KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

JDBC 프로그래밍

[KB] IT's Your Life

sql를 자바와 연동하는 프로그래밍

jdbc 프로그래밍.
java data base connectivity

프로그래밍 언어마다
데이터베이스와 연동하는
자체 프레임 워크들이 각각 있다.

자바측에서 제공하는 프레임워크이다

요즘엔 JPA를 만힝 사용한다.

어떻게 자바에서 쿼리와 결과를 실행 동작 얻을 수 있는지

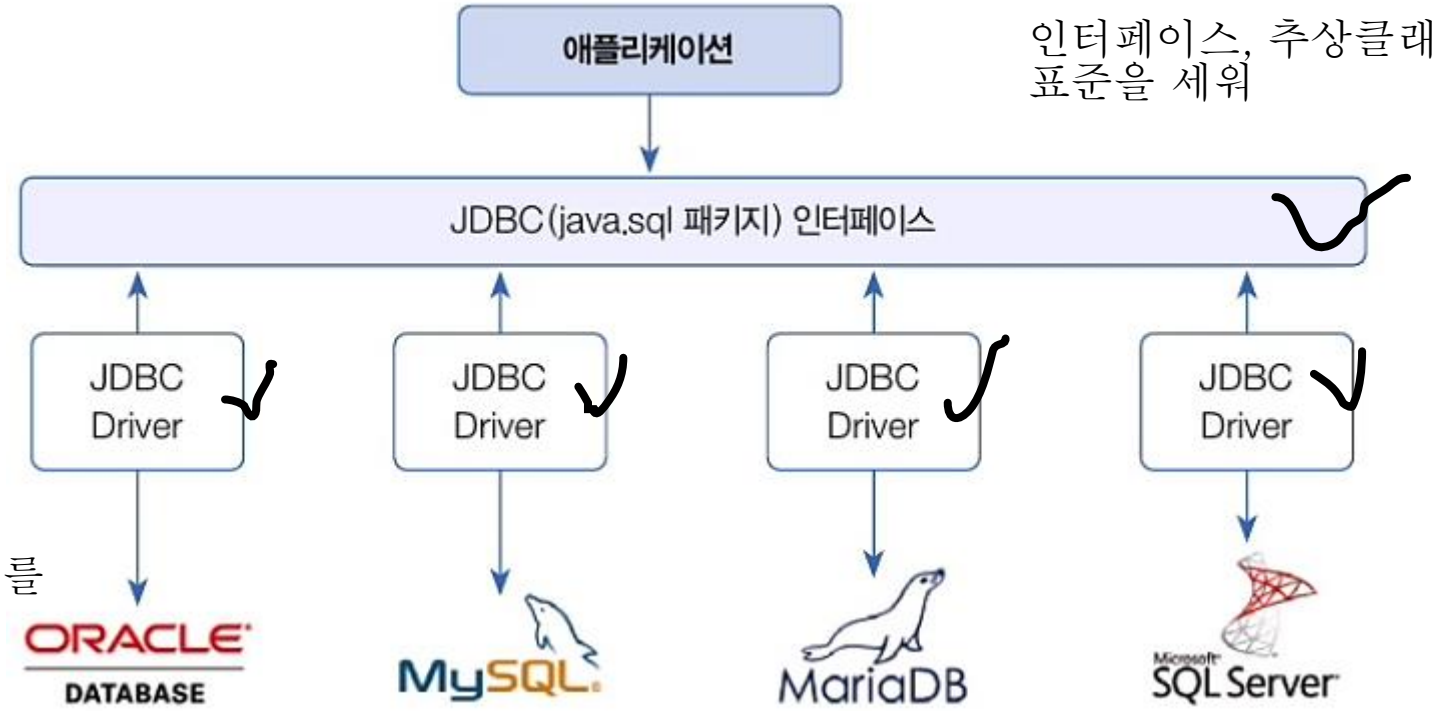
KB 국민은행

1 JDBC 프로그래밍

- **JDBC Java Database Connectivity**
  - 데이터베이스와 연결해서 입출력을 지원
  - 데이터베이스 관리시스템(DBMS)의 종류와 상관없이 동일하게 사용할 수 있는 클래스와 인터페이스로 구성

인터페이스로
추상클래스로
사용법을 다 정의해놈

이들의 구현체를
각 DB회사에서 제공함.
이를 jdbc 드라이버라고 함



인터페이스, 추상클래스로
표준을 세워

우리는 자바에서
제공하는
jdbc인터페이스를
통해서
프로그래밍을 하고

실제 디비 연결은

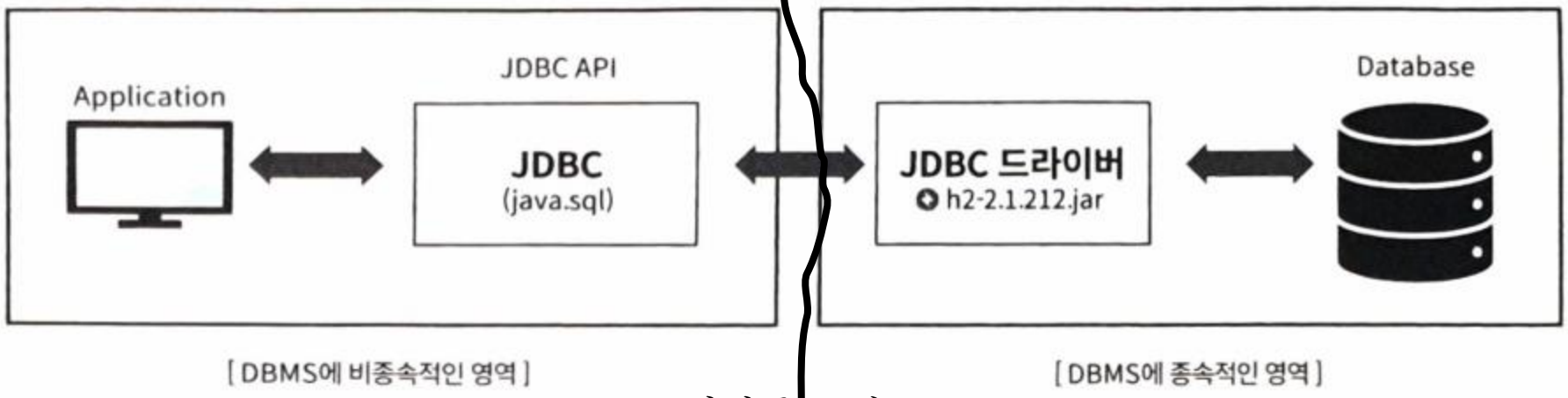
회사에서 만든 드라이버를
통해서 연동된다.

JDBC

JDBC 개념

JDBC는 2개로 나뉘어진다

자바에서 제공하는 인터페이스영역 | 구현체(제조사에서 제공)

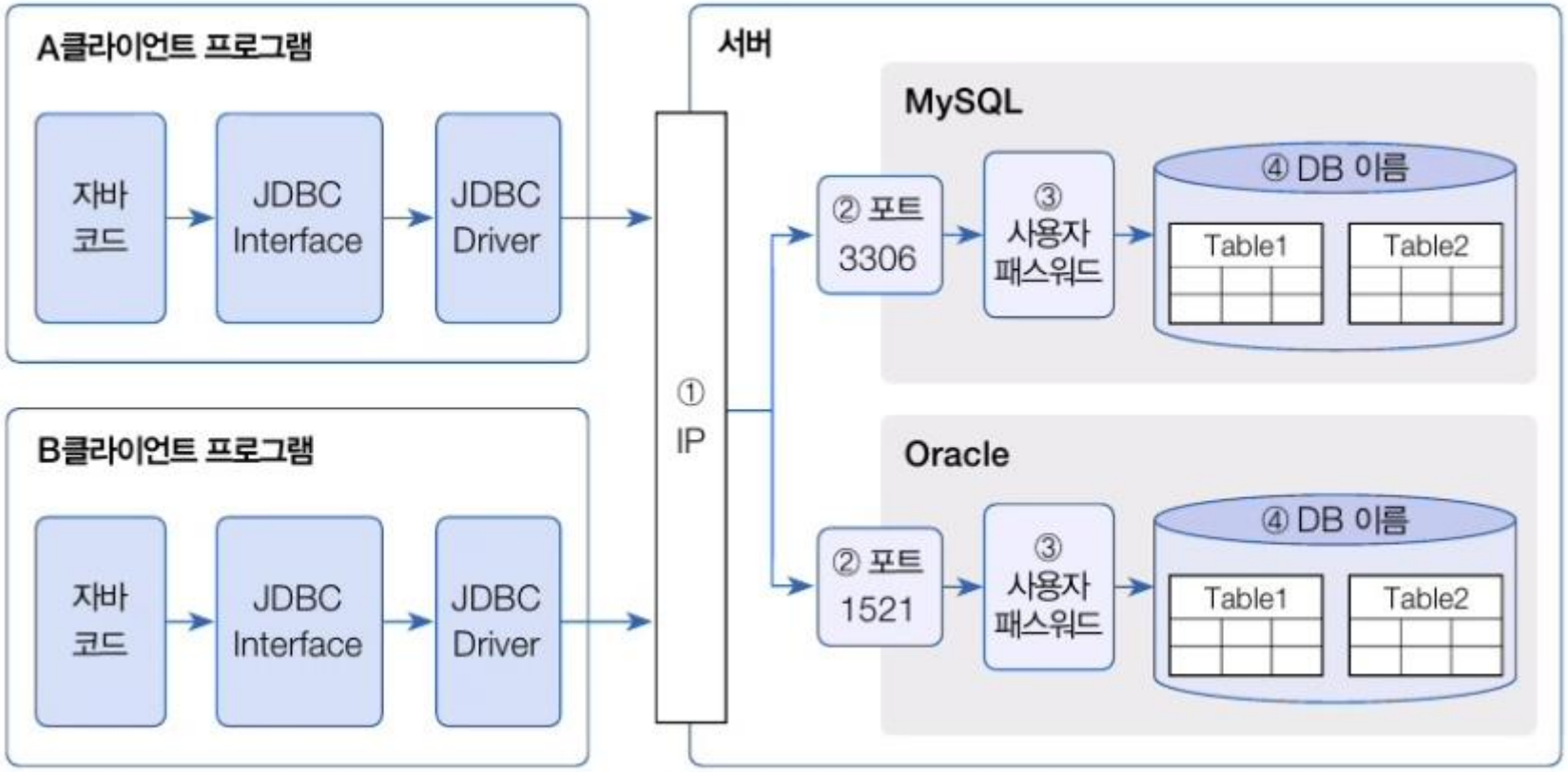


쿼리를 보내고
결과를 조회하고 ...

JDBC

얻을 수 잇는 장점. 데이터 베이스가 달라도 어플리케이션에서는 수정이 경의 없다.

