



It's Your Life

with





# 간단한

# DAO 구현 실습

# 전체 코드는!



- [https://github.com/xenosign/jdbc\\_dao\\_ex](https://github.com/xenosign/jdbc_dao_ex)
- 위의 링크를 참고 하세요!
- 다만 가급적 코드는 제일 마지막에 참고하시고 일단 피피티를 정독 & 따라하면서 Dao 구현의 감을 잡으셔야 합니다!
- 하다가 막힐 때, 코드를 확인하시면 좋습니다!



# 실습 1


SQL로 테이블


생성하고 데이터 넣기



## MySQL Connections

Local instance MySQL80

 root

 localhost:3306

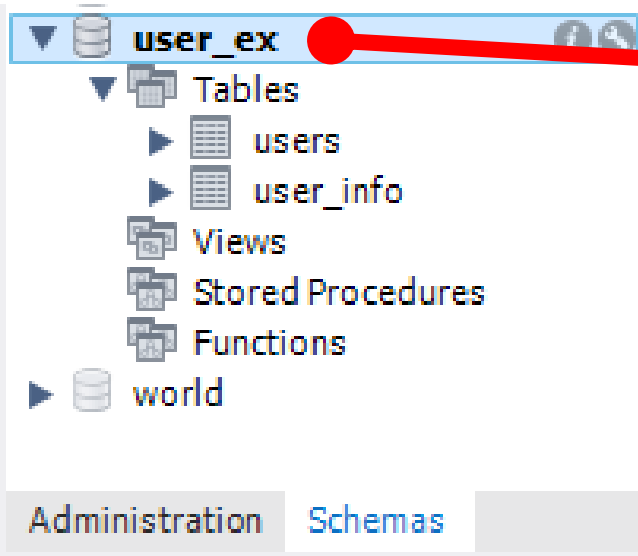
root 권한으로 DB에  
접속 해 주세요



```
1 # user_ex 데이터 베이스 생성
2 • CREATE DATABASE user_ex;
3
4 # user_ex 데이터 베이스 사용
5 • USE user_ex;
6
```


user\_ex 데이터 베이스 생성

방금 생성한 user\_ex 데이터 베이스 사용



왼쪽 스키마 화면에서  
user\_ex 데이터 베이스의 글씨가  
두껍게 처리 되었는지 확인!

```
1  # user_ex 데이터 베이스 생성
2 • CREATE DATABASE user_ex;
3
4  # user_ex 데이터 베이스 사용
5 • USE user_ex;
6
7  # users 테이블 생성
8 • CREATE TABLE users
9  (
10     id          INT AUTO_INCREMENT PRIMARY KEY,
11     email       VARCHAR(100) UNIQUE NOT NULL,
12     password    VARCHAR(100)      NOT NULL
13 );
```



## CREATE 로 users 테이블 생성하기

### 각각 컬럼에 대한 설명

id : 회원 정보를 간단한 숫자 값으로  
관리하기 위한 컬럼 / 숫자 값이며 데이터  
생성 시 AUTO\_INCREMENT 에 의해  
자동으로 숫자 값이 1 씩 올라가며  
생성된다

email : 사용자의 이메일을 저장하는 컬  
럼 / 최대 100 글자의 문자열 / 다른 데이  
터와 중복 금지 / 빈 값이 들어오면 안 됨

password : 사용자의 비밀번호를 저장하  
는 컬럼 / 최대 100 글자의 문자열 / 빈 값  
이 들어오면 안 됨

```
15 # 생성한 users 테이블 확인
16 • SELECT *
17 FROM users;
```

생성한 users 테이블 확인

Result Grid			
	id	email	password
•	NULL	NULL	NULL



```
19 # user 테이블에 회원 정보 삽입
20 # id 값은 자동으로 생성되므로 전달 필요 X
21 • INSERT INTO users (email, password)
22   VALUES ('tetz', '1234'),
23           ('siwan', '1234'),
24           ('na', '1234');
```

실습을 위한 데이터 삽입!

id 컬럼은 자동 생성 되므로  
데이터 전달 필요 없음!




```
29 • CREATE TABLE user_info
30 (
31     id    INT PRIMARY KEY,
32     name  VARCHAR(50) NOT NULL,
33     # user 테이블의 id 와 user_info 의 id 가 서로 참조하는 관계임을 외래키로 설정
34
35     FOREIGN KEY (id) REFERENCES users (id) ON DELETE CASCADE
36     # JOIN 문 연습을 위해, 역지로 만든 테이블이며 해당 테이블은 제2 정규형을(2NF)를 위배합니다
37     # name 컬럼도 id 에 종속이기 때문에 해당 테이블은 굳이 따로 나눌 필요가 없기 때문입니다
38 );
```

## user\_info 테이블 작성

users 테이블의 id 값을 외래키로 가지며  
해당 키를 기준으로 나중에 JOIN 사용 예정

```
41 # 생성한 user 테이블 확인
42 • SELECT *
43 FROM user_info;
44
45 # 각각의 테이블에 필요한 데이터 삽입
46 • INSERT INTO user_info (id, name)
47 VALUES ('1', '이효석'),
48          ('2', '김시완'),
49          ('3', '나건우');
```



Result Grid

	id	name
	NULL	NULL



```
SELECT * from users;  
SELECT * from user_info;
```



결과적으로 위의 쿼리를 실행 했을 때, 아래와 같은 테이블이 완성 되어야 합니다

Result Grid			
	id	email	password
▶	1	tetz	1234
	2	siwan	1234
	3	na	1234
⊛	NULL	NULL	NULL

Result Grid		
	id	name
▶	1	이효석
	2	김시완
	3	나건우
⊛	NULL	NULL



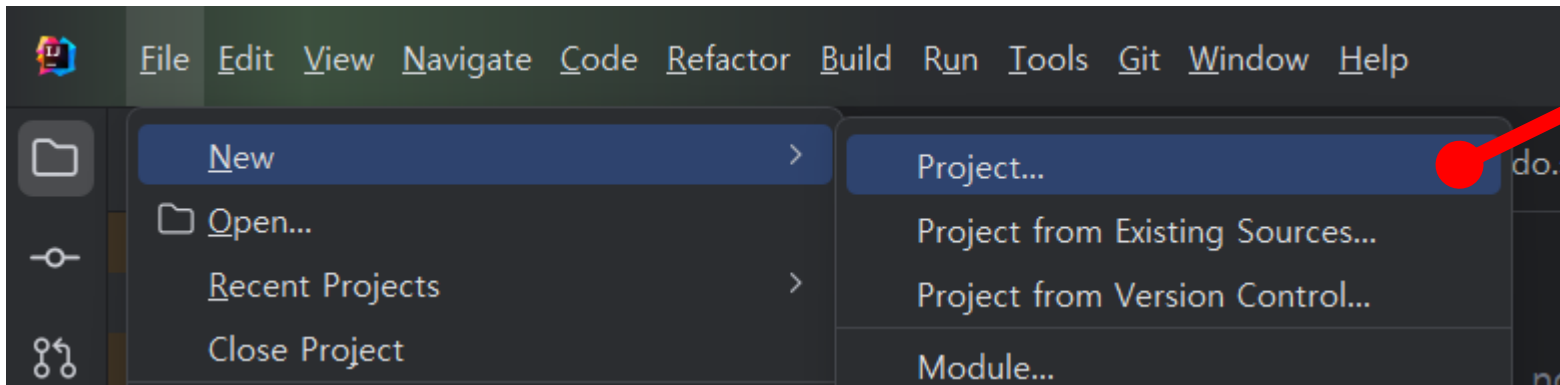
# 실습 2

# Gradle 프로젝트 생성 및 DB 접속하기

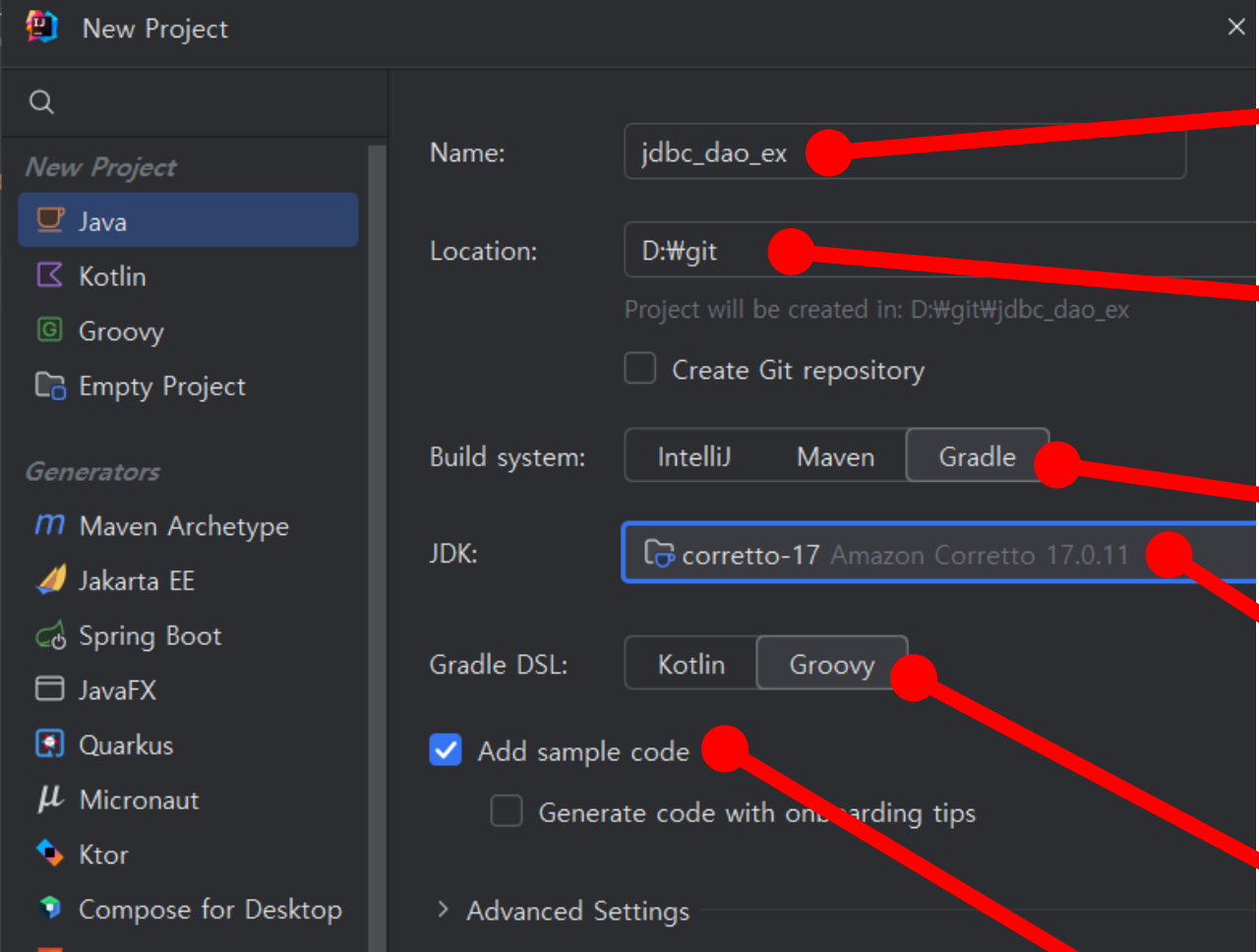


# Gradle 프로젝트

생성 및 설정 설명



메뉴 → File → New  
→ Project 를 선택



원하는 프로젝트 명 입력



저장할 폴더 설정

빌드 시스템은  
Gradle 선택

JDK 는 17버전 중 아무거나 선택  
(corretto 17 추천)

