

2024년 상반기 K-디지털 트레이닝

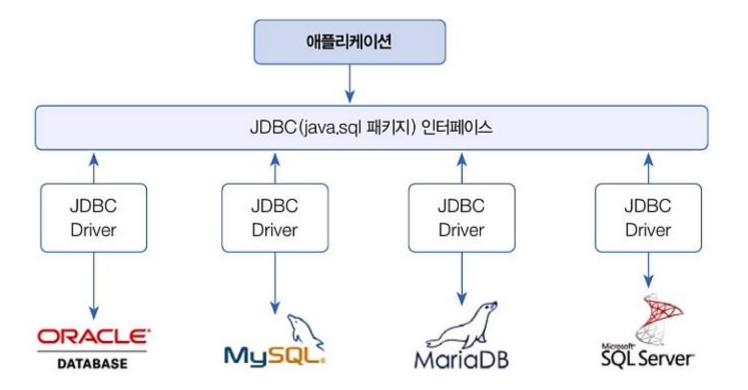
# JDBC 프로그래밍

[KB] IT's Your Life



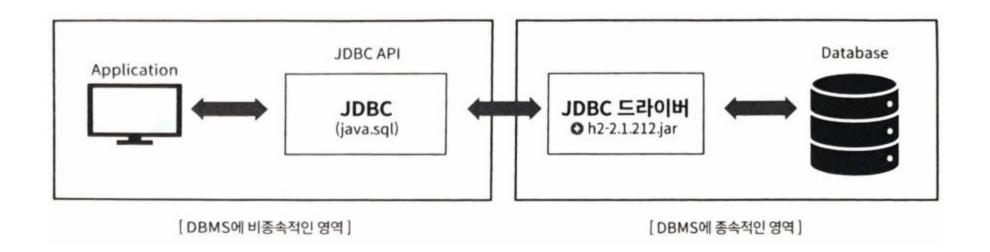
## JDBC Java Database Connectivity

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



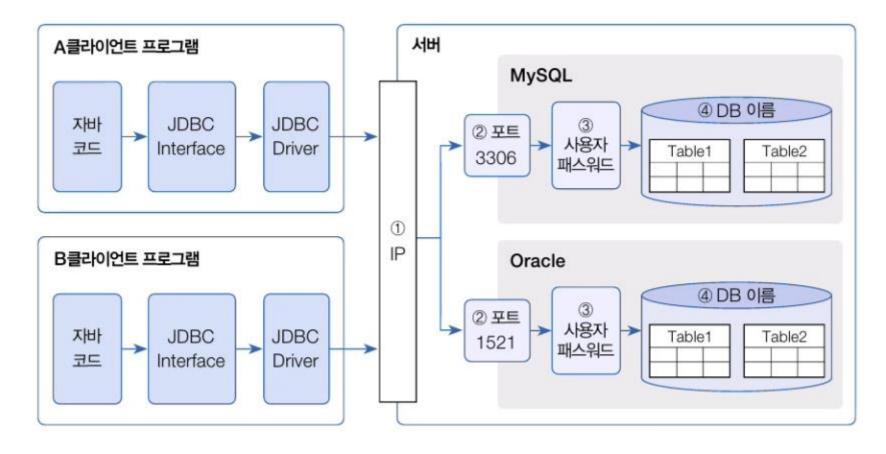
#### **JDBC**

o JDBC 개념



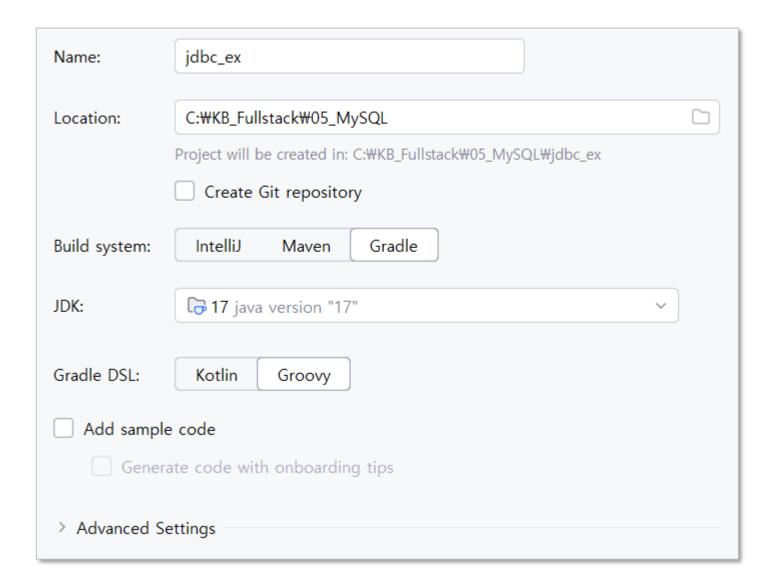
# JDBC

#### o JDBC 개념



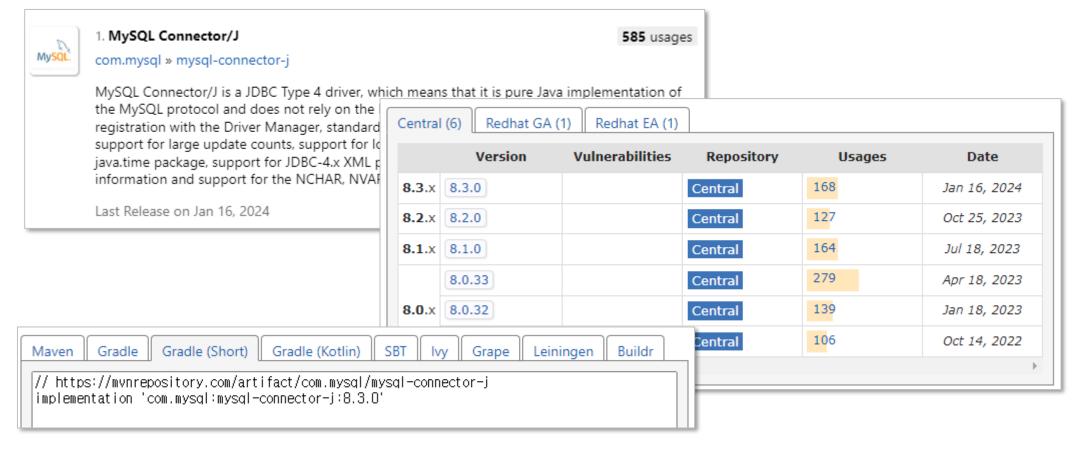
#### 🗸 프로젝트 생성

- Name: jdbc\_ex
- Build System : gradle
- o Group Id: org.scoula



#### **MySQL Connector**

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



