MyBatis

1. MyBatis 기본

1) MyBatis 란?

- 개발자들이 SQL 쿼리를 쉽게 작성하고 관리할 수 있도록 도와주는 퍼시스턴스 프레임워크입니다.
- 이를 사용하여 SQL 쿼리와 프로그래밍 코드를 분리하여 관리할 수 있어서 코드의 가독성과 재사용성을 높이고 유지 보수를 용이하게 합니다.
- 또한 JDBC 코드의 복잡성을 추상화하고 SQL 쿼리의 실행 결과와 자바 객체에 매핑하기 위한 강력한 매핑 기능을 제공합니다.

2) CDATA(Character Data Section)

- SQL 쿼리에 특수 문자를 안전하게 삽입하기 위해 사용됩니다. 해당 안에 포함된 내용은 XML 파서에 의해 파싱 되지 않습니다.
- 비교 연산자를 위해 '<', '>' 기호를 사용하는 경우 기호들은 XML 문법에서 태그를 나타내므로 XML 문서내에서는 오류를 일으킬 수 있습니다.
- 이를 해결하기 위해 SQL 쿼리를 CDATA 섹션 안에 넣어 기호를 안전하게 쿼리에 포함시킬 수 있습니다.

① SQL 전체에 CDATA 사용

- SQL 쿼리에 특수 문자를 안전하게 삽입하기 위한 방법입니다. 이 방법에서는 SQL 쿼리 전체를 <![CDATA[ ]]>로 감싸게 됩니다.
- CDATA 섹션 내에 포함된 내용은 XML 파서에 의해 파싱되지 않습니다.

```
<select id="selectPerson" parameterType="int" resultType="hashmap">
        <![CDATA[
          SELECT * FROM PERSON WHERE AGE < #{age}
        ]]>
</select>
```

② 연산자만 CDATA 사용

- SQL 쿼리에서 특정 연산자에 특수 문자를 안전하게 삽입하기 위한 방법입니다. 이 방법에서는 연산자 부분만 CDATA 섹션(<![CDATA[ ]]>)으로 감싸게 됩니다.

- CDATA 섹션 내에 포함된 내용은 XML 파서에 의해 파싱되지 않습니다. 따라서, 이 방법은 SQL 쿼리의 일부분에 특수 문자를 포함시키고자 할 때 유용하게 사용될 수 있습니다.

```
<select id="selectPerson" parameterType="int" resultType="hashmap">
    SELECT *
    FROM PERSON
    WHERE AGE <![CDATA[ < ]]> #{age}
</select>
```

3) 비교 연산자

- MyBatis 태그 중 <if>, <choose>, <when>, <otherwise>, <where>, <set>, <foreach>, <bind>와 같은 태그 안에서 test 속성에서 사용이 됩니다.

- 이러한 연산자들은 SQL 쿼리의 동적 생성을 위해 사용되며 특히 조건문이나 반복문 등에서 유용하게 활용됩니다.

3.1) 동등 비교 연산자

- 두 값이 같은지를 비교하고자 할 때 사용하는 연산자입니다.
- MyBatis에서는 ==, eq, equals, equalsIgnoreCase 네 가지의 동등 비교 연산자를 제공하며, 이들은 서로 다른 상황에서 사용됩니다.

- == 와 eq 는 두 값이 동일한지를 비교합니다.
- equals 는 두 객체가 동일한 객체인지를 비교합니다.
- equalsIgnoreCase는 대소문자를 구분하지 않고 두 문자열이 동일한지를 비교합니다.

MyBatis 연산자	비교	설명
==	A == B	동등 비교
eq	A eq B	동등 비교
equals	A.equals(B)	동등 비교(동일한 객체 비교)
equalsIgnoreCase	A.equalsIgnoreCase(B)	동등 비교(대소문자 구분 없이 비교)

```
<!-- 사용자 정보를 조회합니다 -->
<select id="selectUser" resultType="userDto">
    SELECT *
    FROM tb_user
    <if test="username != null">
        WHERE username = #{username}
    </if>
    <if test="email eq null">
        AND email = #{email}
    </if>
    <if test="password.equals(null)">
        AND password = #{password}
    </if>
    <if test="!firstName.equalsIgnoreCase(null)">
        AND first_name = #{firstName}
    </if>
</select>
```

3.2) 불일치 비교 연산자

- 두 값이 같지 않은지를 비교하는 연산자입니다.
- MyBatis에서는 !=, neq, !equals, !equalsIgnoreCase 네 가지의 불일치 비교 연산자를 제공하며, 이들은 서로 다른 상황에서 사용됩니다.

