6.1. Dynamic SQL

마이바티스의 가장 강력한 기능 중 하나는 동적 SQL을 처리하는 방법이다. JDBC나 다른 유사한 프레임워크를 사용해본 경험이 있다면 동적으로 SQL 을 구성하는 것이 얼마나 힘든 작업인지 이해할 것이다. 간혹 공백이나 콤마를 붙이는 것을 잊어본 적도 있을 것이다. 동적 SQL은 그만큼 어려운 것이다. 마이바티스는 강력한 동적 SQL 언어로써 이 상황을 개선한다.

if

동적 SQL 에서 가장 공통적으로 사용되는 것으로 where의 일부로 포함될 수 있다.

```
<select id="findActiveBlogWithTitleLike" resultType="Blog">
  SELECT * FROM BLOG WHERE state = 'ACTIVE'
  <if test="title != null">
    AND title like #{title}
  </if>
</select>
```

만약에 title 값이 없다면 모든 active 상태의 Blog 가 리턴될 것이다. 하지만 title 값이 있다면 그 값과 비슷한 데이터를 찾게 될 것이다.

추가적으로 author 객체를 사용하여 검색하는 예를 보자.

```
<select id="findActiveBlogLike" resultType="Blog">
  SELECT * FROM BLOG WHERE state = 'ACTIVE'
  <if test="title != null">
    AND title like #{title}
  </if>
  <if test="author != null and author.name != null">
    AND author_name like #{author.name}
  </if>
</select>
```

choose, when, otherwise

우리는 종종 적용 할 모든 조건을 원하는 대신에 한가지 경우만을 원할 수 있다. 자바의 switch 구문과 유사한 choose 엘리먼트를 제공한다.

```
<select id="findActiveBlogLike" resultType="Blog">
  SELECT * FROM BLOG WHERE state = 'ACTIVE'
  <choose>
    <when test="title != null">
      AND title like #{title}
    </when>
    <when test="author != null and author.name != null">
      AND author_name like #{author.name}
```

```
    </when>
    <otherwise>
      AND featured = 1
    </otherwise>
  </choose>
</select>
```

trim, where, set

다음 예제는 잘 못된 사용방법을 보여준다.

```
<select id="findActiveBlogLike" resultType="Blog">
  SELECT * FROM BLOG WHERE
  <if test="state != null">
    state = #{state}
  </if>
  <if test="title != null">
    AND title like #{title}
  </if>
  <if test="author != null and author.name != null">
    AND author_name like #{author.name}
  </if>
</select>
```

어떤 조건에도 해당되지 않는다면 어떤 일이 벌어질까? 다음과 같은 SQL 이 만들어질 것이다. 이건 실패할 것이다.

```
SELECT * FROM BLOG WHERE
```

두번째 조건에만 해당된다면 무슨 일이 벌어질까? 다음과 같은 SQL이 만들어질 것이다. 이것도 실패할 것이다.

```
SELECT * FROM BLOG WHERE
AND title like 'someTitle'
```

실패하지 않기 위해서 조금 수정해야 한다.

```
<select id="findActiveBlogLike" resultType="Blog">
  SELECT * FROM BLOG
  <where>
    <if test="state != null">
         state = #{state}
    </if>
    <if test="title != null">
        AND title like #{title}
    </if>
    <if test="author != null and author.name != null">
        AND author_name like #{author.name}
    </if>
  </where>
</select>
```

where 엘리먼트는 태그에 의해 컨텐츠가 리턴되면 단순히 “WHERE”만을 추가한다. 게다가 컨텐츠가 “AND”나 “OR”로 시작한다면 그 “AND”나 “OR”를 지워버린다.

만약에 where 엘리먼트가 기대한 것처럼 작동하지 않는다면 trim 엘리먼트를 사용할 수도 있다. 예를 들어 다음은 where 엘리먼트에 대한 trim 기능과 동일하다.

```
<trim prefix="WHERE" prefixOverrides="AND |OR ">
  ...
</trim>
```

override 속성은 오버라이드하는 텍스트의 목록을 제한한다. 결과는 override 속성에 명시된 것들을 지우고 with 속성에 명시된 것을 추가한다.

