



# 섹션 2. 생애 최초 Database 조작하기

## 이번 섹션의 목표

10강. Database와 MySQL

11강. MySQL에서 테이블 만들기

    이번 시간의 DDL 요약

12강. 테이블의 데이터를 조작하기

    이번 시간의 DML 요약

13강. Spring에서 Database 사용하기

14강. 유저 업데이트 API, 삭제 API 개발과 테스트

15강. 유저 업데이트 API, 삭제 API 예외 처리하기

16강. Section2 정리. 다음으로!

## 이번 섹션의 목표

1. 디스크와 메모리의 차이를 이해하고, Database의 필요성을 이해한다.
2. MySQL Database를 SQL과 함께 조작할 수 있다.
3. 스프링 서버를 이용해 Database에 접근하고 데이터를 저장, 조회, 업데이트, 삭제할 수 있다.
4. API의 예외 상황을 알아보고 예외를 처리할 수 있다.

## 10강. Database와 MySQL



강의 영상과 PPT를 함께 보시면 더욱 좋습니다 😊

이번 시간에는 지난 시간에 언급되었던, “유저 정보는 메모리에서만 유지되고 있다”의 의미를 살펴보고, 이 문제를 해결하기 위한 Database에 접근해 볼 것이다.

컴퓨터의 핵심 부품 세 가지는 CPU, RAM, DISK이다! 이 셋은 각각 다음과 같은 역할을 맡고 있다.

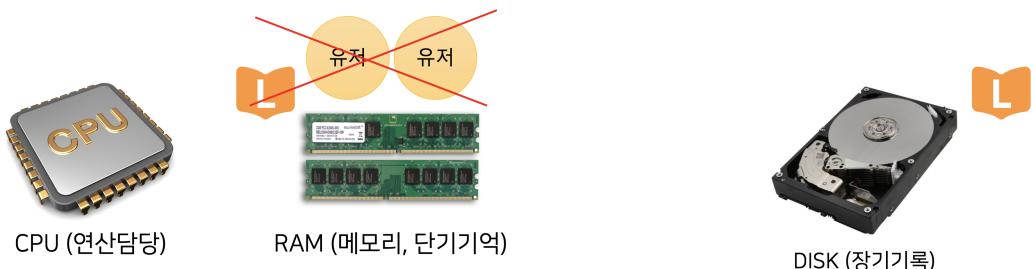
- CPU : 연산담당
- RAM : 메모리, 단기기억
- DISK : 장기기록

우리가 서버를 실행시켜 API를 동작시키기까지 이 3가지 장치는 다음과 같은 역할을 수행한다.

1. 우리가 개발하고 있는 스프링 부트 서버는 DISK에 파일로 잠들어 있다.
2. 서버를 실행시키면 DISK에 있는 코드 정보가 RAM으로 복사된다.
3. API가 실행되면 '연산'이 수행되며, CPU와 RAM을 왔다 갔다 한다.
4. 즉 POST API를 통해 생긴 유저 정보는 RAM에 쓰여 있다.

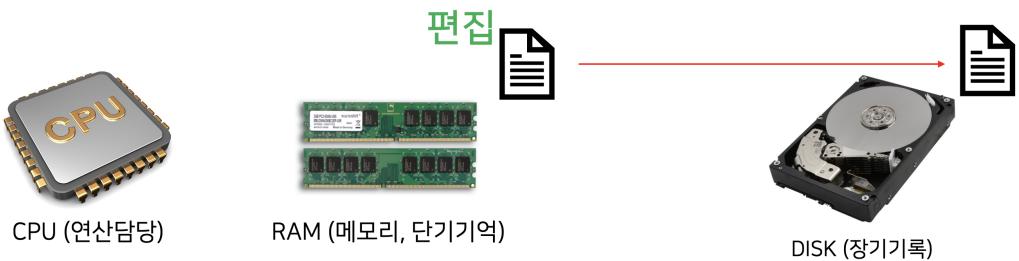


5. 만약 서버가 종료되면 RAM에 있는 모든 정보가 사라진다.



6. 때문에 다시 서버를 시작하면, 유저 정보가 없는 것이다!

비슷하게, 우리가 워드 / 엑셀에서 저장을 한다는 의미는 문서에서 편집한 내용을 DISK에 장기기록한다는 뜻이다!



그렇다면 서버에서는 어떻게 DISK에 정보를 저장할 수 있을까?!

Java에 있는 File 클래스를 사용해 직접 DISK에 접근할 수 있지만, 이럴 때 바로 Database를 사용할 수 있다.

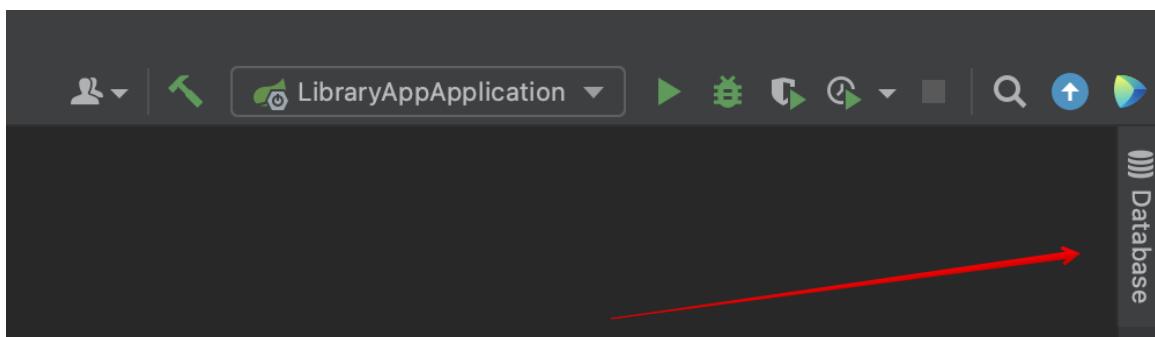
Database란 데이터를 구조화시켜 저장하는 장치이고, RDB(Relational Database)란 데이터를 표처럼 구조화시켜 저장하는 장치이다.

그리고 MySQL이 바로 RDB의 한 종류이다. 이때 SQL이란 단어는 Structured Query Language의 약자로 표처럼 구조화된 데이터를 조회하는 언어를 의미한다.