Lombok도 추가

# **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'
}
```

#### O 수정 후 Sync 실행

#### 〇 프로젝트 설정

Annotation Processing 활성화

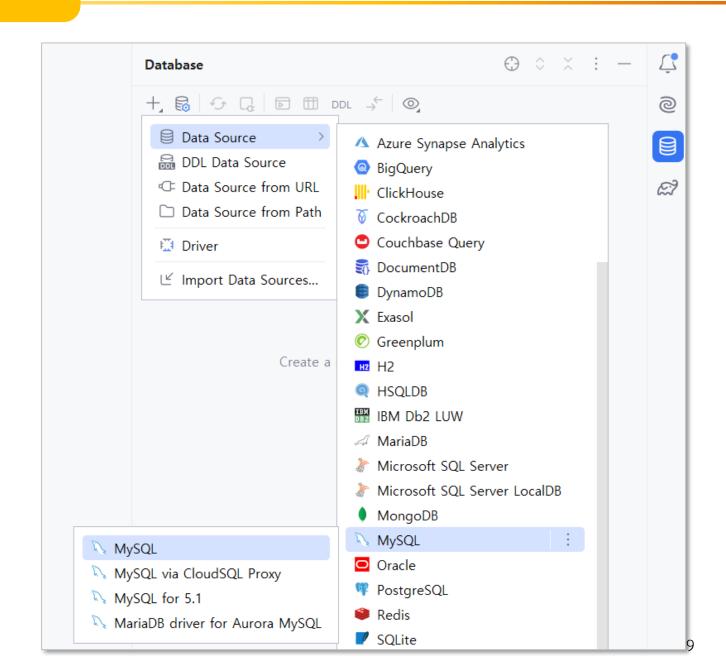
#### ♡ 데이터베이스 준비

CREATE DATABASE jdbc\_ex;

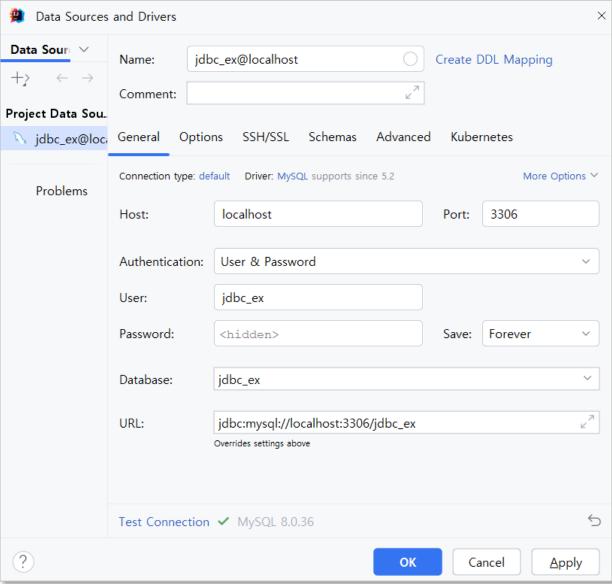
#### 🗸 사용자 준비

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

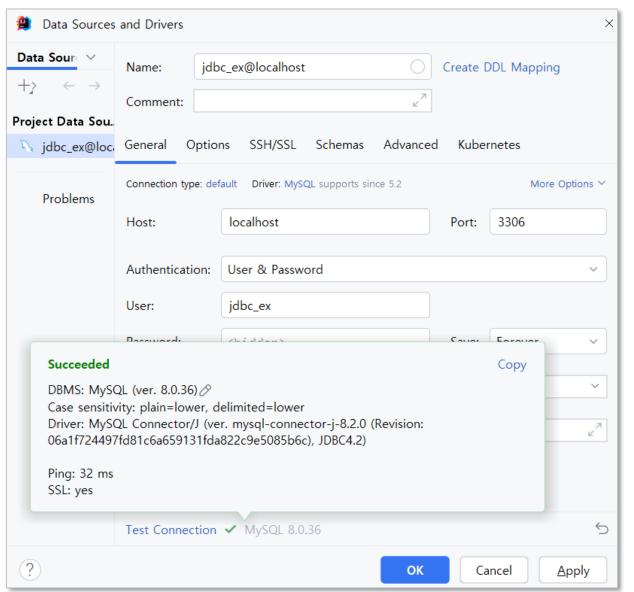
Intellij Datasource 기능 설정



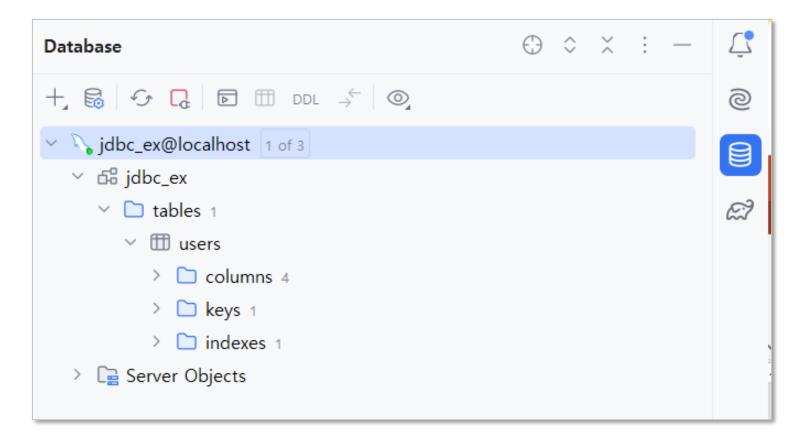
Intellij Datasource 기능 설정



### ✓ Intellij Datasource 기능 설정



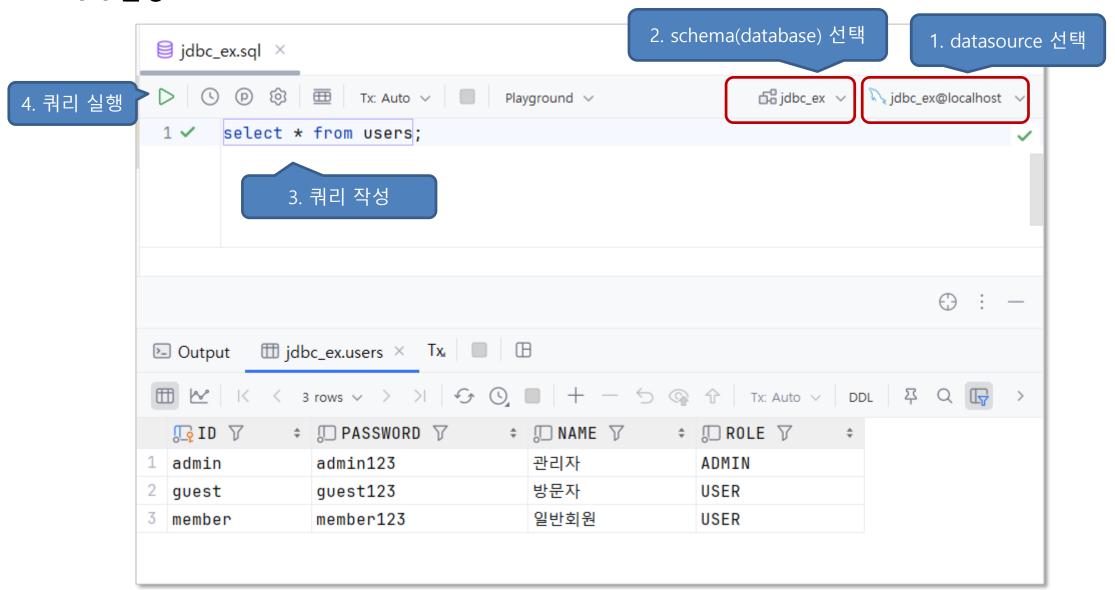
