httpOnly 옵션을 false 로 만들고 다시 해봅시다!



쿠키의 값을 전달

- 프론트에서 쿠키의 저장 된 값을 사용하려면 문자열을 이리저리 잘라서 사용해야 합니다.
- 반면 백엔드에서는 localStorage 처럼 사용이 가능합니다!
- 생성된 쿠키에 대한 접근은 req.cookies.**쿠키이름** 으로 하면 됩니다



```
router.get('/', (req, res) => {  
  res.cookie('alert', true, {  
    expires: new Date(Date.now() + 1000 * 60),  
    httpOnly: false,  
  });  
  console.log(req.cookies.alert);  
  console.log(req.cookies.tetz);  
  res.render('index');  
});
```

```
[nodemon] starting `node app.js`
```

```
서버는 4000번에서 실행 중입니다!
```

```
true
```

```
undefined
```

```
□
```



쿠키의 값을 전달

- 프론트에서 쿠키의 값을 잘라서 쓰기 싫다면?
- 백엔드에서 쿠키의 값을 구해서 프론트에 전달 하면 됩니다!

```
router.get('/', (req, res) => {  
  res.cookie('alert', true, {  
    expires: new Date(Date.now() + 1000 * 60),  
    httpOnly: true,  
  });  
  res.render('index', { alert: req.cookies.alert });  
});
```



쿠키의 값을 전달

```
<script>  
  if ('<%= alert %>' === 'true') alert(`쿠키 팔아요! 쿠키의 값은? ${document.cookie}`);  
</script>
```

- httpOnly 옵션이 켜져 있음에도 값을 전달 하였기 때문에 alert 창이 잘 뜨는 것을 볼 수 있습니다!
- 단, 쿠키 내용은 안뜹니다!



쿠키를 굽는
라우터 생성



특정 주소로 요청을 보내면 쿠키 굽기!

```
<a href="/cookie">쿠키 굽기</a>
```

```
router.get('/cookie', (req, res) => {  
  res.cookie('cookie', true, {  
    expires: new Date(Date.now() + 1000 * 5),  
    httpOnly: false,  
  });  
  res.send('쿠키 굽기 성공!');  
});
```

- 5초 뒤에 사라지나 확인해 봅시다!



쿠키가 있으면 alert 을 띄우기!

```
<script>
  if (<%= cookie %> === true) {
    alert('쿠키 있어요!');
  }
</script>
```

- 백엔드에서 cookie 라는 쿠키가 만들어져 있으면 값이 전달 될 것이므로 해당 값을 확인해서 alert 띄워주기!



쿠키가 있으면 alert 을 띄우기!

```
router.get('/', (req, res) => {  
  res.render('index', { cookie: req.cookies.cookie });  
});
```

- 백엔드에서 만들어진 cookie 는 req.cookies 에서 담겨 있으므로 cookie 라는 쿠키가 있으면 해당 쿠키의 값을 전달!



백엔트 쿠키 기능들!



쿠키의 최대 나이? 정하기!

- 쿠키의 생존은 expires 로 정해도 되지만 maxAge 로도 설정이 가능합니다!
- maxAge 는 쿠키의 생성 시간을 기준으로 밀리 세컨드 단위로 생존 시간을 결정합니다!

```
router.get('/cookie', (req, res) => {  
  res.cookie('cookie', true, {  
    maxAge: 1000 * 60,  
    httpOnly: false,  
  });  
  res.send('쿠키 굿기 성공!');  
});
```