Groovy 선택

Add Sample code 만 체크



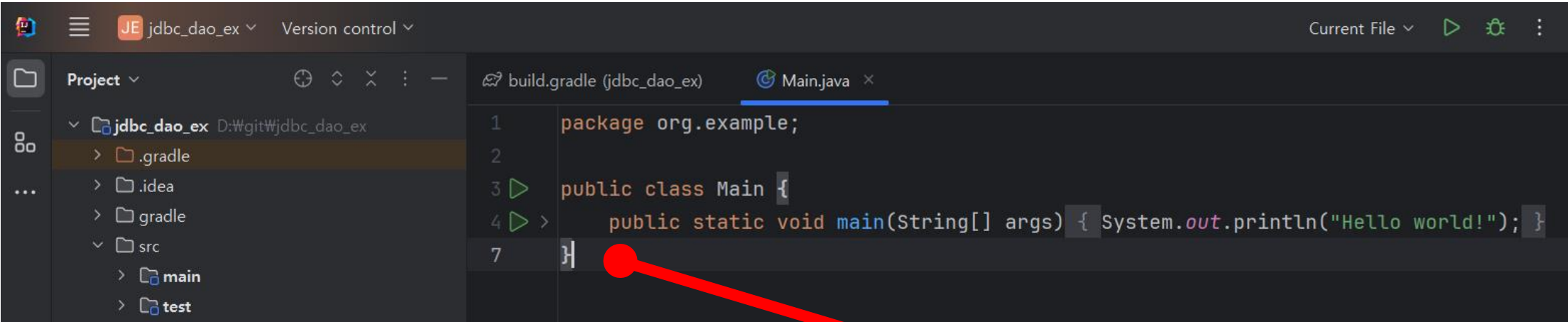
> Advanced Settings

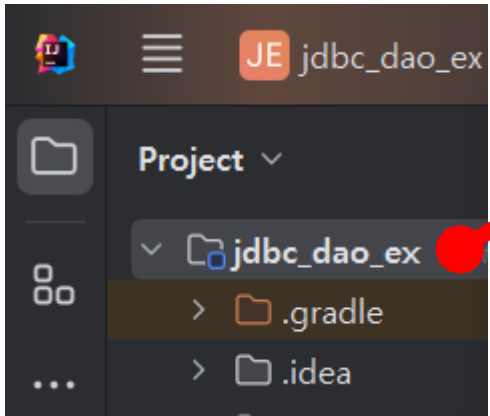
Create

Cancel

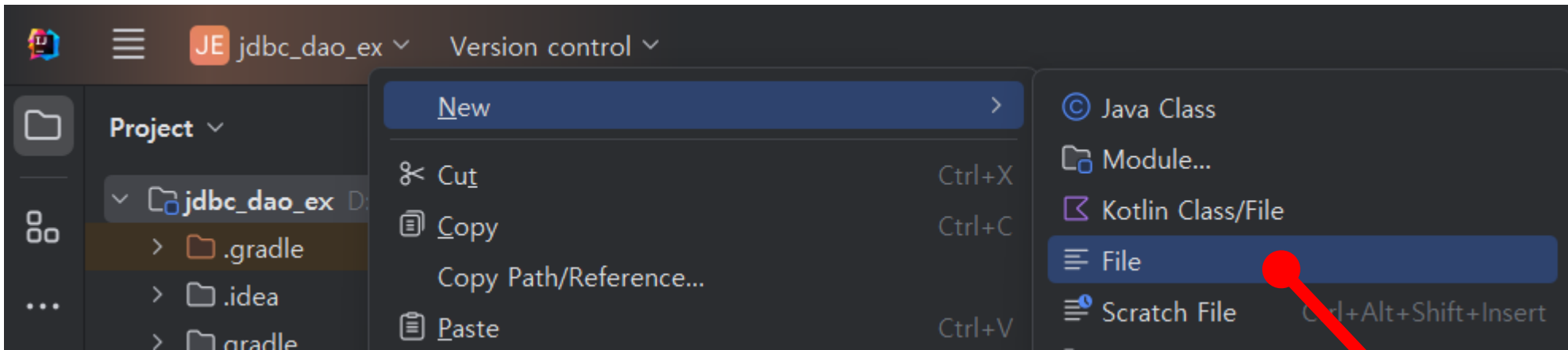
Create 로 프로젝트 생성







프로젝트 최상단 폴더에서 우클릭!



New → File 선택



New File

user.sql

user.sql 파일 생성

build.gradle (jdbc\_dao\_ex)

Main.java

user.sql

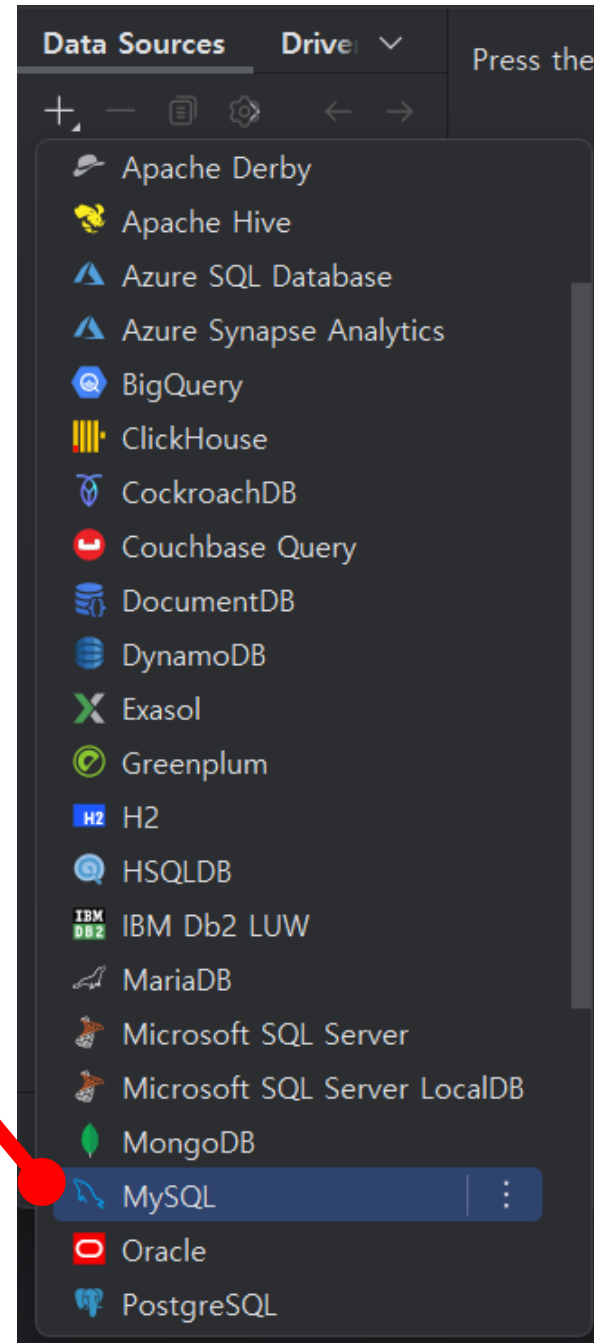
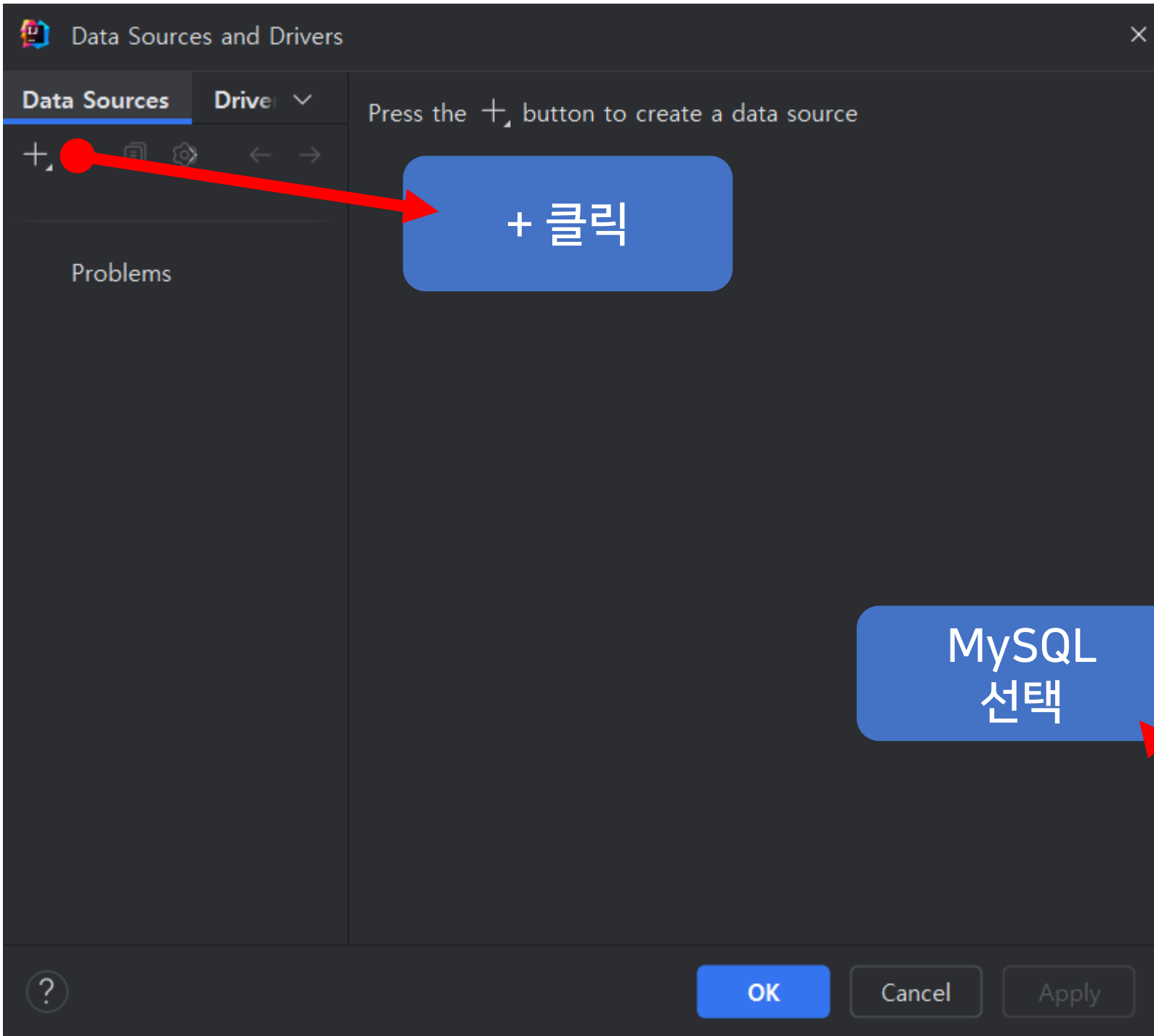
⚠ No data sources are configured to run this SQL and provide advanced code assistance.