- != 와 neq는 두 값이 동일하지 않은지를 비교합니다.
- !equals 는 두 객체가 동일한 객체가 아닌지를 비교합니다.
- !equalsIgnoreCase는 대소문자를 구분하지 않고 두 문자열이 동일하지 않은지를 비교합니다.

MyBatis 연산자	비교	설명
!=	A != B	같지 않음
neq	A neq B	같지 않음
!equals	!A.equals(B)	같지 않음(동일한 객체 비교)
!equalsIgnoreCase	!A.equalsIgnoreCase(B)	같지 않음(대소문자 구분 없이 비교)

```
<!-- 전체 코드를 조회합니다-->
<select id="selectCodeList" resultType="codeDto">
    SELECT t1.*
    FROM tb_code t1
    WHERE t1.use_yn = true
    <if test="grpCd != null">
        AND t1.grp_cd = #{grpCd}
    </if>
                <if test="cd ne null">
        AND t1.cd = #{cd}
    </if>
                <if test="!grpCdNm.equals(null)">
        AND t1.grp_cd_nm = #{grpCdNm}
    </if>
                <if test="!cdNm.equalsIgnoreCase(null)">
        AND t1.cd_nm = #{cdNm}
    </if>
</select>
```

3.3) 논리 연산자

- 주로 조건식에서 두 개 이상의 조건을 결합시키는 데 사용되며, 결과는 참(True) 또는 거짓(False)입니다.
- MyBatis에서 지원하는 논리 연산자에는 '||' (또는 'or')와 '&&' (또는 'and')가 있습니다.

- '||' 또는 'or' 연산자는 두 조건 중 하나 이상이 참일 경우 전체 결과를 참으로 반환합니다.
- '&&' 또는 'and' 연산자는 두 조건 모두 참일 경우에만 전체 결과를 참으로 반환합니다.

```
<!-- 사용자 정보를 조회합니다 -->
<select id="selectUser" resultType="userDto">
    SELECT *
    FROM tb_user
    <if test="username != null and email != null">
        WHERE username = #{username} and email = #{email}
    </if>
    <if test="username != null or email != null">
        WHERE username = #{username} or email = #{email}
    </if>
</select>
```

3.4). 대소 비교 연산자

- 한 값이 다른 값보다 크거나 작은지, 또는 동일한지를 확인합니다.
- 이 연산자들은 XML 문법 오류를 발생시킬 수 있으므로, 이스케이프 처리(<, >, &lte;, &gte;)를 사용하여 안전하게 코드를 작성합니다.

💡 사용자의 나이를 기준으로 정보를 조회하는 SQL 쿼리를 예시로 구성하였습니다.

- 첫 번째 <if> 태그에서는 사용자의 나이가 쿼리에 입력된 나이보다 작거나 같은 모든 사용자를 조회하며, 두 번째 <if> 태그에서는 사용자의 나이가 쿼리에 입력된 나이보다 큰 모든 사용자를 조회합니다.

```
<!-- 사용자의 나이를 기준으로 정보를 조회합니다 -->
<select id="selectUser" resultType="userDto">
    SELECT *
    FROM tb_user
    <if test="age != null">
        WHERE age &amp;lt; #{age}
```

```
    </if>
    <if test="age != null">
        WHERE age &gt; #{age}
    </if>
</select>
```

3.5) 사칙 연산자

- 더하기(+), 빼기(-), 곱하기(*), 나누기(/)와 같은 기본적인 수학 연산을 수행합니다.
- MyBatis에서는 SQL 쿼리 내에서 사용할 수 있으며, 변수 또는 상수값 간의 연산에 사용됩니다.

```
<!-- 사용자의 나이를 기준으로 정보를 조회합니다 -->
<select id="selectUser" resultType="userDto">
    SELECT *
    FROM tb_user
    <if test="age != null">
        WHERE age = #{age} + 10
    </if>
</select>

<!-- 사용자의 나이를 기준으로 정보를 조회합니다 -->
<select id="selectUser" resultType="userDto">
    SELECT *
    FROM tb_user
    <if test="age != null">
        WHERE age = #{age} - 5
    </if>
</select>

<!-- 사용자의 나이를 기준으로 정보를 조회합니다 -->
<select id="selectUser" resultType="userDto">
    SELECT *
    FROM tb_user
    <if test="age != null">
        WHERE age = #{age} * 2
    </if>
</select>
```

```
<!-- 사용자의 나이를 기준으로 정보를 조회합니다 -->
<select id="selectUser" resultType="userDto">
    SELECT *
    FROM tb_user
    <if test="age != null">
        WHERE age = #{age} / 2
    </if>
</select>
```

2. MyBatis 공통 파라미터 종류

1) #{parameter}

- MyBatis에서 SQL 문에 파라미터를 바인딩하는 방법을 의미합니다. 이 방법은 PreparedStatement를 사용하여 SQL쿼리 내에서 안전하게 파라미터를 입력할 수 있습니다.