#### datasource 목록



## 🥝 sql 파일 만들기



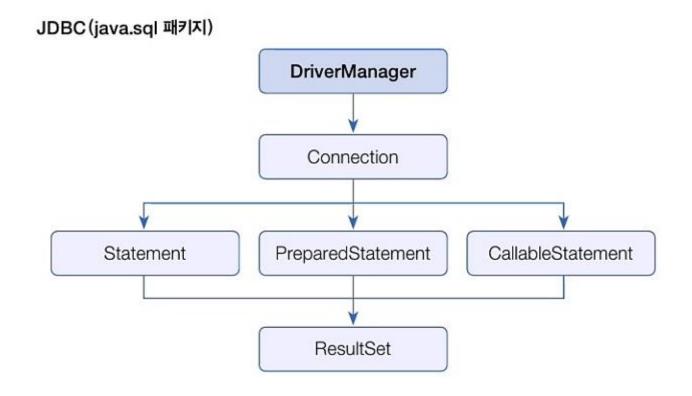
#### 💟 쿼리 실행



### 🕜 데이터 준비

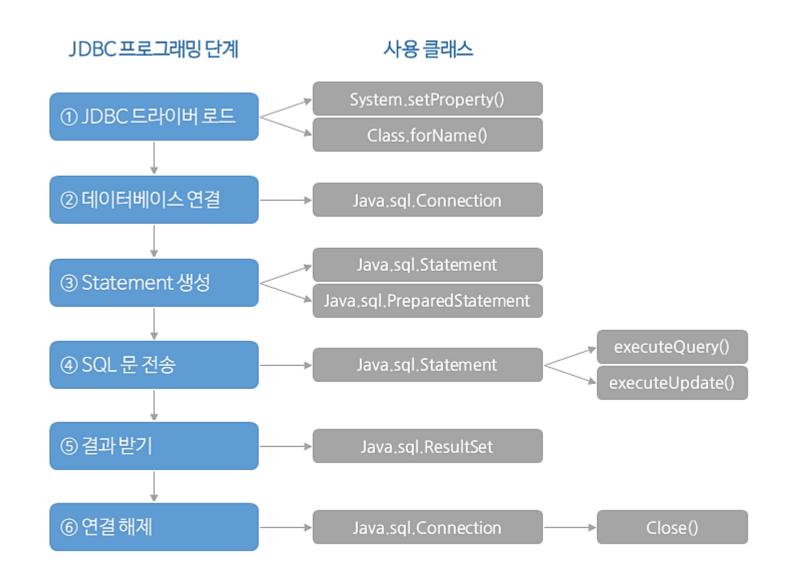
```
use jdbc_ex;
CREATE TABLE USERS (
  ID VARCHAR(12) NOT NULL PRIMARY KEY,
  PASSWORD VARCHAR(12) NOT NULL,
  NAME VARCHAR(30) NOT NULL,
  ROLE VARCHAR(6) NOT NULL
INSERT INTO USERS(ID, PASSWORD, NAME, ROLE)
VALUES('guest', 'guest123', '방문자', 'USER');
INSERT INTO USERS(ID, PASSWORD, NAME, ROLE)
VALUES('admin', 'admin123', '관리자', 'ADMIN');
INSERT INTO USERS(ID, PASSWORD, NAME, ROLE)
VALUES('member', 'member123', '일반회원', 'USER');
SELECT * FROM USERS;
```

### JDBC의 핵심 인터페이스/클래스



#### **JDBC**

JDBC 개발 절차



#### DB 연결

○ 드라이버 확인

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

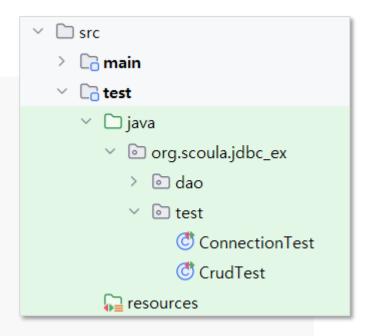
→ 없으면 ClassNotFoundException 발생

#### o Connection 객체

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

# 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 = "idbc ex";
        String password = "jdbc ex";
        Connection conn = DriverManager.getConnection(url, id, password);
        System.out.println("DB 연결 성공");
        conn.close();
                                           DB 연결 성공
```



