

## 목차

### 1장. JDBC

#### 1. JDBC

- 사용전 고려사항
- JDBC 실행순서
- JDBC 실습
  - : 레코드 검색
  - : 레코드 저장
  - : 레코드 수정
  - : 레코드 삭제

#### 2. JDBC 고급기법

- DAO 및 DTO 패턴
- JDBC 트랜잭션 처리

# 1장. JDBC

## [학습목표]

- JDBC 의 개요에 관하여 학습한다.
- DAO (Data Access Object) 패턴에 관하여 학습한다.
- DTO (Data Transfer Object) 패턴에 관하여 학습한다.
- Connection Pool 기능에 관하여 학습한다.

---

01. JDBC

02. DAO 및 DTO 패턴

03. Connection Pool 기능

## 1. JDBC ( Java DataBase Connectivity )

JDBC는 웹 환경이건 아니건 어떤 환경에서건 자바언어를 사용하는 경우에 DBMS 종류에 상관 없이 데이터베이스에 접근할 수 있는 독립적인 프로그래밍 API이다.

### 1.1 JDBC 실행전 고려사항

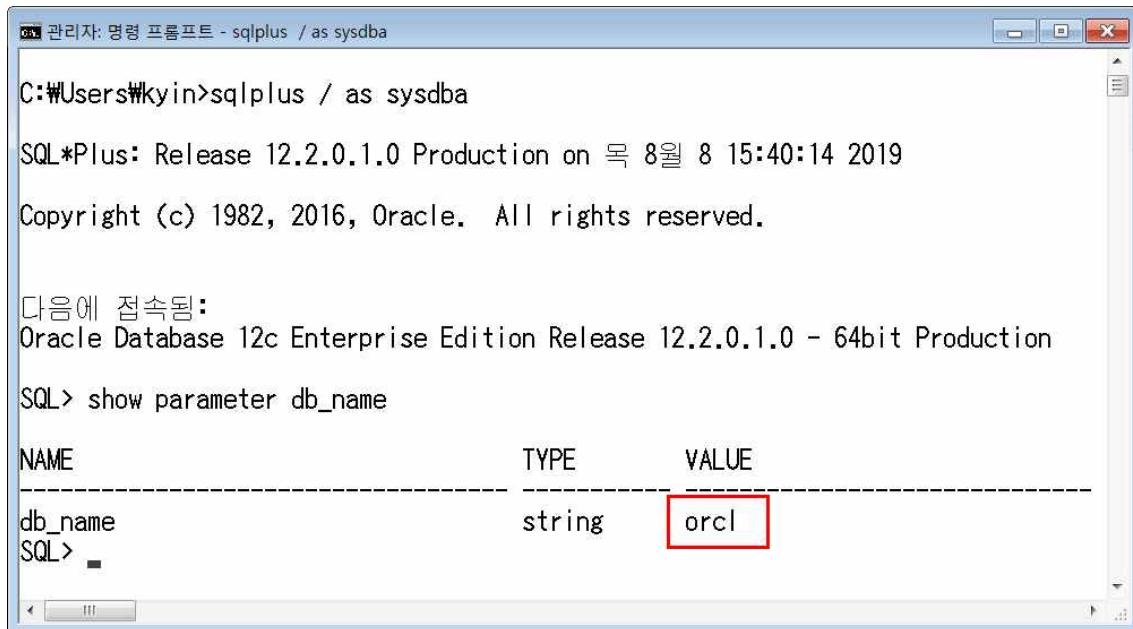
다음은 JDBC를 사용하기 위해서 반드시 알고 있어야 될 사항이다.

- 데이터베이스 정보 확인

데이터베이스가 설치된 IP정보 및 설정된 데이터베이스명(서비스명)을 반드시 알아야 된다.

다음은 DBA 로 접속하여 설정된 데이터베이스명을 알 수 있는 방법이다.

```
SQL> show parameter db_name;
```



- 오라클 데이터베이스의 2가지 서비스 실행 여부 확인

[제어판]-[관리도구]-[서비스]에서 다음 2개의 서비스가 실행중인지를 확인한다.

✓ OracleOraDB12Home1TNSListener

✓ OracleServiceORCL

Office Source Engine	업데...	수동	Local System
Offline Files	오프...	시작... 자동	Local System
OracleJobSchedulerORCL		사용 안 함	NT SERVICE\OracleJ...
OracleOraDB12Home1MTSRecoveryService	시작...	자동	NT SERVICE\OracleO...
OracleOraDB12Home1TNSListener	시작...	자동	NT SERVICE\OracleO...
OracleServiceORCL	시작...	자동	NT SERVICE\OracleS...
OracleVssWriterORCL	시작...	자동	NT SERVICE\OracleV...
Parental Controls	이 ...	수동	Local Service
Peer Name Resolution Protocol	서버...	수동	Local Service

- 오라클 드라이버를 클래스 패스에 설정

자바 어플리케이션과 오라클 데이터베이스가 연동하기 위해서는 오라클에서 제공하는 자바 클래스 파일이 필요하다.

오라클에서 제공한 압축파일(jar)을 ‘오라클 드라이버(oracle driver)’라고 하며, 오라클 드라이버는 오라클 데이터베이스를 설치하면 운영체제에 자동으로 설치된다. 이 드라이버를 명시적으로 클래스 패스에 설정해도 되지만 Maven 프로젝트에서는 pom.xml 파일에 의존성을 설정하는 방법으로 사용한다.

