JSP (Java Sever Page)

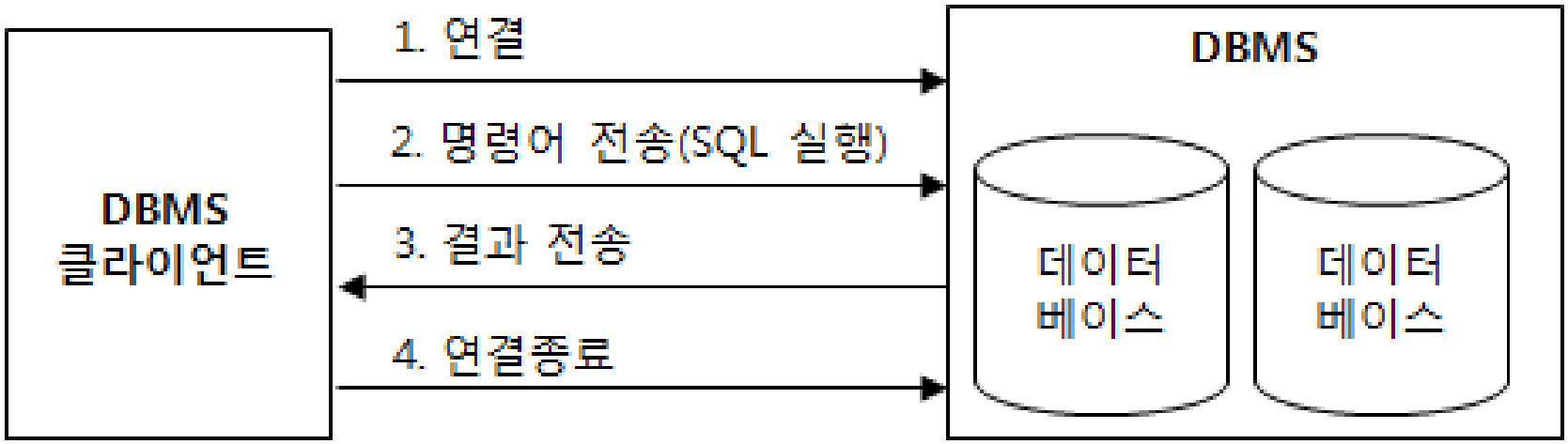
- 데이터 프로그래밍의 기초

CONTENTS

- JDBC 프로그래밍
- 커넥션 풀

데이터베이스 프로그래밍

- 일반적 순서



- 필수 요소
  - DBMS
  - 데이터베이스 :
  - DBMS 클라이언트

■ 테이블 생성 쿼리

- **구문**

```
create table MEMBER (
    MEMBERID        VARCHAR(10) NOT NULL PRIMARY KEY,
    PASSWORD        VARCHAR(10) NOT NULL,
    NAME            VARCHAR(20) NOT NULL,
    EMAIL           VARCHAR(80)
)
```

■ 데이터 삽입 쿼리

- **구문**

```
insert into [테이블이름] ([칼럼1], [칼럼2], .., [칼럼n])
values ([값1], [값2], .., [값n])
```

  - **새로운 레코드를 삽입**
  - **칼럼에 대해 값을 설정**
  - **칼럼 목록을 지정하지 않은 경우 values 에 모든 칼럼에 대한 값을 지정**

- **예**

```
insert into MEMBER (MEMBERID, PASSWORD, NAME, EMAIL)
values (‘cool', ‘1234', ‘유영진‘, ‘cool@gmail.com’);
```

■ 데이터 조회 쿼리

- 구문
  - select [칼럼1], [칼럼2], ..., [칼럼n] from [테이블이름]
- 예
  - select MEMBERID, NAME from MEMBER

- where 절
  - 조건에 맞는 레코드 검색
    - select * from MEMBER where NAME = '유영진'
  - and와 or로 다양한 조건 지정 가능
    - where NAME = ' 유영진' and EMAIL = 'hot@gmail.com'
  - 주요 비교문
    - =, <>, >=, >, <=, <
    - is null, is not null, like

■ 데이터 조회 쿼리 - 정렬, 집합

- order by를 이용한 조회 정렬 순서 지정
  - select .. from [테이블이름] where [조건절]
    order by [칼럼1] asc, [칼럼2] desc, ...

- 집합 관련 함수
  - select max(SALARY), min(SALARY), sum(SALARY) from ...
    - max() - 최대값
    - min() - 최소값
    - sum() - 합

■ 데이터 수정/삭제 쿼리

- 수정 쿼리
  - update [테이블이름] set [칼럼1]=[값1], [칼럼2]=[값2], .. where [조건절]
  - where절을 사용하지 않을 경우 모든 레코드가 수정

