SESSION-20

Hibernate CRUD Operations

C-CREATE
R-READ
U-UPDATE
D-DELETE

A CRUD operation deals with creating, retrieving, updating, and deleting records from the table. In this tutorial we will see how it is done using Hibernate annotations. We are going to discuss 4 main functionalities:

- Creating a Record
- Displaying Records
- Updating a Record
- Deleting a Record

2. Hibernate CRUD Operations

2.1 Tools Used

We are using Eclipse Kepler SR2, JDK 8, MySQL database and Maven. Having said that, we have tested the code against JDK 1.7 and it works well.

2.2 Project Structure

Firstly, let's review the final project structure, in case you are confused about where you should create the corresponding files or folder later!

2.3 Project Creation

This section will demonstrate on how to create a Java based Maven project with Eclipse. In Eclipse IDE, go to `File -> New -> Maven Project`.

In the New Maven Project window, it will ask you to select project location. By default, 'Use default workspace location' will be selected. Select the 'Create a simple project (skip archetype selection)' checkbox and just click on next button to proceed.

It will ask you to 'Enter the group and the artifact id for the project'. We will input the details as shown in the below image. The version number will be by default

Click on Finish and the creation of a maven project is completed. If you observe, it has downloaded the maven dependencies and a `pom.xml` file will be created. It will have the following code:

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>HibernateCrud</groupId>
   <artifactId>HibernateCrud</artifactId>
   <version>0.0.1-SNAPSHOT</version>
</project>
```

We can start adding the dependencies that developers want like Hibernate, MySQL etc. Let’s start building the application!

3. Application Building
Below are the steps involved in developing this application.
3.1 Database & Table Creation
The following MySQL script is used to create a database called `tutorialDb` with table: `student`. Open MySQL terminal or workbench terminal and execute the script:

```
CREATE DATABASE IF NOT EXISTS tutorialDb;


USE tutorialDb;

DROP TABLE IF EXISTS student;

CREATE TABLE IF NOT EXISTS student (
  student_id int(100) NOT NULL AUTO_INCREMENT,
  student_name varchar(50) DEFAULT NULL,
  roll_number varchar(50) DEFAULT NULL,
  course varchar(50) DEFAULT NULL,
  PRIMARY KEY (student_id)
);
```

If everything goes well, the database and the table will be shown in the MySQL workbench.

3.2 Maven Dependencies

Here, we specify three dependencies for Hibernate Core, MySQL Connector and, Log4j. The rest dependencies will be automatically resolved by Maven, such as Hibernate JPA and Hibernate Commons Annotations. The **updated** file will have the following code:

pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>HibernateCrud</groupId>
   <artifactId>HibernateCrud</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <packaging>jar</packaging>
   <dependencies>
      <!-- Hibernate 4.3.6 Final -->
      <dependency>
         <groupId>org.hibernate</groupId>
         <artifactId>hibernate-core</artifactId>
         <version>4.3.6.Final</version>
      </dependency>
      <!-- Mysql Connector -->
      <dependency>
         <groupId>mysql</groupId>
         <artifactId>mysql-connector-java</artifactId>
         <version>5.1.21</version>
      </dependency>
      <!-- Log4j 1.2.16 Final -->
      <dependency>
         <groupId>log4j</groupId>
         <artifactId>log4j</artifactId>
         <version>1.2.16</version>
      </dependency>
   </dependencies>
   <build>
      <finalName>${project.artifactId}</finalName>
   </build>
</project>
```

3.3 Java Class Creation

Let's create the required Java files. Right-click on `src/main/java` folder, `New -> Package`.

A new pop window will open where we will enter the package name as: `com.jcg.hibernate.crud.operations`.

Once the package is created in the application, we will need to create the Model, Database Operations and, Implementation classes. Right click on the newly created package: `New -> Class`.