JDBC 개념



데이터베이스 준비

```
CREATE DATABASE jdbc_ex;
```

사용자 준비

```
CREATE USER 'scoula'@'%' IDENTIFIED BY '1234';
GRANT ALL PRIVILEGES ON jdbc_ex.* TO 'scoula'@'%';
FLUSH PRIVILEGES;
```

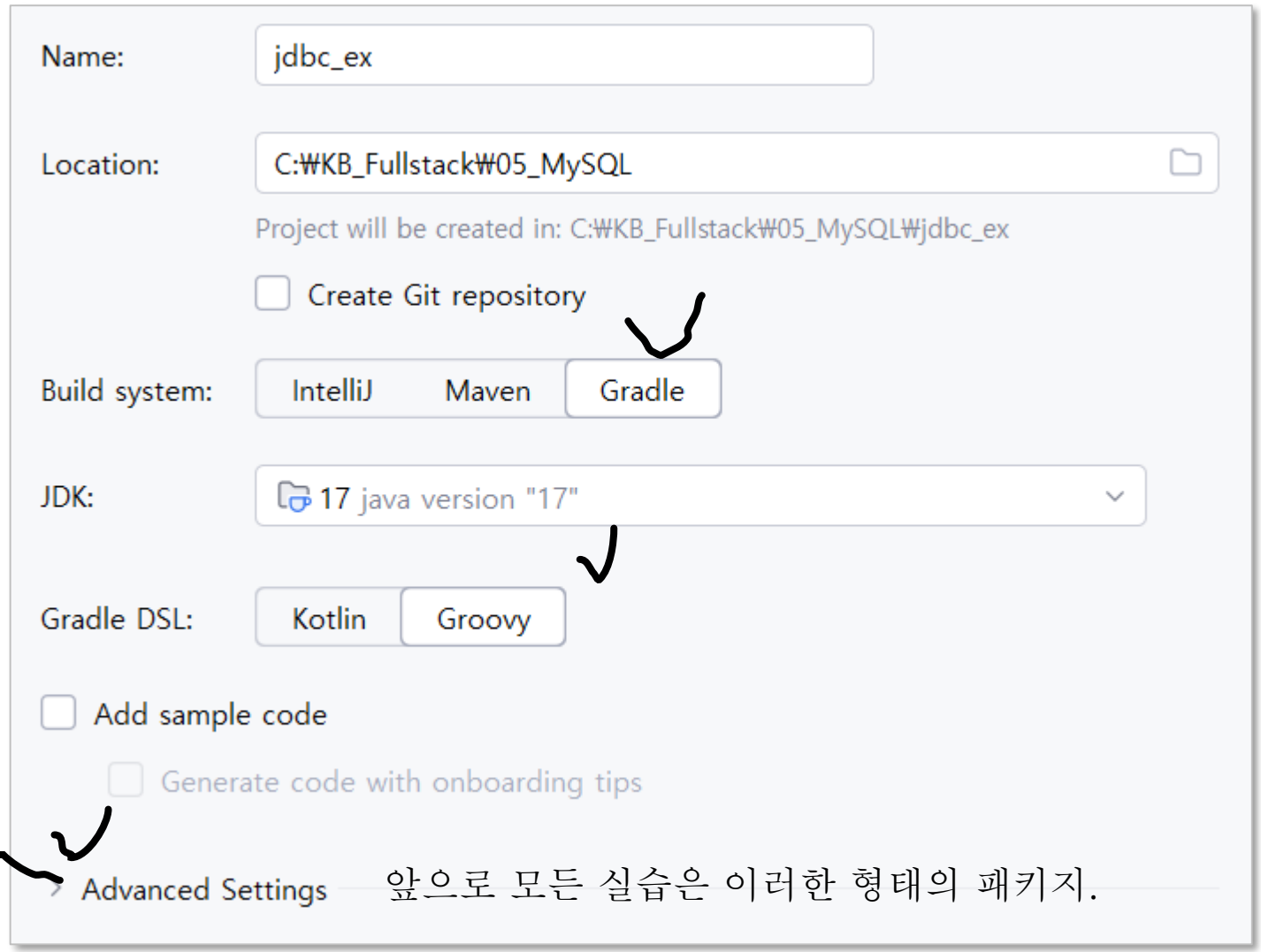
이미 scoula 계정이 만들어져 있따면

권한만 주면 된다.

쿼리 동작후 홈으로 가서 scoula계정이 생성되었고 거기서 지정한 DB가 뜨는지 확인해보자

프로젝트 생성

- Name: jdbc_ex
- Build System : gradle
- Group Id: org.scoula



평소에 썼던
빌드 시스템인 intellij는
외부라이브러리를 설정하는것이
불편한다.

많이 사용할 텐데
시장에 양분하고 있는 툴
메이븐과 그라들을 사용할 것이다.

Gradle dsl domain specific language

그라들 설정파일을 만들고 사용할때
언어를 어떤걸 사용할 것이냐

그루비--구글에서 만든 스크립트 언어이다.
build.gradle 파일 작성에 사용할 것이다

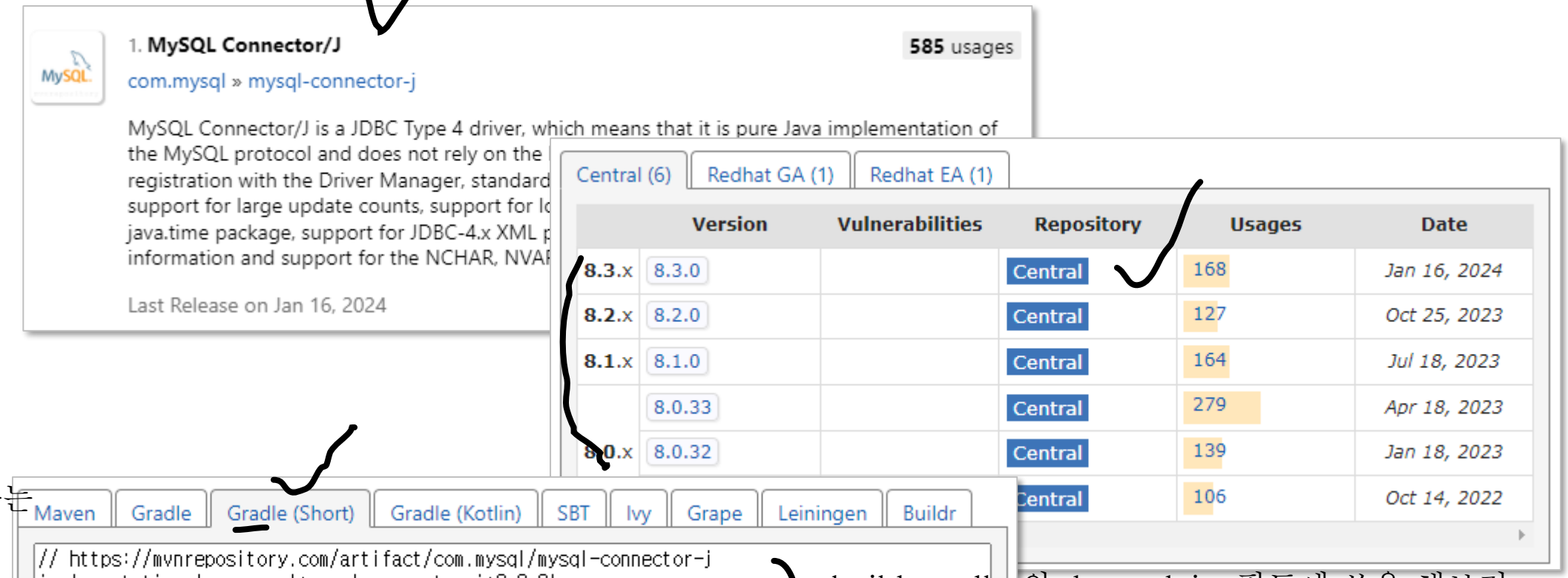
앞으로 모든 실습은 이러한 형태의 패키지.

아티팩트 아이디==산출물. 최종결과물의 이름을 정의하자

- **MySQL Connector**
  - https://mvnrepository.com/
  - MySQL 검색

js의npm과 대응하는 자바의 레포지토리. 메이븐과 그라들

외부라이브러리들이 모인 곳.
mvnrepository
통해 외부에서 mysql커넥터를



툴들에 대응하는 빌드 설정 내용들

build.gradle 의 dependcies필드에 쏘옥 해보려

- Lombok도 추가

롬복도 mysql커넥터처럼 그라들을 통해서 설치해 보자

build.gradle

```
…
dependencies {
    implementation 'com.mysql:mysql-connector-j:8.3.0'
    compileOnly 'org.projectlombok:lombok:1.18.30'
    annotationProcessor 'org.projectlombok:lombok:1.18.30'

    testCompileOnly 'org.projectlombok:lombok:1.18.30'
    testAnnotationProcessor 'org.projectlombok:lombok:1.18.30'

    testImplementation platform('org.junit:junit-bom:5.9.1')
    testImplementation 'org.junit.jupiter:junit-jupiter'
}
…
```

내가 사용할 외부 라이브러리가 뭐냐

여기다가 제시하면
자동으로 다운되고 적용된다

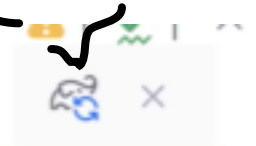
테스트해줌. 앞에 테스트 붙이면.
그라들에서는 저렇게 2문장을 써줘야만 테스트를 함

단위테스트 junit이 디폴트로 추가되어있음

그라들을 통해서 쉽게 적용 가능

○ 수정 후 Sync 실행

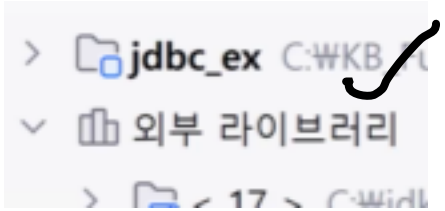
build.gradle에 수정이 되면 동기시켜야 변경된것들이이 적용되어 설치됨