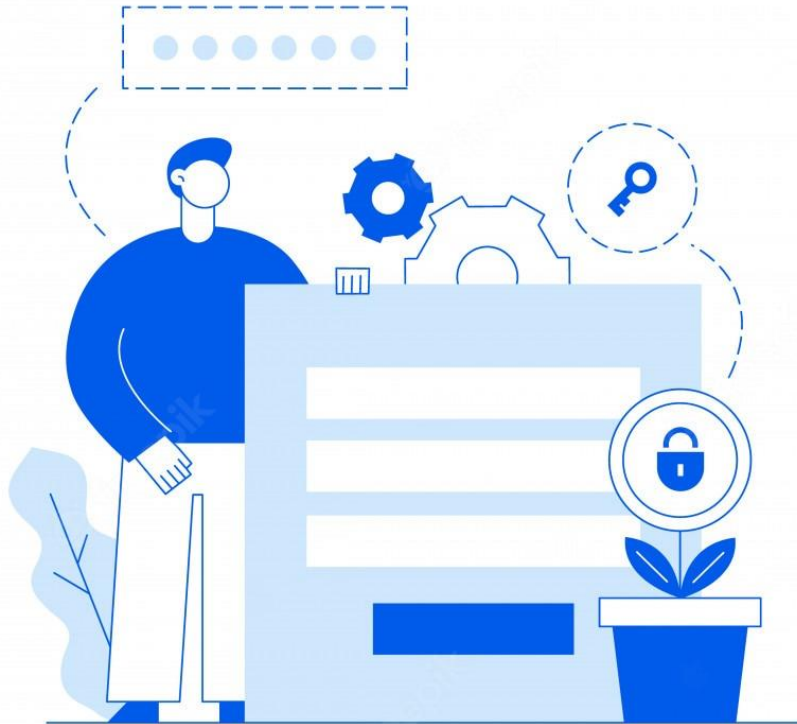


쿠키의 삭제하기!

- 쿠키를 삭제하는 방법은 `res.clearCookie('쿠키 이름')` 을 사용하면 됩니다!

```
router.get('/cookie', (req, res) => {  
  res.cookie('cookie', true, {  
    maxAge: 1000 * 60,  
    httpOnly: false,  
  });  
  res.clearCookie('cookie');  
  res.send('쿠키 굿기 성공!');  
});
```





Welcome!

 Your name

 Your e-mail

 Create password

Password strength



Create account

Sign in



**회원테이블
만들기!**



```
2 • CREATE TABLE user (  
3     `ID_PK` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,  
4     `USERID` VARCHAR(20) NOT NULL UNIQUE,  
5     `PASSWORD` VARCHAR(20) NOT NULL,  
6     `REGISTER_TIME` DATETIME DEFAULT CURRENT_TIMESTAMP,  
7     `UPDATE_TIME` DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP  
8 );
```

```
10 • SELECT * FROM user;
```

```
11 • INSERT INTO user (USERID, PASSWORD) VALUES ('tetz', '11');
```