- 삭제 쿼리
  - delete from [테이블이름] where [조건절]
  - where 절을 사용하지 않을 경우 모든 레코드가 삭제

조인

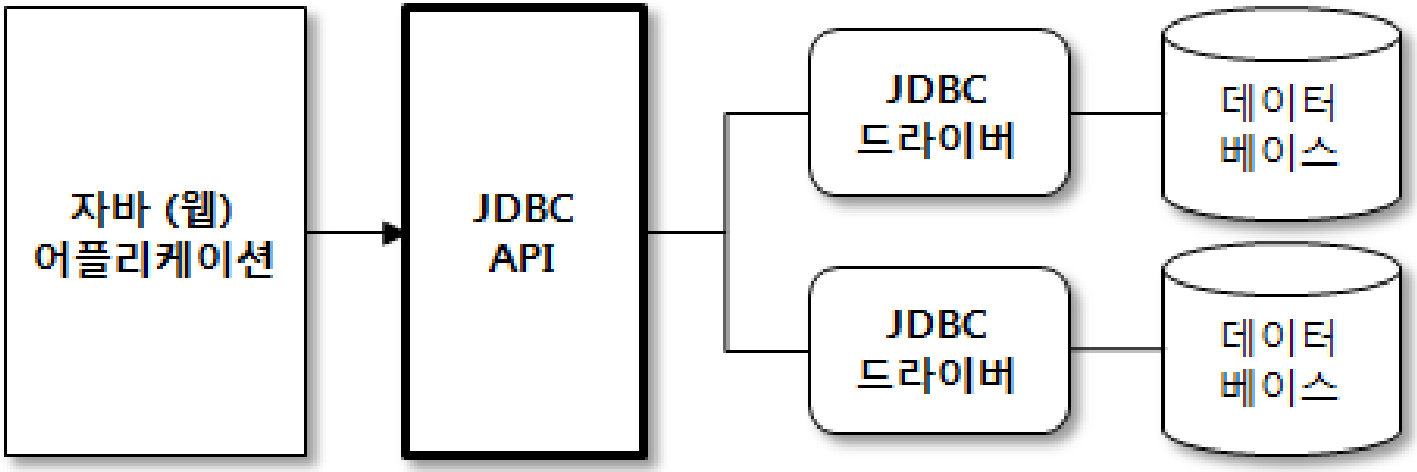
- 두 개 이상의 테이블로부터 관련 있는 데이터를 읽어올 때 사용
- 기본 구문

```
select A. 칼럼1, A. 칼럼2, B. 칼럼3, B. 칼럼4
from [테이블1] as A, [테이블2] as B
where A.[ 칼럼x] = B.[ 칼럼y]
```

- 조인 사용에 따른 장단점
  - 다수의 테이블을 한번에 조회할 때 유용
  - 조인이 복잡해 질수록 조회 속도가 느려질 가능성 높음
    - 복잡한 인덱스 설계 등을 필요로 함

JDBC

- Java Database Connectivity
- 자바에서 DB 프로그래밍을 하기 위해 사용되는 API
- JDBC API 사용 어플리케이션의 기본 구성



  - JDBC 드라이버 : 각 DBMS에 알맞은 클라이언트
    - 보통 jar 파일 형태로 제공

JDBC 프로그래밍 코딩 스타일

```
// 1. JDBC 드라이버 로딩
Class.forName("com.mysql.jdbc.Driver"); // mysql
        // oracle.jdbc.driver.OracleDriver
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try {
    // 2. 데이터베이스 커넥션 생성
conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/test",
"user", "pass");
    // 3. Statement 생성
    stmt = conn.createStatement();
    // 4. 쿼리 실행
    rs = stmt.executeQuery("select * from MEMBER order by MEMBERID");
```

JDBC 프로그래밍 코딩 스타일

```
// 5. 쿼리 실행 결과 출력
        while(rs.next()) {
            String name = rs.getString(1);
        }
    } catch(SQLException ex) {
        ex.printStackTrace();
    } finally {
        // 6. 사용한 Statement 종료
        if (rs != null) try { rs.close(); } catch(SQLException ex) {}
        if (stmt != null) try { stmt.close(); } catch(SQLException ex) {}
        // 7. 커넥션 종료
        if (conn != null) try { conn.close(); } catch(SQLException ex) {}
    }
```

JDBC 드라이버

- DBMS와 통신을 담당하는 자바 클래스
- DBMS 별로 알맞은 JDBC 드라이버 필요
  - 보통 jar 파일로 제공
- JDBC 드라이버 로딩
  - DBMS와 통신하기 위해서는 먼저 로딩해 주어야 함
  - 로딩 코드
    - Class.forName("JDBC드라이버 클래스의 완전한 이름");
  - 주요 DBMS의 JDBC 드라이버
    - MySQL - com.mysql.jdbc.Driver
    - 오라클 - oracle.jdbc.driver.OracleDriver
    - MS SQL 서버 - com.microsoft.sqlserver.jdbc.SQLServerDriver

JDBC URL

- DBMS와의 연결을 위한 식별 값
- JDBC 드라이버에 따라 형식 다름
- 일반적인 구성
  - jdbc:[DBMS]:[데이터베이스식별자]
- 주요 DBMS의 JDBC URL 구성
  - MySQL : jdbc:mysql://HOST[:PORT]/DBNAME[?param=value&param1=value2&...]
  - Oracle: jdbc:oracle:thin:@HOST:PORT:SID
  - MS SQL : jdbc:sqlserver://HOST[:PORT];databaseName=DB

DB 연결 생성

- **DriverManager를 이용해서 Connection 생성**
  - **DriverManager.getConnection(String jdbcURL)**
  - **DriverManager.getConnection(String jdbcURL, String user, String password)**
- **일반적인 코드 구성**

```
Connection conn = null;
try {
    String jdbcDriver = "jdbc:mysql://localhost:3306/chap11?" +
                        "useUnicode=true&characterEncoding=utf-8";
    String dbUser = "jspexam";
    String dbPass = "jspex";
    conn = DriverManager.getConnection(jdbcDriver, dbUser, dbPass);
    ...
} catch(SQLException ex) {
    // 에러 발생
} finally {
    if (conn != null) try { conn.close(); } catch(SQLException ex) {}
}
```