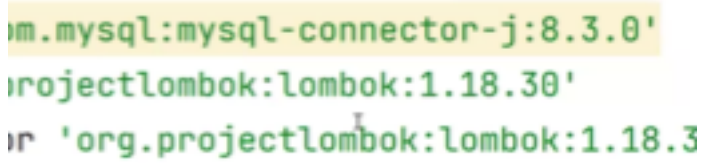


○ 프로젝트 설정

- Annotation Processing 활성화

설치된거 확인





Intellij Datasource 기능 설정

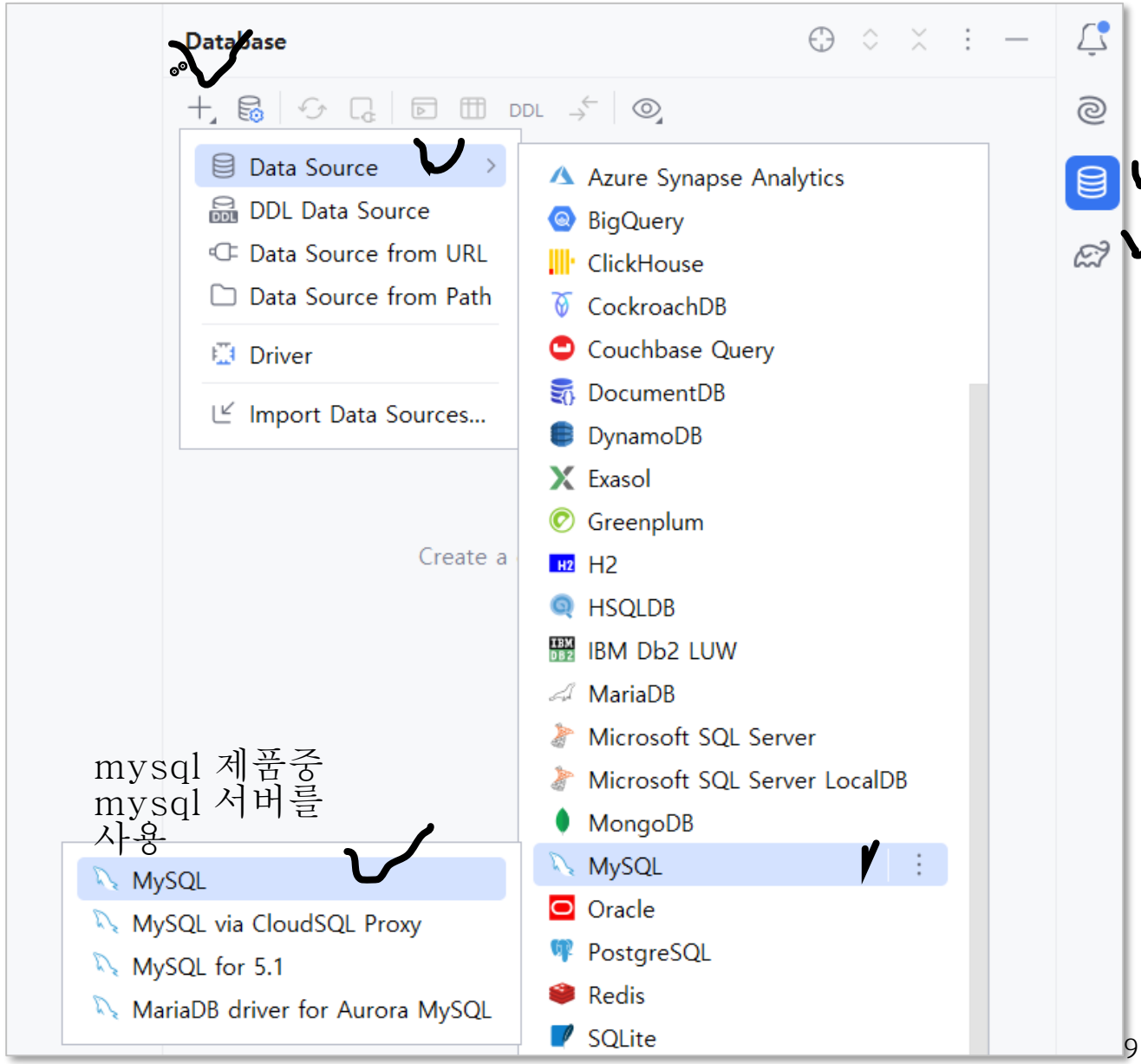
데이터베이스를 항상 조회하고
쿼리가 잘적용됐는지
많이 확인해야함

자바 에디터와 db워크벤치를 왓다갔따 해야하는데

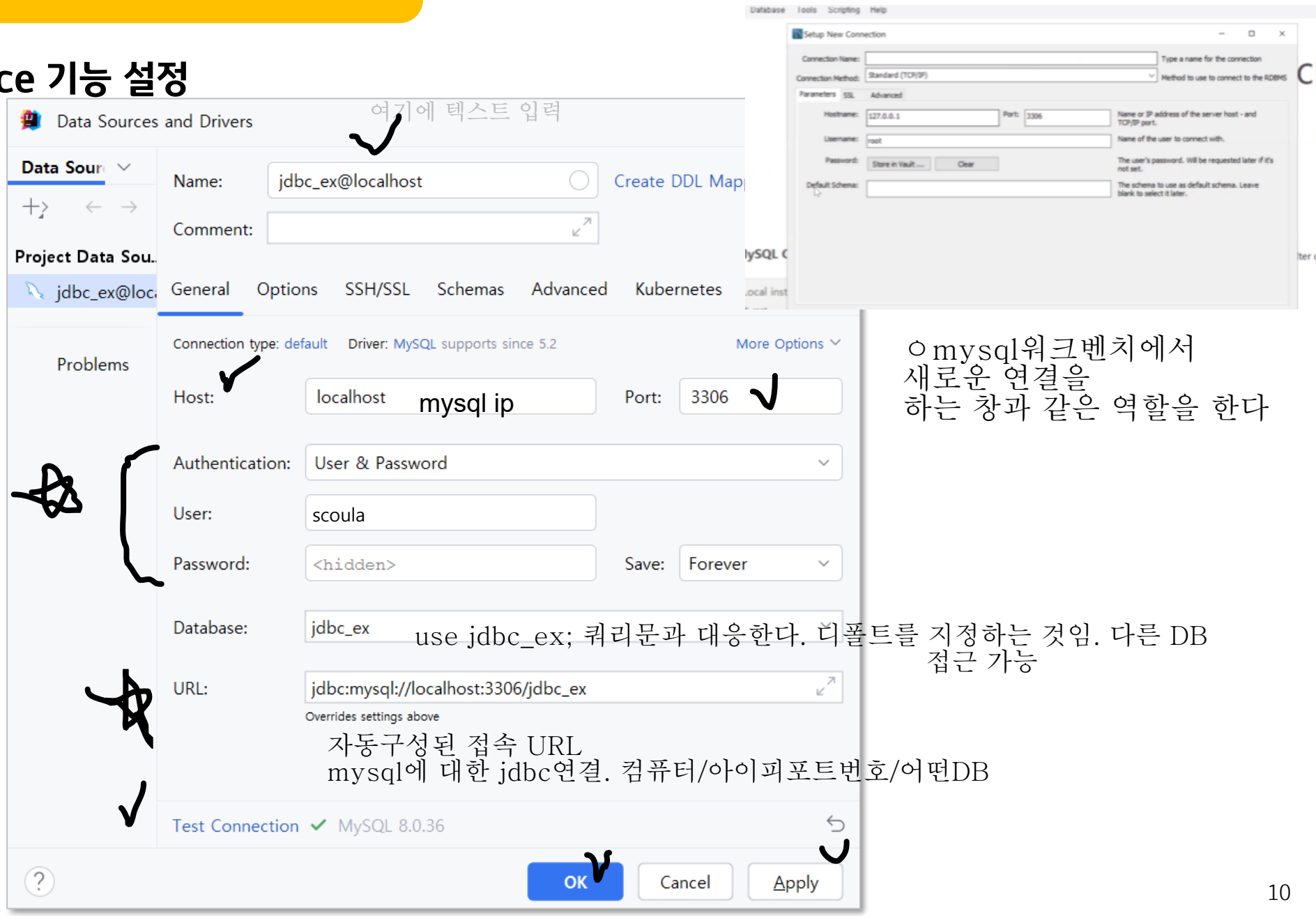
불편하니까

에디터에서 DB상황을 볼수 있고 쿼리를 사용할 수 있게 설정해보자.