다음 예제는 동적인 update 구문의 유사한 경우이다. set 엘리먼트는 update 하고자 하는 칼럼을 동적으로 포함시키기 위해 사용될 수 있다.

```
<update id="updateAuthorIfNecessary">
  update Author
    <set>
      <if test="username != null">username=#{username},</if>
      <if test="password != null">password=#{password},</if>
      <if test="email != null">email=#{email},</if>
      <if test="bio != null">bio=#{bio}</if>
    </set>
  where id=#{id}
</update>
```

여기서 set 엘리먼트는 동적으로 SET 키워드를 붙히고 필요없는 콤마를 제거한다.

trim 엘리먼트로 처리한다면 아래와 같을 것이다.

```
<trim prefix="SET" suffixOverrides=",">
  ...
</trim>
```

이 경우 접두사는 추가하고 접미사를 오버라이딩 한다.

foreach

동적 SQL 에서 공통적으로 필요한 것은 collection 에 대해 반복처리를 하는 것이다. 종종 IN 조건을 사용하게 된다.

```
<select id="selectPostIn" resultType="domain.blog.Post">
  SELECT * FROM POST P
  WHERE ID in
  <foreach item="item" index="index" collection="list"
      open="(" separator="," close=")">
        #{item}
```

```
  </foreach>
</select>
```

foreach 엘리먼트는 매우 강력하고 collection 을 명시하는 것을 허용한다. 엘리먼트 내부에서 사용할 수 있는 item, index 두가지 변수를 선언한다. 이 엘리먼트는 또한 열고 닫는 문자열로 명시할 수 있고 반복간에 둘 수 있는 구분자도 추가할 수 있다.

참고 컬렉션 파라미터로 Map이나 배열객체와 더불어 List, Set 등과 같은 반복가능한 객체를 전달할 수 있다. 반복가능하거나 배열을 사용할때 index 값은 현재 몇번째 반복인지를 나타내고 value 항목은 반복과정에서 가져오는 요소를 나타낸다. Map을 사용할때 index는 key 객체가 되고 항목은 value 객체가 된다.

bind

bind 엘리먼트는 OGNL표현을 사용해서 변수를 만든 뒤 컨텍스트에 바인딩한다.

```
<select id="selectBlogsLike" resultType="Blog">
  <bind name="pattern" value="'%' + _parameter.getTitle() + '%'" />
  SELECT * FROM BLOG
  WHERE title LIKE #{pattern}
</select>
```

_parameter는 메소드가 받는 파라미터를 가리키는 예약어다. 파라미터로 받은 객체안에 title이라는 멤버변수가 있고 getter 메소드로 접근해서 사용하는 예제이다. '%' 기호는 쿼리문에서 LIKE 키워드 다음에서 사용하는 데이터베이스가 선언한 예약기호이다.

XML에서 자바 객체를 생성할 수는 없다. 따라서 bind 태그에서 접근할 수 있는 객체는 파라미터로 받는 객체이거나 이미 메모리에 존재하는 static 자원이 사용 대상이 된다. 파라미터로 Map 자료형의 객체를 받아서 get() 메소드의 키값을 주고 값을 얻는 사용 예를 살펴보자.

```
<select id="search" resultMap="empRowMapper">
  select * from emp
  <trim prefix="WHERE" prefixOverrides="AND|OR">
    <if test="ename != null">
      <bind name="patternEname" value="'%' + _parameter.get('ename') + '%'" />
      ename like #{patternEname}
    </if>
    <if test="job != null">
      <bind name="patternJob" value="'%' + _parameter.get('job') + '%'" />
      and job like #{patternJob}
    </if>
    <if test="salMin != null">
      and sal <![CDATA[>=]]> #{salMin}
    </if>
    <if test="salMax != null">
      and sal <![CDATA[<=]]> #{salMax}
    </if>
  </trim>
</select>
```