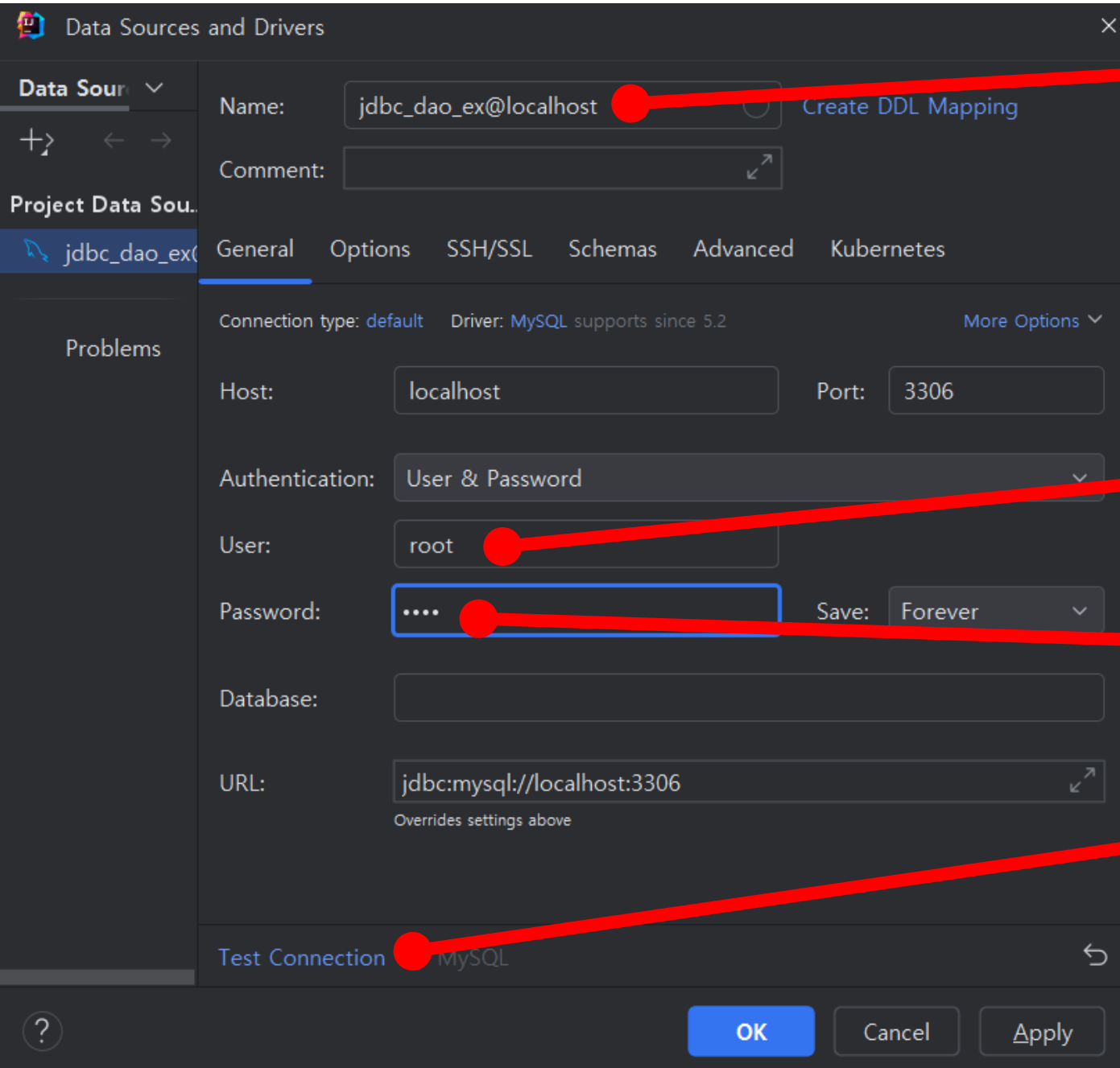
Configure data source

⚠ SQL dialect is not configured.

Change dialect to...

Configure data source 선택





## 원하는 DB 이름 정해주기 (아무거나 해도 됩니다!)




## MySQL 의 root 아이디 사용

## root 아이디의 비번 입력

## Test Connect 클릭

Succeeded

Copy

DBMS: MySQL (ver. 8.0.28) 

Case sensitivity: plain=lower, delimited=lower

Driver: MySQL Connector/J (ver. mysql-connector-j-8.2.0 (Revision: 06a1f724497fd81c6a659131fda822c9e5085b6c), JDBC4.2)


Ping: 20 ms

SSL: yes


성공 메시지가 뜨는지 확인

메시지가 안뜨는 경우  
비밀번호, MySQL 실행 여부  
등등 체크


테스트 성공 후  
OK 클릭

Authentication: User & Password 


User:

Password:  Save: Forever 

Database:

URL:  

