MYBATIS - MAPPER XML

In the previous chapter, we have seen how to configure MyBatis using an XML file. This chapter discusses the Mapper XML file and various mapped SQL statements provided by it.

Before proceeding to mapped statements, assume that the following table named **Student** exists in the MYSQL database −

```
+----+-------+--------+------------+-----------+---------------+
| ID |  NAME | BRANCH | PERCENTAGE |   PHONE   |     EMAIL     |
+----+-------+--------+------------+-----------+---------------+
|  1 | Shyam |   it   |     80     | 954788457 | mail@mail.com |
+----+-------+--------+------------+-----------+---------------+
```

Also assume that POJO class also exists named **Student** with respect to the above table as shown below −

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

   //Setters and getters
}
```

Mapped Statements

Mapper XML is an important file in MyBatis, which contains a set of statements to configure various SQL statements such as select, insert, update, and delete. These statements are known as **Mapped Statements** or **Mapped SQL Statements**.

- All the statements have unique id. To execute any of these statements you just need to pass the appropriate id to the methods in Java Application.*Thisisdiscussedclearlyinlaterchapters*.
- mapper XML file prevents the burden of writing SQL statements repeatedly in the application. In comparison to JDBC, almost 95% of the code is reduced using Mapper XML file in MyBatis.
- All these Mapped SQL statements are resided within the element named**<mapper>**. This element contains an attribute called **'namespace'**.

```
<mapper namespace = "Student">
   //mapped statements and result maps
<mapper>
```

All the Mapped SQL statements are discussed below with examples.

Insert

In MyBatis, to insert values into the table, we have to configure the insert mapped query. MyBatis provides various attributes for insert mapper, but largely we use id and parameter type.

id is unique identifier used to identify the insert statement. On the other hand, **parametertype** is the class name or the alias of the parameter that will be passed into the statement. Below given is an example of insert mapped query −

```
<insert id = "insert" parameterType = "Student">
   INSERT INTO STUDENT1 (NAME, BRANCH, PERCENTAGE, PHONE, EMAIL )
   VALUES (#{name}, #{branch}, #{percentage}, #{phone}, #{email});
</insert>
```

In the given example, we use the parameter of type Student *class*. The class student is a POJO class, which represents the Student record with name, branch, percentage, phone, and email as parameters.

You can invoke the 'insert' mapped query using Java API as shown below −

```
//Assume session is an SqlSession object.
session.insert("Student.insert", student);
```

Update

To update values of an existing record using MyBatis, the mapped query update is configured. The attributes of update mapped query are same as the insert mapped query. Following is the example of the update mapped query −

```
<update id = "update" parameterType = "Student">
   UPDATE STUDENT SET EMAIL = #{email}, NAME = #{name}, BRANCH = #{branch}, PERCENTAGE
= #{percentage}, PHONE = #{phone} WHERE ID = #{id};
</update>
```

To invoke the update query, instantiate Student class, set the values for the variables representing columns which need to be updated, and pass this object as parameter to **update** method. You can invoke the update mapped query using Java API as shown below −

```
//Assume session is an SqlSession object.
session.update("Student.update",student);
```

Delete

To delete the values of an existing record using MyBatis, the mapped query 'delete' is configured. The attributes of 'delete' mapped query are same as the insert and update mapped queries. Following is the example of the delete mapped query −

```
<delete id = "deleteById" parameterType = "int">
   DELETE from STUDENT WHERE ID = #{id};
</delete>
```

You can invoke the delete mapped query using the delete method of **SqlSession** interface provided by MyBatis Java API as shown below −

```
//Assume session is an SqlSession object.
session.delete("Student.deleteById", 18);
```

Select

To retrieve data, 'select' mapper statement is used. Following is the example of select mapped query to retrieve all the records in a table −

```
<select id = "getAll" resultMap = "result">
   SELECT * FROM STUDENT;
</select>
```

You can retrieve the data returned by the select query using the method **selectList.** This method returns the data of the selected record in the form of List as shown below −

```
List<Student> list = session.selectList("Student.getAll");
```

resultMaps

It is the most important and powerful elements in MyBatis. The results of SQL SELECT statements

are mapped to Java objects *beans/POJO*. Once the result map is defined, we can refer these from several SELECT statements. Following is the example of result Map query; it maps the results of the select queries to the Student class −

```
<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 = "getAll" resultMap = "result">
   SELECT * FROM STUDENT;
</select>

<select id = "getById" parameterType = "int" resultMap = "result">
   SELECT * FROM STUDENT WHERE ID = #{id};
</select>
```

Note − It is not mandatory to write the *column* attribute of the resultMap if both the property and the column name of the table are same.