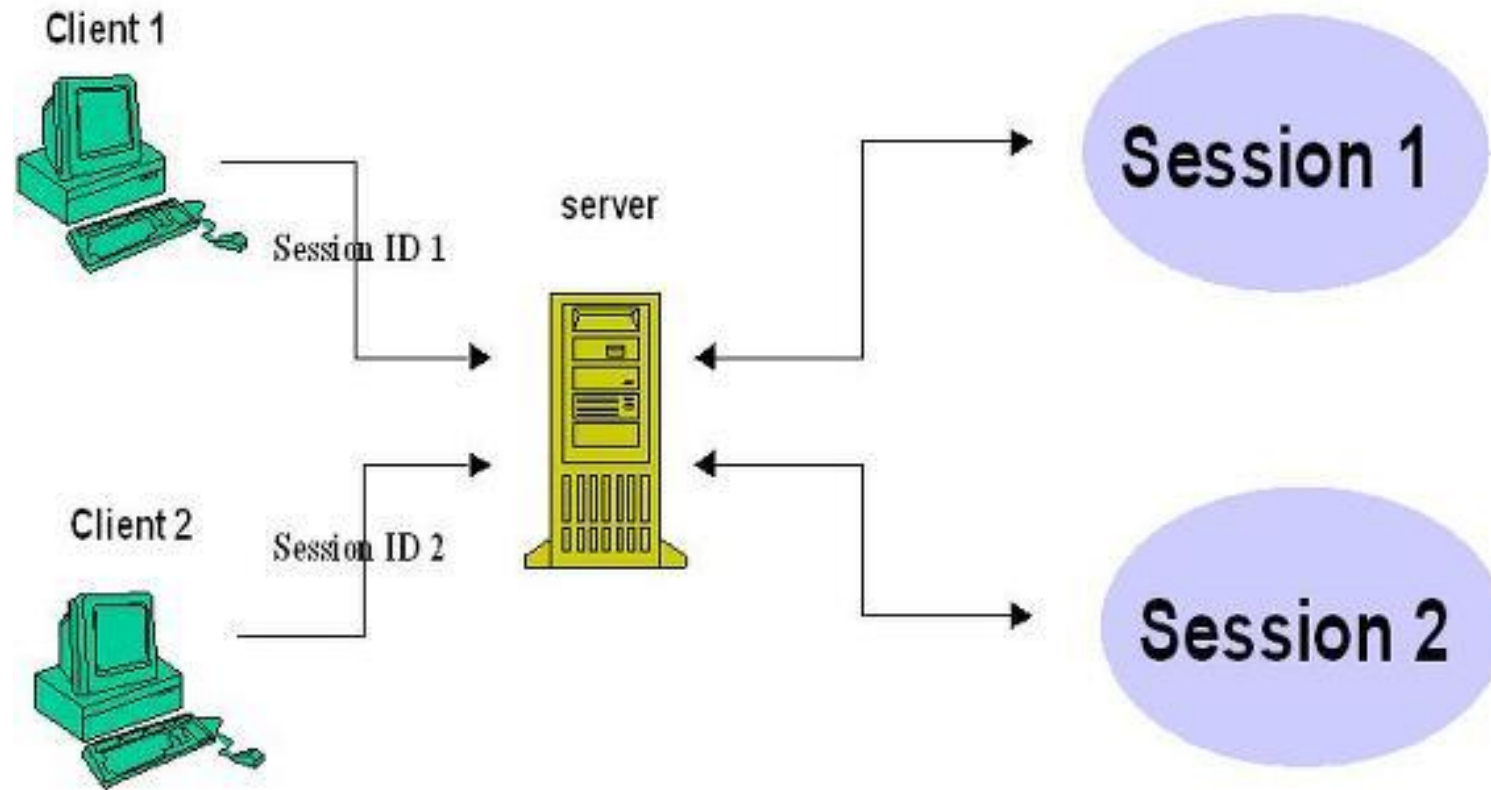
```
12 • INSERT INTO user (USERID, PASSWORD) VALUES ('pororo', '11');
```

	ID_PK	USERID	PASSWORD	REGISTER_TIME	UPDATE_TIME
▶	1	tetz	11	2022-11-26 09:28:54	2022-11-26 09:28:54
	3	pororo	11	2022-11-26 09:29:54	2022-11-26 09:29:54
●	NULL	NULL	NULL	NULL	NULL



Session

(서버의 쿠키)





HTTP Session 이란?

- 브라우저가 아닌 서버에 저장되는 쿠키
- 사용자가 서버에 접속한 시점부터 연결을 끝내는 시점을 하나의 상태로 보고 유지하는 기능을 함 → 로그인 유지
- 서버는 각 사용자에게 대한 세션을 발행하고 서버로 접근(Request)한 사용자를 식별하는 도구로 사용
- 쿠키와 달리 저장 데이터에 제한이 없음
- 만료 기간 설정이 가능하지만, 브라우저가 종료되면 바로 삭제



HTTP Session 의 동작 방식

- 사용자가 최초로 서버 연결을 하면 하나의 Session-ID(임의의 긴 문자열)가 발행 됩니다
- 발행 된 Session-ID 는 서버와 브라우저의 메모리에 쿠키 형태로 저장이 됩니다
- 서버는 사용자가 서버에 접근 시, 쿠키에 저장 된 Session-ID를 통해서 서버는 사용자를 구분하고 요청에 대한 응답을 합니다