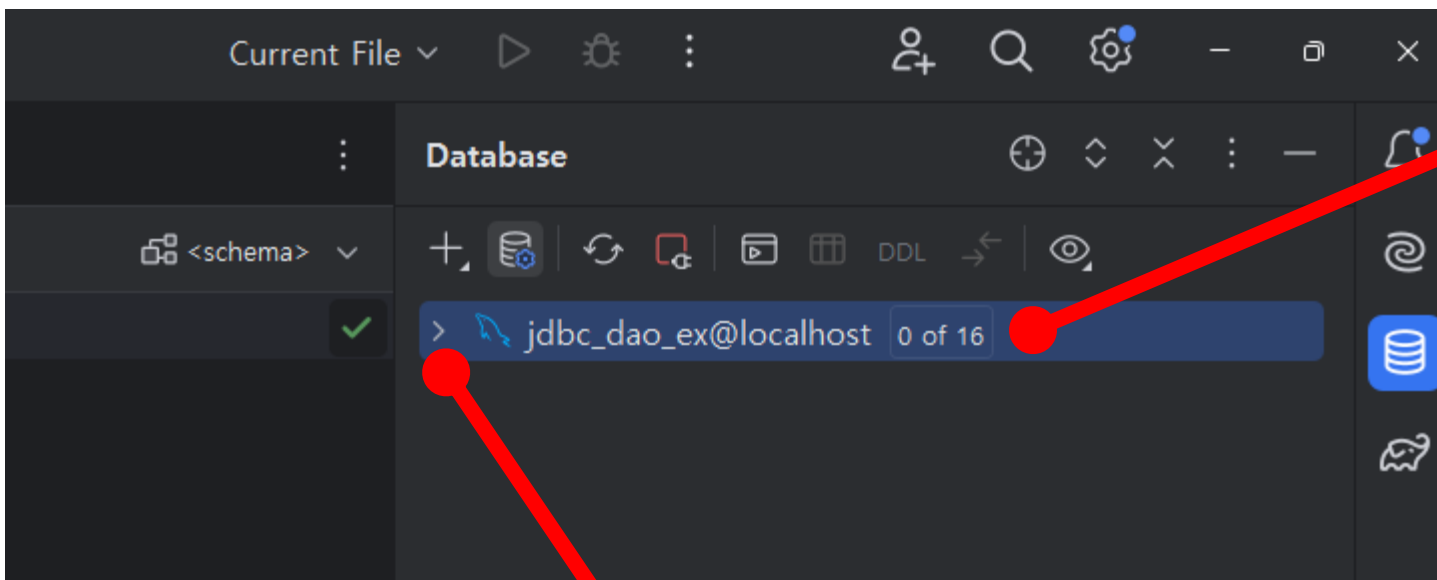
Overrides settings above

Test Connection MySQL 

OK

Cancel

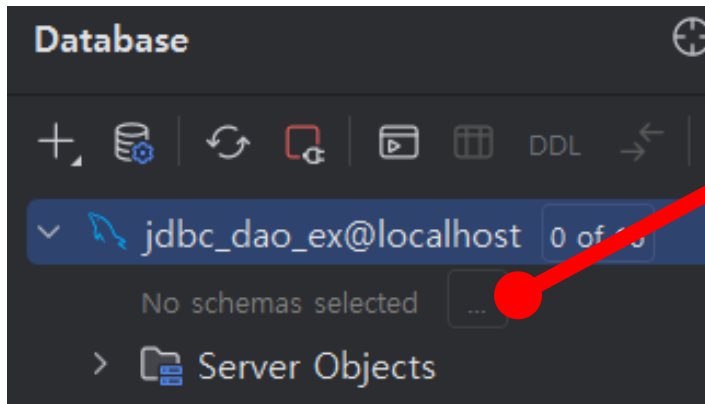
Apply



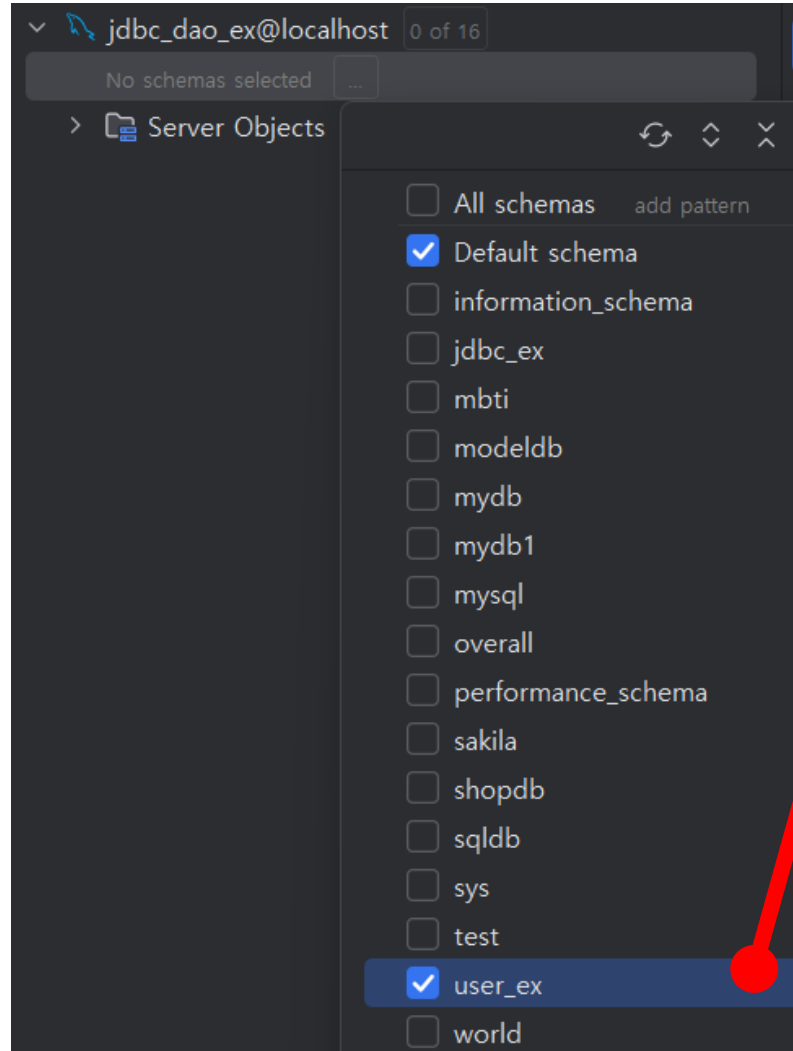
생성 된 DB 접속 정보 확인 🙌

> 버튼 클릭해서  
접속 정보 열기





... 클릭해서 스키마 선택 창 열기



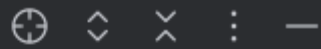
사용할 스키마 목록에서  
user\_ex 데이터베이스 선택

선택을 완료 하면  
스키마 선택 창 바깥쪽 한번  
클릭



선택 된 데이터베이스 및  
테이블 확인하기

Database



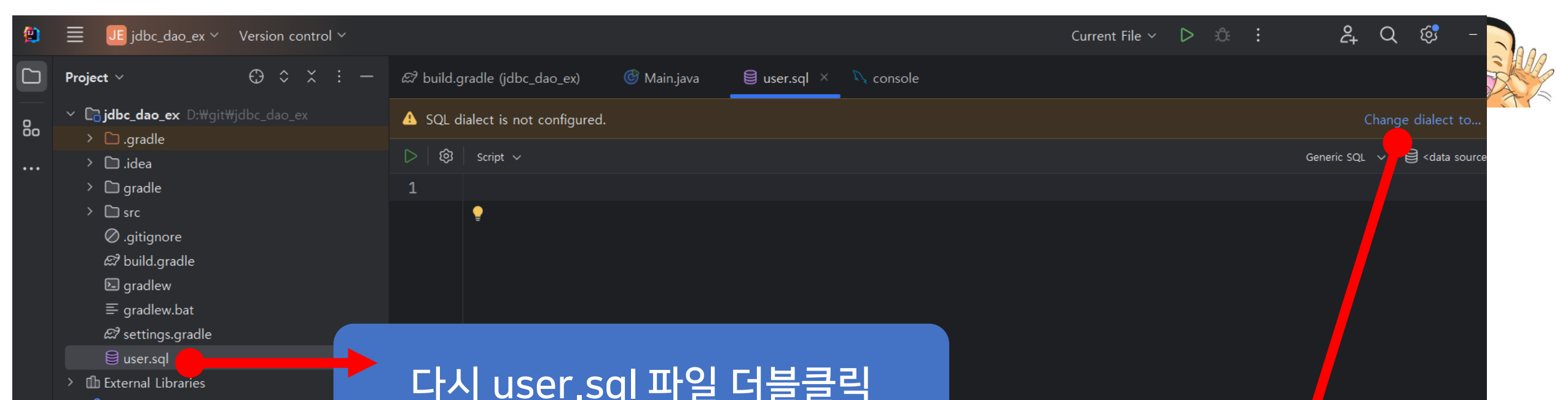
▼ jdbc\_dao\_ex@localhost 1 of 1

▼ user\_ex

▼ tables 2

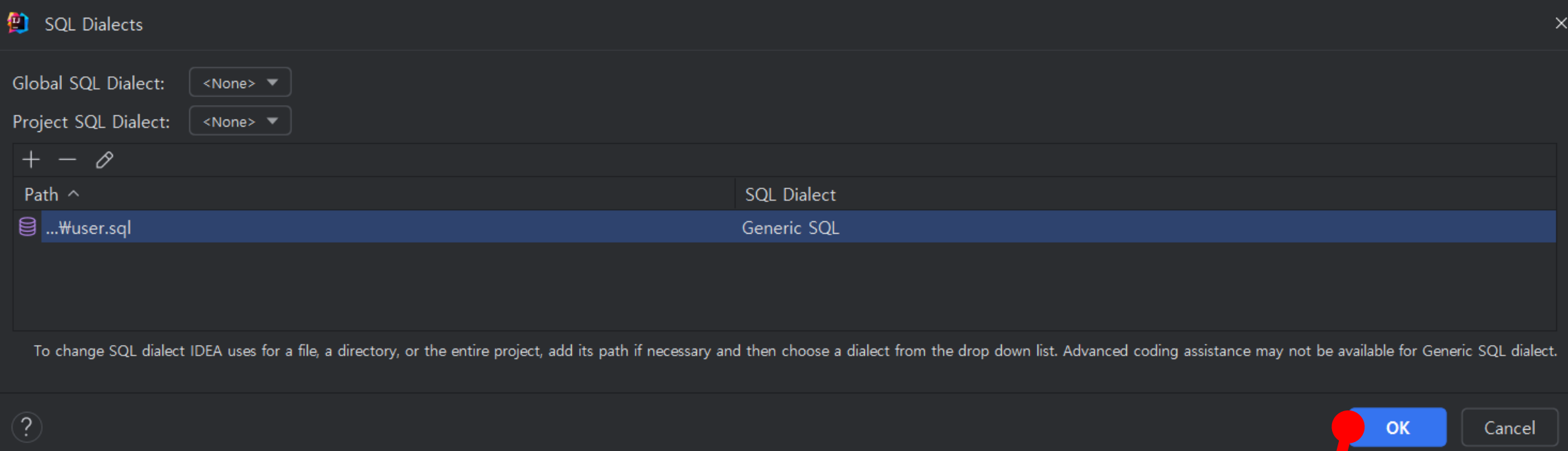
> users

> Server Objects



다시 user.sql 파일 더블클릭

sql 파일 화면에서  
Change dialect to ... 선택



해당 프로젝트에서  
user.sql 을 메인 sql 로 쓰겠다는 설정

→ OK 클릭

```
build.gradle (jdbc_dao_ex)  Main.java  user.sql  console
Script
1  # user_ex 데이터 베이스 생성
2  CREATE DATABASE user_ex;
3
4  # user_ex 데이터 베이스 사용
5  USE user_ex;
6
7  # user 테이블 생성
8  CREATE TABLE user
9  (
10     id        INT AUTO_INCREMENT PRIMARY KEY,
11     email     VARCHAR(100) UNIQUE NOT NULL,
12     password  VARCHAR(100)      NOT NULL
13 );
14
15 # 생성한 user 테이블 확인
16 SELECT *
17 FROM users;
18
19 # user 테이블에 회원 정보 삽입
20 # id 값은 자동으로 생성되므로 전달 필요 x
21 INSERT INTO user (email, password)
22 VALUES ('tetz', '1234'),
23        ('siwan', '1234'),
24        ('na', '1234');
```

워크벤치를 왔다 갔다 하면서 작업하기  
귀찮으므로 user.sql 에  
실습 1에서 작업한 쿼리문을 전부 붙여넣기!

USER user\_ex; 에 커서를 가져다 놓고  
컨트롤 + Enter 키로 해당 구문 실행

SELECT 문 하나 실행

	id	email	password
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234

테이블이 보이면 세팅 완료!!



# DB 접속하기 설명1

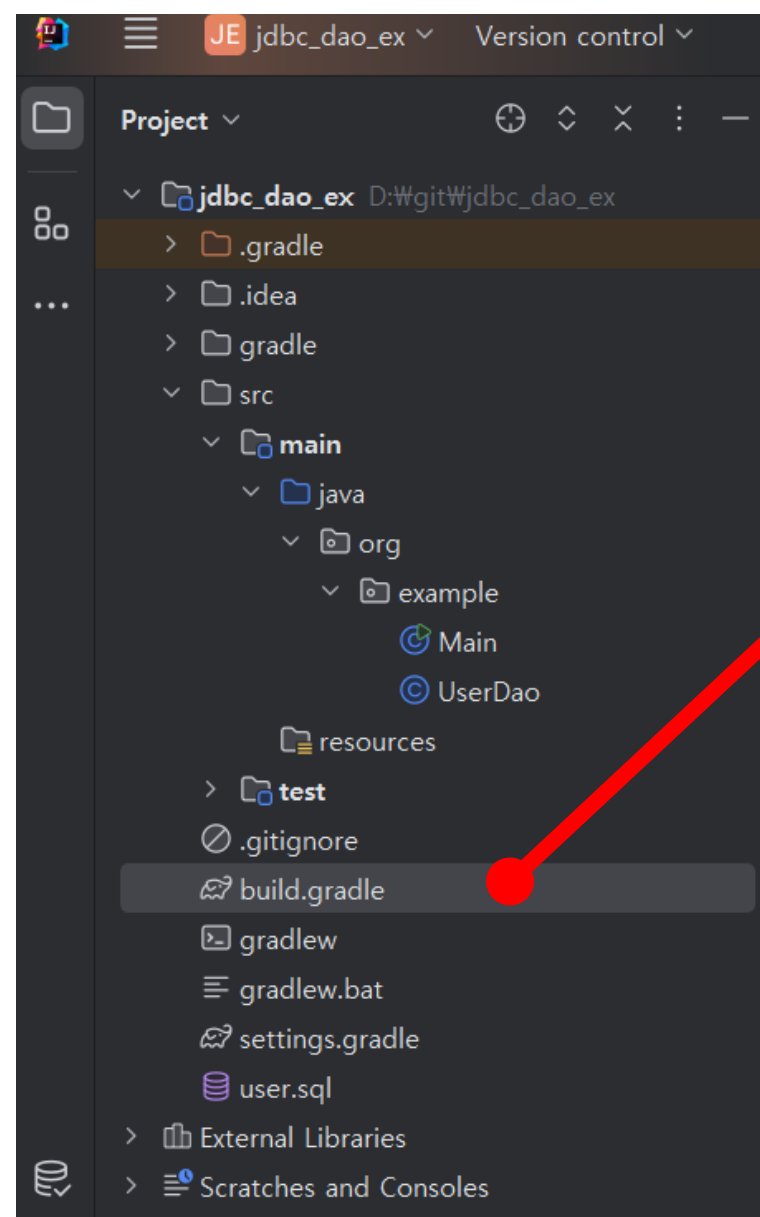
- UserDao 만들기



자바에서 DB에 접속하기 위해서는  
별도의 라이브러리가 필요합니다!

우리는 프로젝트 관리 툴로  
Gradle 을 사용하므로  
build.gradle 파일을 열어 줍니다

해당 파일은 현재 프로젝트에 대한 설정 정보가  
관리되는 파일로 해당 파일을 기반으로  
프로젝트가 설정 됩니다!





```
1 plugins {  
2     id 'java'  
3 }  
4  
5 group = 'org.example'  
6 version = '1.0-SNAPSHOT'  
7  
8 repositories {  
9     mavenCentral()  
10 }  
11  
12 dependencies {  
13     testImplementation platform('org.junit:junit-bom:5.10.0')  
14     testImplementation 'org.junit.jupiter:junit-jupiter'  
15 }  
16  
17 test {  
18     useJUnitPlatform()  
19 }
```

프로젝트 관련 정보

프로젝트에서 사용하는 라이브러리에 관리 정보

여기에 MySQL 과 통신을 도와주는 라이브러리와 Lombok 을 설치해 줘야 합니다!

```
6 version = '1.0-SNAPSHOT'
```

```
7  
8 repositories {  
9     mavenCentral()  
10 }  
11
```

```
12 dependencies {  
13     testImplementation platform('org.junit:junit-bom:5.10.0')  
14     testImplementation 'org.junit.jupiter:junit-jupiter'  
15     // MySQL 접속용 라이브러리  
16     implementation 'com.mysql:mysql-connector-j:8.3.0'  
17     // 롬복 관련 라이브러리  
18     compileOnly 'org.projectlombok:lombok:1.18.30'  
19     annotationProcessor 'org.projectlombok:lombok:1.18.30'  
20     testCompileOnly 'org.projectlombok:lombok:1.18.30'  
21     testAnnotationProcessor 'org.projectlombok:lombok:1.18.30'  
22 }
```

dependencies 항목에  
아래의 라이브러리 정보 코드를 추가!  
요 코드는 과제 올릴 때 코드 블록으로 같이 올렸으니  
그걸 붙여 넣으시면 됩니다!

