**Hibernate**

In this section we are going to cover all the CRUD operation using mySQL databse and hibernate.

The first step is to create a user in mySQL database and set your own password and user, to create it, logging into mysqlworkbech and run the following query:

**CREATE USER 'hbstudent'@'localhost' IDENTIFIED BY 'hbstudent';**

**GRANT ALL PRIVILEGES ON \* . \* TO 'hbstudent'@'localhost';**

**#**

**# Starting with MySQL 8.0.4, the MySQL team changed the**

**# default authentication plugin for MySQL server**

**# from mysql\_native\_password to caching\_sha2\_password.**

**#**

**# The command below will make the appropriate updates for your user account.**

**#**

**# See the MySQL Reference Manual for details:**

**# https://dev.mysql.com/doc/refman/8.0/en/caching-sha2-pluggable-authentication.html**

**#**

**ALTER USER 'hbstudent'@'localhost' IDENTIFIED WITH mysql\_native\_password BY 'hbstudent';**

After creating the user test the connection and run the following script to create the database and the table:

**CREATE DATABASE IF NOT EXISTS `Employee`;**

**USE `hb\_student\_tracker`;**

**--**

**-- Table structure for table `student`**

**--**

**DROP TABLE IF EXISTS `Employee`;**

**CREATE TABLE `Employee` (**

**`id` int(11) NOT NULL AUTO\_INCREMENT,**

**`first\_name` varchar(45) DEFAULT NULL,**

**`last\_name` varchar(45) DEFAULT NULL,**

**`company` varchar(45) DEFAULT NULL,**

**PRIMARY KEY (`id`)**

**) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=latin1;**

**Hibernate with Intelij**

Now let’s create the project, in Intelij we are going to generate the project using Spring Boot.

Go to the following link <https://start.spring.io> and generate the project adding the needed dependency, in our case we will select JPA/hibernate and mySQL and click on generated and the download will start.

Open the project with Intelij and under the folder resources create a xml file named: **hibernate.cfg.xml** and paste the following code inside it:

**<!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>**

**<!-- JDBC Database connection settings -->**

**<property name="connection.driver\_class">com.mysql.cj.jdbc.Driver</property>**

**<property name="connection.url">jdbc:mysql://localhost:3306/hb\_student\_tracker?useSSL=false&amp;serverTimezone=UTC</property>**

**<property name="connection.username">hbstudent</property>**

**<property name="connection.password">hbstudent</property>**

**<!-- JDBC connection pool settings ... using built-in test pool -->**

**<property name="connection.pool\_size">1</property>**

**<!-- Select our SQL dialect -->**

**<property name="dialect">org.hibernate.dialect.MySQLDialect</property>**

**<!-- Echo the SQL to stdout -->**

**<property name="show\_sql">true</property>**

**<!-- Set the current session context -->**

**<property name="current\_session\_context\_class">thread</property>**

**</session-factory>**

**</hibernate-configuration>**

Inside the xml code there are the all the property needed to establish the connection.

**Database Model**

Create the database model creating a new class, we will declare some variable and we will add the opportune mapping to the code.

We will use some annotation such as:

* @Entity, where specified that the class is a model for the database.
* @Table (name = “tableName”), where we can specify the database name, this annotation is optional and eventually Spring will start to look for the name of the class decapitalized.
* GeneratedValue(Strategy = GeneratedType.IDENTITY) to autogenerate the value of the primary key.
* @Id, to tell to spring that this is a primary key.
* @Column, specify the name of the column that the field is refered to.

Create the field needed for your application and generate setter and getter as well.

Following the snip of the code:

@Entity  
@Table(name="student")  
public class Student {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = "id")  
 private int id;  
  
 @Column(name = "first\_name")  
 private String firstName;  
  
 @Column(name = "last\_name")  
 private String lastName;  
  
 @Column(name = "email")  
 private String email;  
  
 public Student(String firstName, String lastName, String email) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.email = email;  
 }  
  
 public Student() {  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 @Override  
 public String toString() {  
 return "Student{" +  
 "id=" + id +  
 ", firstName='" + firstName + '\'' +  
 ", lastName='" + lastName + '\'' +  
 ", email='" + email + '\'' +  
 '}';  
 }  
}

**Hibernate Create an Entry in the database**

Creating a new entry in the database we need to create a session and factory

//Create Factory and link that hibernate xml file  
SessionFactory factory = new Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Student.class).buildSessionFactory();  
  
//create session  
Session session = factory.getCurrentSession();

Create a new object and add into the database using hiberbernate is really simple.