Oracle 11g Express Edition 버전인 경우에는 다음과 같이 의존성을 설정한다.

또한 Oracle 11g 버전인 경우에는 서비스명이 xe 로 고정되어 있다.

```
<dependencies>
  <dependency>
    <groupId>com.jslsolucoes</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.2.0.1.0</version>
  </dependency>
</dependencies>
```

JDBC 연결을 위한 환경설정이 모두 끝났으면 자바코드로 데이터베이스를 연동할 수 있다.

대부분의 데이터베이스 연동 코드가 일정한 패턴으로 구현하기 때문에 설명하는 순서대로 코드 작업을 하면 무리없이 데이터베이스 연동을 할 수 있다.

## 1.2 JDBC 실행 순서 정리

다음은 JDBC를 구현하는 기본적인 프로그램을 개발하는 순서이다.

1. 오라클 데이터베이스 연동을 위한 4가지 정보를 문자열에 저장한다. ( SCOTT, TIGER 는 대문자로 지정한다.)

```
String driver = "oracle.jdbc.driver.OracleDriver";
String url = "jdbc:oracle:thin:@localhost:1521:orcl";
String userid = "SCOTT";
String passwd = "TIGER";
```

드라이버(압축파일)내에 핵심이 되는 클래스 파일이 있다.

이 클래스 파일은 oracle.jdbc.driver 패키지내에 있는 OracleDriver 클래스이다. 이 값을 driver 변수에 저장한다. 오라클의 위치 및 포트번호, 데이터베이스명의 정보를 url 변수에 저장한다. 오라클의 기본 포트번호는 1521번이고 데이터베이스 위치는 현재 로컬 컴퓨터에 설치했기 때문에 localhost라고 지정한다. 만약 오라클 데이터베이스가 원격에 있다면 원격 IP 번호를 지정한다. 서비스명은 기본적으로 orcl 이다. (Oracle 11g 버전인 경우에는 서비스명이 xe로 고정되어

있음.) 마지막으로 접속하고자 하는 계정명과 비밀번호를 userid 변수와 passwd 변수에 저장한다.

## 2. 드라이버 로딩

앞에서 살펴봤던 OracleDriver 클래스를 메모리에 올려야 되는데 다음과 같은 방법으로 클래스 파일을 메모리에 로딩한다.

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

## 3. Connection 맺기

자바코드와 오라클 데이터베이스를 연결하는 것을 의미한다.

연결은 java.sql 패키지의 Connection을 사용한다.

다음과 같이 DriverManager 클래스를 이용해서 Connection을 구한다.

```
Connection con = DriverManager.getConnection( url, userid , passwd );
```

getConnection 메서드 인자에 앞에서 저장했던 url 값과 userid, passwd 값을 지정하면 이 정보를 이용해서 데이터베이스에 연결된다.

## 4. SQL 문 작성

자바에서 데이터베이스에 요청할 SQL문은 문자열로 저장한다. SQL\*PLUS 툴에서의 요청과 같다. 주의할 점은 ;(세미콜론)은 입력하지 않는다.

dept 테이블의 데이터를 검색하기 위한 SQL 문은 다음과 같다.

```
String query = "SELECT deptno,dname,loc FROM dept";
```

다음은 dept 테이블의 데이터 중에서 deptno 가 40인 레코드를 삭제하는 SQL문이다.

```
String query = "DELETE FROM dept WHERE deptno = 40";
```

## 5. PreparedStatement 생성

요청할 SQL문을 전송할 때 사용되는 API이다.

다음과 같이 Connection을 이용해서 PreparedStatement를 생성한다.

```
PreparedStatement pstmt = con.prepareStatement( query );
```

Connection의 prepareStatement() 메서드를 이용해서 PreparedStatement를 생성할 수 있다.

## 6. SQL 문 전송 및 결과값 얻기

SQL문 전송은 PreparedStatement 객체를 이용해서 데이터베이스에 전송한다.

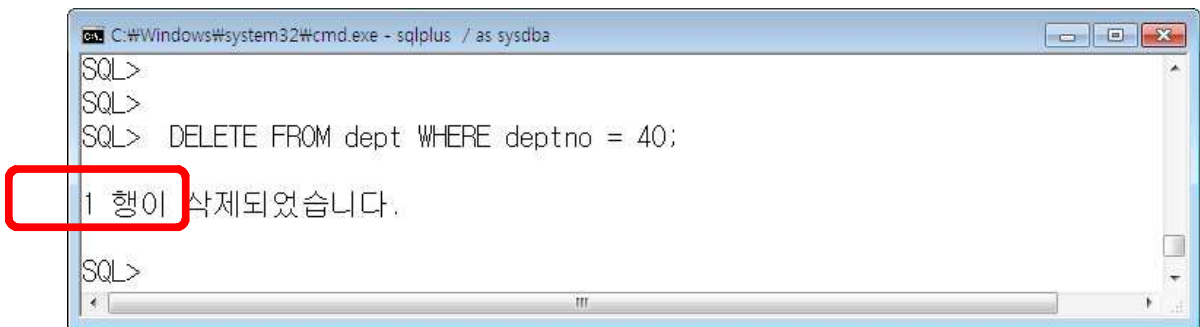
요청하는 SQL문에 따라서 다음과 같이 2 가지 메서드를 사용할 수 있다.