Cookie vs Session

- 하는 역할은 비슷
- 쿠키는 로컬에 저장 되므로 보안 이슈가 발생 가능
- 세션은 로컬에 session-id 만 저장하고, 데이터는 서버에서 처리하므로 보안이 더 좋음
- 단, 쿠키는 데이터를 바로 저장하고 있으므로 속도가 빠름.
- 세션은 쿠키에서 session-id 를 읽어서 서버에서 데이터를 받아야 하므로 속도는 더 느림



Session 으로 로그인 구현하기



먼저 회원 가입, 로그인 페이지 만들기!

- 먼저 회원 가입, 로그인 페이지 부터 만들겠습니다
- 부트스트랩을 이용하여 간단하게 만든 페이지 입니다!
- login.ejs / register.ejs 파일로 추가!



회원 가입 페이지

https://github.com/xenosign/git_4th_backend/blob/main/views/login.ejs



로그인 페이지

https://github.com/xenosign/git_4th_backend/blob/main/views/posts.ejs



랜딩 페이지 변경



랜딩 페이지를 변경

- 세션 테스트를 위해 회원가입 / 로그인 / 게시판 서비스로만 이동이 가능 하도록 랜딩 페이지 변경

```
<div class="container mt-5">  
  <h1 class="text-center">Hello, Express Service</h1>  
  <h2 class="mt-2 text-center"><a href="/login">로그인 바로가기</a></h2>  
  <h2 class="mt-2 text-center"><a href="/register">회원가입 바로가기</a></h2>  
  <h2 class="mt-2 text-center"><a href="/dbBoard">게시판 바로가기</a></h2>  
</div>
```



Session

모듈 추가하기



세션 모듈 추가하기

- 먼저 express-session 모듈 부터 설치 합시다
 - npm i express-session
- 모듈 추가 및 미들웨어 연결

```
const session = require('express-session');
const app = express();
app.use(
  session({
    secret: 'tetz',
    resave: false,
    saveUninitialized: true,
    cookie: {
      maxAge: 1000 * 60 * 60,
    },
  })
);
```



세션 모듈 옵션 설명

- secret: 세션을 발급할 때 사용되는 키 값(아무거나 입력 가능)
- resave: 모든 request 마다 기존에 있던 session에 아무런 변경사항이 없어도 session 을 다시 저장하는 옵션
- saveUninitialized: 세션에 저장할 내역이 없더라도 처음부터 세션을 생성할지 설정
- secure → https 에서만 세션을 주고받을 수 있습니다. http 에서는 세션을 주고받는 것이 불가능
- cookie : 세션 쿠키 설정 (세션 관리 시 클라이언트에 보내는 쿠키)
 - maxAge: 쿠키의 생명 기간이고 단위는 ms입니다.
 - httpOnly → 자바스크립트를 통해서 세션을 사용할 수 없도록 강제



resister.js

구현하기



회원 가입 용 라우터 구현

- register.js 를 만들어서 회원가입 기능을 모듈화

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

const router = express.Router();

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

module.exports = router;
```

- 서버에 라우터 등록

```
const registerRouter = require('./routes/register');
app.use('/register', registerRouter);
```




Register.ejs 에서 데이터 받기

- POST 방식 /register 주소로 회원 가입을 요청하므로 처리
- 'uesr' 테이블에서 회원 데이터를 관리
- Req.body 에 담겨있는 ID 가 DB 에 존재하는지 확인하고, 존재할 경우 id 중복 안내 → 회원 가입 페이지로 이동
- ID가 중복되지 않을 경우, DB에 새로운 회원을 등록하고 회원 가입 성공 메시지를 출력 → 로그인 페이지로 이동



중복 회원 찾기 컨트롤러 작성

- User 테이블에서 중복 된 회원이 있는지 검사하는 컨트롤러 부터 만들어 봅시다!
- 중복이 없어야만 회원 가입이 가능하므로, 해당 기능을 만들고 난 뒤 실제 회원 가입 컨트롤러를 만들어 주시면 됩니다!