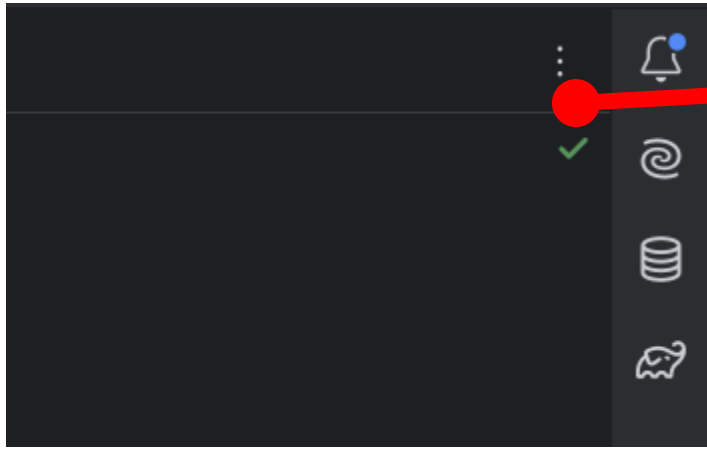
```
6 version = '1.0-SNAPSHOT'
```

```
7  
8 repositories {  
9     mavenCentral()  
10 }  
11
```

```
12 dependencies {  
13     testImplementation platform('org.junit:junit-bom:5.10.0')  
14     testImplementation 'org.juni          :  
15     // MySQL 접속용 라이브러리  
16     implementation 'com.mysql:mysql      ✓  
17     // 롬복 관련 라이브러리  
18     compileOnly 'org.projectlomb  
19     annotationProcessor 'org.pro  
20     testCompileOnly 'org.projectl  
21     testAnnotationProcessor 'org.projectlombok:lombok:1.18.30'  
22 }
```

새롭게 추가 된 라이브러리 정보 반영을 위해서  
코끼리 아이콘 클릭(Gradle 을 다시 빌드)





빌드 후 체크 표시 확인

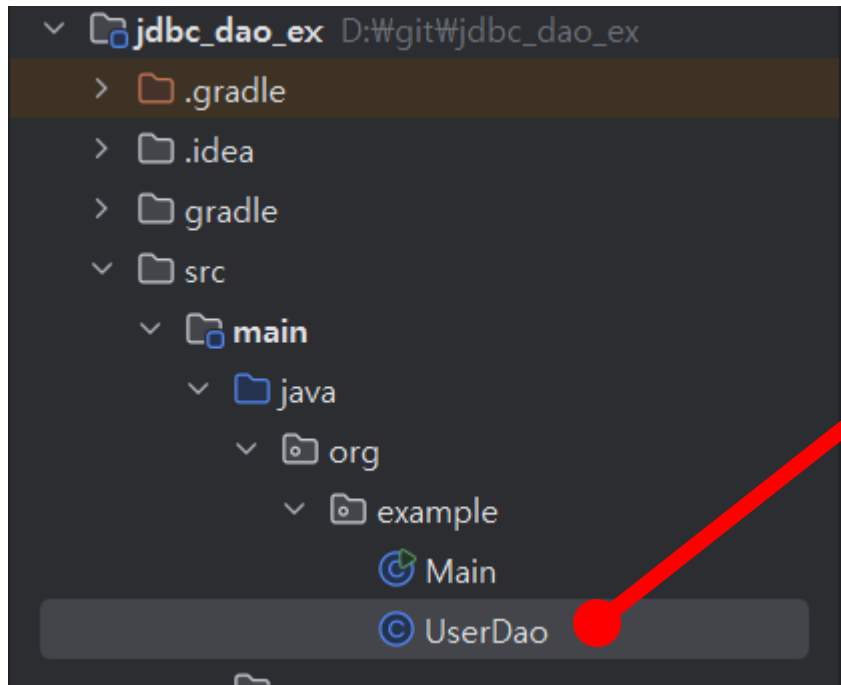




## 자동으로 생성된 example 패키지에 UserDao 클래스를 생성

example 패키지 및 Main 클래스가 없는 분은  
프로젝트 생성 시, Add Sample Code 옵션을  
체크 안해서 그런 것이니 org.example 패키지를  
생성 하신 다음 따라하시면 됩니다!

Main 클래스는 필요 없어요~!



```
1 package org.example;  
2  
3 public class UserDao {  
4 }  
5
```

생성된 UserDao 클래스는 이제  
DAO(Data Access Object) 로써  
실습 1에서 만든 user 데이터 베이스와의  
통신을 전담할 예정입니다!



```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

```
public class UserDao { no usages
    static Connection conn = null; 5 usages
```

```
    static {
```

```
        try {
```

```
            // 데이터베이스 연결 정보 문자열로 작성한 버전
```

```
            String driver = "com.mysql.cj.jdbc.Driver";
```

```
            String url = "jdbc:mysql://localhost:3306/yourDatabaseName";
```

```
            String id = "yourUsername";
```

```
            String password = "yourPassword";
```

```
            Class.forName(driver);
```

```
            conn = DriverManager.getConnection(url, id, password);
```

```
        } catch (Exception e) {
```

```
            e.printStackTrace();
```

```
        }
```

```
    }
```

DB 접속에 필요한 클래스들을 임포트  
요건 직접 타이핑 할 필요 X

아래 코드를 타이핑 하면서  
필요할 때 인텔리제이를 통해  
추가하면 됩니다!

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

```
public class UserDao { no usages
    static Connection conn = null; usages
```

```
    static {
```

```
        try {
```

```
            // 데이터베이스 연결 정보 문자열로 작성한 버전
```

```
            String driver = "com.mysql.cj.jdbc.Driver";
```

```
            String url = "jdbc:mysql://localhost:3306/yourDatabaseName";
```

```
            String id = "yourUsername";
```

```
            String password = "yourPassword";
```

```
            Class.forName(driver);
```

```
            conn = DriverManager.getConnection(url, id, password);
```

```
        } catch (Exception e) {
```

```
            e.printStackTrace();
```

```
        }
```

```
    }
```

접속 정보를 저장하는 Connection 객체  
당연히 처음에는 접속이 안되어 있으므로  
빈 값인 null 을 가진다



```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

```
public class UserDao { no usages
```

```
    static Connection conn = null; 5 usages
```

```
    static {
```

```
        try {
```

```
            // 데이터베이스 연결 정보 문자열로 작성한 버전
```

```
            String driver = "com.mysql.cj.jdbc.Driver";
```

```
            String url = "jdbc:mysql://localhost:3306/yourDatabaseName";
```

```
            String id = "yourUsername";
```

```
            String password = "yourPassword";
```

```
            Class.forName(driver);
```

```
            conn = DriverManager.getConnection(url, id, password);
```

```
        } catch (Exception e) {
```

```
            e.printStackTrace();
```

```
        }
```

```
    }
```

static 키워드를 통하여  
자바 프로그램이 시작되면  
UserDao 를 인스턴스화가 안되어도  
JVM 에 의해 해당 코드가 실행

DB 서버 접속 시에는 어떤 예외가  
발생할지 모르므로

try / catch 문을 사용하여  
예외를 처리 → 프로그램이 터지는 것을 방지

```
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;
```

```
public class UserDao { no usages
```

```
    static Connection conn = null; 5 usages
```

```
    static {
```

```
        try {
```

```
            // 데이터베이스 연결 정보 문자열로 작성한 버전
```

```
            String driver = "com.mysql.cj.jdbc.Driver";
```

```
            String url = "jdbc:mysql://localhost:3306/yourDatabaseName";
```

```
            String id = "yourUsername";
```

```
            String password = "yourPassword";
```

```
            Class.forName(driver);
```

```
            conn = DriverManager.getConnection(url, id, password);
```

```
        } catch (Exception e) {
```

```
            e.printStackTrace();
```

```
        }
```

```
    }
```



```
public class UserDao { no usages
    static Connection conn = null; 5 usages
    static {
        try {
            // 데이터베이스 연결 정보 문자열로 작성한 버전
            // DB 접속에 필요한 Driver 클래스를 지정하는 문자열
            String driver = "com.mysql.cj.jdbc.Driver";
            // DB 에서 어떤 데이터베이스에 접속할지 정하는 문자열, 우리는 user_ex 를 선택해야 한다
            String url = "jdbc:mysql://localhost:3306/user_ex";
            // 접속 계정은 root 를 사용하므로 id 는 root 로 설정, 비번은 각자의 비번에 맞게 입력
            String id = "root";
            String password = "각자 지정한 비번을 입력";

            Class.forName(driver);
            conn = DriverManager.getConnection(url, id, password);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

DB 접속에 필요한 정보를  
문자열로 전달하기 위해  
문자열로 필요한 값을 넣어 준다

비번 꼭 자신이 설정한 비번으로  
넣어주세요!!!



```
public class UserDao { no usages
    static Connection conn = null; 5 usages
    static {
        try {
            // 데이터베이스 연결 정보 문자열로 작성한 버전
            // DB 접속에 필요한 Driver 클래스를 지정하는 문자열
            String driver = "com.mysql.cj.jdbc.Driver";
            // DB 에서 어떤 데이터베이스에 접속할지 정하는 문자열, 우리는 user_ex 를 선택해야 한다
            String url = "jdbc:mysql://localhost:3306/user_ex";
            // 접속 계정은 root 를 사용하므로 id 는 root 로 설정, 비번은 각자의 비번에 맞게 입력
            String id = "root";
            String password = "각자 지정한 비번을 입력";

            Class.forName(driver);
            conn = DriverManager.getConnection(url, id, password);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

문자열로 지정한 정보를 바탕으로  
서버에 접속을 시도



```
public class UserDao { no usages
    static Connection conn = null; 6 usages
    static {
        try {
            // 데이터베이스 연결 정보 문자열로 작성한 버전
            // DB 접속에 필요한 Driver 클래스를 지정하는 문자열
            String driver = "com.mysql.cj.jdbc.Driver";
            // DB 에서 어떤 데이터베이스에 접속할지 정하는 문자열, 우리는 user_ex 를 선택해야 한다
            String url = "jdbc:mysql://localhost:3306/user_ex";
            // 접속 계정은 root 를 사용하므로 id 는 root 로 설정, 비번은 각자의 비번에 맞게 입력
            String id = "root";
            String password = "각자 지정한 비번을 입력";

            Class.forName(driver);
            conn = DriverManager.getConnection(url, id, password);

            if (conn != null) {
                System.out.println("DB 접속에 성공!");
            }
        }
    }
}
```

DB 접속에 성공하면 접속 정보 클래스가 반환되므로 conn 의 정보를 확인 후 접속이 성공했으면 성공 메시지 출력

```
public static Connection getConnection() {  
    return conn;  
}  
  
public static void close() {  
    try {  
        if (conn != null) {  
            conn.close();  
            conn = null;  
        }  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

다른 클래스에서 DB에 접속한 정보를 바탕으로 쿼리를 실행해야 하므로 getConnection 메서드를 통해서 접속에 성공한 정보를 담은 conn을 리턴 시킨다

DB 서버와의 접속을 종료 하고 싶으면 close() 메서드를 호출하면 conn의 접속 정보를 빈 값으로 바꿔서 접속을 종료 시킨다



# DB 접속하기 설명2

- UserDao 로

DB 접속하기

A screenshot of an IDE interface. On the left, a project tree shows the following structure: jdbc\_dao\_ex (D:\git\jdbc\_dao\_ex) > .gradle > .idea > gradle > src > main > java > org > example. Under 'example', there are three files: Main, UserDao, and UserMain. 'UserMain' is selected, highlighted with a red circle, and a red arrow points from it to a blue callout box on the right. The main editor area shows the following code:

```
1 package org.example;
2
3 public class UserMain { n
4 }
5
```

실제로 UserDao 를 운영할  
UserMain 클래스를 작성!