- DML 요청 ( INSERT, DELETE, UPDATE )

DML 요청은 executeUpdate 메서드를 사용한다.

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

SQL\*PLUS 툴에서 DML을 요청하면 항상 적용된 레코드의 개수가 다음과 같이 출력되었다.



따라서 executeUpdate 메서드의 리턴되는 정수값은 변경된 레코드의 개수이다. DELETE 문을 요청했을 때 리턴되는 정수값이 10이라면 삭제된 레코드 개수가 10개라는 것이다.

반환된 값이 저장되는 n 변수값을 이용해서 적용된 레코드의 개수를 파악할 수 있다.

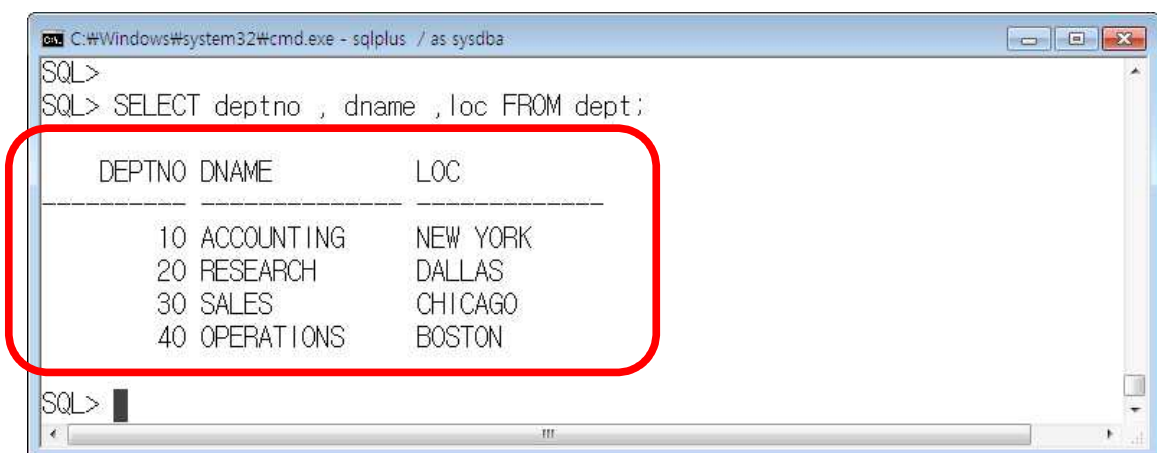
또한 JDBC의 DML처리는 기본적으로 자동으로 commit이 발생된다. 따라서 JDBC를 수행한후 ROLLBACK 처리가 불가능하다.

- SELECT 요청

SELECT 요청은 executeQuery 메서드를 사용한다.

```
ResultSet rs = pstmt.executeQuery( );
```

SQL\*PLUS 툴에서 SELECT 문을 실행하면 항상 테이블 형태로 다음과 같이 결과가 출력되었다.



이 테이블 결과를 자바의 객체로 표현한 것이 ResultSet 객체이다.

ResultSet 객체는 포인터를 이용해서 원하는 데이터를 얻을 수 있다.

먼저 포인터를 이용해서 레코드를 선택하고 나중에 포인터가 가리키는 레코드의 컬럼을 지정해

서 데이터를 얻는다. 레코드를 선택하는 메서드는 next() 메서드를 사용하고 컬럼은 데이터형에 따라서 getInt("컬럼명|별칭|위치값"), getString("컬럼명|별칭|위치값") 메서드를 사용할 수 있다.

다음은 결과값인 ResultSet에서 데이터를 얻는 방법이다.

```
while( rs.next()){
    int deptno = rs.getInt("deptno");
    String dname = rs.getString("dname");
    String loc = rs.getString("loc");
    System.out.println( deptno + " " + dname + " " +loc );
}
```

결과값의 레코드가 여러 개 있을 수 있기 때문에 while 문을 사용하며 next() 메서드는 레코드가 있는 경우에는 true 값을 없으면 false 값을 리턴한다.

deptno 컬럼은 NUMBER 형이기 때문에 getInt 메서드를 사용해서 데이터를 얻는다. dname 컬럼과 loc 컬럼은 VARCHAR2 형이기 때문에 getString 메서드를 사용한다. 이때 메서드의 인자값으로 컬럼명을 사용하거나 SELECT 문에서 사용한 컬럼의 위치값을 사용할 수도 있다.(위치값은 1부터 시작) 하지만 alias를 지정한 경우에는 getXXX메서드에 alias 또는 위치값만 사용해야 되고 컬럼명 사용은 불가능하다.

## 7. 자원 반납

파일 및 데이터베이스는 자바에서 사용하는 외부 자원이기 때문에 반드시 사용한 다음에는 자원을 해제시켜야 된다. 데이터베이스에서 사용한 자원은 ResultSet, PreparedStatement, Connection 등이다. 이 자원을 해제 시킬 때에는 사용한 역순으로 해제 시킨다. 데이터베이스를 사용시 예외가 발생되거나 발생되지 않거나 항상 자원을 해제시켜야 되기 때문에 일반적으로 finally 문에서 자원 반납코드를 구현한다.