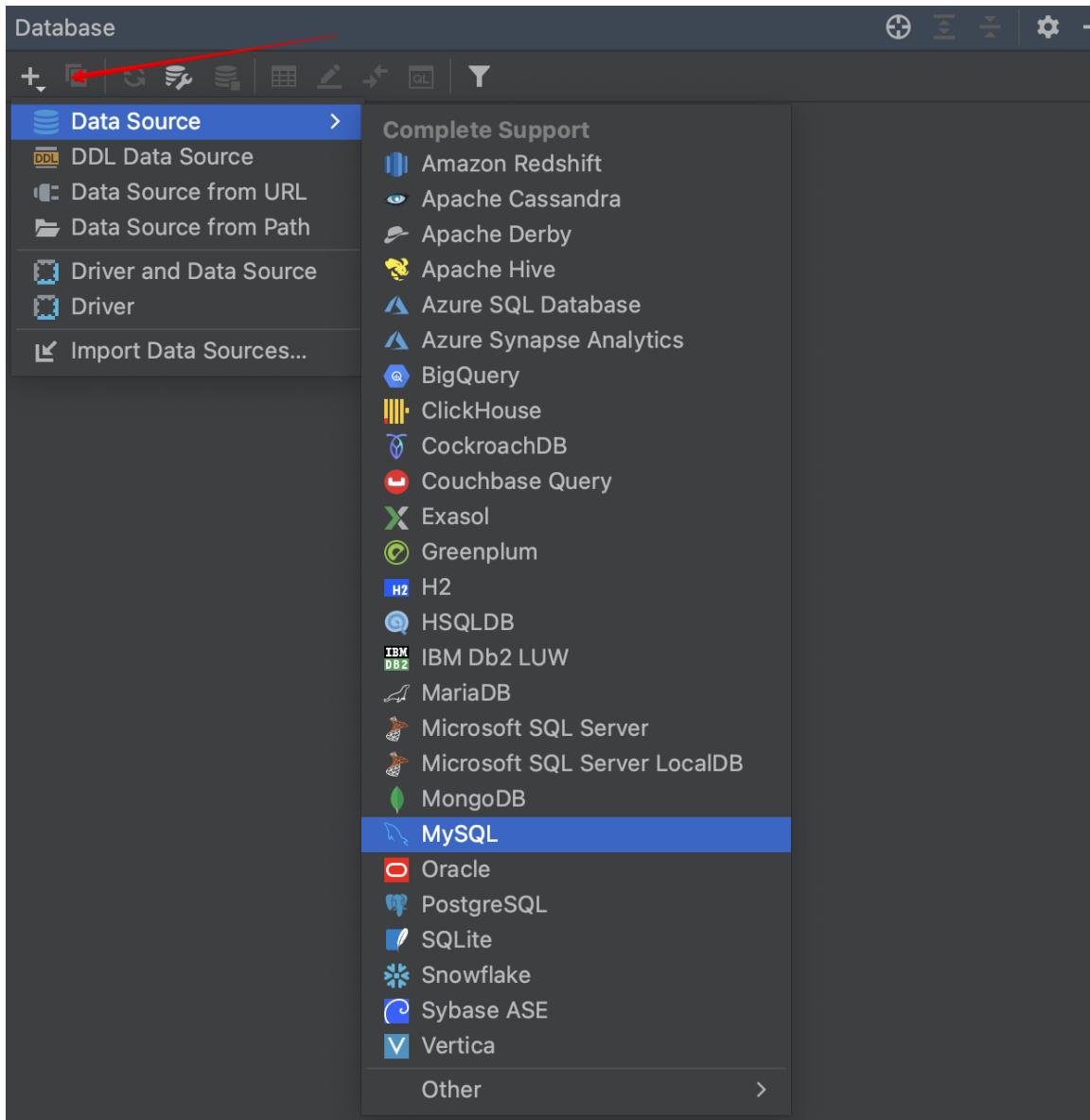
MySQL에 접근하기 위한 방법은 여러가지가 있는데, 그 중 2가지만을 소개해보려고 한다.

### [1. IntelliJ Ultimate 버전 - Database]

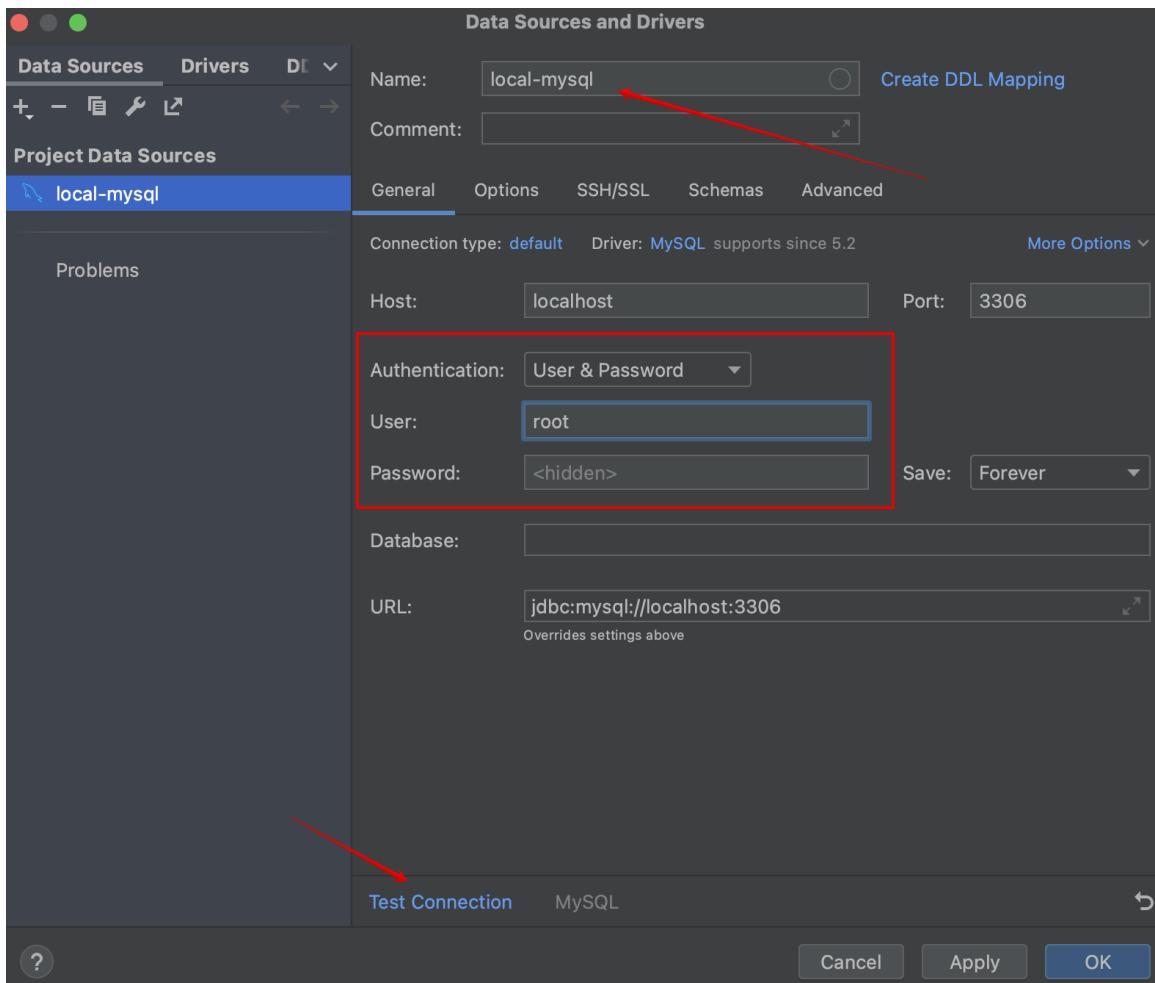
첫 번째 방법은 유로 IntelliJ를 사용하고 있는 경우, 사용할 수 있는 방법이다.