A new pop window will open and enter the file name as `Student`. The `POJO` model class will be created inside the package: `com.jcg.hibernate.crud.operations`

Repeat the step (i.e. Fig. 10) and enter the filename as `DbOperations`. This class will be used to perform the database operations and is created inside the package: `com.jcg.hibernate.crud.operations`

3.3.1 Implementation of Model Class

Add the following code to it:

Student.java

```
package com.jcg.hibernate.crud.operations;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
```

```
@Table(name="student")
public class Student implements Serializable {

   private static final long serialVersionUID = 1L;

   @Id
   @Column(name="student_id")
   @GeneratedValue(strategy=GenerationType.AUTO)
   private int id;

   @Column(name="student_name")
   private String studentName;

   @Column(name="roll_number")
   private int rollNumber;

   @Column(name="course")
   private String course;

   public int getId() {
      return id;
   }

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

   public String getStudentName() {
      return studentName;
   }

   public void setStudentName(String studentName) {
      this.studentName = studentName;
   }

   public int getRollNumber() {
      return rollNumber;
   }

   public void setRollNumber(int rollNumber) {
      this.rollNumber = rollNumber;
   }

   public String getCourse() {
      return course;
   }
```

```
    public void setCourse(String course) {
        this.course = course;
    }

    @Override
    public String toString() {
        return "Student Details?= Id: " + this.id + ", Name: " + this.studentName + ", Roll No.:
" + this.rollNumber + ", Course: " + this.course;
    }
}
```

3.3.2 Implementation of DAO Class

This class has methods that interact with the database for performing the CRUD operation on the records. Add the following code to it:

DbOperations.java

```
package com.jcg.hibernate.crud.operations;

import java.util.ArrayList;
import java.util.List;

import org.apache.log4j.Logger;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

public class DbOperations {

    static Session sessionObj;
    static SessionFactory sessionFactoryObj;

    public final static Logger logger = Logger.getLogger(DbOperations.class);

    // This Method Is Used To Create The Hibernate's SessionFactory Object
    private static SessionFactory buildSessionFactory() {
        // Creating Configuration Instance & Passing Hibernate Configuration File
        Configuration configObj = new Configuration();
        configObj.configure("hibernate.cfg.xml");

        // Since Hibernate Version 4.x, ServiceRegistry Is Being Used
        ServiceRegistry serviceRegistryObj = new
StandardServiceRegistryBuilder().applySettings(configObj.getProperties()).build();

        // Creating Hibernate SessionFactory Instance
        sessionFactoryObj = configObj.buildSessionFactory(serviceRegistryObj);
        return sessionFactoryObj;
    }

    // Method 1: This Method Used To Create A New Student Record In The Database Table
    public static void createRecord() {
        int count = 0;
        Student studentObj = null;
        try {
            // Getting Session Object From SessionFactory
            sessionObj = buildSessionFactory().openSession();
            // Getting Transaction Object From Session Object
            sessionObj.beginTransaction();

            // Creating Transaction Entities
            for(int j = 101; j <= 105; j++) {
                count = count + 1;
                studentObj = new Student();
                studentObj.setRollNumber(j);
                studentObj.setStudentName("Editor " + j);
                studentObj.setCourse("Bachelor Of Technology");
                sessionObj.save(studentObj);
            }

            // Committing The Transactions To The Database
            sessionObj.getTransaction().commit();
            logger.info("\nSuccessfully Created '" + count + "' Records In The Database!\n");
        } catch(Exception sqlException) {
            if(null != sessionObj.getTransaction()) {
                logger.info("\n.......Transaction Is Being Rolled Back.......\n");
                sessionObj.getTransaction().rollback();
            }
            sqlException.printStackTrace();
        } finally {
            if(sessionObj != null) {
                sessionObj.close();
            }
        }
    }

    // Method 2: This Method Is Used To Display The Records From The Database Table
    @SuppressWarnings("unchecked")
    public static List displayRecords() {
        List studentsList = new ArrayList();
        try {
            // Getting Session Object From SessionFactory
            sessionObj = buildSessionFactory().openSession();
            // Getting Transaction Object From Session Object
            sessionObj.beginTransaction();

            studentsList = sessionObj.createQuery("FROM Student").list();
        } catch(Exception sqlException) {
            if(null != sessionObj.getTransaction()) {
                logger.info("\n.......Transaction Is Being Rolled Back.......\n");
                sessionObj.getTransaction().rollback();
            }
            sqlException.printStackTrace();
        } finally {
            if(sessionObj != null) {
                sessionObj.close();
            }
        }
        return studentsList;
    }

    // Method 3: This Method Is Used To Update A Record In The Database Table
    public static void updateRecord(int student_id) {
        try {
            // Getting Session Object From SessionFactory
            sessionObj = buildSessionFactory().openSession();
            // Getting Transaction Object From Session Object
            sessionObj.beginTransaction();

            // Creating Transaction Entity
            Student stuObj = (Student) sessionObj.get(Student.class, student_id);
            stuObj.setStudentName("Java Code Geek");
            stuObj.setCourse("Masters Of Technology");

            // Committing The Transactions To The Database
            sessionObj.getTransaction().commit();
            logger.info("\nStudent With Id?= " + student_id + " Is Successfully Updated In The
Database!\n");
        } catch(Exception sqlException) {
            if(null != sessionObj.getTransaction()) {
                logger.info("\n.......Transaction Is Being Rolled Back.......\n");
                sessionObj.getTransaction().rollback();
            }
            sqlException.printStackTrace();
        } finally {
            if(sessionObj != null) {
```

```
            sessionObj.close();
         }
      }
   }

   // Method 4(a): This Method Is Used To Delete A Particular Record From The Database
Table
   public static void deleteRecord(Integer student_id) {
      try {
         // Getting Session Object From SessionFactory
         sessionObj = buildSessionFactory().openSession();
         // Getting Transaction Object From Session Object
         sessionObj.beginTransaction();

         Student studObj = findRecordById(student_id);
         sessionObj.delete(studObj);

         // Committing The Transactions To The Database
         sessionObj.getTransaction().commit();
         logger.info("\nStudent With Id?= " + student_id + " Is Successfully Deleted From
The Database!\n");
      } catch(Exception sqlException) {
         if(null != sessionObj.getTransaction()) {
            logger.info("\n.......Transaction Is Being Rolled Back.......\n");
            sessionObj.getTransaction().rollback();
         }
         sqlException.printStackTrace();
      } finally {
         if(sessionObj != null) {
            sessionObj.close();
         }
      }
   }

   // Method 4(b): This Method To Find Particular Record In The Database Table
   public static Student findRecordById(Integer find_student_id) {
      Student findStudentObj = null;
      try {
         // Getting Session Object From SessionFactory
         sessionObj = buildSessionFactory().openSession();
         // Getting Transaction Object From Session Object
         sessionObj.beginTransaction();

         findStudentObj = (Student) sessionObj.load(Student.class, find_student_id);
      } catch(Exception sqlException) {
         if(null != sessionObj.getTransaction()) {
            logger.info("\n.......Transaction Is Being Rolled Back.......\n");
```

```
                sessionObj.getTransaction().rollback();
            }
            sqlException.printStackTrace();
        }
        return findStudentObj;
    }

    // Method 5: This Method Is Used To Delete All Records From The Database Table
    public static void deleteAllRecords() {
        try {
            // Getting Session Object From SessionFactory
            sessionObj = buildSessionFactory().openSession();
            // Getting Transaction Object From Session Object
            sessionObj.beginTransaction();

            Query queryObj = sessionObj.createQuery("DELETE FROM Student");
            queryObj.executeUpdate();

            // Committing The Transactions To The Database
            sessionObj.getTransaction().commit();
            logger.info("\nSuccessfully Deleted All Records From The Database Table!\n");
        } catch(Exception sqlException) {
            if(null != sessionObj.getTransaction()) {
                logger.info("\n.......Transaction Is Being Rolled Back.......\n");
                sessionObj.getTransaction().rollback();
            }
            sqlException.printStackTrace();
        } finally {
            if(sessionObj != null) {
                sessionObj.close();
            }
        }
    }
}
```