다음과 같이 자원을 해제한다.

```
if( rs !=null) rs.close(); // ResultSet을 사용한 경우
if( stmt !=null) stmt.close();
if( con !=null) con.close());
```

앞에서 설명했던 각 단계를 잘 이해하고 숙지한다면 JDBC를 사용하는 프로그램 개발에 큰 어려움이 없을 것으로 확신한다.

## 1.3 JDBC 실습

### [실습하기 13-1] 레코드 검색하기

다음은 dept 테이블에 저장된 레코드를 검색하는 자바 프로그램 예제이다.

Ex13\_1.java

```
1 package com.test;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7
8 public class Ex13_1 {
9
10     public static void main(String[] args) {
11
12         String driver = "oracle.jdbc.driver.OracleDriver";
13         String url = "jdbc:oracle:thin:@localhost:1521:orcl";
14         String userid = "SCOTT";
15         String passwd = "TIGER";
16
17         Connection con = null;
18         PreparedStatement pstmt = null;
19         ResultSet rs = null;
20         try {
21             Class.forName(driver);
22             con = DriverManager.getConnection(url, userid, passwd);
23             String query = "SELECT deptno,dname,loc 부서위치 FROM dept";
24
25             pstmt = con.prepareStatement(query);
26             rs = pstmt.executeQuery();
27             while (rs.next()) {
28                 int deptno = rs.getInt("deptno");
29                 String dname = rs.getString("dname"); // getString(2)
30                 String loc = rs.getString("부서위치"); // getString(3)
31                 System.out.println(deptno + " " + dname + " " + loc);
32             }
33         } catch (Exception e) {
34             e.printStackTrace();
35         } finally {
36             try {
37                 if (rs != null)rs.close();
38                 if (pstmt != null)pstmt.close();
39                 if (con != null)con.close();
40             } catch (SQLException e) {
41                 e.printStackTrace();
42             }
43         }
44     }
```

#### <코드설명>

12행~15행: 데이터베이스 접속을 위한 4가지 정보를 String 변수에 저장한다. SCOTT과 TIGER는 대문자로 설정한다.

21행: 명시된 드라이버 클래스를 메모리에 로딩한다.

22행: DriverManager 클래스의 getConnection 메서드를 이용해서 Connection 객체를 얻는다.

23행: 요청할 SQL문을 String 에 저장한다.

24행: SQL문 전송에 필요한 PreparedStatement 객체를 prepareStatement(sql)메서드를 이용



해서 얻는다.

26행: SELECT 문을 요청하기 때문에 executeQuery 메서드를 사용하며 결과는 ResultSet 으로 받는다.

27행~32행: ResultSet의 next() 메서드를 이용해서 행을 선택하고 getInt(컬럼) 또는 getString(컬럼) 메서드를 이용해서 테이블의 컬럼값을 얻는다. 컬럼명 대신에 getString(위치값)을 사용할 수도 있다. 하지만 별칭으로 select한 경우에는 컬럼명은 사용할 수 없다.

따라서 마지막 loc 컬럼값을 얻기 위하여 rs.getString("loc")로 사용하면 예외가 발생된다.

35행~43행: 사용한 자원을 finally문을 이용해서 반납한다. 자원반납은 사용했던 객체의 역순으로 하며 모두 공통적으로 close() 메서드를 사용한다.

</코드설명>

출력결과



```
<terminated> Ex13_1 [Java Application] C:\Program Files\Java\jre:
10 ACCOUNTING NEW YORK
20 RESEARCH DALLAS
30 SALES CHICAGO
40 OPERATIONS BOSTON
```

## [실습하기 13-2] 레코드 저장하기

다음은 dept테이블에 새로운 레코드를 저장하는 자바 프로그램 예제이다.

새로운 데이터로 deptno 값은 50, dname 값은 “개발”, loc 값은 “서울” 로 저장한다. 주의할 점은 deptno 컬럼이 기본키(primary key)이기 때문에 중복 저장이 안된다. 따라서 저장된 레코드값을 먼저 확인하고 실습한다.

```

1 package com.test;
2 import java.sql.Connection;
7
8 public class Ex13_2 {
9     public static void main(String[] args) {
10
11         String driver = "oracle.jdbc.driver.OracleDriver";
12         String url = "jdbc:oracle:thin:@localhost:1521:orcl";
13         String userid = "SCOTT";
14         String passwd = "TIGER";
15
16         Connection con = null;
17         PreparedStatement pstmt = null;
18
19         try{
20             Class.forName(driver);
21             con = DriverManager.getConnection(url, userid, passwd);
22
23             String sql = "INSERT INTO dept (deptno,dname,loc) " +
24                 " VALUES ( ?, ?, ? )";
25             pstmt = con.prepareStatement(sql);
26             pstmt.setInt(1, 50 );
27             pstmt.setString(2, "개발");
28             pstmt.setString(3, "서울");
29
30             int n = pstmt.executeUpdate();
31             System.out.println( n + " 개의 레코드가 저장");
32         }catch(Exception e){
33             e.printStackTrace();
34         }finally{
35             try{
36                 if(pstmt != null ) pstmt.close();
37                 if(con != null ) con.close();
38             }catch(SQLException e){
39                 e.printStackTrace();
40             }
41         }
42     }
43 }
44 }

```

#### 〈코드설명〉

11행~14행: 데이터베이스 접속을 위한 4가지 정보를 String 변수에 저장한다.