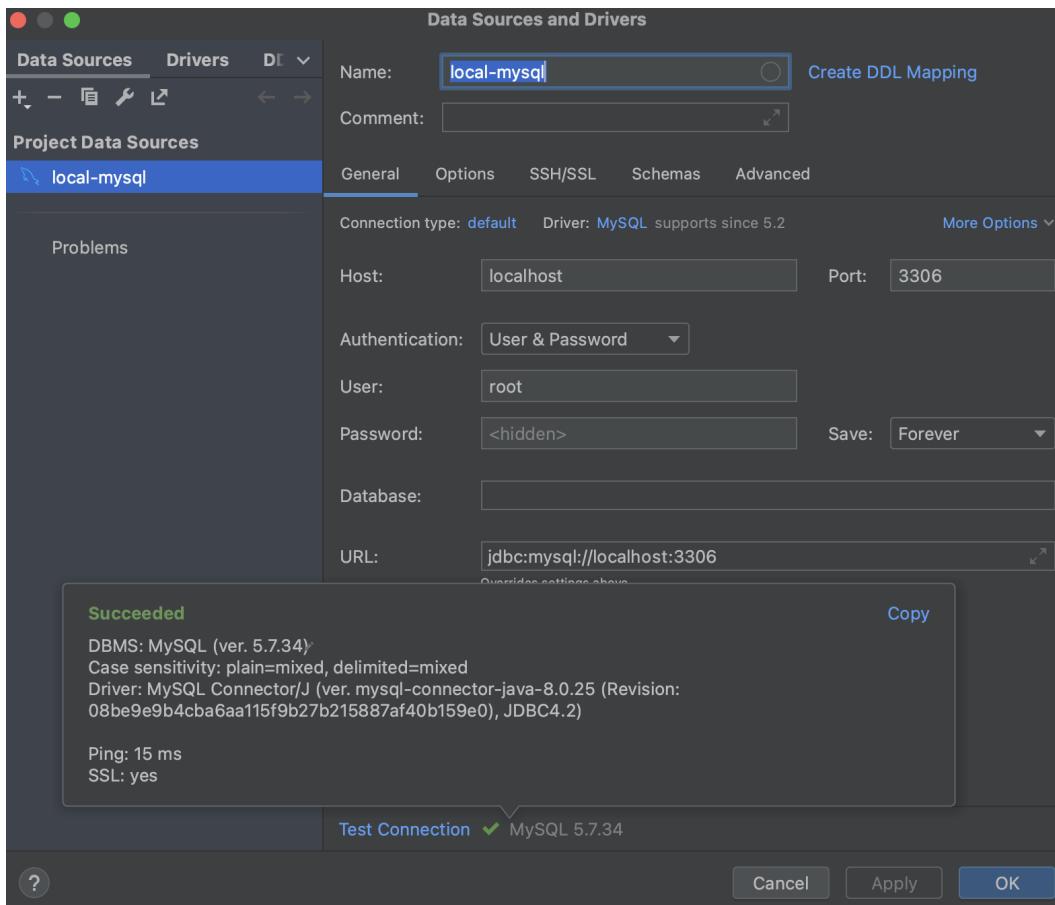
화면 오른쪽 상단에 Database라고 적힌 탭이 있다.



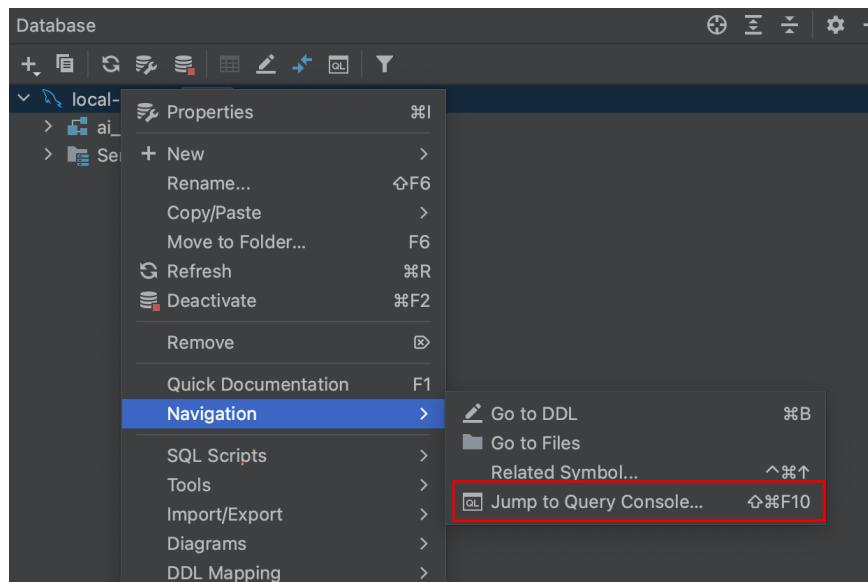
Database 탭을 누르고 + 기호를 누른다. 그 후 Data Source > MySQL을 선택한다.



그런 다음 **Test Connection** 을 누르면 테스트가 성공했다는 화면이 나온다! 이제 OK를 누르고,



설정된 `local-mysql`에 커서를 올려 우클릭 > Navigation > Jump to Query Console을 누르면



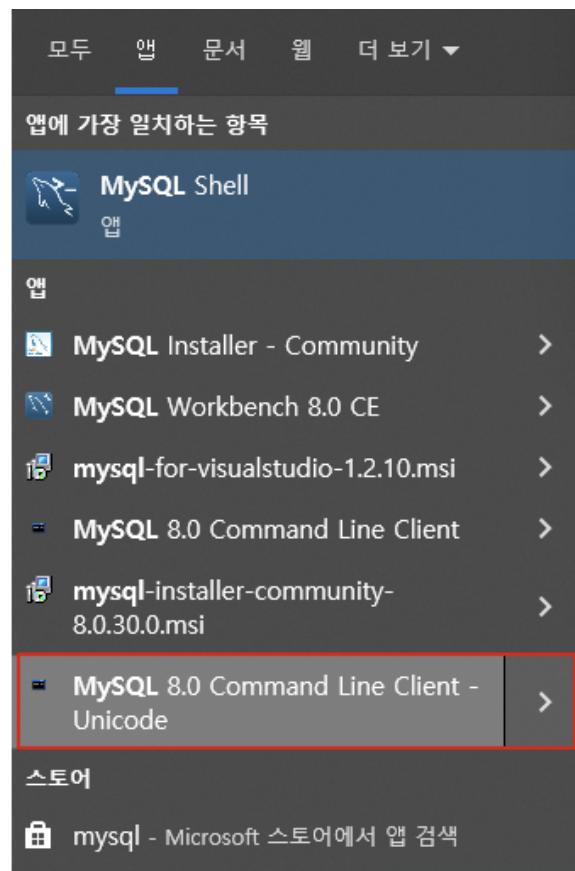
화면 가운데 SQL을 쓸 수 있는 공간이 나오는 것을 확인할 수 있다.

## [2. MySQL CLI 사용]

두 번째 방법은 IntelliJ와 무관하게 MySQL 을 설치하면 사용할 수 있는 CLI를 쓰는 것이다!

CLI란, Command Line Interface의 약자로 쉽게 설명하자면 검정색 화면에 흰색 글씨를 적으며 명령을 내리는 것을 말한다.