Multi-db vendor support

"_databaseId" 변수로 설정된 databaseIdProvider가 동적인 코드에도 사용가능하다면 데이터베이스 제품별로 서로다른 구문을 사용할 수 있다. 다음의 예제를 보라:

```
<insert id="insert">
  <selectKey keyProperty="id" resultType="int" order="BEFORE">
    <if test="_databaseId == 'oracle'">
      select seq_users.nextval from dual
    </if>
    <if test="_databaseId == 'db2'">
      select nextval for seq_users from sysibm.sysdummy1"
    </if>
  </selectKey>
  insert into users values (#{id}, #{name})
</insert>
```

application.properties 파일에 관련설정을 추가할 필요가 있다.

```
mybatis.configuration.database-id=h2
```

Search Example Project

새 프로젝트 만들기

```
STS >> New >> Spring Starter Project

Project Name : spring-mybatis-search
Type : Maven
Packaging : Jar
Java : 8
Package : com.example.demo

Project Dependencies : Web, Lombok, MyBatis, H2
```

JSP 디펜던시 추가

pom.xml

```
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>
```

환경설정

application.properties

```
# DataSource(JDBC Connection) Configuration
spring.datasource.url=jdbc:h2:mem:TESTDB;DB_CLOSE_DELAY=-1;
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

# Use schema.sql, data.sql
spring.datasource.initialize=true

# SQL Holder File
mybatis.mapper-locations=/mybatis/mapper/*-mapper.xml

# Underscore: user_id (DB)
# Camel-case: userId (Java)
# Class User.userId <==Mapping==> Table USER#user_id
mybatis.configuration.map-underscore-to-camel-case=true

mybatis.type-aliases-package=com.example.demo.domain

spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
```

`src/main` 폴더밑에 `webapp/WEB-INF/views` 폴더를 생성한다.

schema.sql

```
drop table if exists emp;

create table emp (
    empno bigint identity not null primary key,
    ename varchar(100),
    job varchar(100),
    sal bigint
);
```

data.sql

```
insert into emp(ename, job, sal) values('일길동1', '도적', 100);
insert into emp(ename, job, sal) values('이길동2', '도적', 200);
insert into emp(ename, job, sal) values('삼길동3', '의사', 300);
insert into emp(ename, job, sal) values('사길동4', '도적', 400);
insert into emp(ename, job, sal) values('오길동5', '도적', 500);
insert into emp(ename, job, sal) values('홍길동6', '도적', 600);
insert into emp(ename, job, sal) values('홍길동7', '장군', 700);
insert into emp(ename, job, sal) values('홍길동8', '도적', 800);
insert into emp(ename, job, sal) values('홍길동9', '백수', 900);
```

```
insert into emp(ename, job, sal) values('홍길동10', '도적', 1000);

insert into emp(ename, job, sal) values('일지매1', '의적', 100);
insert into emp(ename, job, sal) values('이지매2', '백수', 200);
insert into emp(ename, job, sal) values('삼지매3', '의적', 300);
insert into emp(ename, job, sal) values('사지매4', '의사', 400);
insert into emp(ename, job, sal) values('오지매5', '의적', 500);
insert into emp(ename, job, sal) values('일지매6', '의적', 600);
insert into emp(ename, job, sal) values('일지매7', '의적', 700);
insert into emp(ename, job, sal) values('일지매8', '장군', 800);
insert into emp(ename, job, sal) values('일지매9', '의적', 900);
insert into emp(ename, job, sal) values('일지매10', '의적', 1000);

insert into emp(ename, job, sal) values('임꺽정1', '산적', 500);
```

도메인 클래스

Emp.java

```
package com.example.demo.domain;

import lombok.Data;

@Data
public class Emp {
    private int empno;
    private String ename;
    private String job;
    private double sal;
}
```