인텔리제이는 상용버전에서만
DB와 연동하여 우크벤치의 역할을
본인에게서 할 수 있게 기능을 제공한다



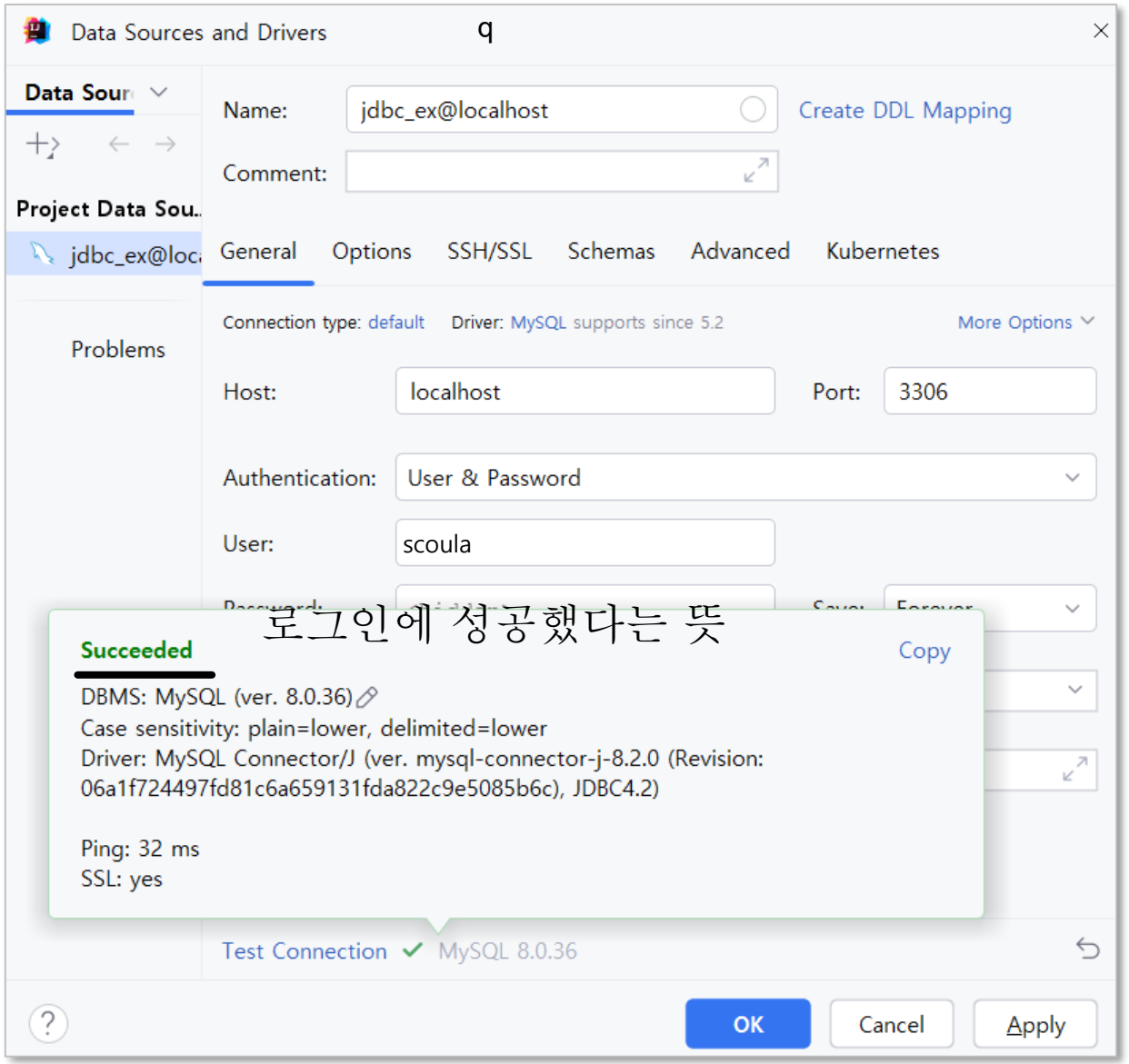
Intellij Datasource 기능 설정



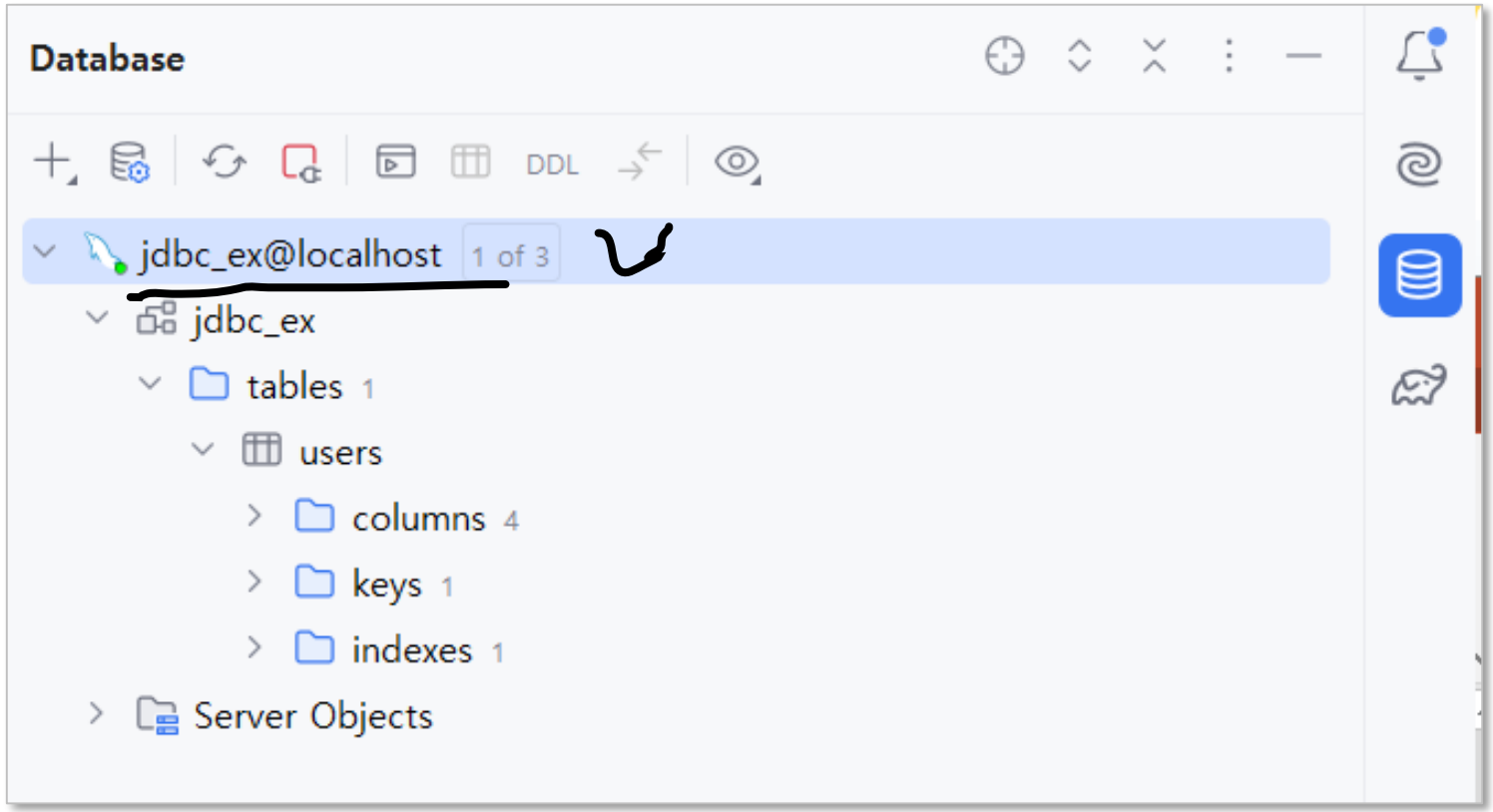
ㅇmysql워크벤치에서 새로운 연결을 하는 창과 같은 역할을 한다

Intellij Datasource 기능 설정

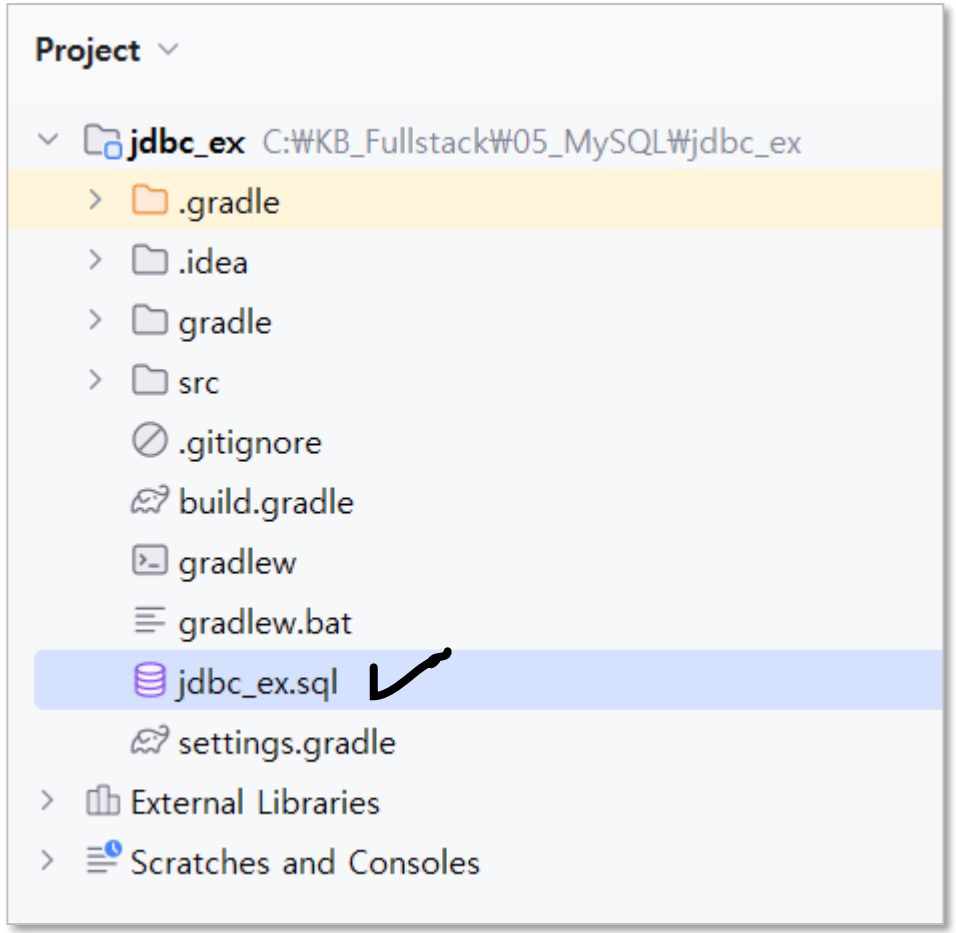
과정에서
인텔리제이에
jdbc드라이버 설치가
있을 수 잇음
dB쪽뿐만 아니라
에디터쪽에도 필요해서



datasource 목록



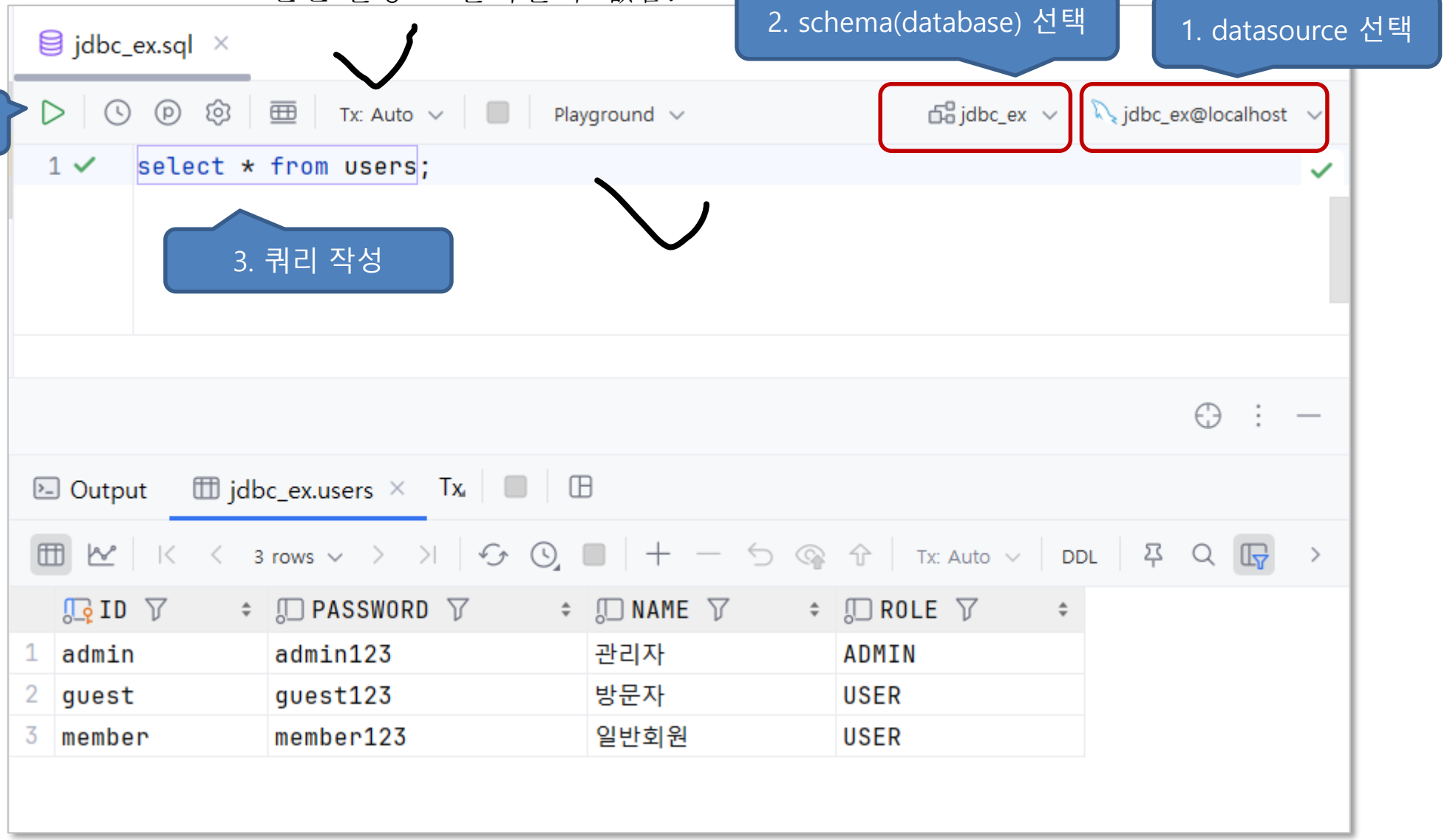
sql 파일 만들기


Project
jdbc_ex C:₩KB_Fullstack₩05_MySQL₩jdbc_ex
.gradle
.idea
gradle
src
.gitignore
build.gradle
gradlew
gradlew.bat
jdbc_ex.sql
settings.gradle
External Libraries
Scratches and Consoles

쿼리 실행

TX==transaction
현재모드는 오토 커밋.
한번 실행==돌이킬 수 없음.

쿼리파일과
데이터 소스 연결



데이터 준비

안써도 되긴함
디폴트 설정 했잖아

```
use jdbc_ex;

CREATE TABLE USERS (
  ID VARCHAR(12) NOT NULL PRIMARY KEY,
  PASSWORD VARCHAR(12) NOT NULL,
  NAME VARCHAR(30) NOT NULL,
  ROLE VARCHAR(6) NOT NULL
);

INSERT INTO USERS(ID, PASSWORD, NAME, ROLE)
VALUES('guest', 'guest123', '방문자', 'USER');

INSERT INTO USERS(ID, PASSWORD, NAME, ROLE)
VALUES('admin', 'admin123', '관리자', 'ADMIN');

INSERT INTO USERS(ID, PASSWORD, NAME, ROLE)
VALUES('member', 'member123', '일반회원', 'USER');

SELECT * FROM USERS;
```

인텔리제이 에디터에서 jdbc_ex.sql파일에서 작성하여
데이터들을 생성해보자

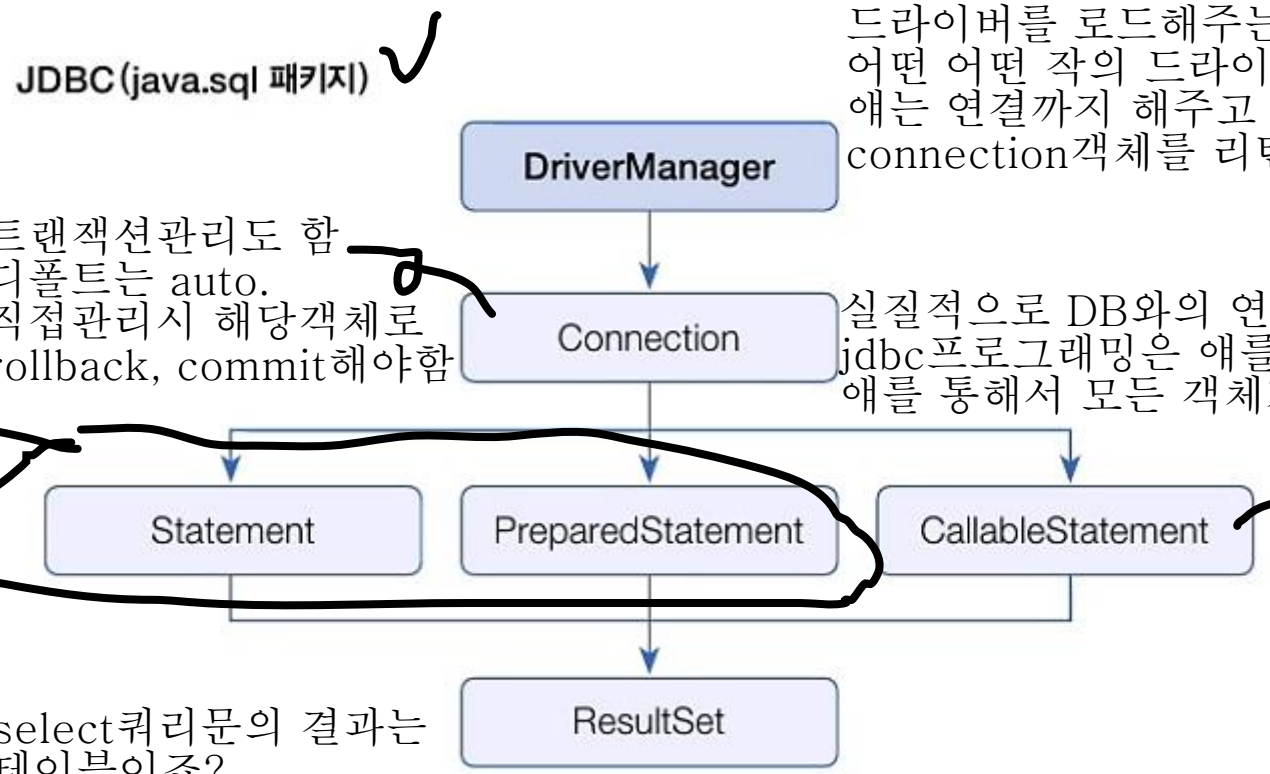
번외
role 열의 정보와
priviledge 정보를 묶어서
따로 테이블을 만들어 관리해야
제3정규형 위배 안하며
역할 지정 정보를 저장할 수 있다

여기까지 준비 과정

프로젝트와관련된 쿼리를 독립된 에디터에서 관리하지 않고
하나의 에디터에서 관리하니 편하다

jdbc프로그래밍에서패키지에서 자주 사용하는 클래스들.

JDBC의 핵심 인터페이스/클래스



드라이버를 로드해주는 매니저.
어떤 어떤 작의 드라이버를 쓸거냐.
얘는 연결까지 해주고
connection객체를 리턴함.

트랜잭션관리도 함
디폴트는 auto.
직접관리시 해당객체로
rollback, commit해야함

실질적으로 DB와의 연결을 유지하고 있는 실체.
jdbc프로그래밍은 얘를 획득하는 것으로 시작함.
얘를 통해서 모든 객체가 파생해서 얻어짐.