20행: 명시된 드라이버 클래스를 메모리에 로딩한다.

21행: DriverManager 클래스의 getConnection 메서드를 이용해서 Connection 객체를 얻는다.

23행: 요청할 SQL문을 String 에 저장한다. 이때 저장할 데이터 대신에 ? 기호를 사용한다. 나중에 PreparedStatement 객체의 set 메서드를 사용하여 값을 동적으로 설정한다.

24행~27행: SQL문 전송에 필요한 PreparedStatement 객체를 prepareStatement(sql)메서드

를 이용해서 얻고 ? 의 순서( 1부터 시작)를 사용하여 저장값을 set메서드로 설정한다. set 메서드는 컬럼 데이터형에 따라서 NUMBER 컬럼이면 setInt( 순서, 값 )로 설정하고 VARCHAR2 컬럼이면 setString( 순서, 값)을 사용하면 된다.

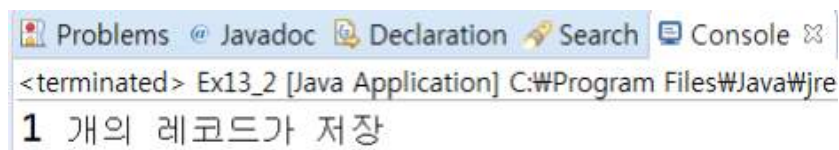
30행: INSERT 문을 요청하기 때문에 executeUpdate 메서드를 사용하며 결과는 int형인 정수값을 반환한다.

31행: 반환된 정수값을 출력하면 SQL문이 실행후 적용된 레코드의 개수를 알 수 있다.

36행~37행: 사용한 자원을 finally문을 이용해서 반납한다. 자원반납은 사용했던 객체의 역순으로 하며 모두 공통적으로 close() 메서드를 사용한다.

</코드설명>

출력결과



실행후에 Ex14\_1 클래스파일을 다시 실행해보면 저장시킨 레코드까지 포함하여 출력됨을 확인할 수 있다. 이 실습은 여러 번 실행하면 deptno 가 PRIMARY KEY로 지정되었기 때문에 예외가 발생된다. 여러 번 실행하려면 deptno 값인 50을 중복되지 않은 값으로 변경후에 실행한다.

### [실습하기 13-3] 레코드 수정하기

다음은 dept테이블에 저장된 레코드를 수정하는 실습이다.

수정할 데이터로 deptno 값이 50인 레코드를 dname을 '인사'로 변경하고 loc을 '부산'으로 변경한다.

```

1 package com.test;
2 import java.sql.Connection;
3
4 public class Ex13_3 {
5     public static void main(String[] args) {
6
7         String driver = "oracle.jdbc.driver.OracleDriver";
8         String url = "jdbc:oracle:thin:@localhost:1521:orcl";
9         String userid = "SCOTT";
10        String passwd = "TIGER";
11
12        Connection con = null;
13        PreparedStatement pstmt = null;
14
15        try {
16            Class.forName(driver);
17            con = DriverManager.getConnection(url, userid, passwd);
18
19            String sql = "UPDATE dept SET dname = ? , loc = ? " + " WHERE deptno = ?";
20            pstmt = con.prepareStatement(sql);
21            pstmt.setString(1, "인사");
22            pstmt.setString(2, "부산");
23            pstmt.setInt(3, 50);
24            int n = pstmt.executeUpdate();
25            System.out.println(n + " 개의 레코드가 수정");
26        } catch (Exception e) {
27            e.printStackTrace();
28        } finally {
29            try {
30                if (pstmt != null) pstmt.close();
31                if (con != null) con.close();
32            } catch (SQLException e) {
33
34            }
35        }
36    }
37 }

```

#### <코드설명>

11행~14행: 데이터베이스 접속을 위한 4가지 정보를 String 변수에 저장한다.

20행: 명시된 드라이버 클래스를 메모리에 로딩한다.

21행: DriverManager 클래스의 getConnection 메서드를 이용해서 Connection 객체를 얻는다.

23행: 요청할 SQL문을 String 에 저장한다. 이전 실습과 마찬가지로 저장할 데이터 대신에 ? 기호를 사용한다. 나중에 PreparedStatement 객체의 set 메서드를 사용하여 값을 동적으로 설정한다.

24행~27행: SQL문 전송에 필요한 PreparedStatement 객체를 prepareStatement(sql) 메서드를 이용해서 얻고 ? 대신에 설정할 값을 위치값( 1부터 시작)를 사용하여 set 메서드로 설정한다.

28행: UPDATE 문을 요청하기 때문에 executeUpdate 메서드를 사용하며 결과는 int형 으로 저장한다.

29행: 반환된 정수값을 출력하면 SQL문이 실행후 적용된 레코드의 개수를 알 수 있다.

32행~39행: 사용한 자원을 finally문을 이용해서 반납한다. 자원반납은 사용했던 객체의 역순으로 하며 모두 공통적으로 close() 메서드를 사용한다.

#### </코드설명>

#### 출력결과

```

Problems Javadoc Declaration Search Console
<terminated> Ex13_3 [Java Application] C:\Program Files\Java\jre1.
1 개의 레코드가 수정

```

실행후에 Ex14\_1 클래스파일을 다시 실행해보면 deptno가 50인 레코드가 수정되어 있는 것을 확인할 수 있다.