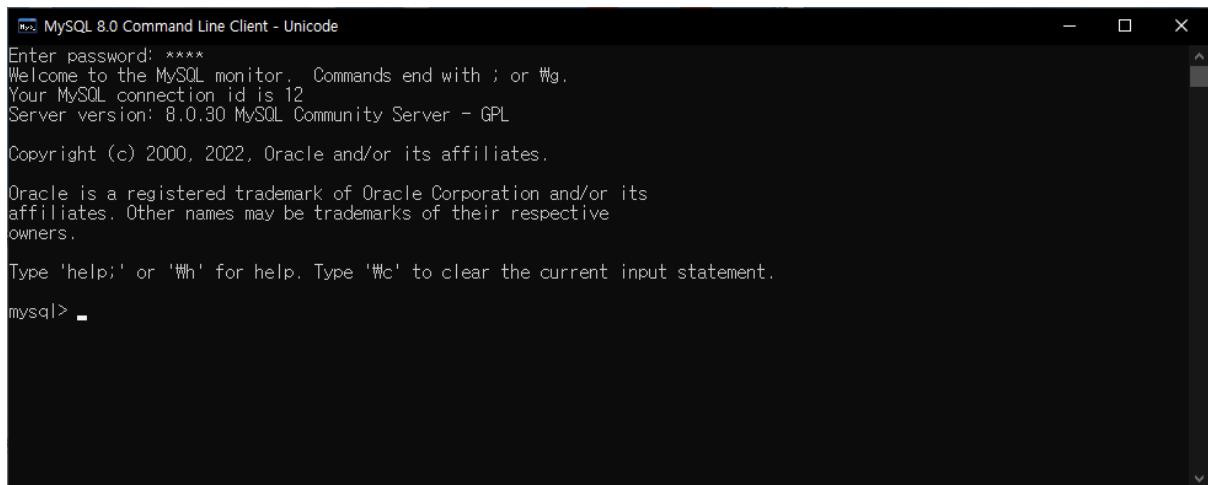
윈도우라면, 윈도우 키를 열어 [MySQL 8.0 Command Line Client - Unicode](#) 를 찾아 열면 CLI에 들어갈 수 있다!



해당 프로그램을 실행시키면 아래와 같이 비밀번호를 입력하라고 나온다.



비밀번호를 모두 입력하면 아래와 같이 `mysql>` 이라 나오고, 이제 강의에서 다룰 SQL을 타이핑할 수 있다.



```
MySQL 8.0 Command Line Client - Unicode
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 8.0.30 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.

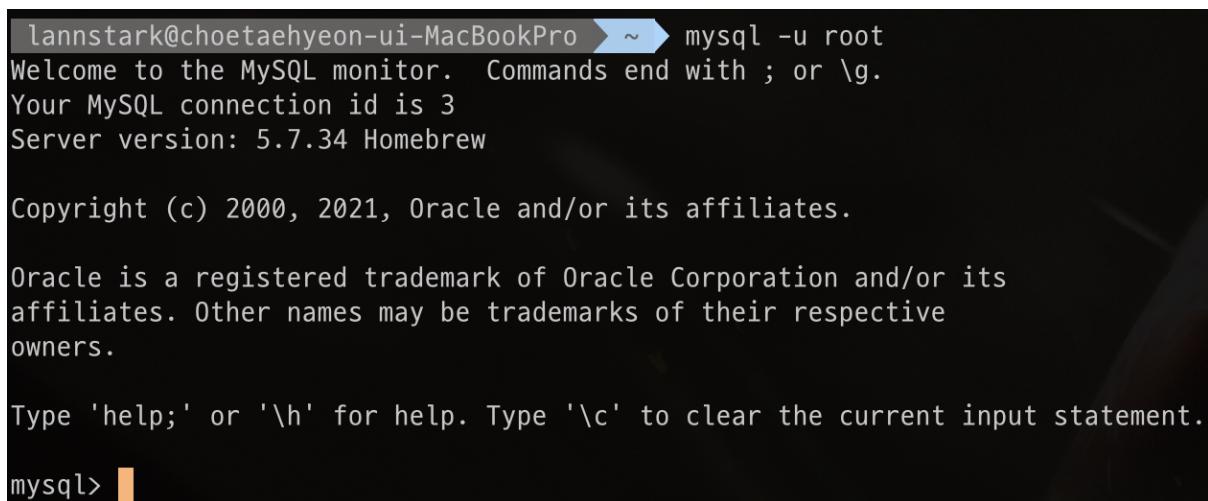
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> -
```

Mac이라면, 터미널을 열어 다음과 같이 타이핑해줄 수 있다. `mysql -u root -p 비밀번호`

만약 비밀번호가 없다면 `mysql -u root` 까지만 타이핑해도 괜찮다.

그러면 다음과 같이 `mysql>` 이라 나오고, 이제 SQL을 타이핑할 수 있다!



```
lannstark@choetaehyeon-ui-MacBookPro ~ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.34 Homebrew

Copyright (c) 2000, 2021, 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> -
```

2가지 방법 중 어떤 방법을 사용해도 상관없다! 다만, 강의에서는 1번 방법을 사용할 예정이다. 연결이 되었다면 `SELECT 1 + 1;` 을 타이핑해 실행시켜 보자!

결과가 다음과 같이 정상적으로 2가 나왔다면 MySQL에 접근해 SQL을 실행시켜 본 것이다!!! 🎉

이제 다음 시간에는 SQL을 이용해 MySQL에 데이터 구조를 잡아볼 것이다!!! 🔥

## 11강. MySQL에서 테이블 만들기



강의 영상과 PPT를 함께 보시면 더욱 좋습니다 😊

이번 시간에는 MySQL에서 테이블을 만들어 볼 것이다!! 자 그전에, 테이블을 만든다는 의미가 무엇일까?!

우선 우리가 컴퓨터에서 엑셀 파일에 과일 정보를 기록하려 한다고 해보자! 자 그러면 우리는 다음과 같은 순서로 작업하게 될 것이다!

1. 우선 엑셀 파일을 담을 폴더를 하나 만든다.
2. 폴더에 들어간다.
3. 그리고 그 폴더 안에 엑셀 파일을 만든다.
4. 엑셀 파일(표)에 Header를 입력한다! 그래야 Header 항목에 맞추어 내용을 입력할 수 있을 것이다.
5. 그리고 각 Header 별로 서식을 지정한다.

우리가 MySQL에 테이블을 만든다는 것 역시 비슷하다. 위의 예시는 실제 MySQL에 다음과 같이 대응된다.

- 폴더 = 데이터베이스
  - 여기서의 데이터베이스는 저장 장치 Database를 말하는 것이 아니라, MySQL 내의 테이블들을 묶어줄 그룹을 말한다.
- 엑셀 파일 = 테이블
- 엑셀 파일의 헤더 = 테이블의 필드 정의
- 엑셀 파일의 서식 = 테이블의 필드 타입

매우 좋다~! 😊 자 그러면 mysql에 들어가 데이터베이스를 만들어보자!

데이터베이스를 만드는 명령어는 다음과 같다. SQL은 모두 마지막에 ; (세미콜론) 을 붙여주어야 한다.

```
create database [데이터베이스 이름];
```

[ ] 에는 우리가 원하는 이름을 집어 넣으면 된다. 예를 들어,

```
create database library;
```

라고 사용할 수 있다.