DAO 클래스

EmpDao.java

```
package com.example.demo.dao;

import java.util.List;
import java.util.Map;

import com.example.demo.domain.Emp;

public interface EmpDao {
    public int insert(Emp emp);
    public int update(Emp emp);
    public int delete(int empno);

    public List<Emp> findAll();
    public int count();
    public Emp findOne(int empno);

    /*
     * 테이블의 특정 범위의 로우들만을 조회하는 메소드 예제
     * - findByStartEnd
     * - findBySkipLimit
```

```
     * - findByPageSize
     *
     * 전제: 정렬은 이미 되어 있다. 거의 대부분의 디비는 PK의 오름차순으로
     * 정렬한 상태로 데이터를 유지한다. 또는 조회 시 마다 order by 구문을 사용한다.
     */

    /*
     * 1. 자바 개발자 입장에서 유리하다.
     * start: empno의 시작 값, end: empno의 종료 값
     */
    public List<Emp> findByStartEnd(int start, int end);

    /*
     * 2. 디비 쿼리 작성 입장에서 유리하다.
     * skip: 앞에서부터 얼마나 스킵할 것인지에 대한 값, limit: 구하는 로우의 개수
     */
    public List<Emp> findBySkipLimit(int skip, int limit);

    /*
     * 3. 사용자 입장(UX)에서 유리하다.
     *   전체 로우의 개수 = 21
     *   size 지정 값 = 10
     *   총 페이지 수 = ceil(21/10) = 3
     * page: 사용자의 선택 페이지 번호, size: 구하는 로우의 개수
     */
    public List<Emp> findByPageSize(int page, int size);

    /*
     * <bind> 태그 사용 예제
     */
    public List<Emp> findByPageSizeUsingBind(int page, int size);

    /*
     * 동적 쿼리 사용 예제
     * ename, job, sal 칼럼으로 검색하는 기능의 메소드를 제공하고 싶다.
     * 다음은 쿼리의 작성 예제이다. 필터링 조건을 단수 또는 복수로 연결하여 질의한다.
     * select * from EMP where ENAME like '%길동%'
     * select * from EMP where JOB like '%적%'
     * select * from EMP where SAL >= 800 and SAL <= 900
     * select * from EMP where SAL >= 800
     * select * from EMP where SAL <= 900
     */
    public List<Emp> search(Map<String, String> map);
}
```

