14장 JDBC

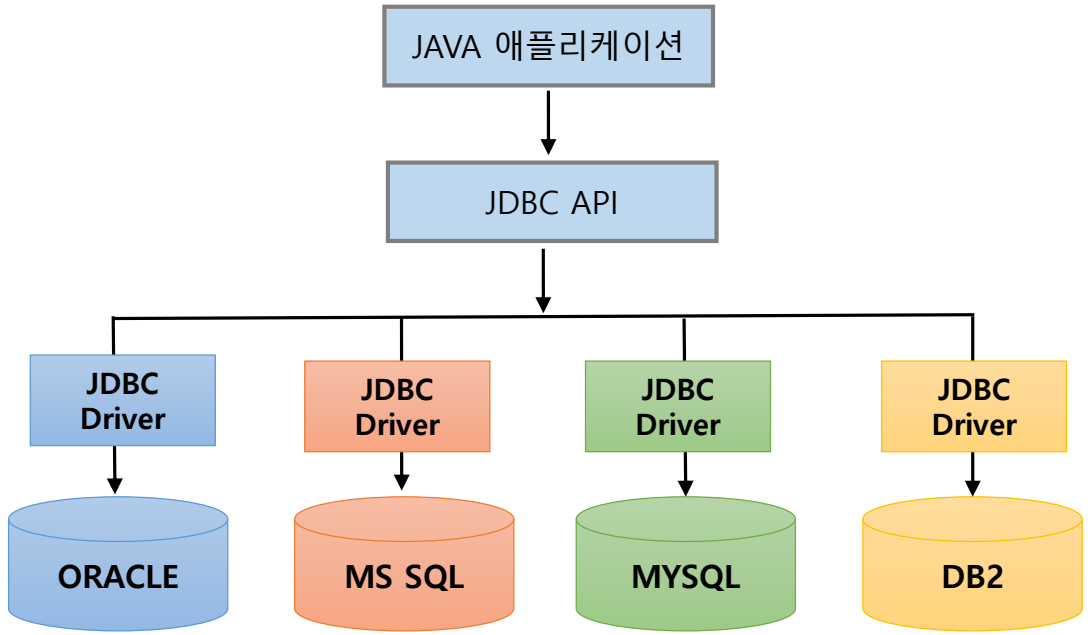
JDBC

◈ JDBC(Java Database Connectivity)

데이터베이스에 연결 및 작업을 하기 위한 자바 표준 인터페이스(API)이다.

JDBC를 사용하여 데이터베이스에 연결 및 데이터에 대하여 검색하고, 데이터를 변경할 수 있게 한다.

하나의 인터페이스로 모든 데이터베이스에 연결할 수 있다.



JDBC

◈ JDBC 준비

Oracle jdbc 드라이버인 ojdbc6.jar 파일 다운받기

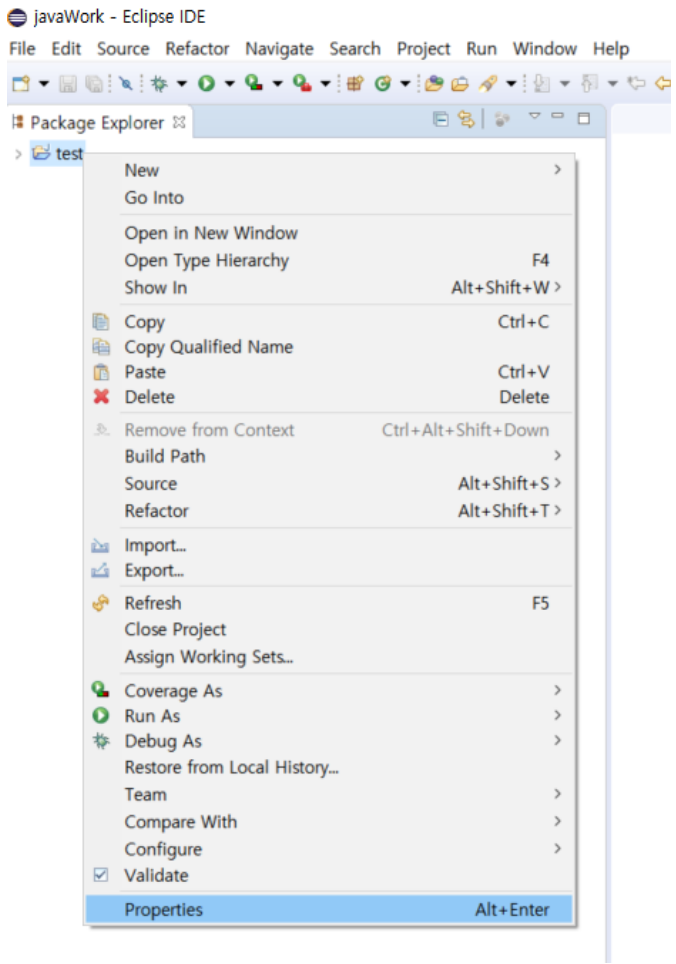
다운로드: https://repo1.maven.org/maven2/com/oracle/database/jdbc/ojdbc6/11.2.0.4/