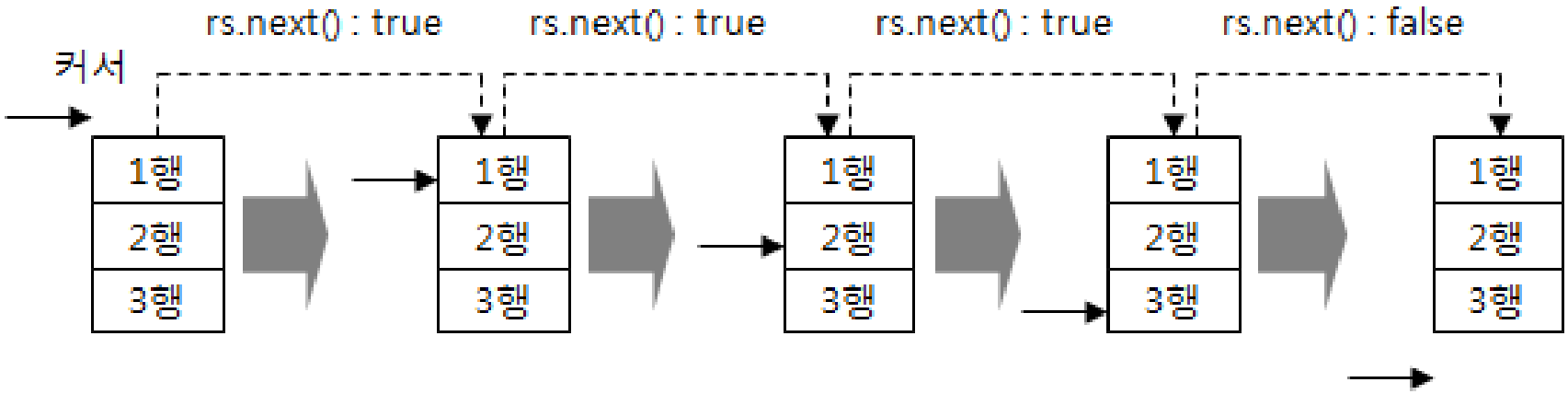
Statement를 이용한 쿼리 실행

- **Connection.createStatement()로 Statement 생성**
- **Statement가 제공하는 메서드로 쿼리 실행**
  - **ResultSet executeQuery(String query) - SELECT 쿼리를 실행**
  - **int executeUpdate(String query) - INSERT, UPDATE, DELETE 쿼리를 실행**

```
Statement stmt = null;
ResultSet rs = null;
try {
    stmt = conn.createStatement();
    int insertedCount = stmt.executeUpdate("insert .....");
    rs = stmt.executeQuery("select * from ....");
    ...
} catch(SQLException ex) {
    ...
} finally {
    if (rs != null) try { rs.close(); } catch(SQLException ex) {}
    if (stmt != null) try { stmt.close(); } catch(SQLException ex) {}
}
```

ResultSet에서 값 조회

- next() 메서드로 데이터 조회 여부 확인



- 데이터 조회 위한 주요 메서드
  - getString()
  - getInt(), getLong(), getFloat(), getDouble()
  - getTimestamp(), getDate(), getTime()

ResultSet에서 데이터 조회하는 코드

- **1개 행 처리**

```
rs = stmt.executeQuery("select * from member");
if (rs.next()) {  // 다음 행(첫 번째 행)이 존재하면 rs.next()는 true를 리턴
    // rs.next()에 의해 다음 행(첫 번째 행)으로 이동
    String name = rs.getString("NAME");
} else {
    // 첫 번째 행이 존재하지 않는다. 즉, 결과가 없다.
}
```

- **1개 이상 행 처리**

```
rs = stmt.executeQuery(...);
if (rs.next()) {
    do {
        String name = rs.getString("NAME");
        ...
    } while( rs.next() );
}
```

■ ResultSet에서 데이터 조회하는 코드

/database/viewMemberList.jsp

```
<%@ page contentType = "text/html; charset=utf-8" %>
<%@ page import = "java.sql.DriverManager" %>
<%@ page import = "java.sql.Connection" %>
<%@ page import = "java.sql.Statement" %>
<%@ page import = "java.sql.ResultSet" %>
<%@ page import = "java.sql.SQLException" %>

<html>
<head><title>회원 목록</title></head>
<body>

MEMBMER 테이블의 내용
<table width="100%" border="1">
<tr>
        <td>이름</td><td>아이디</td><td>이메일</td>
</tr>
<%
        // 1. JDBC 드라이버 로딩
        Class.forName("oracle.jdbc.driver.OracleDriver");
```

ResultSet에서 데이터 조회하는 코드

/database/viewMemberList.jsp

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

try {
        String jdbcDriver = "jdbc:oracle:thin:localhost:1521:orcl" +

"useUnicode=true&characterEncoding=utf8";
        String dbUser = "scott";
        String dbPass = "tiger";

        String query = "select * from emp order by empno";

        // 2. 데이터베이스 커넥션 생성
        conn = DriverManager.getConnection(jdbcDriver, dbUser, dbPass);

        // 3. Statement 생성
        stmt = conn.createStatement();
```

■ ResultSet에서 데이터 조회하는 코드

/database/viewMemberList.jsp