- 또한 이는 입력값을 문자열로 변환하지 않고, SQL이 실행되기 전에 안전하게 값을 대입하여 SQL Injection 공격에 안전합니다.

```
<!--사용자 리스트를 조회합니다.-->
<select id="selectUserList" parameterType="UserDto" resultType="UserDto">
    SELECT t1.*
    FROM TB_USER t1
    WHERE t1.delYn = 0
    AND t1.user_name = #{userName}
</select>
```

[더 알아보기]

💡 PreparedStatement

- SQL을 실행하기 전에 미리 컴파일하여 실행계획을 준비해 놓는 방식입니다. 이를 통해 동일한 SQL을 반복적으로 실행할 때, 성능을 향상할 수 있습니다.
- 또한, SQL Injection 공격을 방지하기 위해 사용됩니다. 사용자 입력을 이용해 쿼리를 작성할 때, #{}를 이용해 PreparedStatement를 생성하면, 입력값이 SQL 구문에 영향을 미치지 않고 안전하게 쿼리를 실행할 수 있습니다.

2) select 태그 속성

① select 태그 속성 종류

parameter 속성	설명
jdbcType	JDBC에서 사용할 SQL 타입을 지정합니다.
javaType	Java에서 사용할 타입을 지정합니다. (기본자료형, 래퍼클래스, 패키지 명, 커스텀 패키지 명을 사용이 가능합니다)
mode	프로시저를 위한 IN, OUT, INOUT 파라미터를 지정합니다.
property	SQL 문에서 참조할 때 사용할 속성 이름을 지정합니다.
jdbcTypeName	JDBC 타입의 이름을 지정합니다.
typeHandler	특정 타입의 핸들러를 지정합니다.
numericScale	BigDecimal 객체를 사용할 때 필요한 스케일을 지정합니다.
resultMap	JOIN이나 복잡한 SQL 쿼리 결과를 매핑하기 위해 사용되는 resultMap의 id를 지정합니다.

[더 알아보기]

💡 select 태그에서 parameterType 속성을 안 적는 경우가 많은데 왜 그럴까?
- select에서 parameterType은 필수가 아닌 선택적 속성입니다. 이는 MyBatis가 파라미터를 자동으로 매핑하고, 자바의 오버로딩 규칙을 따라서 파라미터 타입을 결정하기 때문입니다.
- SQL 쿼리에서 파라미터를 사용하지 않거나, MyBatis가 자동으로 파라미터 타입을 추론할 수 있는 경우 parameterType을 명시적으로 지정할 필요가 없습니다.

3) select 태그의 속성 사용 예시 -1: parameterType, resultType

- 일반적으로 가장 많이 사용하는 select 태그의 속성들에 대해 확인해 봅니다.
- parameterType : 파라미터로 전달될 데이터 타입 유형이며 해당 속성은 선택 사항이며 해당 속성을 사용하지 않더라도 자동으로 매핑됩니다.
- resultType : 결과값을 반환될 타입 유형

```
<select id="selectCodeList" parameterType="codeDto" resultType="codeDto" >
    SELECT t1.*
    FROM multiflex_scma.tb_code t1
    WHERE t1.use_yn = true
    <if test="grpCd != null">
        AND t1.grp_cd = #{grpCd}
    </if>
</select>
```

💡 [더 알아보기]

💡 해당 코드에서 codeDto라고만 적었는데 어떻게 매핑이 될까?

- MyBatis의 설정을 구성하는 DBConfig 파일 내에서 setTypeAliasesPackage로 매핑하였기에 기본적으로 model 패키지 아래 구성한 DTO를 보고 있습니다.

4) insert, update, delete 태그의 속성 기본 사용예시 -1 : parameterType

- 해당 예시에서는 parameterType만 명시하여 전달하려는 파라미터 객체나 타입을 지정합니다.

```
<insert id="insertCode" parameterType="codeDto">
    INSERT INTO tb_code(grp_cd, cd, grp_cd_nm, cd_nm, sort_order, reg_dt, use_yn)
    values ( #{grpCd}
            , #{cd}
            , #{grpCdNm}
            , #{cdNm}
            , #{sortOrder}
            , #{regDt}
            , #{useYn})
</insert>

<update id="updateCode" parameterType="codeDto">
    UPDATE tb_code
    SET cd_nm = #{cdNm}
    WHERE cd = #{cd}
</update>
```

```
<update id="deleteCode" parameterType="codeDto">
    UPDATE tb_code
    WHERE cd = #{cd}
</update>
```

[더 알아보기]

- MyBatis에서 insert, update, delete 작업은 왜 결과값을 반환하지 않은가?
  - 특정 '타입'의 객체를 반환하지 않기 때문에 resultType 속성을 지정할 필요가 없습니다. 이들 작업은 대신 성공적으로 수행된 행의 수를 반환합니다.

