MYBATIS - DYNAMIC SQL

Dynamic SQL is a very powerful feature of MyBatis. It enables programmers to build queries based on the scenario dynamically. For example, if you want to search the Student data base, based on the name of the student in MyBatis, you have to write the query using the dynamic SQL.

MyBatis uses a powerful Dynamic SQL language that can be used within any mapped SQL statement. Following are the OGNL based Dynamic SQL expressions provided by MyBatis.

- if
- choose *when, otherwise*
- trim *where, set*
- foreach

The if Statement

The most common thing to do in dynamic SQL is conditionally include a part of a where clause. For example −

```
<select id = "getRecByName" parameterType = "Student" resultType = "Student">

   SELECT * FROM STUDENT
   <if test = "name != null">
      WHERE name LIKE #{name}
   </if>
</select>
```

This statement provides an optional text search type of functionality. If you pass in no name, then all active records will be returned. But if you do pass in a name, it will look for a name with the given **like** condition.

You can include multiple **if** conditions as −

```
<select id = "getRecByName_Id" parameterType = "Student" resultType = "Student">

   SELECT * FROM STUDENT
   <if test = "name != null">
      WHERE name LIKE #{name}
   </if>

   <if test = "id != null">
      AND id LIKE #{id}
   </if>
</select>
```

The choose, when, and otherwise Statements

MyBatis offers a **choose** element, which is similar to Java's switch statement. It helps to choose only one case among many options.

The following example will search only by name if it is provided, and if the name is not given, then only by id −

```
<select id = "getRecByName_Id_phone" parameterType = "Student" resultType = "Student">
   SELECT * FROM Student WHERE id != 0

   <choose>
      <when test = "name != null">
         AND name LIKE #{name}
      </when>
```

```
      <when test = "phone != null">
         AND phone LIKE #{phone}
      </when>
   </choose>

</select>
```

The where Statement

Take a look at our previous examples to see what happens if none of the conditions are met. You would end up with an SQL that looks like this −

```
SELECT * FROM Student
WHERE
```

This would fail, but MyBatis has a simple solution with one simple change, everything works fine −

```
<select id = "getName_Id_phone" parameterType = "Student" resultType = "Student">
   SELECT * FROM STUDENT

   <where>
      <if test = "id != null">
         id = #{id}
      </if>

      <if test = "name != null">
         AND name LIKE #{name}
      </if>
   </where>

</select>
```

The **where** element inserts a *WHERE* only when the containing tags return any content. Furthermore, if that content begins with *AND* or *OR,* it knows to strip it off.

The foreach Statement

The foreach element allows you to specify a collection and declare item and index variables that can be used inside the body of the element.

It also allows you to specify opening and closing strings, and add a separator to place in between iterations. You can build an **IN** condition as follows −

```
<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>
```

Dynamic SQL Example

This is an example if using dynamic SQL. Consider, we have the following Student table in MySQL −

```
CREATE TABLE details.student(
   ID int(10) NOT NULL AUTO_INCREMENT,
   NAME varchar(100) NOT NULL,
   BRANCH varchar(255) NOT NULL,
   PERCENTAGE int(3) NOT NULL,
   PHONE int(11) NOT NULL,
```

```
   EMAIL varchar(255) NOT NULL,
   PRIMARY KEY (`ID`)
);
Query OK, 0 rows affected (0.37 sec)
```

Let's assume this table has two records as −

```
mysql> select * from student;
+----+----------+--------+------------+-----------+----------------------+
| ID |   NAME   | BRANCH | PERCENTAGE |   PHONE   |        EMAIL         |
+----+----------+--------+------------+-----------+----------------------+
|  1 | Mohammad |   It   |     80     | 900000000 | mohamad123@yahoo.com |
|  2 | Shyam    |   It   |     75     | 984800000 | shyam@gmail.com      |
+----+----------+--------+------------+-----------+----------------------+
2 rows in set (0.00 sec)
```

Student POJO Class

To perform read operation, let us have a Student class in Student.java as −

```
public class Student {
   private int id;
   private String name;
   private String branch;
   private int percentage;
   private int phone;
   private String email;

   public Student(int id, String name,  String branch, int percentage, int phone, String
email) {
      super();
      this.id = id;
      this.name = name;
      this.branch = branch;
      this.percentage = percentage;
      this.phone = phone;
      this.email = email;
   }

   public Student() {}

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public int getPhone() {
      return phone;
   }

   public void setPhone(int phone) {
      this.phone = phone;
   }

   public String getEmail() {
      return email;
   }
```

```
   public void setEmail(String email) {
      this.email = email;
   }

   public String getBranch() {
      return branch;
   }

   public void setBranch(String branch) {
      this.branch = branch;
   }

   public int getPercentage() {
      return percentage;
   }

   public void setPercentage(int percentage) {
      this.percentage = percentage;
   }

}
```

Student.xml File

This file contains the result map named Student, to map the results of the SELECT Query. We will define an "id" which will be used in mybatisRead.java for executing Dynamic SQL SELECT query on database.

```
<?xml version = "1.0" encoding = "UTF-8"?>

&l;t!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace = "Student">

   <resultMap id = "result" type = "Student">
      <result property = "id" column = "ID"/>
      <result property = "name" column = "NAME"/>
      <result property = "branch" column = "BRANCH"/>
      <result property = "percentage" column = "PERCENTAGE"/>
      <result property = "phone" column = "PHONE"/>
      <result property = "email" column = "EMAIL"/>
   </resultMap>

   <select id = "getRecByName" parameterType = "Student" resultType = "Student">
      SELECT * FROM STUDENT

      <if test = "name != null">
         WHERE name LIKE #{name}
      </if>

   </select>

</mapper>
```

GetRecordByName.java File

This file has application level logic to read conditional records from the Student table −

```
import java.io.IOException;
import java.io.Reader;
import java.util.List;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
```

```
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

public class GetRecordByName {

   public static void main(String args[]) throws IOException{

      String req_name = "Mohammad";
      Reader reader = Resources.getResourceAsReader("SqlMapConfig.xml");
      SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
      SqlSession session = sqlSessionFactory.openSession();
      Student stud = new Student();
      stud.setName(req_name);

      //select contact all contacts
      //List<Student> student = session.selectList("getRecByName",stud);

      stud.setId(1);
      List<Student> student = session.selectList("getRecByName_Id",stud);

      for(Student st : student ){

         System.out.println("++++++++++++++details of the student named Mohammad are
"+"++++++++++++++++++" );

         System.out.println("Id :  "+st.getId());
         System.out.println("Name :  "+st.getName());
         System.out.println("Branch :  "+st.getBranch());
         System.out.println("Percentage :  "+st.getPercentage());
         System.out.println("Email :  "+st.getEmail());
         System.out.println("Phone :  "+st.getPhone());

      }

      System.out.println("Records Read Successfully ");
      session.commit();
      session.close();
   }
}
```

Compilation and Run

Here are the steps to compile and run the above mentioned software. Make sure, you have set PATH and CLASSPATH appropriately before proceeding for compilation and execution.

- Create Student.xml as shown above.
- Create Student.java as shown above and compile it.
- Create GetRecordByName.java as shown above and compile it.
- Execute GetRecordByName binary to run the program.

You would get the following result, and a record would be read from the Student table.

```
++++++++++++++details of the student named Mohammad are ++++++++++++++++++
Id :  1
Name :  Mohammad
Branch :  It
Percentage :  80
Email :  mohamad123@yahoo.com
Phone :  90000000
Records Read Successfully
```