이렇게 만들어진 데이터베이스는 다음 SQL을 이용해 목록을 볼 수 있다.

```
show databases;
```

그리고 만약 데이터베이스를 제거하고 싶다면 다음 SQL을 사용할 수 있다.

```
drop database [데이터베이스 이름];
```

자 데이터베이스(폴더) 안으로 들어가 이제 테이블(엑셀 파일)을 만들 때이다. 데이터 베이스 안으로 들어가고 싶다면 다음 SQL을 활용해야 한다.

```
use [데이터베이스 이름];
```

매우 좋다~! 이제 데이터베이스 안으로 들어왔다! 여기서 현재 존재하는 테이블 목록을 보고 싶다면 다음 SQL을 타이핑해보자.

```
show tables;
```

당연히 지금은 데이터베이스 안에 테이블이 들어 있지 않다. 이제 테이블을 만들어보자! 테이블을 만든다는 것은, 엑셀 파일에 column 헤더와 서식을 지정해주는 것과 유사하다. SQL은 다음과 같다.

```
create table [테이블 이름] (
    [필드1 이름] [타입] [부가조건],
    [필드2 이름] [타입] [부가조건],
    ...
    primary key ([필드 이름])
);
```

```
create table fruit (
    id bigint auto_increment,
    name varchar(20),
    price int,
    stocked_date date,
    primary key (id)
);
```

MySQL에서 필드에 제공하는 타입에 대해서는 잠시 뒤에 알아보고, 다른 내용부터 살펴보자.

- **create table**
  - SQL에 의해 고정된 명령어이다.
- **fruit**
  - 테이블 이름을 fruit으로 한다.
- **auto\_increment**
  - id에 부가 조건을 auto\_increment 라고 설정해주었다.
  - auto\_increment를 설정해준 필드는 데이터를 명시적으로 집어 넣지 않더라도 1부터 1씩 증가하며 자동 기록된다.
  - 엑셀의 왼쪽 번호를 생각하면 된다!

| A  | B  | C    |
|----|----|------|
| 이름 | 가격 | 입고일자 |
| 1  |    |      |
| 2  |    |      |
| 3  |    |      |
| 4  |    |      |
| 5  |    |      |
| 6  |    |      |

- primary key (id)
  - id라는 필드를 유일한 키로 지정한다! 1씩 자동으로 증가한다면 어떤 데이터건 유일한 키가 될 것이다.

좋다 이제 탑입을 전체적으로 살펴보자!

### 정수 탑입

- tinyint : 1바이트 정수
- int : 4바이트 정수,
- bigint : 8바이트 정수

### 실수 탑입

- double : 8바이트 실수
- decimal(A, B) : 소수점을 B개 가지고 있는 전체 A자릿수 실수

### 문자열 탑입

- char(A) : A글자가 들어갈 수 있는 문자열
- varchar(A) : 최대 A글자가 들어갈 수 있는 문자열

### 날짜, 시간 탑입

- date : 날짜, `yyyy-MM-dd` 형식으로 들어간다.
- time : 시간, `HH:mm:ss` 형식으로 들어간다.
- datetime : 날짜와 시간을 합친 형식, `yyyy-MM-dd HH:mm:ss` 형식으로 들어간다.

이 탑 정보를 바탕으로, 위의 fruit 테이블을 엑셀로 비유해 생각해보면 다음과 같다.

|   | A                 | B       | C                 |
|---|-------------------|---------|-------------------|
| 1 | 이름 (문자열 - 최대 20자) | 가격 (정수) | 입고일자 (yyyy-MM-dd) |
| 2 |                   |         |                   |
| 3 |                   |         |                   |
| 4 |                   |         |                   |
| 5 |                   |         |                   |
| 6 |                   |         |                   |

- id bigint auto\_increment
  - 엑셀 가장 왼쪽의 넘버링, 자동으로 1씩 증가한다.
- name varchar(20)
  - 최대 20자가 들어갈 수 있는 이름
- price int
  - 정수가 들어갈 수 있는 가격
- stocked\_date
  - 날짜가 들어가는 입고일자

혹시 테이블을 제거하고 싶다면, 아래의 SQL을 사용할 수도 있다.

```
drop table [테이블 이름];
```

## 이번 시간의 DDL 요약

이번 시간에 다루었던 SQL은 DDL(Data Definition Language)이라고 한다. 데이터를 정의하기 위한 SQL이라 데이터 정의 언어라 부르는 것이다. 간단히 요약해보면 다음과 같다.

```
### 데이터베이스 ###
create database [데이터베이스 이름];
show databases;
drop database [데이터베이스 이름];
use [데이터베이스 이름];

### 테이블 ###
create table [테이블 이름] (
    [필드1 이름] [타입] [부가조건],
```

```
[필드2 이름] [타입] [부가조건],  
...  
primary key ([필드 이름])  
);  
show tables;  
drop table [테이블 이름];
```

매우 좋다~!!! 이제 우리는 MySQL에 간단한 테이블을 만들 수 있다 😊

다음 시간에는 이어서 만들어진 테이블에 데이터를 넣고, 삭제하고, 조회하고, 업데이트하는 조작 SQL을 알아보자~!!! 🔥

## 12강. 테이블의 데이터를 조작하기



강의 영상과 PPT를 함께 보시면 더욱 좋습니다 😊

이번 시간에는 지난 시간에 만들었던 fruit 테이블에 데이터를 넣고, 조회하고, 수정하고, 삭제해 볼 것이다! 이 4가지 기능은 각각의 앞 글자를 따서 CRUD라고도 한다!! 👍

- 데이터를 넣는다 (생성 - Create)
- 조회한다 (읽기 - Retrieve or Read)
- 수정한다 (업데이트 - Update)
- 삭제한다 (제거 - Delete)

가장 먼저 fruit 테이블에 데이터를 넣어보자. 문법은 다음과 같다. 대소문자는 크게 중요하지 않다! (여기서는 변화되는 부분이 아니라는 강조를 위해 대문자를 사용했다.)

```
INSERT INTO [테이블 이름] (필드1이름, 필드2이름, ...) VALUES (값1, 값2, ...)
```

예를 들면 다음과 같다.

```
INSERT INTO fruit (name, price, stocked_date)  
VALUES ('사과', 1000, '2023-01-01');
```

- 소문자를 사용해도 괜찮다. 본 강의에서는 SQL의 예약어를 강조하기 위해 대문자로 표기하였다.
- 괄호 안의 필드와 값의 순서가 중요하다.
- id는 지정해 주지 않아도 auto\_increment 덕분에 자동으로 들어간다.

다음으로는 데이터를 조회해보자! 🔥

```
SELECT * FROM [테이블 이름];
```

조회와 관련해서는 2가지를 추가적으로 알고 있어야 한다. 먼저, `SELECT * FROM`에서 `*` 대신 필드 이름을 사용하면 해당 필드만을 가져올 수 있다는 점이다.