```
userCheck: (userId, cb) => {  
  userDB.query(  
    `SELECT * FROM mydb1.user WHERE USERID = '${userId}';`,  
    (err, data) => {  
      if (err) throw err;  
      console.log(data);  
      cb(data);  
    },  
  );  
},
```





회원 가입 컨트롤러 작성

- 중복이 없다면 이제 회원 가입을 하는 컨트롤러를 만들어 주시면 됩니다!

```
registerUser: (newUser, cb) => {  
  userDB.query(  
    `INSERT INTO mydb1.user (USERID, PASSWORD) VALUES ('${newUser.id}', '${newUser.password});`,  
    (err, data) => {  
      if (err) throw err;  
      console.log(data);  
      cb(data);  
    },  
  );  
},
```



회원 가입 라우터 작성

- 먼저 중복 여부를 체크한 다음, 중복 회원이 없을 경우 회원 가입을 진행 시키면 됩니다!



```
router.post('/', (req, res) => {
  userDB.userCheck(req.body.id, (data) => {
    if (data.length === 0) {
      userDB.registerUser(req.body, (result) => {
        if (result.protocol41) {
          res.send(
            '회원 가입 성공!<br><a href="/login">로그인 페이지로 이동</a>',
          );
        } else {
          res.status(404);
          res.send(
            '회원 가입 문제 발생.<br><a href="/register">회원가입 페이지로 이동</a>',
          );
        }
      });
    } else {
      res.send(
        '중복된 id 가 존재합니다.<br><a href="/register">회원가입 페이지로 이동</a>',
      );
    }
  });
});
```



login.js

구현하기



로그인 용 라우터 구현

- login.js 를 만들어서 로그인 기능을 모듈화

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

router.get('/', async (req, res) => {
  res.render('login');
});

module.exports = router;
```

- 서버에 라우터 등록

```
const loginRouter = require('./routes/login');
app.use('/login', loginRouter);
```




로그인 구현



로그인 처리

- 먼저 입력 받은 ID 가 DB 에 존재 하는지를 체크! → 이미 만든 컨트롤러가 있네요!? userCheck 를 사용!
- DB에 입력 받은 ID가 존재하면 입력받은 password 와 DB에 있는 회원의 password 가 동일한지를 체크하고 로그인 처리를 하면 됩니다!
- req.session 을 사용하여 로그인 여부 / 로그인된 ID 를 session 에 저장
→ 게시판 서비스로 이동
- 비밀번호가 틀리면 비밀번호가 불일치 안내 → 로그인 페이지로 이동 안내

```
router.post('/', (req, res) => {
  userDB.userCheck(req.body.id, (data) => {
    if (data.length > 0) {
      if (data[0].PASSWORD === req.body.password) {
        req.session.login = true;
        req.session.userId = req.body.id;
        res.redirect('/dbBoard');
      } else {
        res.send(
          '비밀번호가 다릅니다.<br><a href="/login">로그인 페이지로 이동</a>',
        );
      }
    } else {
      res.send(
        '해당 id 가 존재하지 않습니다.<br><a href="/register">회원가입 페이지로 이동</a>',
      );
    }
  });
});
```



로그 아웃 구현



로그아웃 버튼 만들기

- dbBoard.ejs 파일에 로그아웃 버튼 추가
- GET 방식 /login/logout 주소로 로그아웃 요청

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



로그 아웃 처리

- 로그 아웃 요청이 들어오면 생성 된 req.session 을 삭제 처리 → 최초 화면으로 이동

```
router.get('/logout', async (req, res) => {  
  req.session.destroy((err) => {  
    if (err) throw err;  
    res.redirect('/');  
  });  
});
```