```
3 ▶ public class UserMain {  
4 ▶     public static void main(String[] args) {  
5     UserDao userDao = new UserDao();  
6     }  
7 }
```

바로 실행!!

UserDao 를 인스턴스화!

✓ jdbc\_dao\_ex [:Us 1 sec, 178 ms

오후 1:46:16: Executing ':UserMain.main()'...

> Task :compileJava

> Task :processResources NO-SOURCE

> Task :classes

> Task :UserMain.main()

DB 접속에 성공!



위와 같이 DB 접속에 성공 메시지가 뜨면 접속 성공!

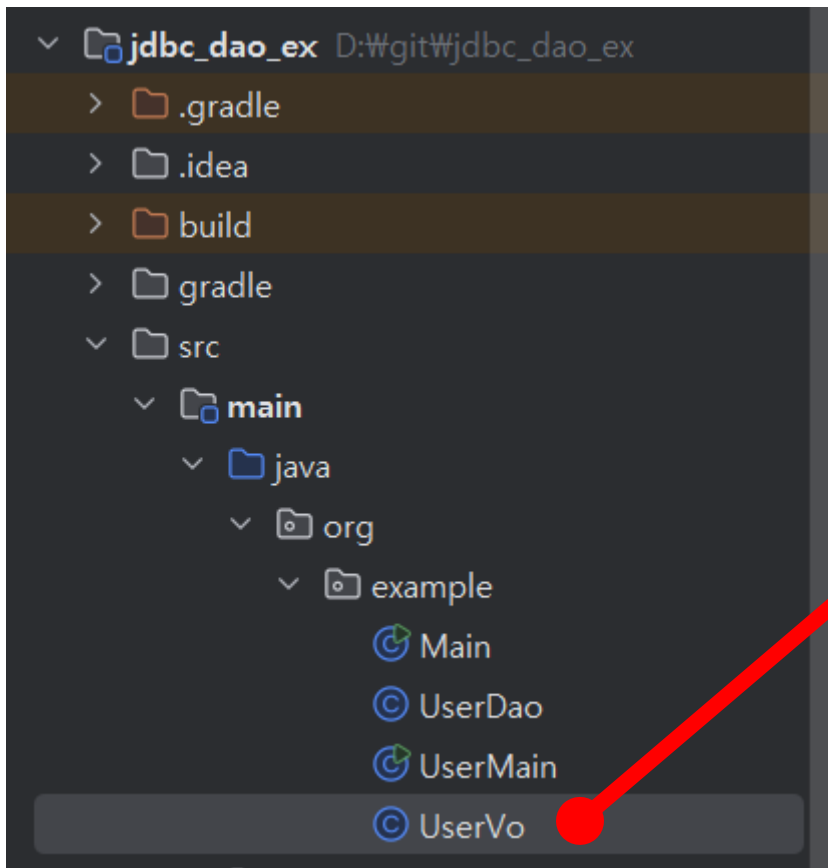
에러가 뜬다면!? DB 접속 정보를 확인하시거나  
Gradle 빌드가 잘 되었는지 확인 필요!

or 에러 메시지를 GPT 에 검색!  
or 저한테 편하게 DM 주세요 😊



# 실습 3

UserVo(Value  
Object) 구현하기



회원 정보를 편리하게 관리하기 위한  
UserVo(Value Object) 클래스를 만들어 봅시다!



현재 우리가 관리하는 회원 정보 테이블은 아래와 같습니다

	id	email	password
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234

id : 숫자  
email : 문자열  
password 문자열

따라서 해당 정보를 편하게 관리하기 위해서  
위의 테이블의 데이터를 객체로 관리할 수 있는 Vo 객체를 만들어 봅시다!

## 테이블과 1:1 로 데이터가 대칭되는 클래스를 선언



```
public class UserVo {  
    private final int id;  
    private final String email;  
    private final String password;  
}
```

	id	email	password
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234

당연히 테이블의 데이터 타입과 UserVo 객체의 데이터 타입은 일치해야만 합니다1

```
import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class UserVo {
    private final int id;
    private final String email;
    private final String password;
}
```

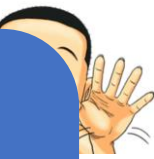
롬복에서 제공하는  
@Data 어노테이션을 추가해서  
UserVo 에 Getter/Setter 함수 자동 생성  
+ toString / equals / hashCode 메서드  
자동 오버라이딩



```
import lombok.AllArgsConstructor;
import lombok.Data;

@Data no usages
@AllArgsConstructor
public class UserVo {
    private final int id;
    private final String email;
    private final String password;
}
```

롬복에서 제공하는  
@AllArgsConstructor 를 추가해서  
UserVo 에 해당하는 생성자도 자동 생성!







```
3 ▶ public class UserMain {  
4 ▶   public static void main(String[] args) {  
5     UserDao userDao = new UserDao();  
6  
7     UserVo tetz = new UserVo(id: 1, email: "tetz", password: "1234");  
8  
9     System.out.println(tetz);  
10   }  
11 }
```

생성한 tetz 인스턴스를 출력

UserVo 테스트를 위해  
다시 UserMain 클래스로 돌아와서  
UserVo 클래스 객체를 생성!

```
> Task :UserMain.main()  
DB 접속에 성공!  
UserVo(id=1, email=tetz, password=1234)
```

생성 된, tetz 인스턴스의 정보를 확인!  
인스턴스의 주소가 아닌 정보가 나오는  
이유는 Lombok 의 @Data 어노테이션  
으로 인해서 toString 이 데이터를  
출력하도록 오버라이딩 되었기 때문!



# 실습 4

## User CRUD 구현하기



# 실습 4-1

## User C(reate) 구현

// 사용자 정보를 데이터베이스에 삽입하는 메서드

```
public void create(String email, String password) { no usages kdtTetz *
    String sql = "INSERT INTO users (email, password) VALUES (?, ?)";

    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(parameterIndex: 1, email);
        pstmt.setString(parameterIndex: 2, password);

        int affectedRows = pstmt.executeUpdate();

        if (affectedRows > 0) {
            System.out.println("회원 추가 성공!");
        } else {
            System.out.println("회원 추가 실패");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

회원 생성  
create() 메서드  
전체 코드



// 사용자 정보를 데이터베이스에 삽입하는 메서드

```
public void create(String email, String password) { no usages kdtTetz *
    String sql = "INSERT INTO users (email, password) VALUES (?, ?)";

    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(parameterIndex: 1, email);
        pstmt.setString(parameterIndex: 2, password);

        int affectedRows = pstmt.executeUpdate();

        if (affectedRows > 0) {
            System.out.println("회원 추가 성공!");
        } else {
            System.out.println("회원 추가 실패");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

users 테이블에 새로운 회원 데이터를  
생성해야 하는 상황입니다!

회원을 생성할 때마다 email 과  
password 정보는 변경이 예상됩니다!

따라서 PreparedStatement 를 사용하여  
편리함과 성능을 챙깁니다!



// 사용자 정보를 데이터베이스에 삽입하는 메서드

```
public void create(String email, String password) { no usages kdtTetz *
    String sql = "INSERT INTO users (email, password) VALUES (?, ?)";

    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(parameterIndex: 1, email);
        pstmt.setString(parameterIndex: 2, password);

        int affectedRows = pstmt.executeUpdate();

        if (affectedRows > 0) {
            System.out.println("회원 추가 성공!");
        } else {
            System.out.println("회원 추가 실패");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

회원 생성 시 마다 변경이 필요한  
email 과 password 값을 ? 로 두어서

쿼리를 보내기 직전에  
PreparedStatement 의 기능을 활용하여  
쿼리문을 완성하여 전달!



// 사용자 정보를 데이터베이스에 삽입하는 메서드

```
public void create(String email, String password) { no usages kdtTetz *
    String sql = "INSERT INTO users (email, password) VALUES (?, ?)";

    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(parameterIndex: 1, email);
        pstmt.setString(parameterIndex: 2, password);

        int affectedRows = pstmt.executeUpdate();

        if (affectedRows > 0) {
            System.out.println("회원 추가 성공!");
        } else {
            System.out.println("회원 추가 실패");
        }
    }
}
```

UserDao 클래스는 이미 DB 접속 정보를 가지고 있으므로 해당 접속 정보(= conn)에 PreparedStatement 를 준비 시킵니다!



// 사용자 정보를 데이터베이스에 삽입하는 메서드

```
public void create(String email, String password) {  
    String sql = "INSERT INTO users (email, password) VALUES (?, ?)";  
  
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {  
  
        pstmt.setString(parameterIndex: 1, email);  
        pstmt.setString(parameterIndex: 2, password);  
  
        int affectedRows = pstmt.executeUpdate();  
  
        if (affectedRows > 0) {  
            System.out.println("회원 추가 성공!");  
        } else {  
            System.out.println("회원 추가 실패");  
        }  
    }  
}
```

첫번째 물음표에 문자열인 email 값을 보내야 하므로 pstmt 에 문자열 값을 세팅하는 setString() 메서드를 사용합니다

첫번째 물음표를 채워야 하니까 1을 넣어주고 첫번째 물음표의 값으로는 매개 변수로 전달받은 email 을 넣어 줍니다





```
// 사용자 정보를 데이터베이스에 삽입하는 메서드
public void create(String email, String password) { no usages kdtTetz *
    String sql = "INSERT INTO users (email, password) VALUES (?, ?)";

    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(parameterIndex: 1, email);
        pstmt.setString(parameterIndex: 2, password);

        int affectedRows = pstmt.executeUpdate();

        if (affectedRows > 0) {
            System.out.println("회원 추가 성공!");
        } else {
            System.out.println("회원 추가 실패");
        }
    }
}
```

두번째 password 쿼리도  
email 과 같은 방식으로 전달하여  
쿼리문을 완성해 줍니다!

전달해야 하는 값이 문자열이 아닌  
숫자(int), 시간(Timestamp) 이면  
해당 타입에 맞는 메서드  
setInt, setTimestamp 등을 사용!



// 사용자 정보를 데이터베이스에 삽입하는 메서드

```
public void create(String email, String password) { no usages kdtTetz *
    String sql = "INSERT INTO users (email, password) VALUES (?, ?)";

    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(parameterIndex: 1, email);
        pstmt.setString(parameterIndex: 2, password);

        int affectedRows = pstmt.executeUpdate();

        if (affectedRows > 0) {
            System.out.println("회원 추가 성공!");
        } else {
            System.out.println("회원 추가 실패");
        }
    }
}
```

완성된 PreparedStatement 를  
DB에 보내는 executeUpdate 를 실행

	id	email	password
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234

생성 쿼리 수행 전의 테이블

	id	email	password
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234
4	4	tetz2	1234

성공적으로 쿼리가 수행된 상태의 테이블  
한 줄(row)의 데이터가  
생성 된 것을 확인 가능

executeUpdate 는 자신이 수행한  
쿼리로 인해서 몇 줄의 변화가 생겼는지를  
리턴해 줍니다!

지금은 1줄이 생성 되었으므로  
1을 리턴!!



// 사용자 정보를 데이터베이스에 삽입하는 메서드

```
public void create(String email, String password) {  
    String sql = "INSERT INTO users (email, password) VALUES (?, ?)";  
  
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {  
  
        pstmt.setString(parameterIndex: 1, email);  
        pstmt.setString(parameterIndex: 2, password);  
  
        int affectedRows = pstmt.executeUpdate();  
  
        if (affectedRows > 0) {  
            System.out.println("회원 추가 성공!");  
        } else {  
            System.out.println("회원 추가 실패");  
        }  
    }  
}
```

회원 등록이 성공하면  
새로운 데이터 1줄이 생성 될 것이므로  
1이 리턴 됩니다!

혹, 예외 발생으로 데이터가  
생성이 안된다면?

0이 리턴 되겠죠?



// 사용자 정보를 데이터베이스에 삽입하는 메서드

```
public void create(String email, String password) { no usages kdtTetz *  
    String sql = "INSERT INTO users (email, password) VALUES (?, ?)";  
  
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {  
  
        pstmt.setString(parameterIndex: 1, email);  
        pstmt.setString(parameterIndex: 2, password);  
  
        int affectedRows = pstmt.executeUpdate();  
  
        if (affectedRows > 0) {  
            System.out.println("회원 추가 성공!");  
        } else {  
            System.out.println("회원 추가 실패");  
        }  
    }  
}
```

affectedRows 가 1이면  
회원이 정상적으로 등록이 된 것이고  
0 이면 등록이 안된 것이므로  
해당 조건에 따라서 결과문을 출력

```
} catch (SQLException e) {  
    e.printStackTrace();  
}  
  
}
```

DB 관련 작업 중 예외가 발생하면  
받아주는 catch 구문



```
3  ▶ public class UserMain {
4  ▶  public static void main(String[] args) {
5  ▶      UserDao userDao = new UserDao();
6  ▶
7  ▶      userDao.create(email: "tetz2", password: "1234");
8  ▶  }
9  ▶ }
```

실제 회원 등록을 위해  
UserMain 클래스로 돌아옵니다!

userDao 에 만든 create 메서드에  
등록할 사용자의 email 과 password 를  
전달하여 회원 생성을 시도!

main 메서드를 실행하여  
결과를 확인!