파일이름: ojdbc6-11.2.0.4

JDBC

◈ 이클립스에 .jar 라이브러리 적용

1. 프로젝트 우클릭 – Properties 클릭



JDBC

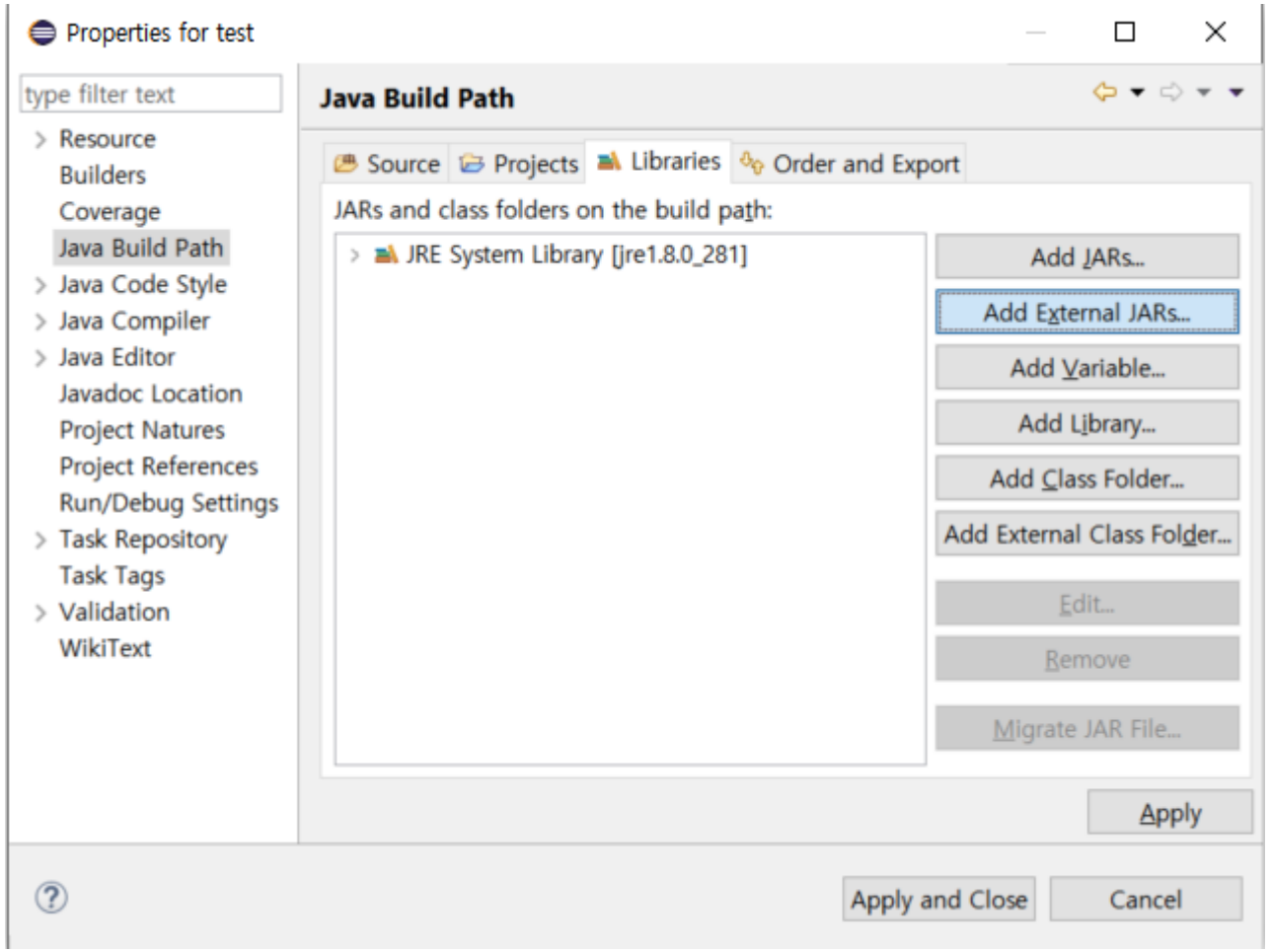
◈ 이클립스에 .jar 라이브러리 적용

2. Java Build Path 항목 선택

2-1. Libraries 탭 선택

2-2. Add External JARs.. 클릭

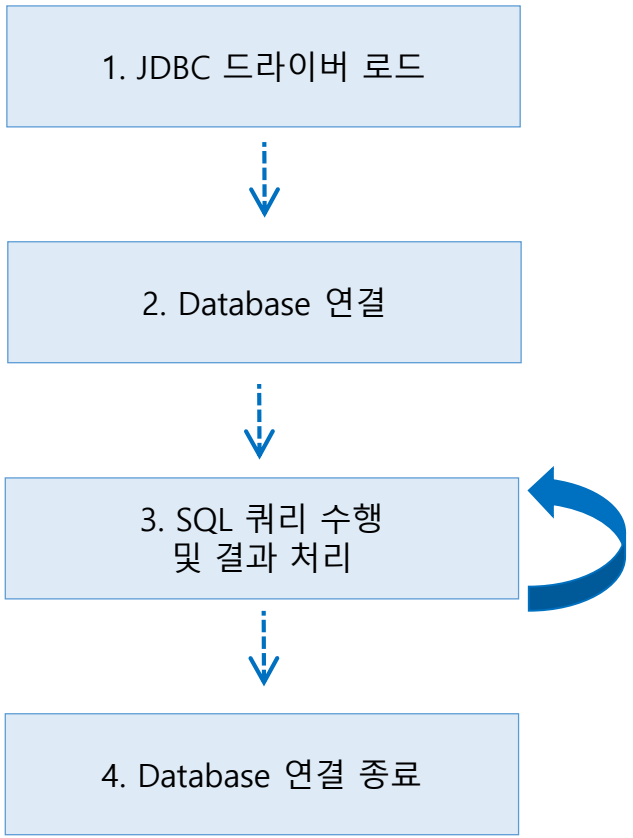
2-3. 다운받은 ojdbc6-11.2.0.4 적용



JDBC

◈ JDBC API 프로그래밍

1. 데이터베이스 연결을 위한 드라이버 로드
2. 데이터베이스 연결을 관리하는 Connection 객체 생성
3. SQL 문 작성 및 수행
4. 연결 종료



JDBC

◈ JDBC 드라이버 로드

프로젝트 내에서 한번만 로드 되면 된다.

```
// 1. 드라이버 로딩 (딱 한번만 로딩 필요)
try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    System.out.println("드라이버 등록 성공");
} catch (ClassNotFoundException e1) {
    e1.printStackTrace();
    System.out.println("드라이버 등록 실패");
    System.exit(0);     // 실행 종료
}
```

← JVM 에 드라이버를 로드

드라이버를 로드해야 이후
DriverManager.getConnection()
이 가능해진다.

JDBC

◈ Connection 객체 생성

try/catch 문으로 연결에 실패하는 경우에 대한 예외 처리(SQLException 사용)를 해야한다.

DriverManager.getConnection() 안에 URL과 계정 ID, 계정 PW 를 입력

URL 예시 -> jdbc:oracle:thin:@127.0.0.1:1521:xe

DB 서버 주소 및 포트번호