```
                    // 4. 쿼리 실행
                    rs = stmt.executeQuery(query);

                    // 5. 쿼리 실행 결과 출력
                    while(rs.next()) {
%>
<tr>
        <td><%= rs.getString("ENAME") %></td>
        <td><%= rs.getString("EMPNO") %></td>
        <td><%= rs.getString("JOB") %></td>
</tr>
<%
                    }
        } catch(SQLException ex) {
                    out.println(ex.getMessage());
                    ex.printStackTrace();
        } finally {
```

■ ResultSet에서 데이터 조회하는 코드

/database/viewMemberList.jsp

```
                    // 6. 사용한 Statement 종료
                    if (rs != null) try { rs.close(); } catch(SQLException ex) {}
                    if (stmt != null) try { stmt.close(); } catch(SQLException ex) {}

                    // 7. 커넥션 종료
                    if (conn != null) try { conn.close(); } catch(SQLException ex) {}
            }
%>
</table>

</body>
</html>
```

ResultSet에서 데이터 조회하는 코드

/database/update/updateForm.jsp

```
<%@ page contentType = "text/html; charset=utf-8" %>
<html>
<head><title>이름 변경폼</title></head>
<body>
<form action="update.jsp" method="post">
<table border="1">
<tr>
        <td>아이디</td>
        <td><input type="text" name="memberID" size="10"></td>
        <td>이름</td>
        <td><input type="text" name="name" size="10"></td>
</tr>
<tr>
        <td colspan="4"><input type="submit" value="변경"></td>
</tr>
</table>
</form>
</body>
</html>
```

■ ResultSet에서 데이터 조회하는 코드

/database/update/update.jsp

```
<%@ page contentType = "text/html; charset=utf-8" %>
<%@ page import = "java.sql.DriverManager" %>
<%@ page import = "java.sql.Connection" %>
<%@ page import = "java.sql.Statement" %>
<%@ page import = "java.sql.SQLException" %>
<%
request.setCharacterEncoding("utf-8");

String memberID = request.getParameter("memberID");
String name = request.getParameter("name");

int updateCount = 0;

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection conn = null;
Statement stmt = null;
```

■ ResultSet에서 데이터 조회하는 코드

/database/update/update.jsp

```
try {
String jdbcDriver = "jdbc:oracle:thin:localhost:1521:orcl" +
"useUnicode=true&characterEncoding=utf8";
String dbUser = "scott";
String dbPass = "tiger";

String query = "update emp1 set ENAME = '"+name+"' "+
   "where EMPNO = '"+memberID+"'";

conn = DriverManager.getConnection(jdbcDriver, dbUser, dbPass);
stmt = conn.createStatement();
updateCount = stmt.executeUpdate(query);
} finally {
if (stmt != null) try { stmt.close(); } catch(SQLException ex) {}
if (conn != null) try { conn.close(); } catch(SQLException ex) {}
}
%>
```

■ ResultSet에서 데이터 조회하는 코드

/database/update/update.jsp

```
<html>
<head><title>이름 변경</title></head>
<body>
<%  if (updateCount > 0) { %>
<%= memberID %>의 이름을 <%= name %>(으)로 변경
<%  } else { %>
<%= memberID %> 아이디가 존재하지 않음
<%  } %>

</body>
</html>
```

■ ResultSet에서 데이터 조회하는 코드

/database/viewMember.jsp

```
<%@ page contentType = "text/html; charset=utf-8" %>
<%@ page import = "java.sql.DriverManager" %>
<%@ page import = "java.sql.Connection" %>
<%@ page import = "java.sql.Statement" %>
<%@ page import = "java.sql.ResultSet" %>
<%@ page import = "java.sql.SQLException" %>
<%
    String memberID = request.getParameter("memberID");
%>
<html>
<head><title>회원 정보</title></head>
<body>
<%
//Class.forName("com.mysql.jdbc.Driver");
  Class.forName("oracle.jdbc.driver.OracleDriver");

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

ResultSet에서 데이터 조회하는 코드

/database/viewMember.jsp

```
    try {
    String jdbcDriver = "jdbc:oracle:thin:localhost:1521:orcl" +
"useUnicode=true&characterEncoding=utf8";
String dbUser = "scott";
String dbPass = "tiger";

        String query =
           "select * from EMP1 where EMPNO= '"+memberID+"'";

        conn = DriverManager.getConnection(jdbcDriver, dbUser, dbPass);
        stmt = conn.createStatement();

        rs = stmt.executeQuery(query);
        if( rs.next() ) {
%>
<table border="1">
<tr>
    <td>아이디</td><td><%= memberID %></td>
</tr>
```

■ ResultSet에서 데이터 조회하는 코드

/database/viewMember.jsp

```
<tr>
    <td>암호</td><td><%= rs.getString("SAL") %></td>
</tr>
<tr>
    <td>이름</td><td><%= rs.getString("ENAME") %></td>
</tr>
<tr>
    <td>이메일</td><td><%= rs.getString("JOB") %></td>
</tr>
</table>
<%
        } else {
%>
<%= memberID %>에 해당하는 정보가 존재하지 않습니다.
<%
        }
    } catch(SQLException ex) {
%>
```