3.3.3 Implementation of Utility Class

This class helps in creating the `SessionFactory` from the Hibernate configuration file and interacts with the DAO class to perform the CRUD operations. Add the following code to it:
AppMain.java

```
package com.jcg.hibernate.crud.operations;

import java.util.List;

import org.apache.log4j.Logger;

public class AppMain {

    public final static Logger logger = Logger.getLogger(AppMain.class);

    public static void main(String[] args) {
        logger.info(".......Hibernate Crud Operations Example.......\n");

        logger.info("\n=======CREATE RECORDS=======\n");
        DbOperations.createRecord();

        logger.info("\n=======READ RECORDS=======\n");
        ListviewStudents = DbOperations.displayRecords();
        if(viewStudents != null & viewStudents.size() > 0) {
            for(Student studentObj : viewStudents) {
                logger.info(studentObj.toString());
            }
        }

        logger.info("\n=======UPDATE RECORDS=======\n");
        int updateId = 1;
        DbOperations.updateRecord(updateId);
        logger.info("\n=======READ RECORDS AFTER UPDATION=======\n");
        List updateStudent = DbOperations.displayRecords();
        if(updateStudent != null & updateStudent.size() > 0) {
            for(Student studentObj : updateStudent) {
                logger.info(studentObj.toString());
            }
        }

        logger.info("\n=======DELETE RECORD=======\n");
        int deleteId = 5;
        DbOperations.deleteRecord(deleteId);
        logger.info("\n=======READ RECORDS AFTER DELETION=======\n");
        List deleteStudentRecord = DbOperations.displayRecords();
        for(Student studentObj : deleteStudentRecord) {
            logger.info(studentObj.toString());
        }

        logger.info("\n=======DELETE ALL RECORDS=======\n");
        DbOperations.deleteAllRecords();
        logger.info("\n=======READ RECORDS AFTER ALL RECORDS DELETION=======");
        List deleteAll = DbOperations.displayRecords();
        if(deleteAll.size() == 0) {
            logger.info("\nNo Records Are Present In The Database Table!\n");
        }
        System.exit(0);
    }
}
```

3.4 Hibernate Configuration File

To configure the Hibernate framework, we need to implement a configuration file i.e. `hiberncate.cfg.xml`. Right-click on `src/main/resources` folder, `New -> Other`.

A new pop window will open and select the wizard as an XML file.

Again, a pop-up window will open. Verify the parent folder location as `HibernateCrud/src/main/resources` and enter the file name as `hibernate.cfg.xml`. Click Finish.

Once the file is created, we will include the database configuration and mapping classes details. Add the following code to it:

hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <!-- SQL Dialect -->
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <!-- Database Connection Settings -->
    <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/tutorialDb</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password"></property>
    <property name="show_sql">true</property>

    <!-- Specifying Session Context -->
    <property
name="hibernate.current_session_context_class">org.hibernate.context.internal.ThreadL
ocalSessionContext</property>

    <!-- Mapping With Model Class Containing Annotations -->
    <mapping class="com.jcg.hibernate.crud.operations.Student" />
  </session-factory>
</hibernate-configuration>
```

Notes:

- Here, we instructed Hibernate to connect to a MySQL database named `tutorialDb` and the Mapping classes to be loaded
- We have also instructed Hibernate framework to use `MySQLDialect` i.e. Hibernate will optimize the generated `SQL` statements for MySQL
- This configuration will be used to create a Hibernate `SessionFactory` object

4. Run the Application

To run the Hibernate application, Right click on the `AppMain` class `-> Run As -> Java Application`.

5. Project Demo

Executing the `AppMain` class, we will save few Student's records and then we will apply the CRUD operations on those records. Developers can debug the example and see what happens in the database after every step.