쿼리를 실행시키는 객체들
둘의 차이는 어떤 방식으로
실행하느냐의 차이로 나뉨

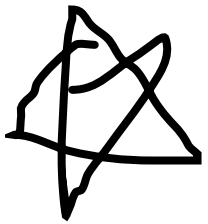
얘는 스토어드 프로시저
를 사용할때 사용됨,

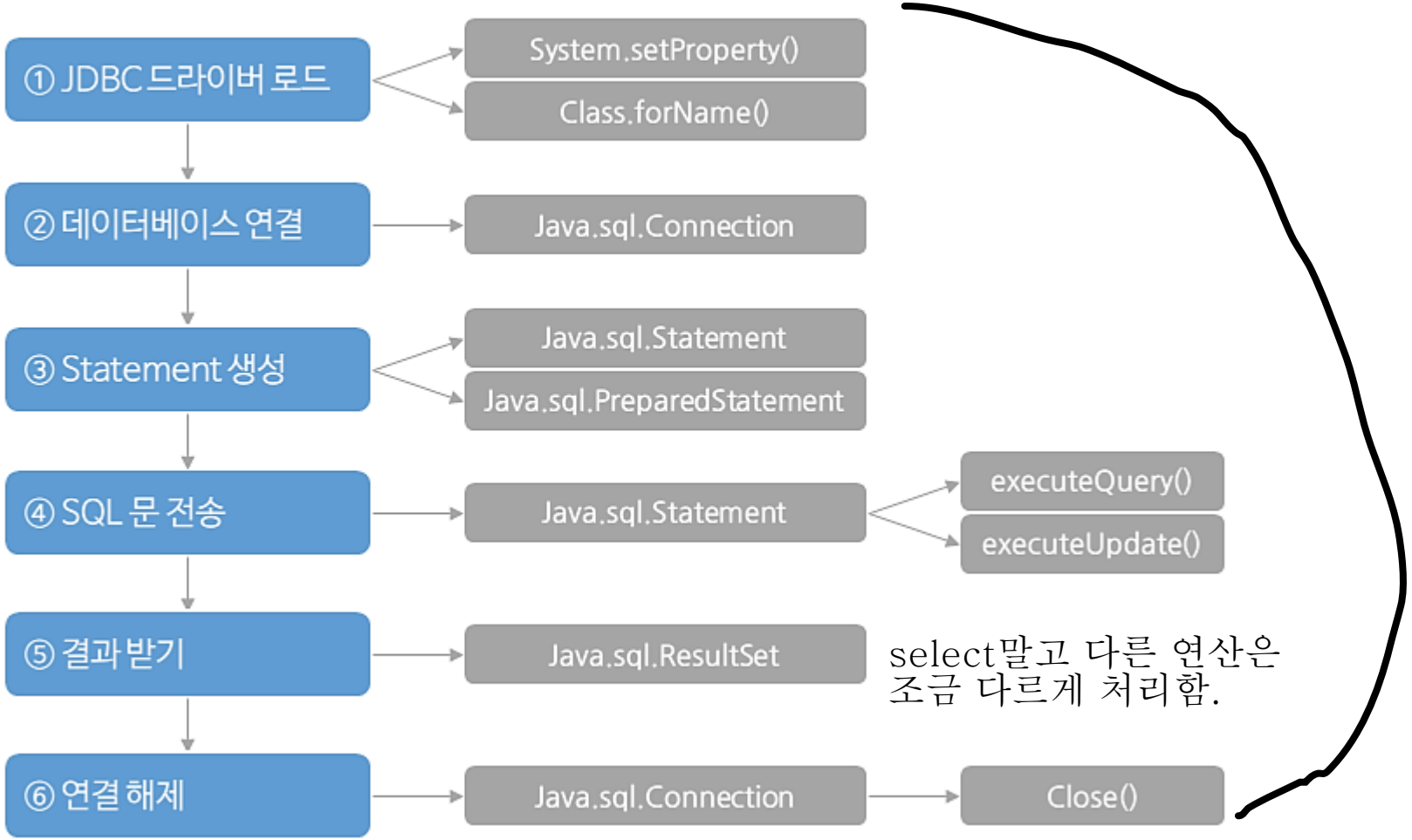
그렇다면 select말고
insert update delete
의 리턴은 뭘까요?
integer이다!
의미는 몇개가 영향을
받았는냐이다.

select쿼리문의 결과는
테이블이죠?
결과(테이블)을 처리하는
클래스

정리!

- JDBC
  - JDBC 개발 절차





모두다 IO작업이기에
모두 다 예외처리해야함!!

select말고 다른 연산은
조금 다르게 처리함.

연결 뿐만 아니라 앞에서 썼던
것들도 매번 다 클로즈해야해!

DB 연결

드라이버 확인

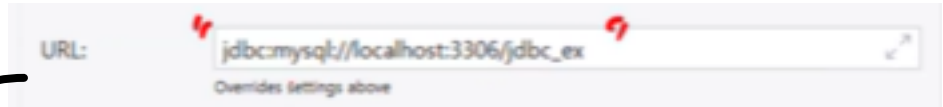
1 Class.forName("com.mysql.cj.jdbc.Driver");

→ 없으면 ClassNotFoundException 발생 없는 경우는. build.gradle에 jdb 드라이버 다운로드 지정을 안했다면

Connection 객체

- 데이터베이스에 연결 세션을 만듦 스태틱
- Connection conn = DriverManager.getConnection(
  "연결 문자열", "사용자", "비밀번호")

연결문자열은 아까 설정했던 거에서 URL



- 연결 문자열
  "jdbc:mysql://[host]:[포트]/[db이름]
  → jdbc:mysql://127.0.0.1:3306/jdbc_ex

- String url = "jdbc:mysql://127.0.0.1:3306/jdbc_ex";
- Connection conn = DriverManager.getConnection(url, "jdbc_ex", "jdbc_ex");

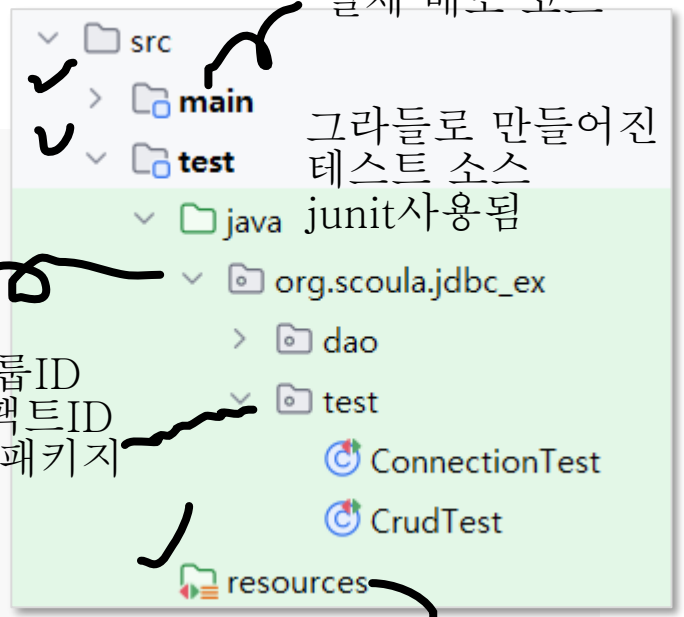
하지만 그 전에
DB연결이 잘되나 안되나
진짜 앱에 연결하기 전에
연결이 잘되는지 테스트 해보자.

ConnectionTest.java

```
package org.scoula.jdbc_ex.test;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.scoula.jdbc_ex.common.JDBCUtil;

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

public class ConnectionTest {
    @Test
    @DisplayName("jdbc_ex 데이터베이스에 접속한다.")
    public void testConnection() throws SQLException, ClassNotFoundException {
        Class.forName("com.mysql.cj.jdbc.Driver");
        String url = "jdbc:mysql://127.0.0.1:3306/jdbc_ex";
        String id = "scoula";
        String password = "1234";
        Connection conn = DriverManager.getConnection(url, id, password);
        System.out.println("DB 연결 성공");
        conn.close();
    }
}
```

실제 배포 코드

그라들로 만들어진 테스트 소스 junit사용됨



테스트할 패키지 하나 만드세요
gradle로 치면
org.scoula가 그룹ID
jdbc_ex가 아티팩트ID
test폴더가 base패키지

운영서버에서 연결해야함

JUNIT테스트 프레임워크의 어노테이션

@Test어노테이션은 단위(메소드혹은클래스단위)테스트를 하는 친구다.

@DisplayName 결과를 출력할때 테스트명을 나타내는 어놑

발생할 수 있는 예외들

코드 파일을 제외한 모든 관련 사용된 파일(자원)

테스트 어노테이션을 사용하면 마크가 생기는데 누르면 단위별로 테스트 가능하다!
실행해보면 테스트 결과와테스트명이 나온다!

```
DB 연결 성공
```

공통된 과정을 매번 할건데 불편하니까 클래스화 시켜야한다. 접속관련 유틸리티를 만들자

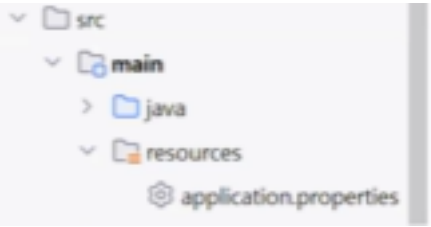
모듈화해야 할 코드

- 데이터베이스 연결 및 닫기 작업은 항상 필요함

→ common.JDBCUtil

JDBCUtil.java

```
package org.scoula.jdbc_ex.common;
```



프로퍼티 파일은`
테스트 패키지 밑에 말고
메인 패키지 밑에다가 새성

19페이지에서 연결관련 코드들은
내용이 박혀잇는 하드 코드.

해당 내용들을 외부에서(설정파일)불러오는 식으로
리팩토링한걸 유틸로 만들자!

1 JDBC 프로그래밍

resources::/application.properties

properties파일에다가 해당 내용 등록 후 사용.

```
driver=com.mysql.cj.jdbc.Driver
url=jdbc:mysql://127.0.0.1:3306/jdbc_ex
id=scoula
password=1234
```

실제로 사용할 것인께
메인패키지에다가

메인 패키지의 리소스폴더와
테스트 패키지의 리소스폴더의
역할 차이는 뭔데 그럼.

테스트용으로만 쓰는 리소스들이 있따.
실제로 테스트하다가 실제배포용 db에 영향이 미칠수있다.
테스트용 db 을 따로 마련한다.

서로 다른 db를 이용하기에 리소스도 다르겟쬬?

테스트시 리소스 사용하는 과정
테스트 패키지의 리소스 폴더에서 지정된 이름을 검색해서 있으먄 그걸로 테스트
없으면
메인 패키지의 리소스 폴더에서 지어된 이름을 찾아서 있으면 그걸로 테스트

JDBCUtil.java

```
package org.scoula.jdbc_ex.common;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class JDBCUtil {
    static Connection conn = null;
    static {
        try {
            Properties properties = new Properties();
            properties.load(JDBCUtil.class.getResourceAsStream("/application.properties"));
            String driver = properties.getProperty("driver");
            String url = properties.getProperty("url");
            String id = properties.getProperty("id");
            String password = properties.getProperty("password");

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

커넥션이 한개만 유지 되도록하기 위해. 그렇다고 싱글톤은 아님. JDBCUtil안에서만 한개니까

스태틱으로 초기화

default 접근제한

절대경로. 루트부터시작.
클래스 패스상의 루트다!
파일 패스가 아니다!
클래스 패스가 뭔데;

JDBCUtil.java

```
  public static Connection getConnection() {
    return conn;
  }

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

}
```

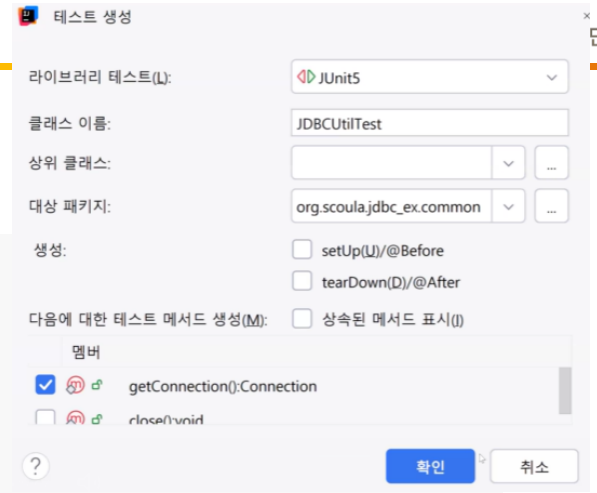
외부로 안에있는 connection객체를 쏴주는 메소드

클로즈할때도! 예외처리!

ConnectionTest.java