■ ResultSet에서 데이터 조회하는 코드

/database/viewMember.jsp

```
에러 발생: <%= ex.getMessage() %>
<%
    } finally {
        if (rs != null) try { rs.close(); } catch(SQLException ex) {}
        if (stmt != null) try { stmt.close(); } catch(SQLException ex) {}
        if (conn != null) try { conn.close(); } catch(SQLException ex) {}
    }
%>

</body>
</html>
```

PreparedStatement를 이용한 처리

- **SQL의 틀을 미리 정해 놓고, 나중에 값을 지정하는 방식**
- **PreparedStatement의 일반적 사용**

```
pstmt = conn.prepareStatement(
    "insert into MEMBER (MEMBERID, NAME, EMAIL) values (?, ?, ?)");
pstmt.setString(1, "madvirus"); // 첫번째 물음표의 값 지정
pstmt.setString(2, "최범균");   // 두번째 물음표의 값 지정
pstmt.executeUpdate();
```

- **쿼리 실행 관련 메서드**
  - **ResultSet executeQuery() - SELECT 쿼리를 실행할 때 사용되며 ResultSet을 결과값으로 리턴한다.**
  - **int executeUpdate() - INSERT, UPDATE, DELETE 쿼리를 실행할 때 사용되며, 실행 결과 변경된 레코드의 개수를 리턴한다**

PreparedStatement의 값 바인딩 관련 메서드

메서드	설명
setString(int index, String x)	지정한 인덱스의 파라미터 값을 x로 지정한다.
setInt(int index, int x)	지정한 인덱스의 파라미터 값을 int 값 x로 지정한다.
setLong(int index, long x)	지정한 인덱스의 파라미터 값을 long 값 x로 지정한다.
setDouble(int index, double x)	지정한 인덱스의 파라미터 값을 double 값 x로 지정한다.
setFloat(int index, float x)	지정한 인덱스의 파라미터 값을 float 값 x로 지정한다.
setTimestamp(int index, Timesetamp x)	지정한 인덱스의 값을 SQL TIMESTMAP 타입을 나타내는 java.sql.Timestamp 타입으로 지정한다.
setDate(int index, Date x)	지정한 인덱스의 값을 SQL DATE 타입을 나타내는 java.sql.Date 타입으로 지정한다.
setTime(int index, Time x)	지정한 인덱스의 값을 SQL TIME 타입을 나타내는 java.sql.Time 타입으로 지정한다.

■ PreparedStatement를 이용한 처리

/database/insert/viewMember.jsp

```
<%@ page contentType = "text/html; charset=utf-8" %>
<html>
<head><title>MEMBER 테이블 레코드 삽입</title></head>
<body>
<form action="insert.jsp" method="post">
<table border="1">
<tr>      <td>아이디</td>
          <td><input type="text" name="memberID" size="10"></td>
          <td>급여</td>
          <td><input type="text" name="sal" size="10"></td>
</tr>
<tr>      <td>이름</td>
          <td><input type="text" name="name" size="10"></td>
          <td>업무</td>
          <td><input type="text" name="job" size="10"></td>
</tr>
<tr>      <td colspan="4"><input type="submit" value="삽입"></td>
</tr>
</table></form></body></html>
```

PreparedStatement를 이용한 처리

/database/insert/viewMember.jsp

```
<%@ page contentType = "text/html; charset=utf-8" %>
<%@ page import = "java.sql.DriverManager" %>
<%@ page import = "java.sql.Connection" %>
<%@ page import = "java.sql.PreparedStatement" %>
<%@ page import = "java.sql.SQLException" %>
<%
request.setCharacterEncoding("utf-8");
String memberID = request.getParameter("memberID");
String sal= request.getParameter("sal");
String name = request.getParameter("name");
String job = request.getParameter("job");
//Class.forName("com.mysql.jdbc.Driver");
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection conn = null;
PreparedStatement pstmt = null;
try {
String jdbcDriver = "jdbc:oracle:thin:localhost:1521:orcl";
String dbUser = "scott";
String dbPass = "tiger";
```

PreparedStatement를 이용한 처리

/database/insert/viewMember.jsp

```
conn = DriverManager.getConnection(jdbcDriver, dbUser, dbPass);
pstmt = conn.prepareStatement(
"insert into EMP1 values (?, ?, ?, ?)");
pstmt.setString(1, memberID);
pstmt.setString(2, sal);
pstmt.setString(3, name);
pstmt.setString(4, job);
pstmt.executeUpdate();
} finally {
if (pstmt != null) try { pstmt.close(); } catch(SQLException ex) {}
if (conn != null) try { conn.close(); } catch(SQLException ex) {}
}
%>
<html>
<head><title>삽입</title></head>
<body>
MEMBER 테이블에 새로운 레코드를 삽입했습니다
</body>
</html>
```

PreparedStatement의 사용 이유

- 반복해서 실행되는 동일 쿼리의 속도를 향상
  - DBMS가 PreparedStatement와 관련된 쿼리 파싱 회수 감소
- 값 변환 처리
  - 작은 따옴표 등 값에 포함된 특수 문자의 처리
- 코드의 간결함
  - 문자열 연결에 따른 코드의 복잡함 감소