EmpDaoImpl.java

```
package com.example.demo.dao;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.ibatis.session.SqlSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.example.demo.domain.Emp;

/*
 * 개발자가 @Mapper 애노테이션을 사용하는 대신 직접 인터페이스를
```

```
 * 구현한 클래스를 작성하면 매퍼 XML에게 전달하는 파라미터 값을
 * 조작할 수 있는 메소드 공간을 가질 수 있다.
 * 더불어서 질의 결과를 조작할 수 있는 로직도 배치할 수 있다.
 */
@Repository
public class EmpDaoImpl implements EmpDao {
    /*
     * SqlSessionTemplate 객체를 주입 받는다.
     * 스프링 부트는 마이바티스 디펜던시를 선택하는 것만으로
     * 빈 컨테이너에 해당 객체가 등록되지만
     * 스프링 레가시에서는 개발자가 직접 등록해야 한다.
     */
    @Autowired
    private SqlSession session;

    @Override
    public int insert(Emp emp) {
        return session.insert(
                "com.example.demo.dao.EmpDao.insert", emp);
    }

    @Override
    public int update(Emp emp) {
        return session.update(
                "com.example.demo.dao.EmpDao.update", emp);
    }

    @Override
    public int delete(int empno) {
        return session.update(
                "com.example.demo.dao.EmpDao.delete", empno);
    }

    @Override
    public List<Emp> findAll() {
        return session.selectList(
                "com.example.demo.dao.EmpDao.findAll");
    }

    @Override
    public int count() {
        return session.selectOne(
                "com.example.demo.dao.EmpDao.count");
    }

    @Override
    public Emp findOne(int empno) {
        return session.selectOne(
                "com.example.demo.dao.EmpDao.findOne", empno);
    }

    @Override
    public List<Emp> findByStartEnd(int start, int end) {
        Map<String, Integer> map = new HashMap<>();
        map.put("start", start); // 범위의 시작
        map.put("end", end); // 범위의 끝

        return session.selectList(
                "com.example.demo.dao.EmpDao.findByStartEnd", map);
    }

    @Override
    public List<Emp> findBySkipLimit(int skip, int limit) {
        Map<String, Integer> map = new HashMap<>();
```

```
        map.put("skip", skip); // 스킵(건너 뜀)
        map.put("limit", limit); // 개수

        return session.selectList(
                "com.example.demo.dao.EmpDao.findBySkipLimit", map);
    }

    @Override
    public List<Emp> findByPageSize(int page, int size) {
        // page, size 값을 이용하여 skip 값을 계산한다.
        int skip = 0;
        if (page > 0) {
            skip = (page - 1) * size;
        }

        // size와 limit는 논리적으로 같은 의미이다.
        Map<String, Integer> map = new HashMap<>();
        map.put("skip", skip); // 스킵(건너 뜀)
        map.put("limit", size); // 개수

        return session.selectList(
                "com.example.demo.dao.EmpDao.findByPageSize", map);
    }

    @Override
    public List<Emp> findByPageSizeUsingBind(int page, int size) {
        Map<String, Integer> map = new HashMap<>();
        map.put("page", page); // 페이지(몇 번째 화면)
        map.put("size", size); // 개수

        return session.selectList(
            "com.example.demo.dao.EmpDao.findByPageSizeUsingBind", map);
    }

    @Override
    public List<Emp> search(Map<String, String> map) {
        /*
         * 매퍼 XML의 <if> 태그에서 제대로 된 값의 상태를 체크하기 위한 처리작업이 필요하다.
         * 1. 빈 문자열을 null로 바꾼다. if 조건에서 null 값으로 비교할 수 있다.
         * 2. if 조건에서 null 값으로 비교하는 대신 빈 문자열을 대상으로 체크한다.
         */

        // #1 방식으로 사용하기 위한 처리작업을 수행한다.
        map.forEach((key, value) -> {
            if ("".equals(value)) {
                map.put(key, null);
            }
        });

        return session.selectList(
                "com.example.demo.dao.EmpDao.search", map);
    }

}
```

MyUtil.java

```
package com.example.demo.util;

public class MyUtil {
    // page 값을 skip 값으로 변환해서 리턴한다.
```

```
    public static String pageToSkip(String pageString, String sizeString) {
        int page = Integer.parseInt(pageString);
        int size = Integer.parseInt(sizeString);

        int skip = 0;
        if (page > 0) {
            skip = (page - 1) * size;
        }

        return String.valueOf(skip);
    }
}
```

emp-mapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.demo.dao.EmpDao">

    <resultMap type="Emp" id="empRowMapper">
        <result property="empno" column="empno"/>
        <result property="ename" column="ename"/>
        <result property="job" column="job"/>
        <result property="sal" column="sal"/>
    </resultMap>

    <insert id="insert">
        insert into EMP (ename, job, sal)
        values (#{ename}, #{job}, #{sal})

        <selectKey order="AFTER" keyProperty="empno" resultType="int" >
            select LAST_INSERT_ID() as empno
        </selectKey>
    </insert>

    <update id="update">
        update EMP set ename=#{ename}, job=#{job}, sal=#{sal}
        where empno=#{empno}
    </update>

    <delete id="delete">
        delete from EMP where empno=#{empno}
    </delete>

    <select id="findAll" resultMap="empRowMapper">
        select * from emp order by empno asc
    </select>

    <select id="count" resultType="int">
        select count(*) from emp
    </select>

    <select id="findOne" resultType="Emp">
        select * from emp where empno=#{empno}
    </select>

    <!--
        쿼리문에서 >, < 기호 사용 시 에러가 발생할 때 해결 방법 3가지

        1. between 문법을 대신 사용한다.
```

```
        select * from emp
        where empno between #{start} and #{end}
        order by empno asc