- 🗸 모듈화해야 할 코드
  - 데이터베이스 연결 및 닫기 작업은 항상 필요함
  - → common.JDBCUtil

# jdbc\_ex.common/JDBCUtil.java

```
package org.scoula.jdbc_ex.common;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class JDBCUtil {
  static Connection conn = null;
  static {
    try {
      Class.forName("com.mysql.cj.jdbc.Driver");
      String url = "jdbc:mysql://127.0.0.1:3306/jdbc_ex";
      String id = "jdbc_ex";
      String password = "jdbc_ex";
      conn = DriverManager.getConnection(url, id, password);
    } catch (Exception e) {
      e.printStackTrace();
```

# jdbc\_ex.comon/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();
```

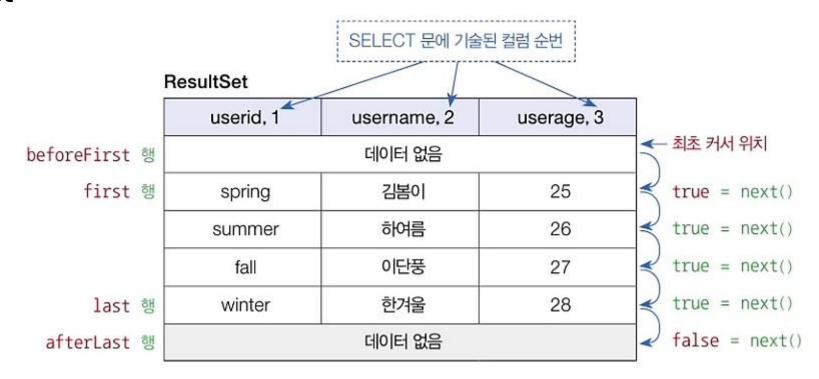
# ConnectionTest.java