■ 웹 어플리케이션 구동시 JDBC 드라이버 로딩

src/Jdbc/Loader.java

```
package Jdbc;
import javax.servlet.http.HttpServlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import java.util.StringTokenizer;
public class Loader extends HttpServlet {
        public void init(ServletConfig config) throws ServletException {
                try {
                        String drivers = config.getInitParameter("jdbcdriver");
                        StringTokenizer st = new StringTokenizer(drivers, ",");
                        while (st.hasMoreTokens()) {
                                String jdbcDriver = st.nextToken();
                                Class.forName(jdbcDriver);
                        }
                } catch(Exception ex) {
                        throw new ServletException(ex);
                }
        }
}
```

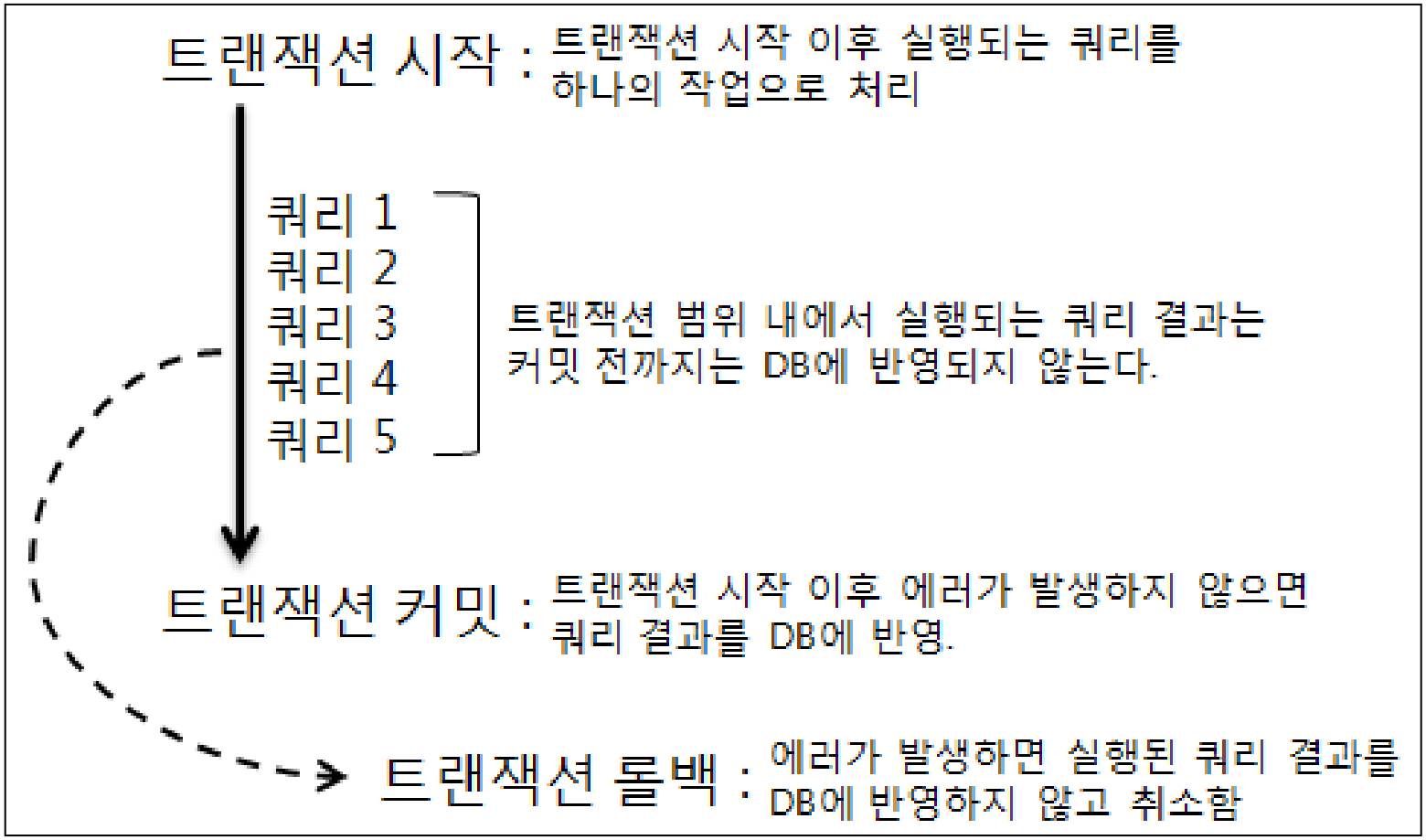
■ 웹 어플리케이션 구동시 JDBC 드라이버 로딩

web.xml

```
<servlet>
        <servlet-name>JDBCDriverLoader</servlet-name>
        <servlet-class>Jdbc.Loader</servlet-class>
        <init-param>
                <param-name>jdbcdriver</param-name>
                <param-value>oracle.jdbc.driver.OracleDriver</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
</servlet>
```

트랜잭션

- 데이터의 무결성을 위해 하나의 작업을 위한 쿼리는 트랜잭션으로 처리될 필요



- 트랜잭션 구현 방법: 오토 커밋 해제

JDBC API에서의 트랜잭션 처리

- **Connection.setAutoCommit(false)**

```
try {
    conn = DriverManager.getConnection(...);
    // 트랜잭션 시작
    conn.setAutoCommit(false);
    ... // 쿼리 실행
    ... // 쿼리 실행
    // 트랜잭션 커밋
    conn.commit();
} catch(SQLException ex) {
    if (conn != null) {
        // 트랜잭션 롤백
        conn.rollback();
    }
} finally {
    if (conn != null) {
        try {
            conn.close();
        } catch(SQLException ex) {}
    }
}
```