5) insert 태그의 속성 사용 예시 : useGeneratedKeys, keyPropert

- 해당 예시에서는 insert를 수행한 이후 Auto Increment 된 키 값을 다시 반환하기 위함입니다

```
-useGeneratedKeys:자동 생성된 키를 사용할지 여부
- keyProperty: 자동 생성된 키를 설정할 속성
<insert id="insertUser" useGeneratedKeys="true" keyProperty="userSq">
    INSERT INTO tb_user(user_id, user_pw, user_nm, user_st)
    VALUES (#{userId}, #{userPw}, #{userNm}, #{userSt})
</insert>
```

- insert 태그의 속성 사용 예시 -2: selectKey
  - insert 구문에 selectKey 태그를 포함하여서 insert의 트랜잭션 시작 전 후에 바로 수행할 수 있도록 구성할 수 있습니다.

```
<insert id="insertUser" parameterType="userDto">
    <selectKey keyProperty="userSq" order="AFTER" resultType="int">
        SELECT LAST_VALUE FROM public.tb_user_user_sq_seq
    </selectKey>
    INSERT INTO tb_user(user_id, user_pw, user_nm, user_st)
    VALUES (#{userId}, #{userPw}, #{userNm}, #{userSt})
</insert>
```

3. 동적 SQL(Dynamic SQL)

- 프로그램 실행 중에 SQL 문장이 생성되는 방식을 의미합니다. 즉 프로그램이 실행되는 동안 SQL 문장의 구조나 조건이 변경되는 것을 의미합니다.
- 이렇게 하면 프로그램이 유연해지고 사용자의 입력이나 프로그램의 상태에 따라 데이터베이스 쿼리를 동적으로 조정할 수 있습니다.

1) if

- 특정 조건에 충족될 때만 SQL문이 수행되도록 하는 기능을 제공하는 태그를 의미합니다.

속성	필수여부	설명
test	필수	조건을 검사하는데 사용됩니다. test 속성의 조건 값이 참이라면 if 블록 내의 SQL이 실행됩니다.

💡 기본구조

- 기본 구조는 "조건" 부분에는 참 또는 거짓을 결정하는 조건이 들어갑니다. 이 조건이 참일 때만 "실행할 SQL 구문"이 실행됩니다.

```
<if test="조건">
  실행할 SQL 구문
</if>
```

💡 사용예시

- 해당 예시에서는 tb_user 테이블을 전체 조회하는 SQL문에서 WHERE 구문을 조건부 수행하는 구문입니다.
- 해당 예시에서는 파라미터로 userId, userName, userEmail, userPw 값이 null이 아니고 빈 값이 아닌 경우에 따라서 조건부 수행이 됩니다.
- 각각 값이 유효하다면 WHERE 조건절의 다음 조건문으로 수행이 됩니다.

```
<!-- 사용자 정보를 조회합니다 -->
<select id="selectUser" resultType="userDto">
    SELECT *
    FROM tb_user
    WHERE 1=1
    <if test="userId != null and !userId.equals('')">
        AND user_id = #{userId}
    </if>
    <if test="userName != null and !userName.equals('')">
        AND user_name = #{userName}
    </if>
    <if test="userEmail != null and !userEmail.equals('')">
        AND user_email = #{userEmail}
    </if>
    <if test="userPw != null and !userPw.equals('')">
        AND user_pw = #{userPw}
    </if>
</select>
```

2) choose (when, otherwise)

- Java에서 Switch문과 같은 기능을 수행하는 태그입니다. choose 태그 안에는 when과 otherwise 태그를 추가하며 when 만으로 구성이 될 수 있거나 otherwise가 추가된 태그로 구성이 될 수 있습니다.
- 'otherwise' 태그의 경우 'when' 태그의 모든 조건이 충족되지 않을 경우 해당 태그의 내부 SQL문이 실행이 됩니다.

조건문	설명
choose	SQL의 CASE문과 비슷한 역할을 수행하는것 처럼 여러 개의 when 중에서 조건에 맞는 하나를 선택하여 실행하는 기능을 제공합니다.
when	choose 내에서 사용이 되며 주어진 조건이 참일 때 실행이 됩니다. 또한, 여러개의 when이 있는 경우 맨 처음 참인 조건을 갖는 when만 실행합니다.
otherwise	choose 내에서 사용되며 어떠한 when도 참이 아닐 때 실행이 됩니다. 즉, when 조건이 모두 거짓일 경우의 기본 동작합니다.