✓ jdbc\_dao\_ex [:Use 1 sec, 63 ms

오후 1:59:30: Executing ':UserMain.main()'...

> Task :compileJava

> Task :processResources NO-SOURCE

> Task :classes

> Task :UserMain.main()

DB 접속에 성공!

회원 추가 성공!

create() 메서드 내부의  
"회원 추가 성공!" 이 출력!

→ 회원 추가가 잘 되었다는 의미!

	id	email	password
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234
4	4	tetz2	1234

실제 데이터도 잘 생성 된 것을 확인 가능!





# 실습 4-2

## User R(ead) 구현

// 모든 회원 정보를 조회하는 메서드

```
public void getAllUsers() { 1 usage  👤 kdtTetz *  
    List<UserVo> userList = new ArrayList<>();  
    String sql = "SELECT id, email, password FROM users";  
  
    try (Statement stmt = conn.createStatement();  
        ResultSet rs = stmt.executeQuery(sql)) {  
  
        while (rs.next()) {  
            int id = rs.getInt(columnLabel: "id");  
            String email = rs.getString(columnLabel: "email");  
            String password = rs.getString(columnLabel: "password");  
  
            UserVo user = new UserVo(id, email, password);  
            userList.add(user);  
        }  
        userList.forEach((user) -> System.out.println(user));  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

전체 회원 조회  
getAllUsers() 메서드  
전체 코드



```
// 모든 회원 정보를 조회하는 메서드  
public void getAllUsers() { 1 usage  
    List<UserVo> userList = new ArrayList<>();  
    String sql = "SELECT id, email, password FROM users";
```

users 테이블의 전체 데이터를  
가져오는 쿼리문 작성

전체 회원 정보의 수는 알 수 없으므로  
데이터의 추가/삭제가 용이한  
ArrayList 컬렉션을 선언

물론, ArrayList 의 데이터는  
UserVo 객체로 저장!

```
try (Statement stmt = conn.createStatement();  
    ResultSet rs = stmt.executeQuery(sql)) {
```

DB 서버와의 통신에서 발생 가능한  
예외에 대비하기 위해 try/catch 문을 사용

변경이 필요 없는 Statement 구문이므로  
별도의 세팅 없이 바로 DB로 보내서 실행!

이번에는 쿼리문이 변경될 필요가 없으므로  
PreparedStatement 가 아닌 일단  
Statement 사용



# ResultSet

## 데이터 설명



```
try (Statement stmt = conn.createStatement();  
    ResultSet rs = stmt.executeQuery(sql)) {
```



몇 개의 데이터가 들어올지 모르는 쿼리 구문을 실행 했기 때문에  
SQL 데이터를 받아주는 ResultSet 이라는 데이터에 저장!

	id	email	password
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234
4	4	tetz2	1234

현재 우리는 이렇게 생긴  
테이블을 가지고 있으므로  
ResultSet 도 왼쪽과 같은 형태의  
데이터로 생성이 됩니다!

단, ResultSet 은 자신의 전체 모습은 모르고 있으며 단순히 2가지 정보를 가진다!

1. 자신의 다음에 데이터가 존재 하는지 / 2. 한 줄의 데이터

데이터  
Cursor

	id	email	password
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234
4	4	tetz2	1234

rs.next() 값 : true

현재 rs(ResultSet) 의 데이터는 이런 모습을 가집니다!

데이터 Cursor 가 현재 자신의 위치를 가르키고 있으며  
자신이 가르키는 데이터 한 줄의 데이터와,  
다음 데이터가 있는지 여부를 rs.next() 메서드를 통해  
확인 가능!



아까의 상태에서 rs.next() 가 실행 되면 커서는 다음 줄로 넘어가게 되고  
우리는 다음 데이터에 접근이 가능합니다!

데이터  
Cursor

	id	email	password
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234
4	4	tetz2	1234

rs.next() 값 : true

데이터 커서가 마지막 데이터를 가르키게 되면 rs.next() 값이 false 가 되므로 while 을 사용한 전체 데이터 순회를 마칠 수 있다

데이터  
Cursor

	id ▾	email ▾	password ▾
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234
4	4	tetz2	1234

rs.next() 값 : false

```
while (rs.next()) {  
    int id = rs.getInt(columnLabel: "id");  
    String email = rs.getString(columnLabel: "email");  
    String password = rs.getString(columnLabel: "password");  
}
```

while 의 조건인  
rs.next() 가 false 가 되므로  
while 문을 이용한 데이터 순회가  
종료된다!

rs 데이터의 각각의 값에 대한 접근은 rs.getInt / rs.getString 등의 메서드를 이용하여 접근합니다!

데이터  
Cursor

	id	email	password
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234
4	4	tetz2	1234

rs.next() 값 : true

rs 에서 데이터를 꺼낼 때에는 각각의 데이터 타입에 맞는 메서드를 사용해야 하며, 메서드에는 데이터가 있는 column 의 이름을 전달해야 한다

rs 에서 데이터를 꺼낼 때에는 각각의 데이터 타입에 맞는 메서드를 사용해야 하며, 메서드에는 데이터가 있는 column 의 명을 전달해야 합니다!

데이터  
Cursor

	id ▾	email ▾	password ▾
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234
4	4	tetz2	1234

rs.next() 값 : true

```
int id = rs.getInt(columnLabel: "id");  
String email = rs.getString(columnLabel: "email");  
String password = rs.getString(columnLabel: "password");
```

column 이 id 인 데이터의 값을 꺼내기  
→ 현재 데이터 커서는 1번째 줄을 가르키므로 1 이라는 숫자 값을 가져옵니다

1은 숫자 이므로 rs.getInt 를 통해  
값을 int 로 가져 옵니다!

rs 에서 데이터를 꺼낼 때에는 각각의 데이터 타입에 맞는 메서드를 사용해야 하며, 메서드에는 데이터가 있는 column 의 명을 전달해야 합니다!

데이터  
Cursor

	id	email	password
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234
4	4	tetz2	1234

rs.next() 값 : true

```
int id = rs.getInt(columnLabel: "id");  
String email = rs.getString(columnLabel: "email");  
String password = rs.getString(columnLabel: "password");
```

column 이 email 인 데이터의 값을 꺼내기  
→ 현재 데이터 커서는 1번째 줄을 가르키므로 "tetz" 이라는 문자열을 가져옵니다

"tetz" 은 문자열이므로 rs.getString 을 통해 값을 문자열로 가져옵니다!

# 다시 코드로!



```
try (Statement stmt = conn.createStatement();
     ResultSet rs = stmt.executeQuery(sql)) {

    while (rs.next()) {
        int id = rs.getInt(columnLabel: "id");
        String email = rs.getString(columnLabel: "email");
        String password = rs.getString(columnLabel: "password");

        UserVo user = new UserVo(id, email, password);
        userList.add(user);
    }
    userList.forEach((user) -> System.out.println(user));
}
```

쿼리를 실행하고  
DB 로 부터 데이터를 받아서  
ResultSet 데이터에 받는 부분

while 문과 rs.next 를 사용하여  
ResultSet 의 전체 데이터를  
순회하는 while 문



```
try (Statement stmt = conn.createStatement();
     ResultSet rs = stmt.executeQuery(sql)) {

    while (rs.next()) {
        int id = rs.getInt(columnLabel: "id");
        String email = rs.getString(columnLabel: "email");
        String password = rs.getString(columnLabel: "password");

        UserVo user = new UserVo(id, email, password);
        userList.add(user);
    }
    userList.forEach((user) -> System.out.println(user));
}
```

rs 로 부터 데이터를 꺼내서  
각각의 변수에 담는 부분

rs 로 부터 가져온 데이터를  
UserVo 생성자에 전달하여  
UserVo 타입 객체로 만드는 코드





```
try (Statement stmt = conn.createStatement();
     ResultSet rs = stmt.executeQuery(sql)) {

    while (rs.next()) {
        int id = rs.getInt(columnLabel: "id");
        String email = rs.getString(columnLabel: "email");
        String password = rs.getString(columnLabel: "password");

        UserVo user = new UserVo(id, email, password);
        userList.add(user);
    }
    userList.forEach((user) -> System.out.println(user));
}
```

ResultSet 이 아닌  
JAVA 에서 컨트롤이 가능한  
데이터로 저장하기 위해

ArrayList 에 방금 생성 된  
회원 객체를 추가!

while 문이 종료 되면 전체 데이터가  
전부 ArrayList 에 저장 되었을 것이므로  
ArrayList 에 저장된 UserVo 객체를 출력!

```
} catch (SQLException e) {  
    e.printStackTrace();  
}  
  
}
```

DB 관련 작업 중 예외가 발생하면  
받아주는 catch 구문



```
public class UserMain {  
    public static void main(String[] args) {  
        UserDao userDao = new UserDao();  
  
        // 회원 추가  
        // 회원이 계속 추가되면 귀찮으므로 주석 처리  
        // userDao.create("tetz2", "1234");  
  
        // 전체 회원 조회  
        userDao.getAllUsers();  
    }  
}
```

방금 작성한 getAllUsers 실행을 위해  
UserMain 클래스로 돌아오기!

방금 작성한 getAllUsers 실행!!

✓ jdbc\_dao\_ex [:Us 7 sec, 422 ms

오후 7:29:43: Executing ':UserMain.main()'...

Starting Gradle Daemon...

Gradle Daemon started in 1 s 278 ms

> Task :compileJava

> Task :processResources NO-SOURCE

> Task :classes

> Task :UserMain.main()

DB 접속에 성공!

UserVo(id=1, email=tetz, password=1234)

UserVo(id=2, email=siwan, password=1234)

UserVo(id=3, email=na, password=1234)

UserVo(id=4, email=tetz2, password=1234)

정상적으로 회원 목록이 잘 출력 되는 것  
확인 가능!!





# 실습 4-3

## User U(pdate) 구현

// 회원 정보를 수정하는 메서드

```
public void updateUser(int id, String newEmail, String newPassword) {  
    String sql = "UPDATE users SET email = ?, password = ? WHERE id = ?";  
  
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {  
  
        pstmt.setString(parameterIndex: 1, newEmail);  
        pstmt.setString(parameterIndex: 2, newPassword);  
        pstmt.setInt(parameterIndex: 3, id);  
  
        int affectedRows = pstmt.executeUpdate();  
  
        if (affectedRows > 0) {  
            System.out.println("회원 정보 수정 성공!");  
        } else {  
            System.out.println("회원 정보 수정 실패");  
        }  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

회원 정보 수정  
updateUser 메서드  
전체 코드



```
// 회원 정보를 수정하는 메서드
```

```
public void updateUser(int id, String newEmail, String newPassword) { no use  
    String sql = "UPDATE users SET email = ?, password = ? WHERE id = ?";
```

회원 정보 update 를 위한  
기본 쿼리문 작성

id, email, password 값은  
수정할 때마다 변경 될 것이므로  
PreparedStatement 사용을 위해  
? 로 남겨두기

숫자 id 가 일치하는 회원의  
email, password 를 수정하는 쿼리!

```
try (PreparedStatement pstmt = conn.prepareStatement(sql)) {  
  
    pstmt.setString(parameterIndex: 1, newEmail);  
    pstmt.setString(parameterIndex: 2, newPassword);  
    pstmt.setInt(parameterIndex: 3, id);  
}
```

PreparedStatement 준비

준비된 PreparedStatement 에  
각각의 ? 값을 매개변수로 받은  
newEmail, newPassword,  
id 값으로 세팅



```
int affectedRows = pstmt.executeUpdate();  
  
if (affectedRows > 0) {  
    System.out.println("회원 정보 수정 성공!");  
} else {  
    System.out.println("회원 정보 수정 실패");  
}
```

회원 수정 쿼리문을 DB로 보내서 실행!