## [실습하기 13-4] 레코드 삭제하기

다음은 dept 테이블에 저장된 레코드를 삭제하는 자바 프로그램 예제이다.

삭제할 데이터로 deptno 값이 50인 레코드를 삭제한다. 만약 50이 아닌 deptno 값이 10,20,30 중에서 하나를 삭제할 때에는 emp 테이블에서 참조하기 때문에 에러가 발생된다.

```
Ex13_4.java
1 package com.test;
2 import java.sql.Connection;
7
8 public class Ex13_4 {
9     public static void main(String[] args) {
10
11         String driver = "oracle.jdbc.driver.OracleDriver";
12         String url = "jdbc:oracle:thin:@localhost:1521:orcl";
13         String userid = "SCOTT";
14         String passwd = "TIGER";
15
16         Connection con = null;
17         PreparedStatement pstmt = null;
18
19         try {
20             Class.forName(driver);
21             con = DriverManager.getConnection(url, userid, passwd);
22
23             String sql = "DELETE FROM dept WHERE deptno = ?";
24             pstmt = con.prepareStatement(sql);
25             pstmt.setInt(1, 50);
26             int n = pstmt.executeUpdate();
27             System.out.println(n + " 개의 레코드가 삭제");
28         } catch (Exception e) {
29             e.printStackTrace();
30         } finally {
31             try {
32                 if (pstmt != null) pstmt.close();
33                 if (con != null) con.close();
34             } catch (SQLException e) {
35                 e.printStackTrace();
36             }
37         }
38     }
39 }
```

### <코드설명>

11행~14행: 데이터베이스 접속을 위한 4가지 정보를 String 변수에 저장한다.

20행: 명시된 드라이버 클래스를 메모리에 로딩한다.

21행: DriverManager 클래스의 getConnection 메서드를 이용해서 Connection 객체를 얻는다.

23행: 요청할 SQL문을 String 에 저장한다. 이전 실습과 마찬가지로 저장할 데이터 대신에 ? 기호를 사용한다. 나중에 PreparedStatement 객체의 set 메서드를 사용하여 값을 동적으로 설정한다.

24행~25행: SQL문 전송에 필요한 PreparedStatement 객체를 prepareStatement(sql) 메서드를 이용해서 얻고 ? 대신에 설정할 값을 위치값( 1부터 시작)를 사용하여 set 메서드로 설정한다.

27행: DELETE 문을 요청하기 때문에 executeUpdate 메서드를 사용하며 결과는 int형 으로 저

장한다.

28행: 반환된 정수값을 출력하면 SQL문이 실행후 적용된 레코드의 개수를 알수 있다.

32행~39행: 사용한 자원을 finally문을 이용해서 반납한다. 자원반납은 사용했던 객체의 역순으로 하며 모두 공통적으로 close() 메서드를 사용한다.

</코드설명>

출력결과



## 2. JDBC 고급기법

### 2.1 DAO 패턴 및 DTO 패턴

데이터베이스를 연동하는 프로그램을 개발할 때 반드시 사용되는 2 가지 개발 패턴이 있다.

#### (1) DAO (Data Access Object) 패턴

일반적으로 어플리케이션 개발은 GUI화면을 가지며 화면에 보여줄 수 있는 데이터 관리를 위해서 데이터베이스를 사용해서 개발하게 된다.

웹 브라우저에서 보여지는 것처럼 GUI 화면을 구성하는 코드를 Presentation Logic 이라고 하며 GUI화면에 데이터를 보여주기 위해서 데이터베이스를 검색하는 코드 및 GUI화면에서 새로 발생된 데이터 (예를 들면 회원가입)를 데이터베이스에 저장하는 코드와 같은 실제적인 작업을 처리하는 코드를 business logic 이라고 한다. presentation logic 과 business logic을 하나의 클래스로 모두 구현 할 수도 있고 여러 클래스로 모듈화 시켜서 구현할 수도 있다. 하나의 클래스로 구현하면 유지보수가 어려워지기 때문에 바람직하지 않다. 예를들어 2개의 로직이 하나의 클래스 파일에 저장되면 화면을 변경하거나 또는 business logic이 변경될 때 복잡한 코드로 인해서 유지보수가 어려워진다. 따라서 모듈화 시켜 개발하게 되며 모듈화한 클래스들중에서 데이터베이스 또는 파일 처리하는 코드만을 관리하는 클래스를 작성할 수 있는데 이 클래스 파일을 DAO 클래스라고 한다. 이렇게 DAO 클래스를 사용해서 개발하는 것이 보편화 되었기 때문에 DAO 패턴이라고 부른다.

일반적으로 DAO 클래스는 테이블 당 한 개씩 생성해서 사용한다. DAO 클래스 안에는 특정 테이블에서 수행할 작업을 메서드로 정의해서 구현하며 트랜잭션 처리와 같은 특별한 작업을 구현하기 위하여 일반적으로 Service 라고 부르는 클래스까지 구현한다.

