
6.1. Dynamic SQL

마이바티스의 가장 강력한 기능 중 하나는 동적 **SQL**을 처리하는 방법이다. **JDBC**나 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 **SQL**은 그만큼 어려운 것이다. 마이바티스는 강력한 동적 **SQL** 언어로 이 상황을 개선한다.

if

동적 **SQL**에서 가장 공통적으로 사용되는 것으로 **where**의 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>
  </choose>
```

```
</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 콘텐츠가 리턴되면 단순히 **"WHERE"**만을 추가한다. 게다가 콘텐츠가 **"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>
</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 **Map**이나 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. **Map**을 사용할때 **index**는 **key**객체가 되고 항목은 **value**객체가 된다.

bind

bind 엘리먼트는 **OGNL**표현을 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. '%' 기호는 쿼리문에서 **LIKE** 키워드 다음에서 사용하는 데이터베이스가 선언한 예약기호이다.

Multi-db vendor support

"**_databaseId**" 변수로 설정된 **databaseIdProvider**가 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
```