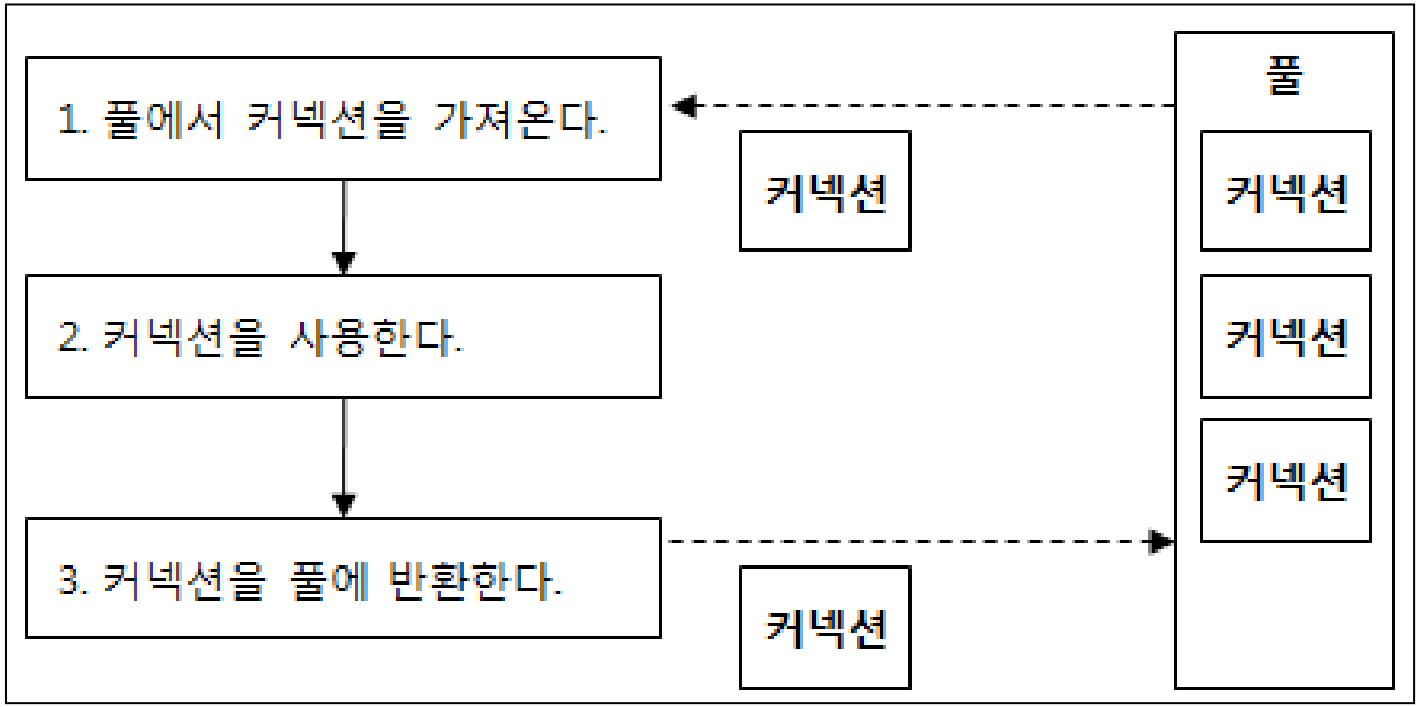
JDBC API에서의 트랜잭션 처리

- **Connection.setAutoCommit(false)**

- **폰북에서 회사 친구와 학교 친구의 경우 두 개의 테이블에 입력이 되어야 한다.**
- **두 개의 테이블이 정상적으로 입력이되도록 트랜잭션 처리 해 보자.**

커넥션 풀

- 데이터베이스와 연결된 커넥션을 미리 만들어서 풀(pool) 속에 저장해 두고 있다가 필요할 때에 커넥션을 풀에서 가져다 쓰고 다시 풀에 반환하는 기법



- 특징
  - 커넥션을 생성하는 데 드는 연결 시간이 소비되지 않는다.
  - 커넥션을 재사용하기 때문에 생성되는 커넥션 수가 많지 않다.

DBCP API를 이용한 커넥션 풀 사용

- 필요 라이브러리
  - Commons-DBCP API 관련 Jar 파일: commons-dbcp2-2.1.1.jar
  - Commons-Pool API의 Jar 파일: commons-pool2-2.4.2.jar
- DBCP가 제공하는 JDBC 드라이버 로딩
  - Class.forName("org.apache.commons.dbcp2.PoolingDriver");
    PoolingDriver driver = (PoolingDriver)
    DriverManager.getDriver("jdbc:apache:commons:dbcp:");
    driver.registerPool(“pootest", connectionPool);
  - 실제 DBMS에 연결할 때 사용될 JDBC 드라이버도 로딩해야 함
- DBCP가 제공하는 커넥션 풀로부터 커넥션 가져오기
  - JDBC URL: jdbc:apache:commons:dbcp:풀이름
  - DriverManager.getConnection("jdbc:apache:commons:dbcp:pooltest");

커넥션 풀 초기화

src/DBCPInit.java

```
package Jdbc;
public class DBCPInit extends HttpServlet {
        @Override
        public void init() throws ServletException {
                loadJDBCDriver();
                initConnectionPool();
        }

        private void loadJDBCDriver() {
                try {
//커넥션 풀이 내부에서 사용할 jdbc 드라이버를 로딩함.
                        //Class.forName("com.mysql.jdbc.Driver");
                        Class.forName("oracle.jdbc.driver.OracleDriver");
                } catch (ClassNotFoundException ex) {
                        throw new RuntimeException("fail to load JDBC Driver", ex);
                }
        }
```