DB 의 인스턴스

```
try {
    // 데이터베이스 서버와 연결(Connect) 한다.
    conn = DriverManager.getConnection("jdbc:oracle:thin:@127.0.0.1:1521:orcl", "계정ID", "계정PW");
} catch (SQLException e) {
    e.printStackTrace();
}
```

JDBC

◈ 쿼리문(SQL) 객체 생성

만약 값을 조회해야하는 SELECT문의 경우 실행 결과를 담을 ResultSet 객체를 생성해야 한다.

```
Statement stmt = null;
ResultSet rs = null;

// 3. 쿼리문 객체를 생성한다.
stmt = conn.createStatement();

// 4. 쿼리문 실행과 동시에 결과를 변수에 담는다.
rs = stmt.executeQuery("SELECT * FROM test");

// 5. 실행결과 처리(다음 행이 있으면 true를 반환하며 커서가 한칸 내려간다)
while(rs.next()) {
    String id = rs.getString("id");
    String name = rs.getString("name");
    int age = rs.getInt("age");
    System.out.println(id + ": " + name + ", " + age);
}
```

쿼리문 실행 실행될 쿼리문

DB

질의 결과 ×

SQL | 인출된 모든 행: 5(0.159초)

	ID	NAME	AGE
1	20220321121743984684414	융가리	16
2	20220321121743989764568	강건마	14
3	20220321121743990808308	마영웅	15
4	20220321121743991286794	표독수	16
5	20220321121743992858787	지대호	17

위에서부터 한줄씩 while문 안에 담기고, get을 이용하여 값을 확인할 수 있다.