```
package org.scoula.jdbc_ex.test;
…

public class ConnectionTest {
    …
    @Test
    @DisplayName("jdbc_ex에 접속한다.(자동 닫기)")
    public void testConnection2() throws SQLException {
        try(Connection conn = JDBCUtil.getConnection()) {
            System.out.println("DB 연결 성공");
        }
    }
}
```

테스트 클래스
자동 구성 생성됨

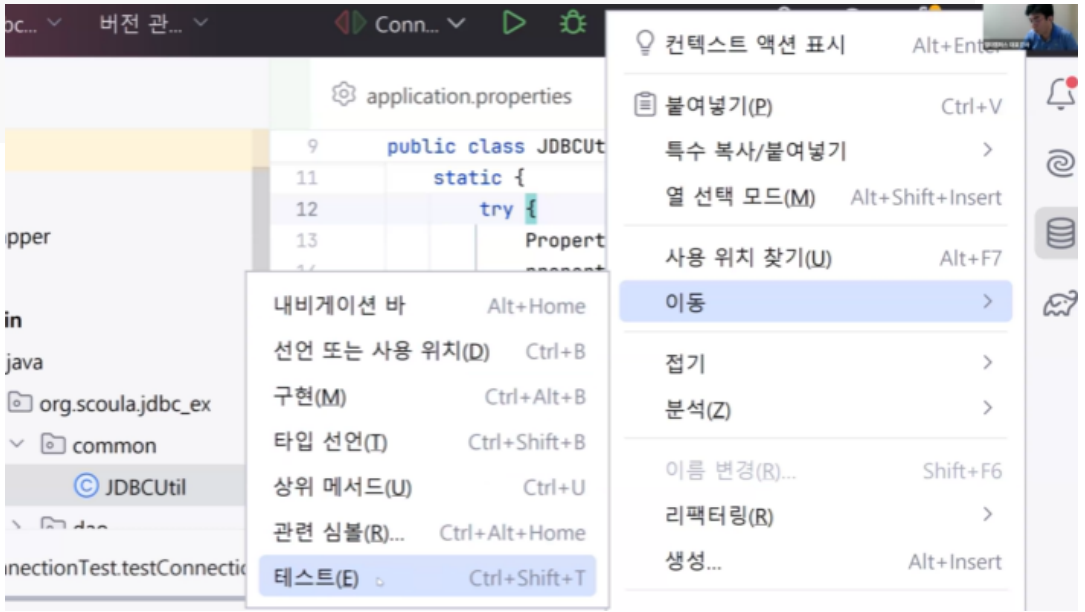


아까 만든 기능들. JUNIT @Test어노테이션으로 테스트해보자

일반적으로 이렇게 안하고

JDBCUtil테스트용 클래스를 하나 만들어서
테스트 하는 것이 단위테스트 관례다.

매번 그렇게 테스트 전용 클래스를 만들면 힘드니 인텔리제이 에디터에서는 테스트 할 수 있는 툴을 자동화 시켜준다!



ctrl+shft+t
가 단축키
외우장

DB연결을 하고 connection객체를 배출하는 util만들고 테스트검증하고 connection객체를 획득햇으니 이제는

Statement

sql문 실행할때. 사용되는

- SQL 문 실행 클래스
- Connection 객체를 통해 생성

```
Statement stmt = conn.createStatement();
```

- SQL 실행 메서드
  - ResultSet executeQuery(SQL문) : select문 실행
  - int executeUpdate(SQL문): insert, update, delete 문 실행

실행하는 2종류

수정

테이블이므로 행단위로 처리해야함.
순회하며.

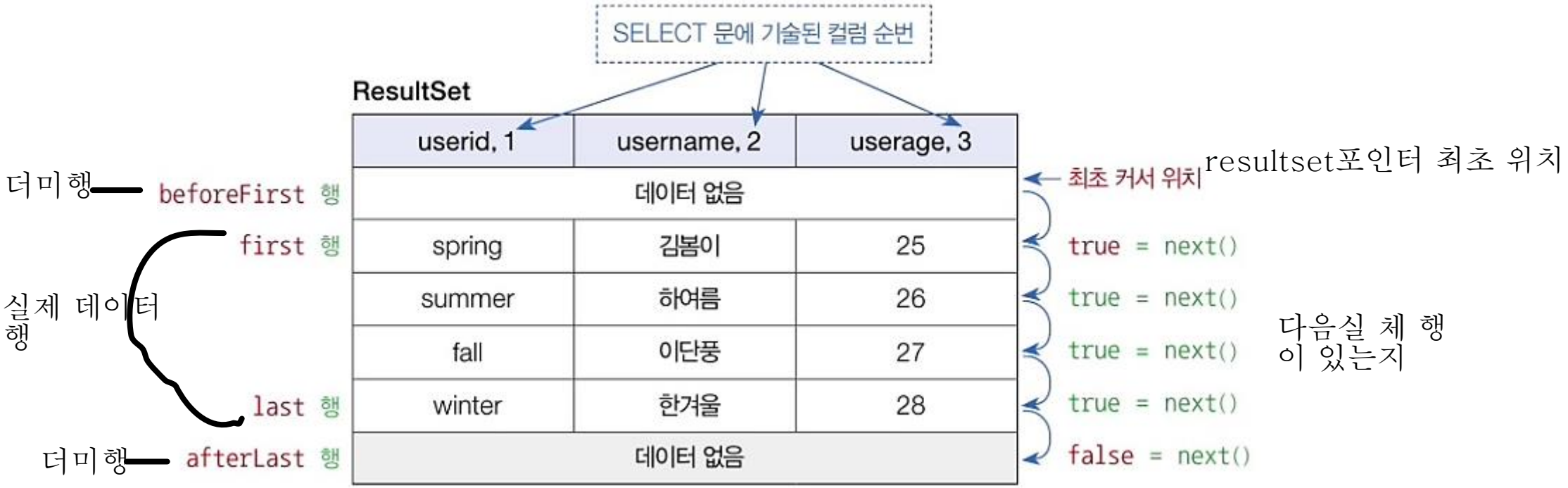
resultset이 지원함

statement 객체의 실행메서드는 문자열의 sql문을
넘기면서 그때 컴파일(해독)되고 실행된다.
즉 호출할때마다 그때그때 컴파일하고실행.
똑같은 구문을 반복 할때 같은 구문이어도(물론데이터는다름)
할때마다 컴파일해야함!성능별로.->컴파일미리해놓고
바뀐데이터만 적용해서 사용하자!->preparedstatement클래스사용

번외 close를 하는 것도 바로해야하지만
이도 예외처리해야하므로 코드가 굉장히 지저분해진다.
자동닫기 구문을 잘 활용해야한다.=> 어떻게 하는건데!!!
아 try-with-resource구문있잖아. try(.....) {....}

select문을 실행하면 메모리에 테이블 하나 저장됨
요소를 가리키는 포인터(커서)가 있음.
실제 데이터 앞뒤로 행이 하나씩 더 있다.

ResultSet



컬럼 값 추출

매개면수로 칼럼인덱스로 줄수 있고 칼럼명을 줄수있따

- getXxxx("컬러명)
  - Xxx : 추출하고자하는 데이터 타입명 리턴되는 타입에 따라. 컬럼의 타입에 따라.메소드를 달리 써야함
  - getString(), getInt(), getLong(), getDouble() getDate( ) ...

컴파일을 미리해놓게다의 prepared.

앞서본 또다른 sql실행 클래스

PreparedStatement

- SQL문에 값을 넣을 때 파라미터화 해서 처리

```
String sql = "INSERT INTO USERS(ID, PASSWORD, NAME, ROLE) " +
             "VALUES(?, ?, ?, ?)";
```

변수처리하는 법. 실제 데이터 들어갈 곳에 "?"를 제시

- Connection 객체를 통해 생성

```
PreparedStatement pstmt = conn.prepareStatement(sql);
```

이때 문자열 넘어가고 미리 컴파일해놓음

그래서 데이터 파트는 변수처리해야함

- 파라미터 설정
  - pstmt.setXxxx(파라미터번호, 값) 실행하기전에 변수처리했던 곳에 값을 채워줘야함
    - setString(), setInt(), setLong(), setDouble()

- SQL문 실행

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

애를 주로 더 씀.

statement와 쿼리가 넘어가는 시점이 다르다.
statement객체를 생성할때 쿼리가 안넘어감.
실제 쿼리를 실행할때(메소드 호출할때). 넘어감.
prepared는 객체가 생성될때 쿼리가 넘어가 컴파일함.

Statement로 Insert 문 실행하기

```
String sql = "INSERT INTO USERS(ID, PASSWORD, NAME, ROLE)" +
             "VALUES('member2', 'member123', '일반회원', 'USER')";

int count = stmt.executeUpdate(sql);
```

- 값을 변수로 대체한다면 ?

```
String userId = "member2";
String password = "member123";
String name = "일반회원";
String role = "USER";

String sql = "INSERT INTO USERS(ID, PASSWORD, NAME, ROLE)" +
             "VALUES('" + userId + "', '" + password + "', '" +
        name + "', '" + role + "')";
```

실수하기 좋음

→ PreparementStatement로 처리

다음페이지로CRUD하기 전에
JUNIT피피티를 보고 오자자

connection기능은 아까 단위테스트했으니
이제는 CRUD(데이터를 수정하는 연산들)
sql문을 실행하는 동작을 하는 statement계열
의 객체들의 메서드들을 단위테스트해보자.

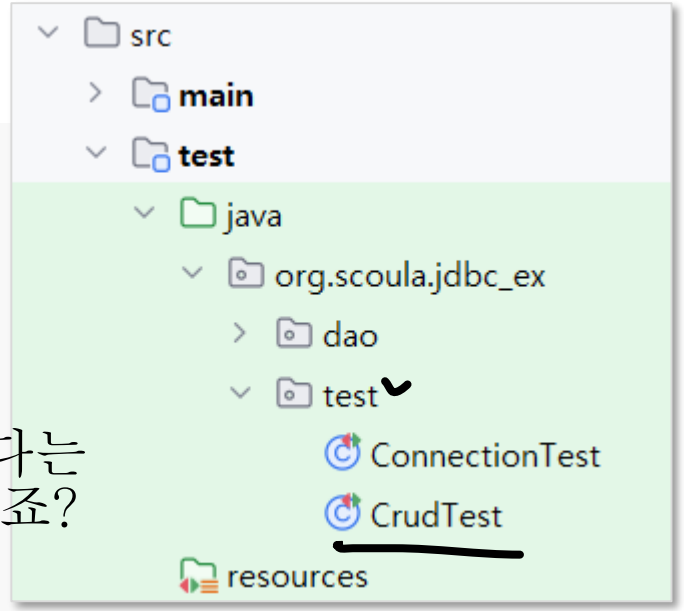
CrudTest.java

```
package org.scoula.jdbc_ex.test;

import org.junit.jupiter.api.*;
import org.scoula.jdbc_ex.common.JDBCUtil;

import java.sql.*;

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class CrudTest {
    Connection conn = JDBCUtil.getConnection();

    @AfterAll
    static void tearDown() {
        JDBCUtil.close();
    }
```

테스트 순서 직접 지정하겠다는 내용` 클래스 레벨에 지정햇죠?

sql문 실행 테스트를 하기전에 연결부터 해야함.
@BeforeAll메소드에서도 connection객체를 얻어도 됨.
지금은 그냥 초기화시 얻게함.

모든게 끝난후 실행되는 static 메소드



CrudTest.java

```
@Test
@DisplayName("새로운 user를 등록한다.")
@Order(1)
public void insertUser()  throws SQLException {
    String sql = "insert into users(id, password, name, role) values(?, ?, ?, ?)";
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, "scoula");
        pstmt.setString(2, "scoula3");
        pstmt.setString(3, "스콜라");
        pstmt.setString(4, "USER");

        int count = pstmt.executeUpdate();
        Assertions.assertEquals(1, count);
    }
}
```

순서 지정함

물음표 갯수는 ㅋ컴파일러가 확인 못하니 주의하자!!!!!

단위테스트할 메서드

미리 컴파일하게끔 넘겨주고(데이터부분제외

데이터 지정후

sql문 실행

insert니까 예상값 1로 단정문 확인

주의! 여기서 메소드가 SQL예외를 던지는데
try로 왜 감싸고 있지?
해당 구문에는 catch절이 없다!

해당 스코프에서는
예외처리하지 않고
try자동닫기 구문 기능만 사용하기 위해서이다!!

자동 닫기 구문 써야지 닫기를 수동으로 안해서 코드가 안 지저분해진다!!

만약 catch절을 사용해서 여기서 처리한다면 메서드 밖으로 예외가 안던져지기에 테스트 성공으로 인식할수 있다

catch문은 테스트에서는 조심

CrudTest.java