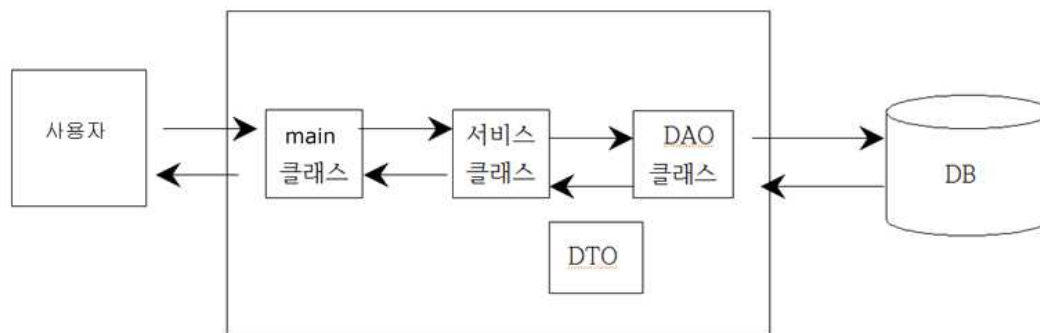
#### (2) DTO (Data Transfer Object) 패턴

presentation logic과 business logic을 여러 클래스로 분리해서 작업은 하지만 서로 간에 긴밀한 관계가 유지되면서 작업이 이루어진다. presentation logic 에서 보여줄 데이터를 얻기 위해



서 business logic에게 요청을 하면 business logic은 필요한 데이터를 데이터베이스에서 검색해서 presentation logic에게 반환하는 작업등을 하게 된다. 이렇게 데이터를 다른 logic에게 전송 및 반환할 때 효율적으로 데이터를 전송 및 사용할 수 있게 클래스를 작성할 수 있는데 이 클래스를 DTO 클래스라고 한다. 이름 그대로 데이터를 전송할 때 사용되는 클래스이다. DTO 클래스를 사용하면 데이터를 전송할 때와 전송된 데이터를 얻어서 사용할 때 효율적으로 사용할 수 있는 장점이 있다. 일반적으로 도메인객체(Domain Object), VO (Value Object)라고도 한다.

실습하기 위한 전체적인 아키텍처는 다음과 같다.



### [실습하기 13-5] DAO 및 DTO 패턴 사용

다음은 dept 테이블에 저장된 레코드를 조회할 수 있는 기능을 DAO 및 DTO 패턴을 적용하여 구현한 실습이다. 매우 중요한 아키텍처이기 때문에 반드시 숙지하도록 한다. 패키지는 com.test2 로 지정한다.

먼저 다음과 같은 정보로 클래스를 작성한다.

- Main 클래스명: Ex13\_5.java
- DAO 클래스명: DeptDAO.java ( 패키지명: com.dao)
- DTO 클래스명: DeptDTO.java ( 패키지명: com.dto)
- Service 클래스명: DeptService.java ( 패키지명: com.service)

다음은 dept 테이블의 레코드 저장 및 다른 클래스로 전달할 목적으로 만든 DeptDTO 클래스이다. DeptDTO 클래스의 변수명은 dept테이블의 컬럼명과 동일하게 지정하고 getter/setter 메서드와 생성자를 추가로 설정한다.

```

1 package com.test2;
2
3 public class DeptDTO {
4
5     private int deptno;
6     private String dname;
7     private String loc;
8
9     public int getDeptno() {
10         return deptno;
11     }
12     public void setDeptno(int deptno) {
13         this.deptno = deptno;
14     }
15     public String getDname() {
16         return dname;
17     }
18     public void setDname(String dname) {
19         this.dname = dname;
20     }
21 }

```

다음은 오라클 데이터베이스와 연결하기 위해 Connection을 얻는 작업을 하기 위한 DeptService 클래스를 작성한다. 이 클래스에서 Connection을 얻는 이유는 트랜잭션을 처리하기 위해서이다. 트랜잭션 처리는 2.2 JDBC 트랜잭션(Transaction,tx) 처리를 참조한다. 얻은 Connection을 실제 데이터베이스와 연동하는 DeptDAO 클래스에 인자로 전달하여 사용된다.

```

1 package com.test2;
2
3 import java.sql.Connection;
4
5 public class DeptService {
6
7     String driver = "oracle.jdbc.driver.OracleDriver";
8     String url = "jdbc:oracle:thin:@localhost:1521:xe";
9     String userid = "SCOTT";
10    String passwd = "TIGER";
11
12    public DeptService() {
13        try {
14            Class.forName(driver);
15        } catch (ClassNotFoundException e) {
16            e.printStackTrace();
17        }
18    }
19 }

```



```

23= public List<DeptDTO> select() {
24     List<DeptDTO> list = null;
25     Connection con = null;
26     try {
27         con = DriverManager.getConnection(url, userid, passwd);
28         DeptDAO dao = new DeptDAO();
29         list = dao.select(con);
30     } catch (SQLException e) {
31         e.printStackTrace();
32     } finally {
33         try {
34             if (con != null)
35                 con.close();
36         } catch (SQLException e) {
37             e.printStackTrace();
38         }
39     }
40     return list;
41 }