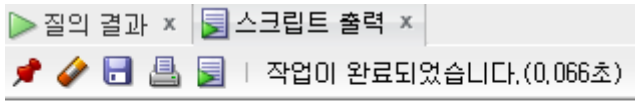
JDBC

◈ INSERT, DELETE, UPDATE 쿼리문의 경우 ResultSet이 필요없으며, 쿼리문의 실행 결과 또한 단순히 실행된 행의 개수를 리턴한다.

```
int cnt = stmt.executeUpdate("INSERT INTO test VALUES('a002', '최수종', 50)");
```

```
if(cnt > 0) {
    System.out.println(cnt + "행 삽입되었습니다.");
}else {
    System.out.println("삽입이 정상적으로 이루어지지 않았습니다.");
}
```

DB




```
1 행 이(가) 삽입되었습니다.
```

JDBC

◈ PreparedStatement

기존 Statement 객체로는 WHERE 절이나 ORDER BY 절 등 조건을 주어야 하는 경우 유연하게 적용하기 힘들며, 보안에도 취약하기 때문에 PreparedStatement 를 사용한다.

쿼리문은 StringBuffer를 이용하여 한줄 한줄 추가(append)해서 PreparedStatement 에 적용한다.

```
StringBuffer query = new StringBuffer();

query.append(" SELECT         ");
query.append("       id       ");
query.append("     , name     ");
query.append("     , age      ");
query.append(" FROM           ");
query.append("     test       ");
query.append(" ORDER BY       ");
query.append("     id         ");

PreparedStatement ps = conn.prepareStatement(query.toString());
ResultSet rs = ps.executeQuery();
```

JDBC

◈ PreparedStatement

? 를 이용하여 쿼리문 조건식에 값 만들기

```
StringBuffer query = new StringBuffer();

query.append("SELECT            ");
query.append("      id          ");
query.append("    ,  name       ");
query.append("    ,  age        ");
query.append("FROM              ");
query.append("    test          ");
query.append("WHERE 1=1         ");
query.append("  AND id = 'a001' ");
query.append("ORDER BY          ");
query.append("    1             ");


ps = conn.prepareStatement(query.toString());
rs = ps.executeQuery();
```

=

```
StringBuffer query = new StringBuffer();

query.append("SELECT            ");
query.append("      id          ");
query.append("    ,  name       ");
query.append("    ,  age        ");
query.append("FROM              ");
query.append("    test          ");
query.append("WHERE 1=1         ");
query.append("  AND id = ?      ");
query.append("ORDER BY          ");
query.append("    ?             ");


ps = conn.prepareStatement(query.toString());
// PrepareStatement의 setString은 쿼리문 내의 물음표(?)를 1부터 순서대로 채운다.
ps.setString(1, "a001");
ps.setInt(2, 1);
rs = ps.executeQuery();
```

JDBC

◈ Connection Factory

데이터베이스와의 연결 객체인 Connection 객체를 제공하는 클래스

Connection Factory가 있다면 매번 드라이버를 로딩 하거나, conn 객체를 만들 필요가 없어진다.

◈ Connection Pool

미리 Connection 객체를 일정 수만큼 생성시킨 후 쿼리문이 실행될 때마다 보유 중인 Connection 객체를 빌려주고, 쿼리문 실행이 종료되면 다시 Connection 객체를 반환 받으며 Connection 객체를 관리한다.

미리 연결된 상태의 Connection 객체들을 확보한 상태로 쿼리문이 실행되기 때문에 매번 쿼리문이 실행될 때 새로 연결되는(Connection 객체를 생성할) 시간을 절약할 수 있다.

쿼리문이 동시다발적으로 실행되는 경우에도 보유중인 Connection의 수만큼 이를 처리할 수 있다.

JDBC

◈ DAO (Data Access Object)

데이터베이스와 연계하여 처리할 메소드들(쿼리문)을 모아 둔 클래스

쿼리문 또한 StringBuffer를 이용하여 한줄 한줄 추가해서 PreparedStatement 에 적용한다.

◈ DTO (Data Transfer Object)

데이터를 하나의 객체로 관리할 목적으로 만들어 둔 클래스

SELECT 문으로 실행한 결과에 대해 한 줄(행)에 해당하는 데이터와 속성(컬럼)이 일치해야 한다.

JDBC

◈ Service

DAO 클래스에 정의되어 있는 메소드(쿼리문)를 실제 실행하는 메소드들을 보유하고 있는 클래스

```
// DB를 사용하는 각 기능들을 분리함으로써, 메인 코드가 한결 깔끔해진다.
JdbcService service = JdbcService.getInstance();

service.deleteAll();

service.insertTest(new TestVO(UtilClass.makeUniqueId(), "융가리", 16));
service.insertTest(new TestVO(UtilClass.makeUniqueId(), "강건마", 14));
service.insertTest(new TestVO(UtilClass.makeUniqueId(), "마영웅", 15));
service.insertTest(new TestVO(UtilClass.makeUniqueId(), "표독수", 16));
service.insertTest(new TestVO(UtilClass.makeUniqueId(), "지대호", 17));

ArrayList<TestVO> testList = service.selectTestList();

for(TestVO test : testList) {
    System.out.println(test);
}
```