💡 기본구조

- choose 구문내에서 when ~ otherwise가 수행이 됩니다.
- 조건1이 참이면 '실행할 SQL 구문 1'이 실행되고 조건 2가 참이면 '실행할 SQL 구문 2'가 실행됩니다. 만약 어떠한 조건도 충족되지 않으면 "기본 실행 SQL 구문"이 실행됩니다.
- <choose>: 여러 개의 when 중에서 조건에 맞는 하나를 선택하여 실행하는 구조를 제공합니다.
- <when test="조건">: 특정 조건이 충족될 때 수행되는 SQL 구문입니다. 여러 개의 when 구문이 있을 수 있으며, 처음으로 참인 조건을 갖는 when 구문만 실행됩니다.
- <otherwise>: when 구문들 중 어떤 것도 충족되지 않을 때 실행되는 SQL 구문입니다. 즉, 모든 when 조건이 거짓일 경우 수행됩니다.

```
<choose>
    <when test="조건1">
        실행할 SQL 구문1
    </when>
    <when test="조건2">
        실행할 SQL 구문2
    </when>
    <otherwise>
        기본 실행 SQL 구문
    </otherwise>
</choose>
```

사용예시

- 해당 예시에서는 tb_user 테이블을 전체 조회하는 SQL문에서 ORDER BY 구문을 조건에 따라 동적 SQL문으로 처리하는 방법을 이용한 예시입니다.
- 해당 SQL문에서 파라미터로 기본적으로 'cd' 값이 null이 아닌 상태에 값이 'java' 혹은 'kotlin'의 경우에 각각에 맞는 정렬을 수행하는 예시입니다. 또한 해당 경우에 맞지 않는 경우는 다른 정렬을 수행합니다.

```
<!-- 전체 코드를 조회합니다-->
<select id="selectCodeList" resultType="codeDto">
    SELECT t1.*
    FROM tb_code t1
    WHERE t1.use_yn = true
    <choose>
        <when test="cd != null and cd == 'java'">
            ORDER BY cd DESC
        </when>
        <when test="cd != null and cd == kotlin">
            ORDER BY cd ASC
        </when>
        <otherwise>
            ORDER BY sort_order DESC
        </otherwise>
    </choose>
</select>
```

3) trim

- SQL 쿼리의 일부를 동적으로 제어하기 위한 기능입니다. 해당 태그로 감싸 있는 경우 조건에 만족하는 내용이 없다면 trim 태그는 수행되지 않습니다.
- trim 태그는 where, set 등의 예약어를 사용하는 경우에 이를 조건에 따라 동적으로 적용할 수 있게 해 줍니다.

속성	설명
prefix	trim 태그가 선언된 SQL의 블록 맨 앞에 추가되는 접두사입니다.
prefixOverrides	trim 태그가 선언된 SQL의 블록 앞에서 제거해야 하는 문자열을 지정합니다.
suffix	trim 태그가 선언된 SQL의 블록 맨 뒤에 추가되는 접미사입니다.
suffixOverrides	trim 태그가 선언된 SQL의 블록 뒤에서 제거해야 하는 문자열을 지정합니다.

```
<trim prefix="접두사" prefixOverrides="접두사 제거" suffix="접미사" suffixOverrides="접미사 제거">
    SQL 구문
</trim>
```

3.1) Trim 속성 사용예시 :prefix

- 해당 구문은 tb_code 테이블의 컬럼들을 동적으로 변경하는 SQL문입니다.
- 해당 구문에서는 trim 태그 내의 if 태그의 조건이 하나라도 만족한다면 prefix="SET" 속성으로 SET 구문을 생성하며, suffixOverrides 속성을 통해 마지막 ","를 제거합니다.
- 단 해당 구문에서 하나라도 수정하는 컬럼이 없다면 trim 태그는 생성되지 않고 오류가 발생할 수 있습니다.

```
<!--코드를 수정합니다.-->
<update id="updateCode" parameterType="codeDto">
    UPDATE tb_code
    <trim prefix="SET" suffixOverrides=",">
        <if test="cd != null">cd = #{cd},</if>
        <if test="grpCdNm != null">grp_cd_nm = #{grpCdNm},</if>
        <if test="cdNm != null">cd_nm = #{cdNm},</if>
        <if test="sortOrder != null">sort_order= #{sortOrder},</if>
        <if test="useYn != null">use_yn= #{useYn},</if>
    </trim>
    WHERE grp_cd = #{grpCd}
</update>
```