로그인 여부에 따른 게시판 서비스 변경



로그인 여부에 따른 상황 처리

- 로그인 안되어 있으면 게시판에 접속이 불가능 하도록 수정
- Req.session 은 어느 라우터에서나 불러서 쓸 수 있다!
- Req.session.login 의 값을 확인해서 게시판 서비스로 이동 할지, 로그인 페이지 이동을 안내할지 결정
- Board.ejs 파일에 req.session.id 에 저장된 회원 id 정보도 같이 전달!


```
router.get('/', (req, res) => {
  if (req.session.login) {
    db.getAllArticles((data) => {
      const ARTICLE = data;
      const articleCounts = ARTICLE.length;
      res.render('dbBoard', {
        ARTICLE,
        articleCounts,
        userId: req.session.userId,
      });
    });
  } else {
    res.status(404);
    res.send('로그인 해주세요.<br><a href="/login">로그인 페이지로 이동</a>');
  }
});
```





로그인 확인용 함수를 이용

- 미들웨어의 2번째 매개 변수로 로그인 확인용 함수를 넣어서 처리하는 방법도 있습니다!
- 로그인 여부를 req.session 값을 통해 판별하고 로그인되어 있으면 next() 를 이용 뒤의 익명 함수를 수행하는 구조를 가집니다
- If 문을 덜 사용하고 편리하게 사용이 가능합니다



```
function isLogin(req, res, next) {  
  if (req.session.login) {  
    next();  
  } else {  
    res.send('로그인 해주세요.<br><a href="/login">로그인 페이지로 이동</a>');  
  }  
}
```

```
router.get('/', isLogin, (req, res) => {  
  db.getAllArticles((data) => {  
    const ARTICLE = data;  
    const articleCounts = ARTICLE.length;  
    res.render('dbBoard', {  
      ARTICLE,  
      articleCounts,  
      userId: req.session.userId,  
    });  
  });  
});
```



게시글에 작성자 id 정보 추가



게시글에 작성자 id 정보 추가

- 각각의 게시글의 작성자가 누구인지 알려주는 기능을 추가
- 자신이 작성한 글은 수정 및 삭제 버튼이 보이도록 기능을 추가
- 위의 기능 추가를 위해서는 게시글 DB에 작성자의 id 정보가 있어야함!
- Workbench 에 가서 일단 추가 해봅시다!



Navigator

SCHEMAS

Filter objects

- mbti
- mydb
- mydb1**
 - Tables
 - board**
 - us
 - Views
 - Store
 - Func
- sakila
- sys
- world

Query 1 SQL File 3* SQL File 4* board - Table

Table Name: board

Charset/Collation: utf8mb4

Comments:

	Datatype	PK
	INT	<input checked="" type="checkbox"/>
	VARCHAR(100)	<input type="checkbox"/>
	VARCHAR(300)	<input type="checkbox"/>
ATE	DATETIME	<input type="checkbox"/>
TE	DATETIME	<input type="checkbox"/>

Column Name:

Context Menu:

- 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



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
💡 ID_PK	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
💎 TITLE	VARCHAR(100)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
💎 CONTENT	VARCHAR(300)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
💎 REGISTER_DATE	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CURRENT_TIMESTAMP
💎 UPDATE_DATE	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CURRENT_TIMESTAMP ON...
💎 USERID	VARCHAR(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	



<						
Result Grid						
Filter Rows: <input type="text"/>						
Edit:						
Export/Import:						
Wr						
	ID_PK	TITLE	CONTENT	REGISTER_DATE	UPDATE_DATE	USERID
	1	제목1	테스트 컨텐츠 입니다!	2022-11-24 14:27:39	2022-11-26 11:25:24	tetz
	2	제목2	테스트 컨텐츠 입니다!	2022-11-24 14:27:41	2022-11-26 11:25:24	tetz
▶*	NULL	NULL	NULL	NULL	NULL	NULL