```
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");
```

#### Statement

- o SQL 문 실행 클래스
- o Connection 객체를 통해 생성 Statement stmt = conn.createStatement();
- SQL 실행 메서드
  - ResultSet executeQuery(SQL문): select문 실행
  - int executeUpdate(SQL문): insert, update, delete 문 실행

#### ResultSet



#### ○ 컬럼 값 추출

- getXxxx("컬러명)
  - Xxx: 추출하고자하는 데이터 타입명
  - getString(), getInt(), getLong(), getDouble()

#### PreparedStatement

○ SQL문에 값을 넣을 때 파라미터화 해서 처리

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

o Connection 객체를 통해 생성

PreparedStatement pstmt = conn.prepareStatement(sql);

- 파라미터 설정
  - pstmt.setXxxx(파라미터번호, 값)
    - setString(), setInt(), setLong(), setDouble()
- o SQL문 실행

int count = pstmt.executeUpdate();

#### Statement로 Insert 문 실행하기

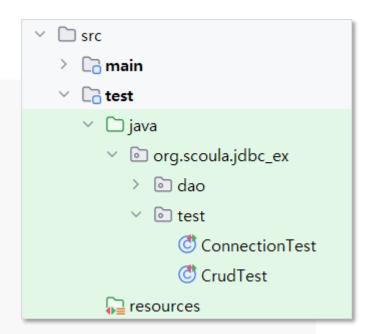
#### ○ 값을 변수로 대체한다면?

→ PreparementStatement로 처리

```
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();
    }
}
```



```
@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);
    }
}
```

```
@Test
@DisplayName("user 목록을 추출한다.")
@0rder(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"));
```

```
@Test
@DisplayName("특정 user 검색한다.")
@0rder(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");
```

```
@Test
@DisplayName("특정 user 수정한다.")
@0rder(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);
```

```
@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);
    }
}
```

#### VO 패턴

- o VO 객체
  - Value Object
  - 특정 테이블의 한 행을 매핑하는 클래스

클래스 정의 → 테이블 필드들 → 컬럼들 인스턴스 → 한 행

# UserVO.java

```
package org.scoula.jdbc_ex.domain;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

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

#### DAO 패턴 적용

- 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

```
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
   int create(UserVO user) throws SQLException;
   // 회원 목록 조회
   List<UserVO> getList() throws SQLException;
   // 회원 정보 조회
   Optional<UserVO> get(String id) throws SQLException;
   // 회원 수정
   int update(UserVO user) throws SQLException;
   // USERS 테이블 관련 CRUD 메소드
   // 회원 삭제
   int delete(String id) throws SQLException;
```

```
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 = ?";
```

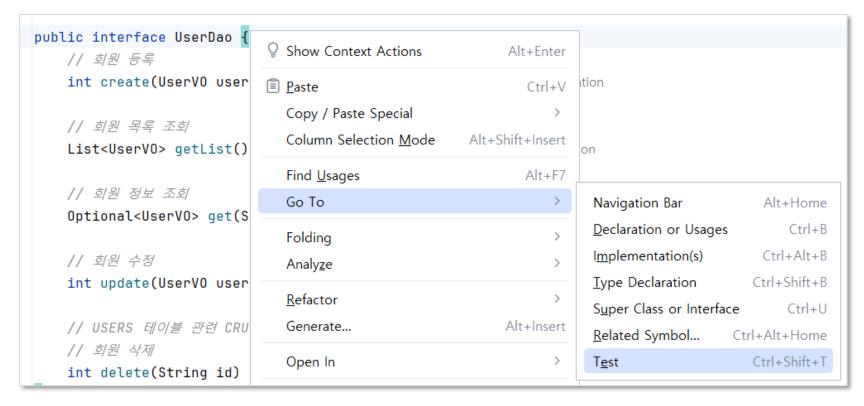
```
// 회원 등록
                                                  private String USER_INSERT = "insert into users values(?, ?, ?, ?)";
@Override
public int create(UserVO user) throws SQLException {
    try (PreparedStatement stmt = conn.prepareStatement(USER_INSERT)) {
        stmt.setString(1, user.getId());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