3.2) Trim 속성 사용예시: prefixOverrides

- 해당 예시에서는 사용자 정보를 조건부 조회하는 SQL문입니다.
- 해당 부분에서는 trim 태그를 사용하여 if 태그의 조건이 하나라도 존재한다면 AND 구문을 제거하고, 존재하지 않으면 생성하지 않도록 하는 구문을 구성하였습니다.

```
<!-- 사용자 정보를 조회합니다 -->
<select id="selectUser" resultType="userDto">
    SELECT *
    FROM tb_user
    WHERE
    <trim prefixOverrides="AND ">
        <if test="userId != null">AND user_id = #{userId}</if>
        <if test="userName != null">AND userName = #{userName}</if>
    </trim>
</select>
```

[참고]

- 그러면 trim 내에서 두 개 모두 만족하는 경우 AND라는 값이 모두 사라지는가?
  - 만약 'userId'와 'userName' 조건이 동시에 만족된다면, 첫 번째 조건 'userId' 앞의 'AND '는 제거되지만, 두 번째 조건 'uerName' 앞의 'AND '는 그대로 유지됩니다.
  - 따라서 두 조건이 모두 만족되면 SQL 구문은 WHERE user_id = #{userId} AND userName = #{userName}가 됩니다.

3.3) Trim 속성 사용예시: suffix

- 해당 예시에서는 tb_user 테이블의 컬럼을 채우는 INSERT 구문입니다.
- INSERT 구문에서도 일부 값만 동적으로 추가하는 경우 trim 태그를 이용하여서 전달받은 파라미터 값에 따라 동적으로 컬럼 값을 추가합니다.
- 단 해당 구문에서 하나라도 추가하는 컬럼이 없다면 trim 태그는 생성되지 않고 오류가 발생할 수 있습니다.

```
<insert id="insertUser">
  insert into tb_user
  <trim prefix="(" suffix=")" suffixOverrides=",">
    <if test="id != null">id,</if>
    <if test="username != null">username,</if>
    <if test="password != null">password,</if>
```

```
    <if test="email != null">email,</if>
    <if test="bio != null">bio</if>
  </trim>
  values
  <trim prefix="(" suffix=")" suffixOverrides=",">
    <if test="id != null">#{id},</if>
    <if test="username != null">#{username},</if>
    <if test="password != null">#{password},</if>
    <if test="email != null">#{email},</if>
    <if test="bio != null">#{bio}</if>
  </trim>
</insert>
```

3.4) Trim 속성 사용예시: suffixOverrides

- 해당 쿼리는 tb_user 테이블을 조회해 오는 SQL문입니다.
- 해당 SQL문은 동적으로 컬럼과 조건절을 추가하여 가져 올 수 있습니다.

```
<select id="selectUser">
  select
  <trim suffixOverrides=",">
    <if test="id != null">id,</if>
    <if test="username != null">username,</if>
    <if test="password != null">password,</if>
    <if test="email != null">email,</if>
    <if test="bio != null">bio</if>
  </trim>
  from tb_user
  <trim prefix="WHERE" prefixOverrides="AND |OR ">
    <if test="id != null">AND id = #{id}</if>
    <if test="username != null">AND username = #{username}</if>
    <if test="password != null">AND password = #{password}</if>
    <if test="email != null">AND email = #{email}</if>
    <if test="bio != null">AND bio = #{bio}</if>
  </trim>
</select>
```

[더 알아보기]

💡 prefixOverrides="AND |OR " 속성은 무슨 의미 일까?

- SQL 구문의 시작 부분에서 "AND " 또는 "OR " 문자열을 제거하라는 것입니다. 이를 통해 SQL 구문이 "AND " 또는 "OR "로 시작하는 것을 방지할 수 있습니다.

4) where

- SQL의 WHERE 절을 동적으로 만들어주는 구문입니다.
- WHERE절 내의 만족하는 조건이 없다면 WHERE 절을 만들지 않고 조건이 하나라도 있다면 WHERE 절을 만들어 줍니다.
- WHERE절 조건 앞에 있는 AND나 OR을 자동으로 제거해 주는 기능도 제공합니다.