Board.ejs

파일 수정



작성자 표시

- 제목 위에 작성자의 id 를 보여주는 부분 추가

```
<li>
  <div class="author">
    작성자 : <%= ARTICLE[i].USERID %>
  </div>
  <div class="title">
    <%= ARTICLE[i].TITLE %>
  </div>
```



자신이 작성한 글에만 수정 삭제 버튼 표시

- 게시글의 작성자 id 와 로그인한 유저의 id 를 비교해서 수정 및 삭제 버튼 표시

```
div class="foot">
  <% if (ARTICLE[i].USERID === userId) { %>
  <a class="btn orange" href="dbBoard/modify/<%= ARTICLE[i].ID_PK %>">수정</a>
  <a class="btn blue" href="#" onclick="deleteArticle('<%= ARTICLE[i].ID_PK %>')">삭제</a>
  <% } %>
</div>
```



게시글 추가 기능

수정



게시글 추가 페이지로 이동

- 로그인 상태가 아니면 해당 페이지로 이동이 안되도록 설정

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



게시글 추가 기능

- 글을 추가 할 때에도 로그인 여부 판별
- 새로운 게시글을 추가할 때, title, content 이외에 id 값으로 로그인한 유저의 id 값을 받아서 글을 추가



```
router.post('/write', isLoggedIn, (req, res) => {
  if (req.body.title && req.body.content) {
    const newArticle = {
      id: req.session.userId,
      title: req.body.title,
      content: req.body.content,
    };
    boardDB.writeArticle(newArticle, (data) => {
      if (data.affectedRows >= 1) {
        res.redirect('/dbBoard');
      } else {
        const err = new Error('글 쓰기 실패');
        throw err;
      }
    });
  } else {
    const err = new Error('글 제목 또는 내용이 없습니다!');
    throw err;
  }
});
```

```
writeArticle: (newArticle, cb) => {  
  boardDB.query(  
    `INSERT INTO mydb1.board (USERID, TITLE, CONTENT) VALUES ('${newArticle.id}',  
    '${newArticle.title}', '${newArticle.content}')`,  
    (err, data) => {  
      if (err) throw err;  
      cb(data);  
    },  
  );  
},
```





게시글 수정 기능

수정



게시글 수정 페이지로 이동

- 로그인 상태가 아니면 해당 페이지로 이동이 안되도록 설정

```
router.get('/modify/:id', isLoggedIn, (req, res) => {  
  boardDB.getArticle(req.params.id, (data) => {  
    if (data.length > 0) {  
      res.render('dbBoard_modify', { selectedArticle: data[0] });  
    }  
  });  
});
```



게시글 수정 기능

- 로그인이 안되어 있으면 게시글 수정 요청이 안되도록 설정 그 외의 부분은 동일하므로 건들 필요가 없음!



```
router.post('/modify/:id', isLogin, (req, res) => {
  if (req.body.title && req.body.content) {
    db.modifyArticle(req.params.id, req.body, (data) => {
      console.log(data);
      if (data.affectedRows >= 1) {
        res.redirect('/dbBoard');
      } else {
        const err = new Error('글 수정 실패');
        throw err;
      }
    });
  } else {
    const err = new Error('글 제목 또는 내용이 없습니다!');
    throw err;
  }
});
```



게시글 삭제 기능

수정



게시글 삭제 기능

- 로그인이 안되어 있으면 게시물 삭제 요청이 안되도록 설정 그 외의 부분은 동일하므로 건들 필요가 없음!

```
router.delete('/delete/:id', isLogin, (req, res) => {
  db.deleteArticle(req.params.id, (data) => {
    console.log(data);
    if (data.affectedRows >= 1) {
      res.send('삭제 완료!');
    } else {
      const err = new Error('글 삭제 실패');
      throw err;
    }
  });
});
```