```

다음은 실제 데이터베이스와 연동하는 DeptDAO 클래스이다. dept 테이블의 하나의 레코드는 DeptDTO 에 저장하고 누적하기 위하여 ArrayList<DeptDTO>를 사용한다. 주의할 점은 Connection을 DAO 클래스에서 close() 하면 안되고 반드시 Service 클래스에서 close() 시켜야 된다.

```

1 DeptDAO.java
1 package com.test2;
2 import java.sql.Connection;
3
4 public class DeptDAO {
5
6     public List<DeptDTO> select(Connection con)
7     {
8         List<DeptDTO> list = new ArrayList<DeptDTO>();
9         PreparedStatement pstmt = null;
10        ResultSet rs = null;
11        try {
12            String query = "SELECT deptno,dname,loc 부서위치 FROM dept";
13            pstmt = con.prepareStatement(query);
14            rs = pstmt.executeQuery();
15            while (rs.next()) {
16                int deptno = rs.getInt("deptno");
17                String dname = rs.getString("dname"); // getString(2)
18                String loc = rs.getString("부서위치"); // getString(3)
19                list.add(new DeptDTO(deptno, dname, loc));
20            }
21        } catch (Exception e) {
22            e.printStackTrace();
23        } finally {
24            try {
25                if (rs != null)rs.close();
26                if (pstmt != null)pstmt.close();
27            } catch (SQLException e) {
28                e.printStackTrace();
29            }
30        }
31        return list;
32    }
33 }

```

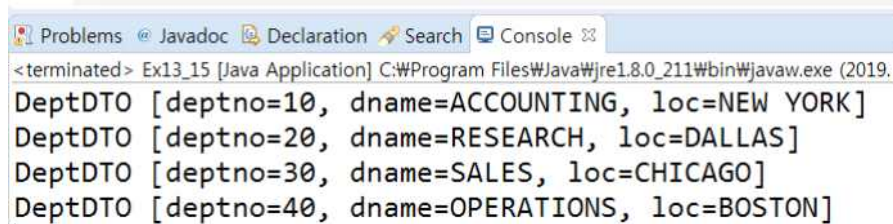
마지막으로 main 메서드에서 요청하고 출력한다.

```

1 package com.test2;
2
3 import java.util.List;
4
5 public class Ex13_5 {
6
7     public static void main(String[] args) {
8
9         DeptService service = new DeptService();
10
11         List<DeptDTO> list = service.select();
12         for (DeptDTO dto : list) {
13             System.out.println(dto);
14         }
15     }
16 }

```

### 실행결과



```

<terminated> Ex13_15 [Java Application] C:\Program Files\Java\jre1.8.0_211\bin\javaw.exe (2019.
DeptDTO [deptno=10, dname=ACCOUNTING, loc=NEW YORK]
DeptDTO [deptno=20, dname=RESEARCH, loc=DALLAS]
DeptDTO [deptno=30, dname=SALES, loc=CHICAGO]
DeptDTO [deptno=40, dname=OPERATIONS, loc=BOSTON]

```

다음은 삭제 기능이다. 주요 코드만 살펴보기로 한다.

DeptDAO 클래스에서 삭제기능을 구현하는 delete 메서드 코드이다.

```

public class DeptDAO {

    public void delete(Connection con , int deptno) {
        PreparedStatement pstmt = null;
        try {
            String query = "delete from dept where deptno = ?";
            pstmt = con.prepareStatement(query);
            pstmt.setInt(1, deptno);

            int n = pstmt.executeUpdate();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (pstmt != null)pstmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

```

다음은 DeptService 클래스에서 삭제기능을 구현하는 delete 메서드 코드이다.

```

    public void delete(int deptno) {
        Connection con = null;
        try {
            con = DriverManager.getConnection(url, userid, passwd);
            DeptDAO dao = new DeptDAO();
            dao.delete(con, deptno);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

다음은 main() 메서드에서 변경된 코드이다.

```

Ex13_5.java
1 package com.test2;
2
3 import java.util.List;
4
5 public class Ex13_5 {
6
7     public static void main(String[] args) {
8
9         DeptService service = new DeptService();
10
11         service.delete(50);
12
13         List<DeptDTO> list = service.select();
14         for (DeptDTO dto : list) {
15             System.out.println(dto);
16         }
17     }
18 }

```

삭제와 동일한 방법으로 저장 및 수정 기능도 구현할 수 있다.

## 2.2 JDBC 트랜잭션(Transaction,tx) 처리

앞에서 언급한대로 JDBC에서는 모든 DML 작업이 자동으로 auto commit으로 처리된다. 하지만 트랜잭션(transaction)으로 처리해야만 하는 작업에서는 명시적으로 auto commit을 비활성화 시킨 후에 작업해야 된다.

다음은 JDBC에서 트랜잭션 처리를 위한 가장 기본적인 코드로서, Service클래스에서 Connection API의 setAutoCommit(false) 메서드를 사용하여 비활성화 시키고 commit() 또는 rollback() 메서드를 사용하여 트랜잭션 처리를 할 수 있다. dao.insert()와 dao.delete() 메서드가 모두 성공하면 commit()하고 만약 하나라도 예외가 발생되면 rollback()으로 처리한다.

```

public void transactionMethod(){
    Connection con = null;
    try {
        con = DriverManager.getConnection(url, userid, passwd);
        con.setAutoCommit(false);
        dao.insert(new EmpDTO(1200, "유관순", 3000, 10));
        dao.delete(1000);
        con.commit();
    } catch (Exception e) {
        con.rollback();
    } finally {
        try {
            if(con!=null)con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

TX 비회상화

TX 커밋

예외발생시 TX 롤백