💡 기본 구조

- WHERE절을 동적으로 만들어주는 구문으로 where 태그 내에 조건이 참인 경우에만 해당 조건을 SQL문장에 포함시킵니다.

```
<where>
    <if test="조건1">
        AND 실행할 SQL 구문1
    </if>
    <if test="조건2">
        AND 실행할 SQL 구문2
    </if>
</where>
```

💡 사용예시

- 조건부 코드 리스트를 조회하는 SQL문입니다.
- 해당 부분에서는 where 태그로 동적 SQL문을 구성합니다.
- if 태그의 조건을 하나라도 만족하는 경우 WHERE 절이 생성이 되며 첫 번째 마주한 SQL 문에서는 AND / OR을 제거하고 WHERE절이 생성이 됩니다.

```
<!--코드를 조회합니다-->
<select id="selectCode" resultType="codeDto">
    SELECT t1.grp_cd
    , t1.cd
    , t1.grp_cd_nm
    , t1.cd_nm
    FROM public.tb_code t1
    <where>
```

```
        <if test="cd != null">
            AND cd = #{cd}
        </if>
        <if test="grpCdNm != null">
            AND grp_cd_nm = #{grpCdNm}
        </if>
        <if test="cdNm != null">
            AND cd_nm = #{cdNm}
        </if>
        <if test="sortOrder != null">
            AND sort_order= #{sortOrder}
        </if>
        <if test="useYn != null">
            AND use_yn= #{useYn}
        </if>
    </where>
```

5) set

- SQL의 SET 절을 동적으로 만들어주는 구문입니다.
- SET절 내부에 만족하는 조건이 없다면 SET 절을 만들지 않고 조건이 하나라도 있다면 SET절을 만들어줍니다.
- SET절 내에 마지막 조건 뒤에 붙는 쉼표를 자동으로 제공하는 기능도 제공합니다.

기본구조

- SET 절을 동적으로 만들어주는 구문으로 set 태그 내에 조건이 참인 경우에만 해당 조건을 SQL문장에 포함시킵니다.
- 만약 조건이 하나도 만족하지 않는다면 SET절 자체가 생성되지 않으며 마지막 조건 뒤에 붙는 쉼표를 자동으로 제거해 줍니다.

```
<set>
    <if test="조건1">
        실행할 SQL 구문1,
    </if>
    <if test="조건2">
        실행할 SQL 구문2,
    </if>
```

```
</set>
```

💡 사용예시

- tb_code 테이블의 컬럼을 수정하는 update문입니다.
- 해당 부분에서 set 태그를 두어서 set 태그 내의 조건이 만족하는 경우 set을 생성하고 존재하지 않으면 생성하지 않습니다.
- 또한 set 태그의 기능으로 구문의 마지막 쉼표를 제거해 주는 기능을 제공합니다.

```
<!--코드를 수정합니다.-->
<update id="updateCode" parameterType="codeDto">
    UPDATE tb_code
    <set>
        <if test="cd != null">cd = #{cd},</if>
        <if test="grpCdNm != null">grp_cd_nm = #{grpCdNm},</if>
        <if test="cdNm != null">cd_nm = #{cdNm},</if>
        <if test="sortOrder != null">sort_order= #{sortOrder},</if>
        <if test="useYn != null">use_yn= #{useYn},</if>
    </set>
    WHERE grp_cd = #{grpCd}
</update>
```

6) foreach

- 반복적인 구문을 처리할 때 사용되는 구문입니다.
- SQL의 IN 연산자와 같이 동적인 값들을 처리할 때 유용합니다.
- foreach는 반복할 컬렉션, 반복문에서 사용할 item 이름, 인덱스 이름, 시작 구문, 반복 구문, 종료 구문을 지정할 수 있습니다.

속성	필수/선택	설명
collection	필수	반복문에 사용할 컬렉션을 지정합니다.
close	선택	반복문이 종료될 때 사용할 구문을 지정합니다.
index	선택	반복문에서 사용할 인덱스 이름을 지정합니다.
item	선택	반복문에서 사용할 아이템 이름을 지정합니다.
nullable	선택	컬렉션이 null인 경우에도 반복문을 수행할지의 여부를 지정합니다.
open	선택	반복문이 시작될 때 사용할 구문을 지정합니다.
separator	선택	각 반복 값 사이에 추가할 구분자를 지정합니다.

💡 기본구조

- 반복적인 구문을 처리할 때 사용되는 구문으로 반복할 컬렉션, 반복문에서 사용할 item 이름, 인덱스 이름, 시작 구문, 반복 구문, 종료 구문을 지정할 수 있습니다.

```
<foreach collection="컬렉션" item="아이템" index="인덱스" open="시작구문" separator="구분자" close="종료구문">
    반복할 SQL 구문
</foreach>
```

