MYBATIS - ANNOTATIONS

In the previous chapters, we have seen how to perform curd operations using MyBatis. There we used a Mapper XML file to store mapped SQL statements and a configuration XML file to configure MyBatis.

To map SQL statements, MyBatis also provides annotations. So, this chapter discusses how to use MyBatis annotations.

While working with annotations, instead of configuration XML file, we can use a java mapper interface to map and execute SQL queries.

Assume, we have the following employee 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)
```

Assume this table has two records as −

```
mysql> select * from STUDENT;
+----+----------+--------+------------+-----------+--------------------+
| ID |  NAME    | BRANCH | PERCENTAGE |   PHONE   |      EMAIL         |
+----+----------+--------+------------+-----------+--------------------+
|  1 | Mohammad |   It   |     80     | 984803322 | Mohammad@gmail.com |
|  2 | Shyam    |   It   |     75     | 984800000 | shyam@gmail.com    |
+----+----------+--------+------------+-----------+--------------------+
```

Student POJO Class

The POJO class would have implementation for all the methods required to perform desired operations.

Create a Student class in Student.java file 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_mapper.java

This is the file, which contains the mapper interface where we declare the mapped statements using annotations instead of XML tags. For almost all of the XML-based mapper elements, MyBatis provides annotations. The following file named Student_mapper.java, contains a mapper interface. Within this file, you can see the annotations to perform CURD operations on the STUDENT table.

```
import java.util.List;

import org.apache.ibatis.annotations.*;

public interface Student_mapper {

   final String getAll = "SELECT * FROM STUDENT";
   final String getById = "SELECT * FROM STUDENT WHERE ID = #{id}";
   final String deleteById = "DELETE from STUDENT WHERE ID = #{id}";
   final String insert = "INSERT INTO STUDENT (NAME, BRANCH, PERCENTAGE, PHONE, EMAIL )
VALUES (#{name}, #{branch}, #{percentage}, #{phone}, #{email})";
```

```
   final String update = "UPDATE STUDENT SET EMAIL = #{email}, NAME = #{name}, BRANCH =
#{branch}, PERCENTAGE = #{percentage}, PHONE = #{phone} WHERE ID = #{id}";

   @Select(getAll)
   @Results(value = {
      @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")
   })

   List getAll();

   @Select(getById)
   @Results(value = {
      @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")
   })

   Student getById(int id);

   @Update(update)
   void update(Student student);

   @Delete(deleteById)
   void delete(int id);

   @Insert(insert)
   @Options(useGeneratedKeys = true, keyProperty = "id")
   void insert(Student student);
}
```

Annotations_Example.java File

This file would have application level logic to insert records in the Student table. Create and save **mybatisInsert.java** file as shown below −

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

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 Annotations_Example {

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

      Reader reader = Resources.getResourceAsReader("SqlMapConfig.xml");
      SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
      SqlSession session = sqlSessionFactory.openSession();
      session.getConfiguration().addMapper(Student_mapper.class);

      Student_mapper mapper = session.getMapper(Student_mapper.class);

      //Create a new student object
      Student student = new Student();

      //Set the values
      student.setName("zara");
      student.setBranch("EEE");
      student.setEmail("zara@gmail.com");
      student.setPercentage(90));
      student.setPhone(123412341);

      //Insert student data
      mapper.insert(student);
      System.out.println("record inserted successfully");
      session.commit();
      session.close();

   }

}
```

Compilation and Execution

Here are the steps to compile and run the Annotations_Example.java file. Make sure, you have set PATH and CLASSPATH appropriately before proceeding for compilation and execution.

- Create Student_mapper.java file as shown above and compile it.
- Create SqlMapConfig.xml as shown in the MYBATIS - Configuration XML chapter of this tutorial.
- Create Student.java as shown above and compile it.
- Create Annotations_Example.java as shown above and compile it.
- Execute Annotations_Example binary to run the program.

You would get the following result, and a record would be created in the STUDENT table.

```
$java Annotations_Example
Record Inserted Successfully
```

If you check the STUDENT table, it should display the following result –

```
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      |
|  3 | Zara     |  EEE   |     90     | 123412341 | zara@gmail.com       |
+----+----------+--------+------------+-----------+----------------------+
3 rows in set (0.08 sec)
```

In the same way, we can perform update, delete, and read operations using annotations by replacing the content of Annotations_Example.java with the respective snippets mentioned below –

Update

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

   Reader reader = Resources.getResourceAsReader("SqlMapConfig.xml");
   SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
   SqlSession session = sqlSessionFactory.openSession();
   session.getConfiguration().addMapper(Student_mapper.class);
   Student_mapper mapper = session.getMapper(Student_mapper.class);

   //select a particular student using id
   Student student = mapper.getById(2);
   System.out.println("Current details of the student are "+student.toString());
```

```
    //Set new values to the mail and phone number of the student
    student.setEmail("Shyam123@yahoo.com");
    student.setPhone(984802233);

    //Update the student record
    mapper.update(student);
    System.out.println("Record updated successfully");
    session.commit();
    session.close();

}
```

Read

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

    Reader reader = Resources.getResourceAsReader("SqlMapConfig.xml");
    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
    SqlSession session = sqlSessionFactory.openSession();
    session.getConfiguration().addMapper(Student_mapper.class);
    Student_mapper mapper = session.getMapper(Student_mapper.class);

    //Get the student details
    Student student = mapper.getById(2);
    System.out.println(student.getBranch());
    System.out.println(student.getEmail());
    System.out.println(student.getId());
    System.out.println(student.getName());
    System.out.println(student.getPercentage());
    System.out.println(student.getPhone());
    session.commit();
    session.close();

}
```

Delete

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

    Reader reader = Resources.getResourceAsReader("SqlMapConfig.xml");
    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
    SqlSession session = sqlSessionFactory.openSession();
    session.getConfiguration().addMapper(Student_mapper.class);

    Student_mapper mapper = session.getMapper(Student_mapper.class);
    mapper.delete(2);
    System.out.println("record deleted successfully");
    session.commit();
    session.close();

}
```