예를 들면 다음과 같다.

```
SELECT name, price FROM fruit;
```

다음으로는 조회를 할 때 조건을 통해 필터를 걸 수 있다는 점이다! SQL 문법은 다음과 같다.

```
SELECT * FROM [테이블 이름] WHERE [조건];
```

예를 들어, 사과만 가져올 수 있다.

```
SELECT * FROM fruit WHERE name = '사과';
```

조건은 `and` 또는 `or`을 통해 여러가지를 볼일 수 있다.

```
SELECT * FROM fruit WHERE name = '사과' and price <= 2000;
```

이때 조건은 `=` 외에도 `>` `<` `!=` `<=` `>=` `between` `in` `not in` 등 다양하게 존재한다. 간단히 몇 가지 예시를 살펴보자.

```
# between 예시 - 가격이 1000 ~ 2000원에 위치한 모든 과일을 가져온다.
SELECT * FROM fruit WHERE price BETWEEN 1000 AND 2000;
```

```
# in 예시 - 이름이 사과이거나 수박인 모든 과일을 가져온다.  
SELECT * FROM fruit WHERE name IN ('사과', '수박');
```

```
# not in 예시 - 이름이 사과가 아닌 모든 과일을 가져온다.  
SELECT * FROM fruit WHERE name NOT IN ('사과');
```

이제 데이터를 업데이트해보자! 업데이트 SQL 문법은 다음과 같다.

```
UPDATE [테이블 이름] SET 필드1이름=값1, 필드2이름=값2, ... WHERE [조건];
```

예를 들어, 모든 사과는 가격을 2000원에서 1500원으로 변경할 수 있다.

```
UPDATE fruit SET price=1500 WHERE name = '사과';
```

만약 조건을 붙이지 않는다면 모든 데이터가 업데이트되므로 매우 주의해야 한다! 😢

다음으로 데이터를 삭제해보자!

```
DELETE FROM [테이블 이름] WHERE [조건];
```

예를 들어, 모든 사과를 없앨 수 있다!!

```
DELETE FROM fruit WHERE name = '사과';
```

업데이트와 마찬가지로, 조건을 붙이지 않고 DELETE 쿼리를 날리게 되면, 모든 데이터가 삭제되기 때문에 매우 주의해서 사용해야 한다!! 😢

## 이번 시간의 DML 요약

이번 시간에 다루었던 SQL은 DML(Data Manipulation Language)이라고 한다. 데이터를 조작하기 위한 SQL이라 데이터 조작 언어라 부르는 것이다. 간단히 요약해 보면 다음과 같

다.

```
# 데이터 넣기  
INSERT INTO [테이블 이름] (필드1이름, 필드2이름, ...) VALUES (값1, 값2, ...)  
  
# 데이터 조회하기  
SELECT * FROM [테이블 이름] WHERE [조건];  
  
# 데이터 업데이트하기  
UPDATE [테이블 이름] SET 필드1이름=값1, 필드2이름=값2, ... WHERE [조건];  
  
# 데이터 삭제하기  
DELETE FROM [테이블 이름] WHERE [조건];
```

매우 좋다~~ 이것으로 MySQL의 사용법을 간단하게 모두 익혔다. 이제 다음 시간에는 다시 우리의 도서 관리 애플리케이션으로 돌아와 서버와 데이터베이스가 통신할 수 있게 설정해 보도록 하자!

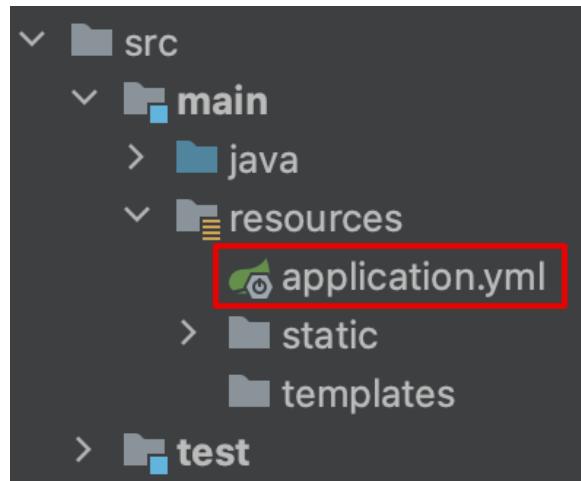
## 13강. Spring에서 Database 사용하기

이번 시간에는 스프링 서버와 MySQL 데이터베이스를 연결하고 기존에 만들었던 유저 저장 API와 조회 API를 리팩토링해볼 것이다. 지금까지 우리는 사람이 직접 MySQL에 접근했다. 하지만 이제는 우리의 스프링 서버가 MySQL에 접근하도록 설정해 주어야 한다.

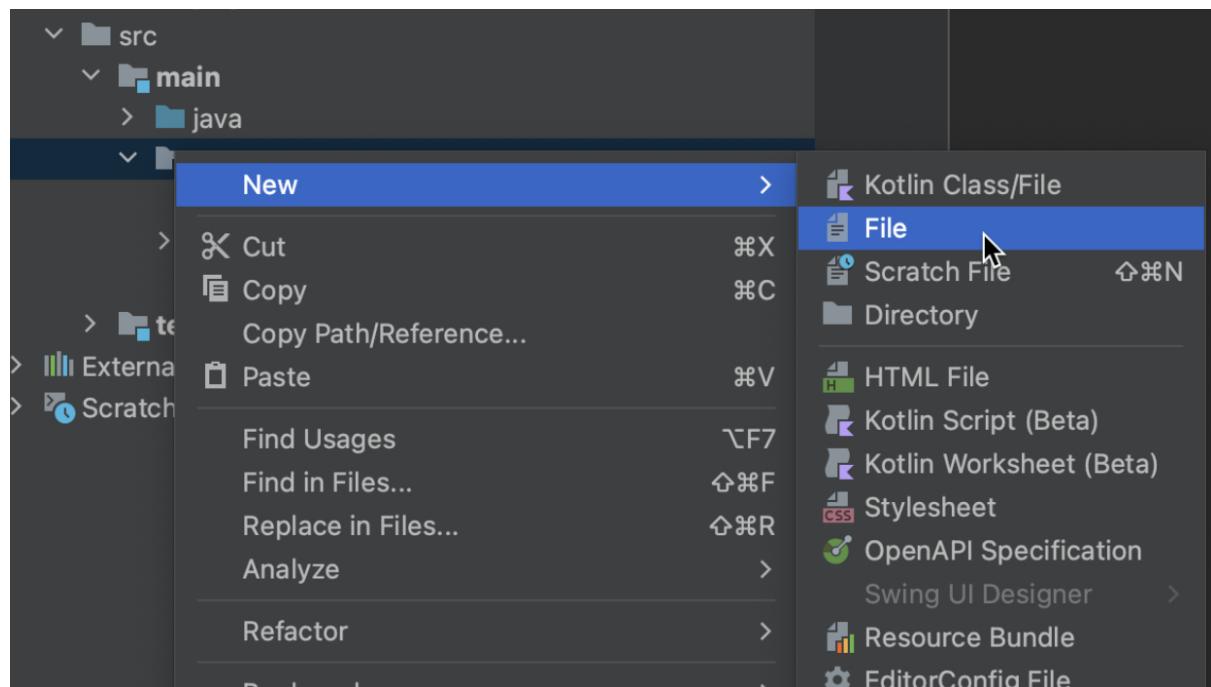
**이제 우리의 스프링 서버가 MySQL DB에 접근하게 하자!**



좋다 이제 스프링 서버와 데이터베이스를 연결해 보자. 우선, 다음 위치에 `application.yml` 이란 파일을 만들어야 한다.



파일을 만드는 것 역시 Java Class를 만드는 것과 유사하게 오른쪽 클릭 > New > File을 선택하여, `application.yml` 을 입력해 주면 된다!