💡 사용예시

- 해당 SQL문은 tb_code 테이블의 전체 컬럼을 조건부 조회하는 SQL문입니다.
- 해당 부분에서 foreach 태그를 통해 list로 들어온 데이터를 처리합니다.
- 예를 들어 item = [“java”, “kotlin”, “node”]와 같은 데이터가 수행될 경우 IN (“java”, “kotlin”, “node”)으로 구성되어 수행됩니다.

```
<!-- 코드 값에 따라 코드 리스트를 전체 조회합니다.-->
<select id="selectCodeByCd" resultType="codeDto">
```

```
    SELECT
    t1.grp_cd
    , t1.cd
    , t1.grp_cd_nm
    , t1.grp_cd
    , t1.grp_cd_nm
    , t1.cd_nm
    , t1.sort_order
    FROM tb_code t1
    WHERE t1.use_yn = true
    AND t1.cd IN
    <foreach item="item" index="index" collection="list" open="("
                                                    separator="," close=")">
        #{item}
    </foreach>
</select>
```

7) bind

- OGNL(객체 그래프를 탐색) 표현식의 결과를 변수에 할당하는 구문입니다. 이렇게 하면 동일한 표현식을 반복하지 않고 변수 이름을 사용하여 쉽게 참조할 수 있습니다.

속성	필수/선택	설명
name	필수	변수의 이름을 지정합니다.
value	필수	변수에 할당할 값을 지정합니다. 이 값은 OGNL 표현식이 될 수 있습니다.

💡 기본구조

- name 속성은 변수의 이름을 지정하며, value 속성은 변수에 할당할 값을 지정합니다. 이 값은 OGNL 표현식이 될 수 있습니다.

```
<bind name="변수명" value="OGNL 표현식" />
```

[더 알아보기]

- OGNL(Object-Graph Navigation Language) 표현식
  - 객체 그래프를 탐색하기 위한 표현식 언어입니다. OGNL은 자바 객체의 속성에 접근하거나 메서드를 호출하는 데 사용되며, 광범위한 연산, 타입 변환, 컬렉션 처리 등 많은 기능을 제공합니다.
- OGNL(Object-Graph Navigation Language) 표현식 사용예시
  - name : 'name'이라는 이름의 속성 값에 접근합니다.
  - person.birthday.month : 'person' 객체의 'birthday' 속성 내의 'month' 속성 값에 접근합니다.
  - @com.example.MyClass@MY_CONSTANT : 'com.example.MyClass'의 'MY_CONSTANT'라는 이름의 정적 필드에 접근합니다

- 사용예시
  - tb_code 테이블을 조회하는데 조건부 조회를 수행하는 SQL문입니다.
  - 해당 SQL문에서는 bind 태그를 통해 매핑하려는 이름을 속성 name로 지정하고 value 값에 OGNL 표현식을 통해 지정합니다.
  - 기본적으로 파라미터로 넘어오는 데이터는 '_parameter'로 매핑이 되며 해당 값에서 getter로 cdNm이라는 값을 가져와 Like 패턴을 만듭니다.
  - 이러한 패턴을 기반으로 동적 Like 문을 만듭니다.

```
<!-- 코드 이름에 LIKE 조건으로 코드 리스트를 전체 조회합니다.-->
<select id="selectCodeByCdNm" resultType="codeDto">
    <bind name="pattern" value="'%' + _parameter.getCdNm() + '%'"/>
    SELECT t1.grp_cd
    , t1.cd
    , t1.grp_cd_nm
    , t1.grp_cd
    , t1.grp_cd_nm
    , t1.cd_nm
    , t1.sort_order
    FROM tb_code t1
    WHERE t1.use_yn = true
    AND t1.cd_nm LIKE #{pattern}
</select>
```

8) sql

- 재사용 가능한 SQL 조각을 정의하는 데 사용됩니다. 이 태그를 사용하면 동일한 SQL 조각을 여러 매퍼에서 사용할 수 있으므로 코드의 중복을 줄이는데 도움이 됩니다.
- sql 태그의 id 속성은 SQL 조각의 식별자로 사용되며, 이를 include 태그의 refid 속성에 지정하여 SQL 조각을 참조할 수 있습니다.
- sql 태그 내부에는 동적 SQL 구문도 포함될 수 있습니다. 이 경우, include 태그를 사용하여 SQL 조각을 참조할 때 동적 SQL 구문에 필요한 값을 전달할 수 있습니다.

💡 기본 구조

```
<sql id="Identifier">
  SQL 구문
</sql>
```

💡 사용예시

- sql 태그를 활용하여 모든 컬럼을 정의하고 구분자로 "codeList"로 지정하였습니다.
- 이렇게 되면 반복적으로 사용하는 모든 컬럼 제외 대상을 공통으로 이용이 가능합니다.

```
<!-- 코드 테이블 모든 컬럼 -->
<sql id="codeList">
      t1.grp_cd
    , t1.cd
    , t1.grp_cd_nm
    , t1.grp_cd
    , t1.grp_cd_nm
    , t1.cd_nm
    , t1.sort_order
</sql>
```