회원 정보 수정이 정상적으로 수행되면  
1줄의 데이터가 변경 될 것이므로  
해당 데이터의 개수를 받아서  
affectedRows 에 저장하기!



```
int affectedRows = pstmt.executeUpdate();

if (affectedRows > 0) {
    System.out.println("회원 정보 수정 성공!");
} else {
    System.out.println("회원 정보 수정 실패");
}
```

affectedRows 의 숫자가 1이면  
회원 수정 성공 1이 아니면 실패이므로  
상황에 맞는 문장을 출력!

```
} catch (SQLException e) {  
    e.printStackTrace();  
}  
  
}
```

DB 관련 작업 중 예외가 발생하면  
받아주는 catch 구문



방금 작성한 getAllUsers 실행을 위해  
UserMain 클래스로 돌아오기!

```
public class UserMain {  
    public static void main(String[] args) {  
        UserDao userDao = new UserDao();  
  
        // 회원 추가  
        userDao.create(email: "tetz2", password: "1234");  
        // 전체 회원 조회  
        userDao.getAllUsers();  
    }  
}
```



	id	email	password
1	1	tetz	1234
2	2	siwan	1234
3	3	na	1234
4	4	tetz2	1234

회원 id 가 4이고 email 이 tetz2,  
password 가 1234 인 회원의 정보를  
email 을 lhs, password 를 abcd 로 수정

불필요한 회원 추가 및 콘솔 출력을  
막기 위해서 주석 처리

```
public class UserMain {
    public static void main(String[] args) {
        UserDao userDao = new UserDao();
        // 회원 추가
        // 회원이 계속 추가되면 귀찮으므로 주석 처리
        // userDao.create("tetz2", "1234");
        // 전체 회원 조회
        // userDao.getAllUsers();

        // 회원 수정 메서드 실행
        userDao.updateUser(id: 4, newEmail: "lhs", newPassword: "abcd");
        // 회원 수정이 성공적으로 되었는지 목록 확인
        userDao.getAllUsers();
    }
}
```

회원 수정 메서드 수행!

수정이 정상적으로 이루어졌는지  
확인하기 위해 전체 목록 조회

✓ jdbc\_dao\_ex [:Use 1 sec, 98 ms

> Task :classes

> Task :UserMain.main()

DB 접속에 성공!

회원 정보 수정 성공!

UserVo(id=1, email=tetz, password=1234)

UserVo(id=2, email=siwan, password=1234)

UserVo(id=3, email=na, password=1234)

UserVo(id=4, email=lhs, password=abcd)



id 가 4인 회원의  
email 이 lhs 로 password 가 abcd 로  
정상적으로 수정 되었음을 확인!



실습 4-4

User D(elete) 구현



// 회원 정보를 삭제하는 메서드

```
public void deleteUser(int id) {  
    String sql = "DELETE FROM users WHERE id = ?";  
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {  
        pstmt.setInt(1, id);  
        int affectedRows = pstmt.executeUpdate();  
        if (affectedRows > 0) {  
            System.out.println("회원 삭제 성공!");  
        } else {  
            System.out.println("회원 삭제 실패");  
        }  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

회원 삭제  
deleteUser 메서드  
전체 코드



```
String sql = "DELETE FROM users WHERE id = ?";
```



## 회원 삭제를 위한 기본 쿼리문 작성

삭제를 하려는 회원의 id 값은  
계속 변할 것이므로 ? 로 처리!

숫자 id 가 일치하는 회원의  
정보를 삭제하는 쿼리!

```
try (PreparedStatement pstmt = conn.prepareStatement(sql)) {  
    pstmt.setInt(parameterIndex: 1, id);
```

PreparedStatement 준비

PreparedStatement 에  
매개 변수로 받아온  
삭제 회원의 id 를 전달

```
int affectedRows = pstmt.executeUpdate();  
if (affectedRows > 0) {  
    System.out.println("회원 삭제 성공!");  
} else {  
    System.out.println("회원 삭제 실패");  
}
```

완성 된 PreparedStatement 를  
DB 서버로 보내서 쿼리를 수행

삭제가 완료 되면 데이터 1개가 변하므로  
해당 결과를 affectedRows 로 받기

쿼리 수행의 결과 값인  
affectedRows 값에 따라서  
적절한 결과 출력!

```
public class UserMain {  
    public static void main(String[] args) {  
        UserDao userDao = new UserDao();  
        // 회원 추가  
        // 회원이 계속 추가되면 귀찮으므로 주석 처리  
        // userDao.create("tetz2", "1234");  
        // 전체 회원 조회  
        // userDao.getAllUsers();  
  
        // 회원 수정 메서드 실행  
        userDao.updateUser(id: 4, newEmail: "lhs", newPassword: "abcd");  
        // id가 4인 회원 삭제 메서드 실행  
        userDao.deleteUser(id: 4);  
        // 회원 수정이 성공적으로 되었는지 목록 확인  
        userDao.getAllUsers();  
    }  
}
```

방금 추가한 deleteUser 메서드  
테스트를 위해서 UserMain 클래스로  
돌아오기

직전에 수정한 회원 정보를 삭제!

삭제 확인을 위해 전체 회원 목록 출력

✓ jdbc\_dao\_ex [:Use 1 sec, 67 ms

> Task :processResources NO-SOURCE

> Task :classes

> Task :UserMain.main()

DB 접속에 성공!

회원 정보 수정 성공!

회원 삭제 성공!

UserVo(id=1, email=tetz, password=1234)

UserVo(id=2, email=siwan, password=1234)

UserVo(id=3, email=na, password=1234)



id 가 4번인 회원의 정보가 삭제 되었음을  
확인 가능!



# 실습 5

JOIN 을 사용하여  
회원 이름 출력하기

// 테이블을 합친 뒤, 회원의 이름 정보까지 전부 출력하는 메서드

```
public void getAllUsersWithName() { 1 usage new *  
    String sql = "SELECT users.id, users.email, users.password, user_info.name " +  
        "FROM users " +  
        "JOIN user_info ON users.id = user_info.id";  
  
    try (Statement stmt = conn.createStatement();  
        ResultSet rs = stmt.executeQuery(sql)) {  
  
        while (rs.next()) {  
            int id = rs.getInt(columnLabel: "id");  
            String email = rs.getString(columnLabel: "email");  
            String password = rs.getString(columnLabel: "password");  
            String name = rs.getString(columnLabel: "name");  
  
            System.out.printf("ID: %d, Email: %s, Password: %s, Name: %s\n", id, email, password, name);  
        }  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

JOIN 을 사용한  
회원 정보 출력  
getAllUsersWithName  
메서드 전체 코드





// 테이블을 합친 뒤, 회원의 이름 정보까지 전부 출력하는 메서드

```
public void getAllUsersWithName() { 1 usage new *  
    String sql = "SELECT users.id, users.email, users.password, user_info.name " +  
        "FROM users " +  
        "JOIN user_info ON users.id = user_info.id";
```

users 테이블의 id 와 user\_info 테이블의 id 는 서로 참조 관계(= 외래키) 이므로  
둘의 데이터는 항상 같게 유지 됩니다!

따라서 id 를 기준으로 서로의 테이블을 아래와 같이 하나로 합칠 수 있습니다!

id	email	password	id	name
1	tetz	1234	1	이효석
2	siwan	1234	2	김시완
3	na	1234	3	나건우



id	email	password	id	name
1	tetz	1234	1	이효석
2	siwan	1234	2	김시완
3	na	1234	3	나건우

두 테이블의 id 는 서로 같기 때문에 테이블을 합쳐도  
데이터에 이상 현상이 발생하지 않습니다!



// 테이블을 합친 뒤, 회원의 이름 정보까지 전부 출력하는 메서드

```
public void getAllUsersWithName() { 1 usage new *
```

```
String sql = "SELECT users.id, users.email, users.password, user_info.name " +  
             "FROM users " +  
             "JOIN user_info ON users.id = user_info.id";
```

user\_info 테이블이 합쳐져서  
기존 users 테이블에서는 알 수 없었던  
회원의 이름 정보 컬럼도 가져오기!

	id	email	password	name
1	1	tetz	1234	이효석
2	2	siwan	1234	김시완
3	3	na	1234	나건우

실제 JOIN 쿼리를 실행하면 얻게 되는  
테이블의 모습

```
try (Statement stmt = conn.createStatement();  
    ResultSet rs = stmt.executeQuery(sql)) {
```

JOIN 쿼리를 실행하고 해당 결과를  
ResultSet 데이터로 받기

```
while (rs.next()) {
```

```
    int id = rs.getInt(columnLabel: "id");
```

```
    String email = rs.getString(columnLabel: "email");
```

```
    String password = rs.getString(columnLabel: "password");
```

```
    String name = rs.getString(columnLabel: "name");
```

```
    System.out.printf("ID: %d, Email: %s, Password: %s, Name: %s\n", id, email, password, name);
```

```
}
```

```
try (Statement stmt = conn.createStatement();  
    ResultSet rs = stmt.executeQuery(sql)) {
```

```
while (rs.next()) {  
    int id = rs.getInt(columnLabel: "id");  
    String email = rs.getString(columnLabel: "email");  
    String password = rs.getString(columnLabel: "password");  
    String name = rs.getString(columnLabel: "name");  
  
    System.out.printf("ID: %d, Email: %s, Password: %s, Name: %s\n", id, email, password, name);  
}
```

ResultSet 데이터를 순회하면서  
각각의 정보(id, email, password,  
name)를 변수에 담기

UserVo 클래스는 name 에 대한  
멤버 변수가 없기 때문에 이름 정보 출력이 불가능!

따라서, while 문에서 바로 출력을 해주면 됩니다!  
name 정보 까지 포함해서 결과를 출력!

```
public class UserMain {  
    public static void main(String[] args) {  
        UserDao userDao = new UserDao();  
  
        // 기존에 구현한 회원 목록 출력 메서드  
        userDao.getAllUsers();  
        // 이번에 구현한 이름 정보를 포함한 회원 목록 출력 메서드  
        userDao.getAllUsersWithName();  
    }  
}
```

방금 추가한 getAllUsersWithName 메서드  
테스트를 위해서 UserMain 클래스로  
돌아오기

비교를 위해 getAllUsers() 메서드와  
getAllUsersWithName() 를 동시에 실행

✓ jdbc\_dao\_ex [:Use 1 sec, 90 ms

오후 8:56:49: Executing ':UserMain.main()'...

> Task :compileJava

> Task :processResources NO-SOURCE

> Task :classes

> Task :UserMain.main()

DB 접속에 성공!

UserVo(id=1, email=tetz, password=1234)

UserVo(id=2, email=siwan, password=1234)

UserVo(id=3, email=na, password=1234)

ID: 1, Email: tetz, Password: 1234, Name: 이효석

ID: 2, Email: siwan, Password: 1234, Name: 김시완

ID: 3, Email: na, Password: 1234, Name: 나건우



getAllUsersWithName() 는  
user\_info 테이블에서 이름 정보를 가져와서  
같이 출력해 주는 것을 확인 가능!!