`application.yml` 파일에는 다음과 같이 입력해 주자! 이 파일을 통해 스프링 서버와 연결할 데이터베이스에 대한 정보를 설정하게 된다.

```
spring:  
  datasource:  
    url: "jdbc:mysql://localhost/library"  
    username: "root"  
    password: ""  
    driver-class-name: com.mysql.cj.jdbc.Driver
```

- url : 우리가 연결할 데이터베이스 주소
  - `jdbc:mysql://` - jdbc를 이용해 mysql에 접근한다
  - `localhost` - 접근하려는 mysql은 localhost에 있다.
  - `/library` - 접근하는 데이터베이스는 library이다.
- username : MySQL에 접근하기 위한 계정명
- password : MySQL에 접근하기 위한 비밀번호
- driver-class-name : 데이터베이스에 접근할 때 사용할 프로그램

좋다~! 다음으로는 유저 API를 List 대신 DB를 사용하도록 변경할 건데, 그전에 유저 테이블을 mysql에 만들어주도록 하자!

```
create table user(
    id bigint auto_increment,
    name varchar(25),
    age int,
    primary key (id)
)
```

이제 UserController에 들어가 POST API부터 변경해 보자! (아래는 전체 Controller이다!)

```
@RestController
public class UserController {

    private final JdbcTemplate jdbcTemplate;

    public UserController(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    @PostMapping("/user")
    public void saveUser(@RequestBody UserCreateRequest request) {
        String sql = "INSERT INTO user(name, age) VALUES(?, ?)";
        jdbcTemplate.update(sql, request.getName(), request.getAge());
    }
}
```

- jdbcTemplate과 생성자
  - jdbcTemplate을 이용해 MySQL에 SQL을 보낼 수 있다.

- 이렇게 final 변수를 만들고 생성자를 만들어두면, 스프링이 알아서 jdbcTemplate 을 넣어준다!!!
- Section3에서 보다 자세한 동작 원리를 설명한다.
- `"INSERT INTO user(name, age) VALUES(?, ?)"`
  - 원래 값이 들어가야 하는 부분에 ? 를 사용했다. 이렇게 하면, 아래 있는 update에 sql과 함께 값을 넣어줄 수 있다.
- `jdbcTemplate.update(sql, request.getName(), request.getAge());`
  - jdbcTemplate.update는 INSERT 쿼리, UPDATE 쿼리, DELETE 쿼리에 사용할 수 있다.
  - 첫 파라미터로는 sql을 받고, ?를 대신할 값을 차례로 넣어주면 된다.

매우 좋다~! 이제 GET API 역시 변경해보자!

```
// UserController.java 안에 들어간다.
@GetMapping("/user")
public List<UserResponse> getUsers() {
    String sql = "SELECT * FROM user";
    return jdbcTemplate.query(sql, new RowMapper<UserResponse>() {
        @Override
        public UserResponse mapRow(ResultSet rs, int rowNum) throws SQLException {
            long id = rs.getLong("id");
            String name = rs.getString("name");
            int age = rs.getInt("age");
            return new UserResponse(id, name, age);
        }
    });
}
```

- `jdbcTemplate.query(sql, RowMapper 구현 의명클래스)`
  - query를 사용하면, SELECT 쿼리를 날릴 수 있다.
- `@Override` 함수
  - ResultSet 객체에는 결과가 담겨있고, 이 객체에 `getType("필드이름")` 을 사용해서 실제 값을 가져올 수 있다.
- UserResponse 역시 id, name, age를 받는 생성자를 만들어주었다.

람다를 사용하면 `getUsers()` 는 다음과 같이 변경된다.

```

@GetMapping("/user")
public List<UserResponse> getUsers() {
    String sql = "SELECT * FROM user";
    return jdbcTemplate.query(sql, (rs, rowNum) -> {
        long id = rs.getLong("id");
        String name = rs.getString("name");
        int age = rs.getInt("age");
        return new UserResponse(id, name, age);
    });
}

```

Java 익명 클래스와 람다에 익숙하지 않다면 아래 유튜브 영상의 내용을 일부 참고할 수도 있다.

- <https://www.youtube.com/watch?v=VRVbcPYGu6I>

이제 POST API와 GET API를 모두 DB를 사용하게끔 변경했다면, 웹 페이지를 열어 유저를 저장하고, 조회해 보자!! 잘 동작하는 것을 확인할 수 있다~!! 😊 심지어 이제는 서버를 껐다 키더라도 정상적으로 기능이 동작한다!! 👍

다음 시간에는 유저 업데이트 API와 삭제 API를 개발하고 테스트할 것이다!! 마지막까지 가보자~!! 🔥

## 14강. 유저 업데이트 API, 삭제 API 개발과 테스트

이번 시간에는 유저 업데이트 API와 삭제 API를 개발하고 웹 UI를 이용해 테스트해 볼 것이다!

생성, 조회 API와 비슷하게 정해진 스펙이 있으니 요구사항과 함께 한 번 살펴보도록 하자.

**도서관 사용자 이름을 업데이트 할 수 있다.**

- HTTP Method : PUT
- HTTP Path : /user
- HTTP Body (JSON)

```
{
    "id": Long,
    "name": String // 변경되어야 하는 이름이 들어온다
}
```

- 결과 반환 X (HTTP 상태 200 OK이면 충분한다)

## 도서관 사용자를 삭제할 수 있다.

- HTTP Method : DELETE
- HTTP Path : /user
- 쿼리 사용
  - 문자열 name (삭제되어야 하는 사용자 이름)
- 결과 반환 X (HTTP 상태 200 OK이면 충분하다)

매우 좋다~  먼저 업데이트 API부터 만들어보자! 지난 시간에 만들었던 POST API와 비슷하게 처리할 수 있다!

```
// UserController 안의 메소드
@PutMapping("/user")
public void updateUser(@RequestBody UserUpdateRequest request) {
    String sql = "UPDATE user SET name = ? WHERE id = ?";
    jdbcTemplate.update(sql, request.getName(), request.getId());
}
```

```
// PUT /user API의 HTTP Body를 위한 새로운 클래스
public class UserUpdateRequest {

    private long id;
    private String name;

    public long getId() {
        return id;
    }

    public String getName() {
        return name;
    }
}
```

다음으로는 삭제 API도 만들어보자!

```

@DeleteMapping("/user")
public void deleteUser(@RequestParam String name) {
    String sql = "DELETE FROM user WHERE name = ?";
    jdbcTemplate.update(sql, name);
}

```

- `@RequestParam`

- 쿼리 파라미터가 1개이기 때문에 `@RequestParam` 을 사용했다.
- 필드를 하나 가지고 있는 객체를 사용할 수도 있다.

자 이제 API를 모두 만들었으니, 침착하게 웹 UI에 접속해 테스트를 해보자.

| 등록하기   |     | 목록                                      |
|--------|-----|---|
| 사용자 이름 | 나이  |   |
| A      | 10세 | <button>수정</button> <button>삭제</button> |
| B      | 20세 | <button>수정</button> <button>삭제</button> |

목록에 들어가 수정과 삭제를 눌러보면 된다! 매우 좋다~!!! 😊

현재 UI를 통한 접근은 모두 정상적으로 동작한다! 다만, 한 가지 생각해 볼 지점이 있다.