```
private UserVO map(ResultSet rs) throws SQLException {
    UserV0 user = new UserV0();
    user.setId(rs.getString("ID"));
    user.setPassword(rs.getString("PASSWORD"));
    user.setName(rs.getString("NAME"));
    user.setRole(rs.getString("ROLE"));
   return user;
                                             private String USER_LIST = "select * from users";
// 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()) {
            UserV0 user = map(rs);
           userList.add(user);
    return userList;
```

```
// 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();
}
```

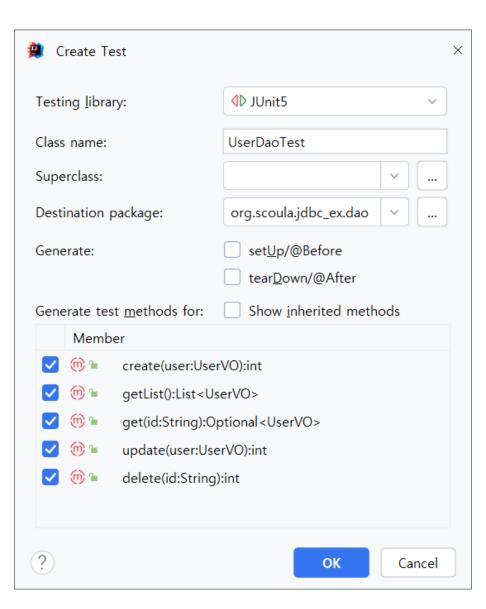
```
private String USER_UPDATE = "update users set name = ?, role = ? where id = ?";
// 회원 수정
@Override
public int update(UserVO user) throws SQLException{
    Connection conn = JDBCUtil.getConnection();
   try ( PreparedStatement stmt = conn.prepareStatement(USER UPDATE)) {
        stmt.setString(1, user.getName());
        stmt.setString(2, user.getRole());
        stmt.setString(3, user.getId());
        return stmt.executeUpdate();
// USERS 테이블 관련 CRUD 메소드
// 회원 삭제
                                                    private String USER DELETE = "delete from users where id = ?";
@Override
public int delete(String id) throws SQLException{
    try(PreparedStatement stmt = conn.prepareStatement(USER DELETE)) {
       stmt.setString(1, id);
        return stmt.executeUpdate();
```

#### UserDao 테스트 클래스



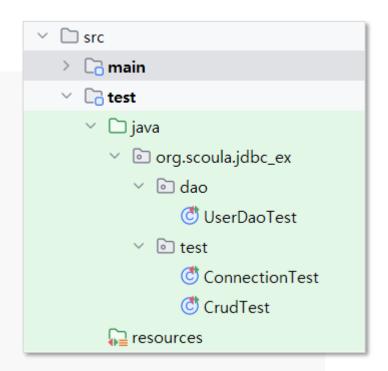


## UserDao 테스트 클래스



# 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();
```



# UserDaoTest.java

```
@Test
@DisplayName("user를 등록합니다.")
@0rder(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.")
@0rder(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.")
@0rder(3)
void get() throws SQLException {
   UserVO user = dao.get("ssamz3").orElseThrow(NoSuchElementException::new);
   Assertions.assertNotNull(user);
@Test
@DisplayName("user의 정보를 수정합니다.")
@0rder(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.")
@0rder(5)
void delete() throws SQLException {
   int count = dao.delete("ssamz3");
   Assertions.assertEquals(1, count);
```