커넥션 풀 초기화

src/DBCPInit.java

```
        private void initConnectionPool() {
            try {
//String jdbcUrl =  "jdbc:mysql://localhost:3306/test?" + "useUnicode=true&characterEncoding=utf8";
                String jdbcDriver = "jdbc:oracle:thin:localhost:1521:orcl";
                String username = "scott";
                String pw = "tiger";
//커넥션풀이 새로운 커넥션을 생성할 때 사용할 커넥션팩토리를 생성.
ConnectionFactory connFactory = new DriverManagerConnectionFactory(jdbcUrl, username, pw);
// PoolableConnection을 생성하는 팩토리 생성.
// DBCP는 커넥션을 보관할 때 PoolableConnection 을 사용
// 실제 커넥션을 담고 있있으며, 커넥션 풀을 관리하는데 필요한 기능을 제공한다.
// 커넥션을 close하면 종료하지 않고 커넥션 풀에 반환
PoolableConnectionFactory poolableConnFactory = new PoolableConnectionFactory(connFactory, null);
//커넥션이 유효한지 여부를 검사할 때 사용하는 쿼리를 지정한다.
                poolableConnFactory.setValidationQuery("select 1");
//커넥션 풀의 설정 정보를 생성한다.
            GenericObjectPoolConfig poolConfig = new GenericObjectPoolConfig();
//유휴 커넥션 검사 주기
                poolConfig.setTimeBetweenEvictionRunsMillis(1000L * 60L * 5L);
//풀에 보관중인 커넥션이 유효한지 검사할지 유무 설정
                poolConfig.setTestWhileIdle(true);
```

커넥션 풀 초기화

src/DBCPInit.java

```
//커넥션 최소 개수
                                        poolConfig.setMinIdle(4);
//커넥션 최대 개수
                                        poolConfig.setMaxTotal(50);
//커넥션 풀을 생성. 생성자는  PoolabeConnectionFactory와 GenericObjectPoolConfig를 사용
                                        GenericObjectPool<PoolableConnection> connectionPool =
                                        new GenericObjectPool<>(poolableConnFactory, poolConfig);
//PoolabeConnectionFactory에도 커넥션 풀을 연결
                                        poolableConnFactory.setPool(connectionPool);
//커넥션 풀을 제공하는 jdbc 드라이버를 등록.
                                        Class.forName("org.apache.commons.dbcp2.PoolingDriver");
            PoolingDriver driver = (PoolingDriver) DriverManager.getDriver("jdbc:apache:commons:dbcp:");
//위에서 커넥션 풀 드라이버에 생성한 커넥션 풀을 등록한다. 이름은 chap14 이다.
                                        driver.registerPool("chap14", connectionPool);
                        } catch (Exception e) {
                                        throw new RuntimeException(e);
                        }
            }
}
```

- 실제 커넥션을 생성할 ConnectionFactory를 생성한다.
- 커넥션 풀로 사용할 PoolableConnection을 생성하는 PoolableConnectionFactory를 생성한다.
- 커넥션 풀 설정 정보를 생성한다.
- 커넥션 풀을 사용할 JDBC 드라이버를 등록한다.

■ 웹 어플리케이션 구동시 JDBC 드라이버 로딩

web.xml

```
<servlet>
        <servlet-name>DBCPInit</servlet-name>
        <servlet-class>Jdbc.DBCPInit</servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
```

■ 커넥션 풀 초기화

database/viewMemberUsingPool.jsp

```
<%@ page contentType = "text/html; charset=utf-8" %>
<%@ page import = "java.sql.DriverManager" %>
<%@ page import = "java.sql.Connection" %>
<%@ page import = "java.sql.Statement" %>
<%@ page import = "java.sql.ResultSet" %>
<%@ page import = "java.sql.SQLException" %>
<html>
<head><title>직원 목록</title></head>
<body>
직원 테이블의 내용
<table width="100%" border="1">
<tr>
    <td>이름</td><td>아이디</td><td>이메일</td>
</tr>
<%
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
```

■커넥션 풀 초기화

database/viewMemberUsingPool.jsp

```
    try {
        String jdbcDriver = "jdbc:apache:commons:dbcp:/pool";
        String query = "select * from EMP order by EMPNO";
        conn = DriverManager.getConnection(jdbcDriver);
        stmt = conn.createStatement();
        rs = stmt.executeQuery(query);
        while(rs.next()) {
%><tr>
    <td><%= rs.getString("ENAME") %></td>
    <td><%= rs.getString("EMPNO") %></td>
    <td><%= rs.getString("JOB") %></td>
</tr>
<%        }
    } finally {
        if (rs != null) try { rs.close(); } catch(SQLException ex) {}
        if (stmt != null) try { stmt.close(); } catch(SQLException ex) {}
        if (conn != null) try { conn.close(); } catch(SQLException ex) {}
    }
%></table></body></html>
```

DBCP를 이용한 회원가입 처리 구현

① 회원가입 폼

② 회원가입 처리

: 데이터 받기, 데이터베이스 저장

③ 로그인

: 데이터 베이스를 이용한 아이디와 비밀번호 체크