        2. 치환기호를 사용한다.
        select * from emp
        where empno &gt;= #{start} and empno &lt;= #{end}
        order by empno asc

        3. CDATA Section으로 감싼다.
        범위 안에 태그가 존재하지 않는다는 뜻이다.
     -->
    <select id="findByStartEnd" resultMap="empRowMapper">
        <![CDATA[
            select * from emp
            where empno >= #{start} and empno <= #{end}
            order by empno asc
        ]]>
    </select>

    <select id="findBySkipLimit" resultMap="empRowMapper">
        <![CDATA[
            select * from emp order by empno asc
            limit #{skip}, #{limit}
        ]]>
    </select>

    <select id="findByPageSize" resultMap="empRowMapper">
        <![CDATA[
            select * from emp order by empno asc
            limit #{skip}, #{limit}
        ]]>
    </select>

    <!--
        static 메소드 pageToSkip()를 호출하여 page, size 값을 주고
        skip 값을 얻어서 bind 태그 내 변수 skip에 담고 쿼리문에서 사용한다.
     -->
    <select id="findByPageSizeUsingBind" resultMap="empRowMapper">
        <bind name="skip" value="@com.example.demo.util.MyUtil@pageToSkip(page, size)" />
        <![CDATA[
            select * from emp
            order by empno asc
            limit #{skip}, #{size}
        ]]>
    </select>

    <!--
        trim 태그 작동방식
        1. 태그 안쪽에 문자열이 존재하면 prefix 값을 먼저 출력한다.
        2. prefix 값인 문자열 다음에 AND|OR가 나오면 삭제한다.

        <select id="search" resultMap="empRowMapper">
            select * from emp
            <trim prefix="WHERE" prefixOverrides="AND|OR">
                <if test="ename != null">
                    ename like CONCAT('%',#{ename},'%')
                </if>
                <if test="job != null">
                    and job like CONCAT('%',#{job},'%')
                </if>
                <if test="salMin != null">
                    and sal <![CDATA[>=]]> #{salMin}
                </if>
```

```
                <if test="salMax != null">
                    and sal <![CDATA[<=]]> #{salMax}
                </if>
            </trim>
        </select>

        CONCAT('%',#{ename},'%') 대신 bind 태그를 사용할 수 있다.
     -->
    <select id="search" resultMap="empRowMapper">
        select * from emp
        <trim prefix="WHERE" prefixOverrides="AND|OR">
            <if test="ename != null">
            <bind name="patternEname" value="'%' + _parameter.get('ename') + '%'" />
                ename like #{patternEname}
            </if>
            <if test="job != null">
            <bind name="patternJob" value="'%' + _parameter.get('job') + '%'" />
                and job like #{patternJob}
            </if>
            <if test="salMin != null">
                and sal <![CDATA[>=]]> #{salMin}
            </if>
            <if test="salMax != null">
                and sal <![CDATA[<=]]> #{salMax}
            </if>
        </trim>
    </select>

</mapper>
```

EmpDaoImplTest.java

```
package com.example.demo.dao;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertThat;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.transaction.annotation.Transactional;

import com.example.demo.domain.Emp;

@RunWith(SpringRunner.class)
@SpringBootTest
public class EmpDaoImplTest {
    @Autowired
    private EmpDao dao;

    @Test
    public void testInsert() {
        int oldCount = dao.count();
```

```
        // 가짜 테스트 데이터 객체, PK인 empno 값은 설정하지 않는다.
        // empno 값은 디비가 자동으로 제너레이트 해서 넣을 것이다.
        Emp emp = new Emp();
        emp.setEname("Tom");
        emp.setJob("Actor");
        emp.setSal(999);

        System.out.println("전: " + emp); // empno 값은 0인 상태

        int affected = dao.insert(emp);
        System.out.println("insert affected = " + affected);

        System.out.println("후: " + emp); // empno 값은 디비가 자동으로 제너레이트 값!