```
@Test
@DisplayName("user 목록을 추출한다.")
@Order(2)
public void selectUser() throws SQLException {
    String sql ="select * from users";
    try(Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql);
        ) {
        while(rs.next()) {
            System.out.println(rs.getString("name"));
        }
    }
}
```

순서 지정

변수처리 파트 없음

그래서 그냥 statement객체로 처리

이시점에서 컴파일됨.

resultset에서 한행식 접근 출력.

resulset도 자동닫기try범위안에 있다.

어차피 얘도 마무리로 닫기해야하니까

try(...; ....;) { .... } 이런 형식이었음

보통 한행씩 접근해 바로 행동(사용)하지 않고 일단 상위레벨로 넘기기 쉽게 리스트로 만들어 넘긴다.

어덯게 사용할지는 넘겨받은 상위레벨에서 정하는 것이 보편적이고 좋다

CrudTest.java

```
@Test
@DisplayName("특정 user 검색한다.")
@Order(3)
public void selectUserById() throws SQLException {
    String userid = "scoula";
    String sql ="select * from users where id = ?";
    try(PreparedStatement stmt = conn.prepareStatement(sql)){
        stmt.setString(1, userid);
        try(ResultSet rs = stmt.executeQuery()) {
            if(rs.next()) {
                System.out.println(rs.getString("name"));
            } else {
                throw new SQLException("scoula not found");
            }
        }
    }
}
```

앞과 다르게 sql에 변수가 있다면 다르게 처리해야한다.

resultset을 같은 try레벨에서 생성 못한다 왜냐하면 ?변수를 먼저 채워야 하기 때문!

변수 지정하고
안에서 중첩으로 try 자동닫기 문을 또 활용하여 resultset생성

예외를 던진 이유
pk조건에 의한 select의 결과는 하나 이거나 데이터가 없는 경우이다.
데이터가 없다고 실패한거나 잘못된 쿼리가 아니다. 그냥 데이터가 없는 것이다.
쿼리실행은 성공 한것이다. 데이터가 없을뿐
하지만 이 테스트는 scoula라는 아이디가 무조건 있다고 가정한 것이기에
왜냐면 앞에서 insert를 했으니까 ㅁ조건 순서(2)에서
문법적으로 논리적으로는 오류가 아니지만
맥락상 가정상의 ㄴ오류가 맞아서 일부러 예외를 던져준것임!!!

번외 테스트에는 어떤 상황을 가정하고 하는 테스트하고
상관없이 수행되어야하는 테스트하고 분리가 된다

바로 앞 34페이지에서 봤던게 전자다.
전자는 내부의 상황을 알아야하기에 화이트박스라고 불린다
후자는 블랙박스라고 불린다.

CrudTest.java

```
@Test
@DisplayName("특정 user 수정한다.")
@Order(4)
public void updateUser() throws SQLException {
    String userid = "scoula";
    String sql ="update users set name= ? where id = ?";
    try(PreparedStatement stmt = conn.prepareStatement(sql)){
        stmt.setString(1, "스콜라 수정");
        stmt.setString(2, userid);
        int count = stmt.executeUpdate();
        Assertions.assertEquals(1, count);
    }
}
```

미리 컴파일

쿼리 실행

static메서드는
static import기능을 써서 편하게 쓸 수 있다.

```
import static org.junit.jupiter.api.Assertions.assertEquals;
```

클래스말고 static메서드를 임포트함.

임포트하고 assertEquals(...)만 쓰면 된다.

그냥 함수 사용하듯이

alt+enter로 빠르게 staticimport를 할 수있다.

CrudTest.java

```
    @Test
    @DisplayName("지정한 사용자를 삭제한다.")
    @Order(5)
    public void deleteUser() throws SQLException {
        String userid = "scoula";
        String sql ="delete from users where id = ?";
        try(PreparedStatement stmt = conn.prepareStatement(sql)){
            stmt.setString(1, userid);
            int count = stmt.executeUpdate();
            Assertions.assertEquals(1, count);
        }
    }
}
```

번외 결과에서 데이터를 추출할때
개별 칼럼 단위로 처리를 하면은 흩어져 있어서
복잡.= >캡슐화 필요.
또한
쿼리문에 사용되는 데이터들을 캡슐화할 필요가 있다.
=>
VO패턴

근데 메서드 별로 실행시키면
오더가 의미가 없잖아.

클래스 레벨에서 테스트 실행시키면
그때
지정한 순서대로 테스트를 실행한다

만약 순서 지어 없이 디폴트면 랜덤 순서로
메서드 테스트 실행

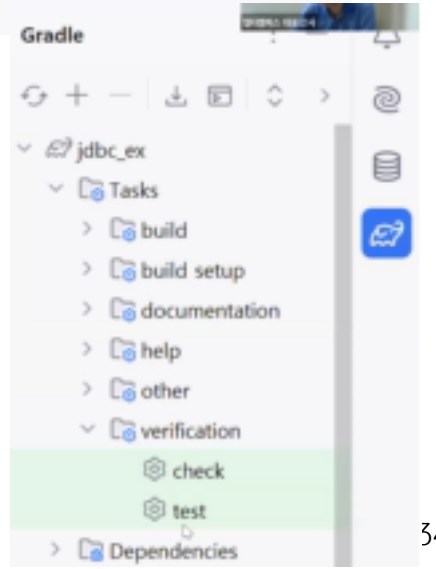
클래스 단위 테스트가 아닌 테스트 패키지에 들어있는 모든 것들을 일괄 테스트하고 싶을 때는

gradle툴을 이용하면 된다.

검증>테스트 를 더블 클릭해주면 모든 테스트를 일괄 진행한다.

방ㄱ므은 gui로 실행했지만 명령어로 할 수도있따.

나중에 ci/cd 지속통활배포 프로세스를 만들때 자동화 시킬수있다.



VO 패턴

VO 객체

- Value Object
- 특정 테이블의 한 행을 매핑하는 클래스

데이터베이스의 테이블과 1대1 대응.

테이블을 그대로 표현한 객체임.

Java파트 <->DB파트

클래스 정의 → 테이블
필드들 → 컬럼들
인스턴스 → 한 행

테이블은 클래스로
한 컬럼은 한 필드로

한 행은 하나의 인슨턴스로
표현

테이블의 모양과 클래스의 모양이 같다
이런 형식의 객체는 VO라고 ㅇ한다

UserVO.java

VO형식객체은 도메인이라고도 불린다. 현재 문제를 나타내는 데이터를 말한다.

```
package org.scoula.jdbc_ex.domain;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor  롬복 사용
@AllArgsConstructor
public class UserVO {
    private String id;
    private String password;
    private String name;
    private String role;
}
```

DB테이블의 칼럼의 데이터 타입과
대응하는 필ㄹ드의 데이터 타입
맞춰야함

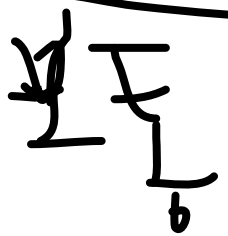
DAO 패턴 적용

앱 개발시 SRP원칙에 따라 DB연동을 담당하는 클래스가 필요하다!!!
이를 Data Access Lyaer라고 부른다 줄여서 DAO.
특징은 비즈니스 로직은 없고 데이터에 대한 CUURD만 가지고 있다

DAO 클래스

- Data Access Object
- 데이터베이스에 접근하여 실질적인 데이터베이스 연동 작업을 담당하는 클래스
- 테이블에 대한 CRUD 연산을 처리

인터페이스 정의 후 구현 클래스 작성



UserDao.java

인터페이스

```
package org.scoula.jdbc_ex.dao;

import org.scoula.jdbc_ex.domain.UserVO;

import java.sql.SQLException;
import java.util.List;
import java.util.Optional;

public interface UserDao {
    // 회원 등록  안에는 insert문이 활용되겠죠?
    int create(UserVO user) throws SQLException;

    // 회원 목록 조회  안에는 select문을 활용하여 R.S.를 받고 list로 변환해야겠죠?
    List<UserVO> getList() throws SQLException;

    // 회원 정보 조회  안에서는 조건있는 select문이 있겠져?
    Optional<UserVO> get(String id) throws SQLException;

    // 회원 수정  안에서 update문이 활용되겠죠
    int update(UserVO user) throws SQLException;

    // 회원 삭제  안에서 dELETE문이 활요되겠죠오오
    int delete(String id) throws SQLException;
}
```

데이터를 활용시
아이디 패스워드 등등의 데이터를 따로 넘기지 않고
VO객체에 담아 캡슐화해서 한꺼번에 넘긴다

편하기도 하지만 추후에 넘겨줄 데이터 정보가
늘어나도
하나만 수정하면 되니까 편하다
제거해도 마찬가지

UserDaoImpl.java

구현체

```
package org.scoula.jdbc_ex.dao;

import org.scoula.jdbc_ex.common.JDBCUtil;
import org.scoula.jdbc_ex.domain.UserVO;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

public class UserDaoImpl implements UserDao {
    Connection conn = JDBCUtil.getConnection();

    // USERS 테이블 관련 SQL 명령어
    private String USER_LIST   = "select * from users";
    private String USER_GET    = "select * from users where id = ?";
    private String USER_INSERT = "insert into users values(?, ?, ?, ?)";
    private String USER_UPDATE = "update users set name = ?, role = ? where id = ?";
    private String USER_DELETE = "delete from users where id = ?";
```

앞에 final 붙이는게 더 적절

UserDaoImpl.java

```
private String USER_INSERT = "insert into users values(?, ?, ?, ?)";
```

```
// 회원 등록
@Override
public int create(UserVO user) throws SQLException {
    try (PreparedStatement stmt = conn.prepareStatement(USER_INSERT)) {
        stmt.setString(1, user.getId());
        stmt.setString(2, user.getPassword());
        stmt.setString(3, user.getName());
        stmt.setString(4, user.getRole());
        return stmt.executeUpdate();
    }
}
```

UserDaoImpl.java

```
private UserVO map(ResultSet rs) throws SQLException {
    UserVO user = new UserVO();
    user.setId(rs.getString("ID"));
    user.setPassword(rs.getString("PASSWORD"));
    user.setName(rs.getString("NAME"));
    user.setRole(rs.getString("ROLE"));
    return user;
}

// 회원 목록 조회
@Override
public List<UserVO> getList() throws SQLException{
    List<UserVO> userList = new ArrayList<UserVO>();
    Connection conn = JDBCUtil.getConnection();
    try (PreparedStatement stmt = conn.prepareStatement(USER_LIST);
         ResultSet rs = stmt.executeQuery()) {
        while(rs.next()) {
            UserVO user = map(rs);
            userList.add(user);
        }
    }
    return userList;
}
```

해당 메소드는
@Builder 어노테이션
하는 것도 적절

```
private String USER_LIST   = "select * from users";
```

recordset->userVo로 변환

UserDaoImpl.java

```
// 회원 정보 조회
@Override
public Optional<UserVO> get(String id) throws SQLException{
    try (PreparedStatement stmt = conn.prepareStatement(USER_GET)) {
        stmt.setString(1, id);
        try(ResultSet rs = stmt.executeQuery()) {
            if(rs.next()) {
                return Optional.of(map(rs));
            }
        }
    }
    return Optional.empty();
}
```

PK

```
private String USER_GET    = "select * from users where id = ?";
```

결과 resultset 안 실제 데이터 행 갯수는 0 아님 1개이기에

현재 상황은 결과에서 데이터가 안나와도 비정상은 아님!

하지만 호출한 쪽에서 널 체크를 해야하는데 이를 강제화하는 optional객체를 사용한다.

optional클래스 특징은 static메소드를 통해서 해당 객체를 생성시켜야한다. 다 private이므로.

데이터(실데이터 행)가 존재하는 경우 Optional.of(...) 로 생성. 데이터가 잇다고 확정짓는 것.
데이터가 없다면 null리턴이 아닌 비어있는 optional객체를 반환한다. 이는 optional.empty()로.

UserDaoImpl.java

