HIBERNATE - MANY-TO-ONE MAPPINGS

A **many-to-one** association is the most common kind of association where an Object can be associated with multiple objects. For example a same address object can be associated with multiple employee objects.

Define RDBMS Tables:

Consider a situation where we need to store our employee records in EMPLOYEE table which will have following structure:

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   address    INT NOT NULL,
   PRIMARY KEY (id)
);
```

Further, many employee can have same address, so this association can be presented using many-to-one association. We will store address related information in a separate table which has following structure:

```
create table ADDRESS (
   id INT NOT NULL auto_increment,
   street_name VARCHAR(40) default NULL,
   city_name VARCHAR(40) default NULL,
   state_name VARCHAR(40) default NULL,
   zipcode VARCHAR(10) default NULL,
   PRIMARY KEY (id)
);
```

Create both the RBDMS tables and keep them ready for the next implementation.

Define POJO Classes:

Let us implement a POJO class **Employee** which will be used to persist the objects related to EMPLOYEE table and having a variable of Address type.

```
import java.util.*;

public class Employee{
   private int id;
   private String firstName;
   private String lastName;
   private int salary;
   private Address address;

   public Employee() {}
   public Employee(String fname, String lname,
                   int salary, Address address ) {
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
      this.address = address;
   }
   public int getId() {
      return id;
   }
   public void setId( int id ) {
      this.id = id;
   }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }
   public int getSalary() {
      return salary;
   }
   public void setSalary( int salary ) {
      this.salary = salary;
   }

   public Address getAddress() {
      return address;
   }
   public void setAddress( Address address ) {
      this.address = address;
   }
}
```

We need to define another POJO class corresponding to ADDRESS table so that address objects can be stored and retrieved into the ADDRESS table.

```
import java.util.*;

public class Address{
   private int id;
   private String street;
   private String city;
   private String state;
   private String zipcode;

   public Address() {}
   public Address(String street, String city,
                  String state, String zipcode) {
      this.street = street;
      this.city = city;
      this.state = state;
      this.zipcode = zipcode;
   }
   public int getId() {
      return id;
   }
   public void setId( int id ) {
      this.id = id;
   }
   public String getStreet() {
      return street;
   }
   public void setStreet( String street ) {
      this.street = street;
   }
   public String getCity() {
      return city;
   }
   public void setCity( String city ) {
      this.city = city;
   }
   public String getState() {
      return state;
   }
   public void setState( String state ) {
      this.state = state;
   }
   public String getZipcode() {
      return zipcode;
   }
   public void setZipcode( String zipcode ) {
      this.zipcode = zipcode;
   }

}
```

Define Hibernate Mapping File:

Let us develop our mapping files which instructs Hibernate how to map the defined classes to the database tables. The <many-to-one> element will be used to define the rule to establish a many-to-one relationship between EMPLOYEE and ADDRESS entities.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator />
      </id>
      <property name="firstName" column="first_name" type="string"/>
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
      <many-to-one name="address" column="address"
       />
   </class>

   <class name="Address" table="ADDRESS">
      <meta attribute="class-description">
         This class contains the address detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator />
      </id>
      <property name="street" column="street_name" type="string"/>
      <property name="city" column="city_name" type="string"/>
      <property name="state" column="state_name" type="string"/>
      <property name="zipcode" column="zipcode" type="string"/>
   </class>

</hibernate-mapping>
```

You should save the mapping document in a file with the format <classname>.hbm.xml. We saved our mapping document in the file Employee.hbm.xml. You are already familiar with most of the mapping detail but let us see all the elements of mapping file once again:

- The mapping document is an XML document having **<hibernate-mapping>** as the root element which contains two <class> elements corresponding to each class.

- The **<class>** elements are used to define specific mappings from a Java classes to the database tables. The Java class name is specified using the **name** attribute of the class element and the database table name is specified using the **table** attribute.

- The **<meta>** element is optional element and can be used to create the class description.

- The **<id>** element maps the unique ID attribute in class to the primary key of the database table. The **name** attribute of the id element refers to the property in the class and the

column attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.

- The **<generator>** element within the id element is used to automatically generate the primary key values. Set the **class** attribute of the generator element is set to **native** to let hibernate pick up either **identity, sequence** or **hilo** algorithm to create primary key depending upon the capabilities of the underlying database.

- The **<property>** element is used to map a Java class property to a column in the database table. The **name** attribute of the element refers to the property in the class and the **column** attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.