        int nowCount = dao.count();

        assertThat("한 행이 추가되었으므로 nowCount는 oldCount 보다 1이 커야 합니다.",
                nowCount, is(oldCount + 1));

        // 읽기 좋은 코드가 최선의 코드다. == 이해가 쉽다. == 협력 작업 시 유리하다.
        // 따라서, 최대한 읽기 좋게 작성하고 싶다. 매처 라이브러리가 지원하는 메소드를 사용하는 이유가 된다.
        // (I) assert that nowCount is oldCount + 1.
    }

    @Test
    public void testUpdate() {
        Emp emp = new Emp();
        emp.setEname("Tom");
        emp.setJob("Actor");
        emp.setSal(999);

        System.out.println("입력 전: " + emp);

        // 수정 대상을 입력한다.
        int affected = dao.insert(emp);
        System.out.println("insert affected = " + affected);

        System.out.println("입력 후/수정 전: " + emp);

        emp.setJob("Developer");
        emp.setSal(2000);

        // 수정 메소드를 테스트한다.
        affected = dao.update(emp);
        System.out.println("update affected = " + affected);

        // 수정된 데이터를 확인하기 위해서 조회한다.
        Emp e = dao.findOne(emp.getEmpno());

        System.out.println("수정 후: " + e);

        assertThat(e.getJob(), is(emp.getJob()));
        assertThat(e.getSal(), is(emp.getSal()));

        // 테스트 시 사용한 로우를 삭제한다.
        // 대신 @Transactional 애노테이션을 @Test와 같이 사용해도 된다.
        affected = dao.delete(emp.getEmpno());
        System.out.println("delete affected = " + affected);
    }

    @Test
    public void testDelete() {
        int oldCount = dao.count();
```

```
        Emp emp = new Emp();
        emp.setEname("Tom");
        emp.setJob("Actor");
        emp.setSal(999);

        // 삭제 대상을 입력한다.
        int affected = dao.insert(emp);
        System.out.println("insert affected = " + affected);

        /// 삭제 메소드를 테스트한다.
        affected = dao.delete(emp.getEmpno());
        System.out.println("delete affected = " + affected);

        int nowCount = dao.count();

        assertEquals(nowCount, oldCount);
    }

    @Test
    public void testFindAll() {
        System.out.println(dao instanceof EmpDaoImpl);

        List<Emp> emps = dao.findAll();
        emps.forEach(System.out::println);

        assertEquals(emps.size(), dao.count());
    }

    @Test
    public void testCount() {
        int count = dao.count();
        System.out.println("count = " + count);

        assertEquals(count, dao.findAll().size());
    }

    @Transactional
    @Test
    public void testFindOne() {
        Emp emp = new Emp();
        emp.setEname("Tom");
        emp.setJob("Actor");
        emp.setSal(999);

        // 조회 대상을 입력한다.
        int affected = dao.insert(emp);
        System.out.println("insert affected = " + affected);

        Emp e = dao.findOne(emp.getEmpno());
        System.out.println(e);

        assertThat(e.getEname(), is(emp.getEname()));
        assertThat(e.getJob(), is(emp.getJob()));
        assertThat(e.getSal(), is(emp.getSal()));
    }

    @Test
    public void testFindBySkipLimit() {
        int skip = 10;
        int limit = 10;
        List<Emp> emps = dao.findBySkipLimit(skip, limit);
        emps.forEach(System.out::println);
    }
```

```
    @Test
    public void testFindByPageSize() {
        int page = 2;
        int size = 10;
        List<Emp> emps = dao.findByPageSize(page, size);
        emps.forEach(System.out::println);
    }

    @Test
    public void testFindByStartEnd() {
        int start = 11;
        int end = 20;
        List<Emp> emps = dao.findByStartEnd(start, end);
        emps.forEach(System.out::println);
    }

    @Test
    public void testFindByPageSizeUsingBind() {
        int page = 2;
        int size = 10;
        List<Emp> emps = dao.findByPageSizeUsingBind(page, size);
        emps.forEach(System.out::println);
    }