```
private String USER_UPDATE = "update users set name = ?, role = ? where id = ?";
```

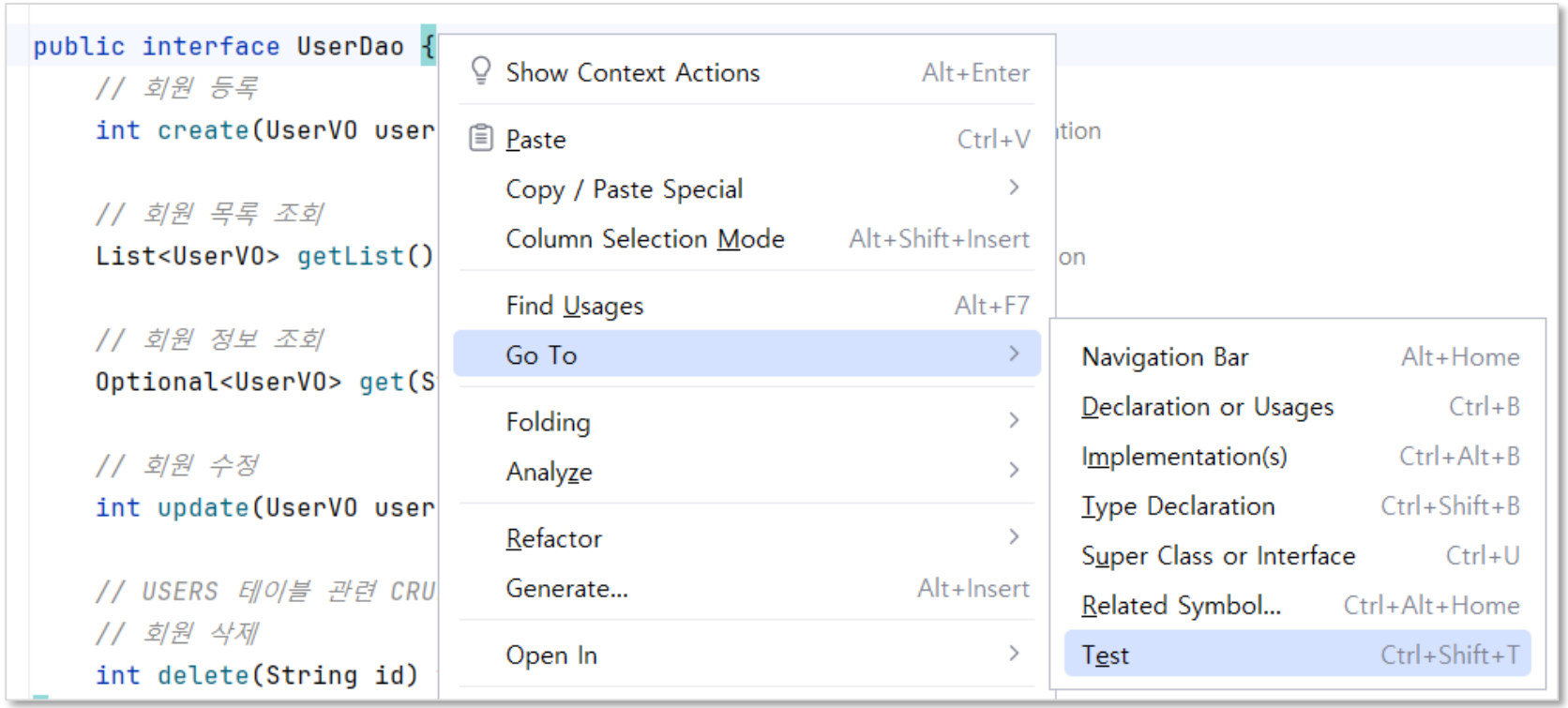
```
    // 회원 수정
    @Override
    public int update(UserVO user) throws SQLException{
        Connection conn = JDBCUtil.getConnection();
        try ( PreparedStatement stmt = conn.prepareStatement(USER_UPDATE)) {
            stmt.setString(1, user.getName());
            stmt.setString(2, user.getRole());
            stmt.setString(3, user.getId());
            return stmt.executeUpdate();
        }
    }

    // USERS 테이블 관련 CRUD 메소드
    // 회원 삭제
    @Override
    public int delete(String id) throws SQLException{
        try(PreparedStatement stmt = conn.prepareStatement(USER_DELETE)) {
            stmt.setString(1, id);
            return stmt.executeUpdate();
        }
    }
}
```

```
private String USER_DELETE = "delete from users where id = ?";
```

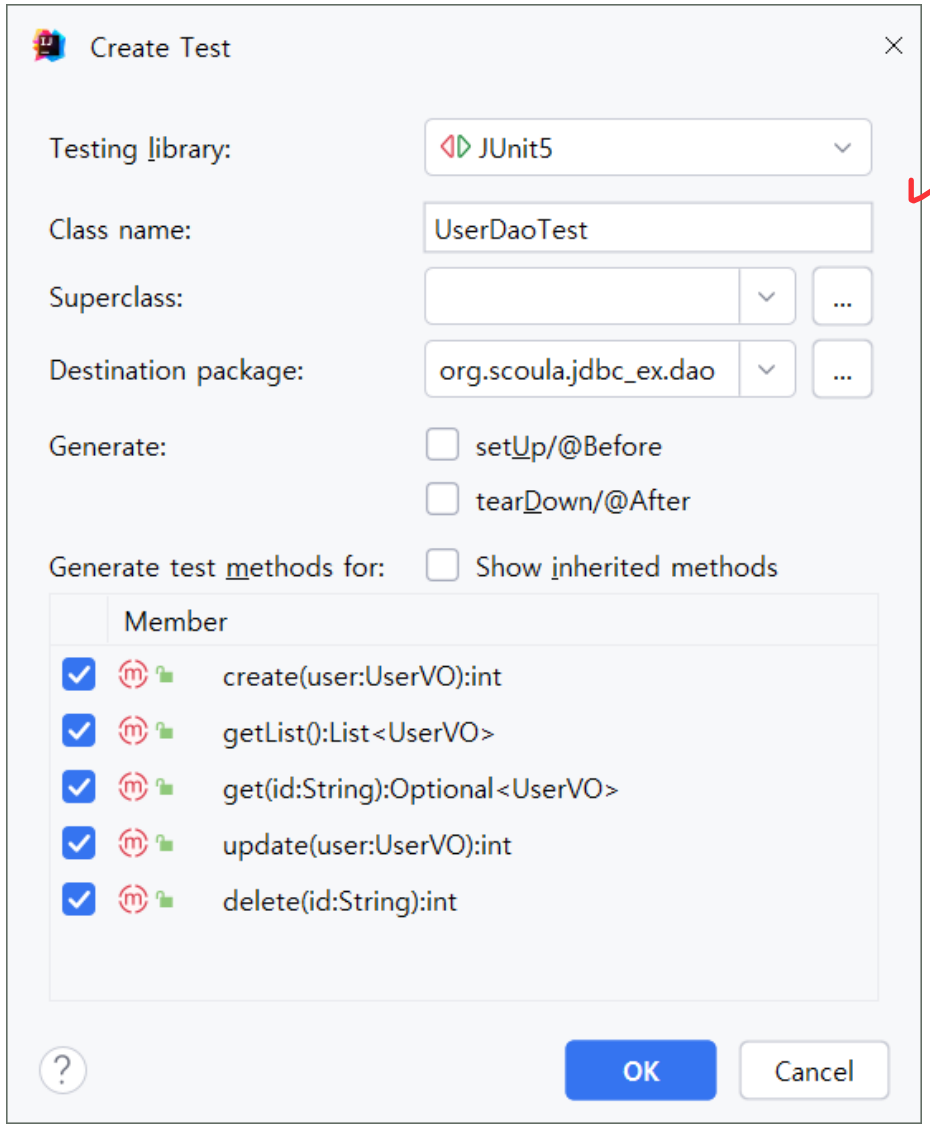
번외.
위까지 데이터 접근 계층. 비즈니스 로직은 없음. 이걸로 뭐할진
몰라. 요구한 데이터 접근 및 처리만 맡음. 비즈니스 로직은 누가 담당.
서비스 계층에서 하면된다.

아까만든 인터페이스 구현체를 단위 테스트

UserDao 테스트 클래스



UserDao 테스트 클래스

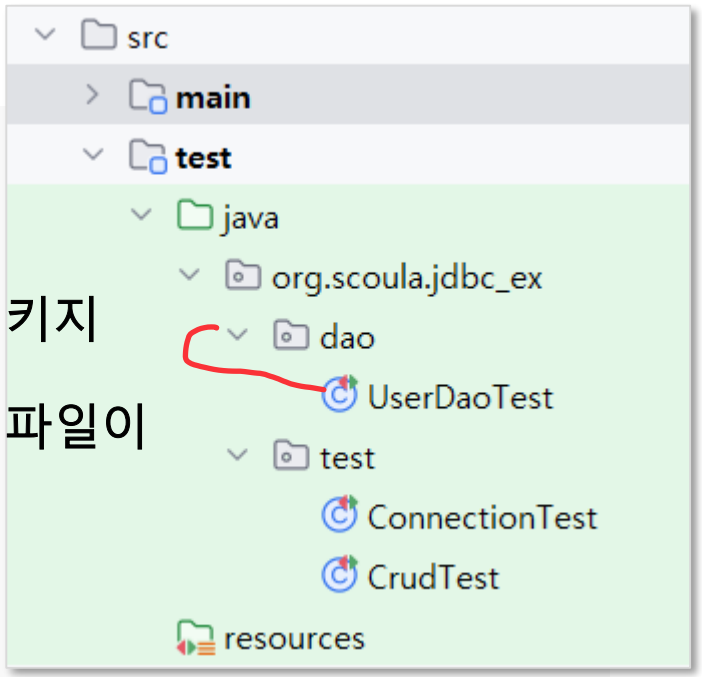


UserDaoTest.java

```
package org.scoula.jdbc_ex.dao;

import org.junit.jupiter.api.*;
import org.scoula.jdbc_ex.common.JDBCUtil;
import org.scoula.jdbc_ex.domain.UserVO;

import java.sql.SQLException;
import java.util.List;
import java.util.NoSuchElementException;

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
class UserDaoTest {
    UserDao dao = new UserDaoImpl();

    @AfterAll
    static void tearDown() {
        JDBCUtil.close();
    }
```

테스트 구성및코드 작성

그럼 같은 패키지 안에서 해당 테스트 파일이 만들어짐



UserDaoTest.java

```
@Test
@DisplayName("user를 등록합니다.")
@Order(1)
void create() throws SQLException {
    UserVO user = new UserVO("ssamz3", "ssamz123", "쌤즈", "ADMIN");
    int count = dao.create(user);
    Assertions.assertEquals(1, count);
}

@Test
@DisplayName("UserDao User 목록을 추출합니다.")
@Order(2)
void getList() throws SQLException {
    List<UserVO> list = dao.getList();
    for(UserVO vo: list) {
        System.out.println(vo);
    }
}
```

UserDaoTest.java

```
    @Test
    @DisplayName("특정 user 1건을 추출합니다.")
    @Order(3)
    void get() throws SQLException {
        UserVO user = dao.get("ssamz3").orElseThrow(NoSuchElementException::new);
        Assertions.assertNotNull(user);
    }

    @Test
    @DisplayName("user의 정보를 수정합니다.")
    @Order(4)
    void update() throws SQLException {
        UserVO user = dao.get("ssamz3").orElseThrow(NoSuchElementException::new);
        user.setName("쌤즈3");
        int count = dao.update(user);
        Assertions.assertEquals(1, count);
    }

    @Test
    @DisplayName("user를 삭제합니다.")
    @Order(5)
    void delete() throws SQLException {
        int count = dao.delete("ssamz3");
        Assertions.assertEquals(1, count);
    }
}
```

get메서드는 optional반환

orelsethrow 데이터 있으면 유저VO반환
없으면 예외를 발생시켜라. 인자값은 생성자 참조.
원래는 ( ) -> new NoSuch...Exception( ) 이코드임.

근데 어차피 디폴트가 NoSuch...예외인데
그냥 ( );로 호출하면 된다

여기까지가 JDBC프로그래밍 파트임. 나머진 서비스 레이어에서 비즈니스 로직을 다루면 된다