현재 수정 API와 삭제 API를 사용할 때 수정이나 삭제가 정상적으로 이루어지지 않더라도 `200 OK` 를 반환한다!!! 예를 들어 ABC라는 유저가 없는 상황에서 DELETE API를 사용하면 성공했다는 의미를 담은 `200 OK` 가 반환되는 것이다. 분명 성공은 아닌데, `200 OK` 가 반환되는 것이 어색하다.

The screenshot shows the Postman application interface. A DELETE request is being sent to the URL `http://localhost:8080/user?name=ABC`. In the 'Params' tab, there is a single parameter named 'name' with the value 'ABC'. The response status is shown as `Status: 200 OK`, indicating a successful deletion.

비슷하게 업데이트 API 역시 존재하지 않는 유저에 대한 업데이트가 일어나도 `200 OK` 가 반환된다. 다음 시간에는 두 API가 예외 상황에서 잘 동작할 수 있도록 보완할 것이다.

## 15강. 유저 업데이트 API, 삭제 API 예외 처리하기

이번 시간에는 유저 업데이트 API와 유저 삭제 API를 사용할 때 대상 유저가 없으면 `200 OK` 가 나오지 않도록 수정할 것이다. 가장 간단하게, `IllegalArgumentException` 과 같은 표준 예외를 던지면 다른 상태가 나오게 된다.

간단히 API를 만들어 확인해보면 이렇다!

```
@GetMapping("/user/error-test")
public void errorTest() {
    throw new IllegalArgumentException();
}
```

The screenshot shows a Postman request to `http://localhost:8080/user/error-test`. The response status is `500 Internal Server Error`. The response body is a JSON object:

```
1  {
2      "timestamp": "2022-08-31T12:14:52.347+00:00",
3      "status": 500,
4      "error": "Internal Server Error",
5      "path": "/user/error-test"
6 }
```

API 내부에서 예외를 던지게 될 경우, API가 성공했다는 `200 OK` 대신 내부적인 에러가 있다 는 `500 Internal Server Error` 가 나오는 것을 확인할 수 있다!

매우 좋다~! 이제 유저 업데이트 API부터 변경해 보도록 하자! 유저 데이터의 존재 여부 를 확인하기 위해 GET API에서 사용했던 `jdbcTemplate.query()` 를 응용할 수 있다.

```
@PutMapping("/user")
public void updateUser(@RequestBody UserUpdateRequest request) {
    String readSql = "SELECT * FROM user WHERE id = ?";
```

```

        boolean isUserNotExist = jdbcTemplate.query(readSql, (rs, rowNum) -> 0, request.getId()).isEmpty();
        if (isUserNotExist) {
            throw new IllegalArgumentException();
        }

        String updateSql = "UPDATE user SET name = ? WHERE id = ?";
        jdbcTemplate.update(updateSql, request.getName(), request.getId());
    }

```

- `String readSql = "SELECT * FROM user WHERE id = ?";`
  - id를 기준으로 유저가 존재하는지 확인하기 위해 SELECT 쿼리를 작성하였다.
- `jdbcTemplate.query(readSql, (rs, rowNum) -> 0, request.getId())`
  - SELECT 쿼리를 DB에 날려 데이터가 있는지 확인했다!
  - `jdbcTemplate.query(sql, RowMapper 구현 익명클래스)` 를 사용하게 되면 익명 클래스의 오버라이딩 된 메소드를 거쳐 최종적으로 `List` 가 반환된다.
  - 때문에 유저 데이터가 있다면 비어 있지 않은 List가 반환될 것이고, 유저 데이터가 없다면 비어 있는 List가 반환될 것이다.
- `if (isUserNotExist)`
  - 만약 유저가 존재하지 않는다면 `IllegalArgumentException` 을 던지도록 작성하였다.

이제 POST MAN으로도 확인을 해보자.

The screenshot shows a POSTMAN interface with the following details:

- Method:** PUT
- URL:** http://localhost:8080/user
- Body:** (JSON selected)
 

```

1 {
2     "id": 10,
3     "name": "ABCD"
4 }
```
- Response Status:** Status: 500 Internal Server Error
- Response Body:**

```

1 {
2     "timestamp": "2022-08-31T12:32:33.770+00:00",
3     "status": 500,
4     "error": "Internal Server Error",
5     "path": "/user"
6 }
```

데이터베이스에 존재하지 않는 10번 유저의 이름을 변경하도록 했더니, `500 Internal Server Error` 가 나온 것을 확인할 수 있다!

물론 `isUserNotExist` 라는 변수를 만들지 않고 다음과 같이 코드를 조금 변경할 수도 있다.

```
if (jdbcTemplate.query(readSql, (rs, rowNum) -> 0, request.getId()).isEmpty()) {  
    throw new IllegalArgumentException();  
}
```

이어서 삭제 API도 비슷한 형식으로 변경해 보자. 차이점이라면, 이번엔 `id` 가 아니라 `name` 이 기준인 것을 주의해야 한다.

```
@DeleteMapping("/user")  
public void deleteUser(@RequestParam String name) {  
    String readSql = "SELECT * FROM user WHERE name = ?";  
    boolean isUserNotExist = jdbcTemplate.query(readSql, (rs, rowNum) -> 0, name).isEmpty();  
    if (isUserNotExist) {  
        throw new IllegalArgumentException();  
    }  
  
    String deleteSql = "DELETE FROM user WHERE name = ?";  
    jdbcTemplate.update(deleteSql, name);  
}
```

다 만들었다면, POST MAN으로도 확인해 보자! `500 Internal Server Error` 가 잘 나올 것이다.

매우 좋다~!!! 😊🎉 이렇게 우리는 유저에 관련된 모든 기능을 보다 완벽하게 만들었다!!!!  
👍

다음 시간에는 이번 Section을 정리해 보자.

## 16강. Section2 정리. 다음으로!

이번 Section을 거치며 우리는 서버를 재시작하면 데이터가 사라지는 문제를 해결하였다  
~!!!!!! 👍 또한, 이 과정에서 아래의 내용들을 익힐 수 있었다.

1. 디스크와 메모리의 차이를 이해하고, Database의 필요성을 이해한다.
2. SQL을 이용해 MySQL Database를 조작할 수 있다.
3. 스프링 서버를 이용해 Database에 접근하고 데이터를 저장, 조회, 업데이트, 삭제할 수 있다.

4. API의 예외 상황을 알아보고 예외를 처리할 수 있다.

그리고 Spring Boot를 활용하여, 우리의 서버에 유저와 관련된 기능을 모두 구현했다.

하지만~~ 사실 현재 코드에는 사실 심각한 문제가 숨겨져 있다!! 바로 한 클래스인 Controller에서 너무 많은 역할을 하고 있다는 것이다. 지금은 구현한 기능이 간단해서 크게 체감되지 않을 수도 있지만, 우리가 앞으로 추가 구현해야 할 요구사항을 생각하면 변화가 필요하다.

또한 실무에서는 이보다 훨씬 복잡한 요구사항도 많다. 한 API에서 무려 10개의 Table을 사용하는 경우도 있다!

다음 섹션에서는 왜 한 함수에서 모든 기능을 구현하면 안 되는지 조금 더 알아보고, 스프링의 핵심 개념을 활용해 이 문제를 해결해 보자! 🔥🏃