The step are:

* Begin the transaction
* Save the object
* Commit the transaction
* Close the factory

Following an example of code:

package com.example.hibernatedemo;  
  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
public class primaryKeyDemo {  
  
  
 public static void main(String[] args) {  
  
 //Create Factory and link that hibernate xml file  
 SessionFactory factory = new Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Student.class).buildSessionFactory();  
  
 //create session  
 Session session = factory.getCurrentSession();  
  
 try {  
  
 //create 3 students object  
 System.*out*.println("creating a new student object ...");  
 Student tempStudent1 = new Student("Mary", "Public", "mary@code.com");  
 Student tempStudent2 = new Student("John", "Doe", "john@code.com");  
 Student tempStudent3 = new Student("Bonita", "Applebum", "bonita@code.com");  
  
 // start a transaction, it similar begin and commit transaction in sql  
 session.beginTransaction();  
  
  
 //save the student object,  
 System.*out*.println("Saving the student.......");  
 session.save(tempStudent1);  
 session.save(tempStudent2);  
 session.save(tempStudent3);  
  
  
 // commit transaction  
 session.getTransaction().commit();  
 System.*out*.println("Done!");  
  
  
 } finally {  
 factory.close();  
 }  
 }  
}

**Hibernate Query database**

Hibernate gives the possibility to query the database using our custom hsql quert.

Below the code:

package com.example.hibernatedemo;  
  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
import java.util.List;  
  
public class QueryStudentDemo {  
 public static void main(String[] args) {  
 // create session factory, session factory is need to configure hirbernate, place the file hibernate.cfg.xml under resources file  
 SessionFactory factory = new Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Student.class).buildSessionFactory();  
 //create session  
  
 Session session = factory.getCurrentSession();  
  
 try {  
 //begin transaction  
 session.beginTransaction();  
  
 //query student  
  
 List<Student> theStudents = session.createQuery("from Student").getResultList();  
  
 //display the students  
  
 *displayStudent*(theStudents);  
  
 //query student: lastName = Doe  
  
 theStudents = session.createQuery("from Student where lastName = 'Doe'").getResultList();  
   
  
  
 //display the students  
 System.*out*.println("Students who have last name of Doe");  
 *displayStudent*(theStudents);  
 //query students: lastName = 'Doe' OR firstName= 'Daffy'  
 theStudents = session.createQuery("from Student s where" + " s.lastName='Doe' OR s.firstName='Daffy'").getResultList();  
 System.*out*.println("Students who have last name of Doe and firstName = Daffy");  
 *displayStudent*(theStudents);  
  
  
 //query students where email LIKE %luv2code.com  
 theStudents = session.createQuery("from Student s where s.email LIKE '%code.com'").getResultList();  
  
 System.*out*.println("\n\nStudents who email ends with luv2code.com");  
 *displayStudent*(theStudents);  
  
 // commit transaction  
 session.getTransaction().commit();  
 System.*out*.println("Done!");  
  
  
 } finally {  
 factory.close();  
 }  
 }  
  
 private static void displayStudent(List<Student> theStudents) {  
 for (Student student : theStudents) {  
 System.*out*.println(student);  
 }  
 }  
}

**Hibernate Update**

Hibernate gives also the possibility to update our data by object and query, following the code:

package com.example.hibernatedemo;  
  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
import java.util.List;  
  
public class UpdateStudentDemo {  
 public static void main(String[] args) {  
 // create session factory, session factory is need to configure hirbernate, place the file hibernate.cfg.xml under resources file  
 SessionFactory factory = new Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Student.class).buildSessionFactory();  
 //create session  
  
 Session session = factory.getCurrentSession();  
  
  
 //set the student id. It will be used to update the database  
 int studentId =1;  
  
 try {  
  
 // begin the transaction  
 session.beginTransaction();  
 // retrieve student based on the id: primary key  
 Student tempStudent = session.get(Student.class, studentId);  
  
 // set the student using the setter from the student class  
  
 System.*out*.println("Updating the student...");  
 tempStudent.setFirstName("Scooby");  
  
 //commit the transaction  
 session.getTransaction().commit();  
  
  
 // NEW CODE  
  
 //Get a new session  
 session = factory.getCurrentSession();  
 //Begin a new transaction  
 session.beginTransaction();  
  
  
 //Update email for all students  
 System.*out*.println("Update email for all the students");  
 session.createQuery("update Student set email='foo@gmail.com' ").executeUpdate();  
 session.getTransaction().commit();  
 System.*out*.println("Done!");  
  
  
 } finally {  
 factory.close();  
 }  
 }  
}

**Hibernate Delete**

package com.example.hibernatedemo;  
  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
public class DeleteStudentDemo {  
 public static void main(String[] args) {  
 // create session factory, session factory is need to configure hirbernate, place the file hibernate.cfg.xml under resources file  
 SessionFactory factory = new Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Student.class).buildSessionFactory();  
 //create session  
  
 Session session = factory.getCurrentSession();  
  
  
 //set the student id. It will be used to update the database  
  
  
 try {  
 int studentId = 1;  
 //now get a new session  
 session = factory.getCurrentSession();  
  
 // begin the transaction  
 session.beginTransaction();  
  
  
 // retrieve student based on the id: primary key  
 Student tempStudent = session.get(Student.class, studentId);  
  
 // delete the student  
 // System.out.println("Deleting student: " + tempStudent);  
 // session.delete(tempStudent);  
  
 System.*out*.println("Deleting Student id=2");  
 session.createQuery("delete from Student where id=2").executeUpdate();  
  
  
 // delete student id=2  
  
  
 //commit the transaction  
 session.getTransaction().commit();  
  
  
 // NEW CODE  
  
 System.*out*.println("Done!");  
  
  
 } finally {  
 factory.close();  
 }  
 }  
}