    @Test
    public void testSearch() {
        Map<String, String> map = new HashMap<>();
        map.put("ename", "일");
        map.put("job", "적");
        map.put("salMin", "200");
        map.put("salMax", "700");

        List<Emp> emps = dao.search(map);
        System.out.println(emps.size());
        emps.forEach(System.out::println);
    }
}
```

마이바티스가 디비에게 질의하는 SQL 쿼리문을 로깅으로 확인하고 싶다면 `부록 E. Logback을 이용한 SQL 로깅` 부분을 살펴보자.

EmpController.java

```
package com.example.demo.controller;

import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import com.example.demo.dao.EmpDao;
import com.example.demo.domain.Emp;

@Controller
```

```
@RequestMapping("/emps")
public class EmpController {
    @Autowired
    private EmpDao empDao;

    @GetMapping
    public String getSearchView() {
        return "search";
    }

    @PostMapping
    public Object postOne(
            @RequestParam Map<String, String> map, Model model) {

        // 브라우저는 검색화면에서 사용자가 직업란의 검색어를 입력하지 않아도
        // job="" 형태로 파라미터를 전달한다.
        // 빈 문자열은 null이 아니므로 주의가 필요하다.
        System.out.println(map);
        // 예: {ename=길동, job=, salMin=, salMax=}

        List<Emp> emps = empDao.search(map);
        model.addAttribute("emps", emps);

        System.out.println(map);
        // 예: {ename=길동, job=null, salMin=null, salMax=null}

        model.addAttribute("conditions", searchConditionToHtml(map));

        return "search";
    }

    private String searchConditionToHtml(Map<String, String> map) {
        StringBuffer sb = new StringBuffer();
        sb.append("검색조건: ");
        if (map.get("ename") != null) {
            sb.append("<mark>Name=" + map.get("ename") + "</mark> ");
        }
        if (map.get("job") != null) {
            sb.append("<mark>Job=" + map.get("job") + "</mark> ");
        }
        if (map.get("salMin") != null) {
            sb.append("<mark>Salary Min=" + map.get("salMin") + "</mark> ");
        }
        if (map.get("salMax") != null) {
            sb.append("<mark>Salary Max=" + map.get("salMax") + "</mark> ");
        }
        if ("검색조건: ".equals(sb.toString())) {
            sb.append("<mark>ALL</mark>");
        }
        return sb.toString();
    }
}
```

search.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
```

```
<title></title>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
<style type="text/css">
    .bs-example{
        margin: 20px;
    }
</style>
</head>
<body>
  <div class="bs-example">
    <h3>Employee's Search</h3>
    <form class="" action="" method="post">
      <table class="table table-hover">
        <tr>
          <th>Name</th>
          <td>
            <input type="text" name="ename" placeholder="직원명을 입력하세요">
          </td>
        </tr>
        <tr>
          <th>Job</th>
          <td>
            <input type="text" name="job">
          </td>
        </tr>
        <tr>
          <th>Salary</th>
          <td>
            <input type="number" name="salMin" min="0">
            ~
            <input type="number" name="salMax" min="0">
          </td>
        </tr>
        <tr>
          <td colspan="2">
            <button type="submit">Search</button>
          </td>
        </tr>
      </table>
    </form>
    <hr>
    <p>${conditions }</p>
    <table class="table table-hover">
      <thead>
        <tr>
          <th>No</th>
          <th>Name</th>
          <th>Job</th>
          <th>Salary</th>
        </tr>
      </thead>
      <tbody>
      <c:forEach var="e" items="${emps }">
        <tr>
          <td>${e.empno }</td>
          <td>${e.ename }</td>
          <td>${e.job }</td>
          <td>${e.sal }</td>
        </tr>
      </c:forEach>
      </tbody>
```

```
    </table>
  </div>
</body>
</html>
```

브라우저를 띄우고 `http://localhost:8080/emps` 주소를 사용하여 테스트를 진행 해 보자.