- The **<many-to-one>** element is used to set the relationship between EMPLOYEE and ADDRESS entities. The **name** attribute is set to the defined variable in the parent class, in our case it is *address*. The **column** attribute is used to set the column name in the parent table EMPLOYEE.

Finally, we will create our application class with the main method to run the application. We will use this application to save few Employee's records alongwith their addresses and then we will apply CRUD operations on those records.

```
import java.util.*;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
   private static SessionFactory factory;
   public static void main(String[] args) {
      try{
         factory = new Configuration().configure().buildSessionFactory();
      }catch (Throwable ex) {
         System.err.println("Failed to create sessionFactory object." + ex);
         throw new ExceptionInInitializerError(ex);
      }
      ManageEmployee ME = new ManageEmployee();

      /* Let us have one address object */
      Address address = ME.addAddress("Kondapur","Hyderabad","AP","532");

      /* Add employee records in the database */
      Integer empID1 = ME.addEmployee("Manoj", "Kumar", 4000, address);

      /* Add another employee record in the database */
      Integer empID2 = ME.addEmployee("Dilip", "Kumar", 3000, address);

      /* List down all the employees */
      ME.listEmployees();

      /* Update employee's salary records */
      ME.updateEmployee(empID1, 5000);

      /* Delete an employee from the database */
      ME.deleteEmployee(empID2);

      /* List down all the employees */
      ME.listEmployees();

   }

   /* Method to add an address record in the database */
   public Address addAddress(String street, String city,
                             String state, String zipcode) {
      Session session = factory.openSession();
      Transaction tx = null;
      Integer addressID = null;
      Address address = null;
      try{
         tx = session.beginTransaction();
         address = new Address(street, city, state, zipcode);
         addressID = (Integer) session.save(address);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
      return address;
   }

   /* Method to add an employee record in the database */
   public Integer addEmployee(String fname, String lname,
                                    int salary, Address address){
      Session session = factory.openSession();
      Transaction tx = null;
      Integer employeeID = null;
      try{
         tx = session.beginTransaction();
         Employee employee = new Employee(fname, lname, salary, address);
         employeeID = (Integer) session.save(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
      return employeeID;
   }

   /* Method to list all the employees detail */
   public void listEmployees( ){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         List employees = session.createQuery("FROM Employee").list();
         for (Iterator iterator =
                           employees.iterator(); iterator.hasNext();){
            Employee employee = (Employee) iterator.next();
            System.out.print("First Name: " + employee.getFirstName());
            System.out.print("  Last Name: " + employee.getLastName());
            System.out.println("  Salary: " + employee.getSalary());
            Address add = employee.getAddress();
            System.out.println("Address ");
            System.out.println("\tStreet: " +  add.getStreet());
            System.out.println("\tCity: " + add.getCity());
            System.out.println("\tState: " + add.getState());
            System.out.println("\tZipcode: " + add.getZipcode());
         }
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
   /* Method to update salary for an employee */
   public void updateEmployee(Integer EmployeeID, int salary ){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         Employee employee =
                    (Employee)session.get(Employee.class, EmployeeID);
         employee.setSalary( salary );
         session.update(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
   /* Method to delete an employee from the records */
   public void deleteEmployee(Integer EmployeeID){
      Session session = factory.openSession();
      Transaction tx = null;
      try{
         tx = session.beginTransaction();
         Employee employee =
                   (Employee)session.get(Employee.class, EmployeeID);
         session.delete(employee);
         tx.commit();
      }catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      }finally {
         session.close();
      }
   }
}
```

Compilation and Execution:

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

- Create hibernate.cfg.xml configuration file as explained in configuration chapter.
- Create Employee.hbm.xml mapping file as shown above.
- Create Employee.java source file as shown above and compile it.
- Create Address.java source file as shown above and compile it.
- Create ManageEmployee.java source file as shown above and compile it.
- Execute ManageEmployee binary to run the program.

You would get following result on the screen, and same time records would be created in EMPLOYEE and ADDRESS tables.

```
$java ManageEmployee
.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........

First Name: Manoj  Last Name: Kumar  Salary: 4000
Address
        Street: Kondapur
        City: Hyderabad
        State: AP
        Zipcode: 532
First Name: Dilip  Last Name: Kumar  Salary: 3000
Address
        Street: Kondapur
        City: Hyderabad
        State: AP
        Zipcode: 532
```

```
First Name: Manoj  Last Name: Kumar  Salary: 5000
Address
         Street: Kondapur
         City: Hyderabad
         State: AP
         Zipcode: 532
```

If you check your EMPLOYEE and ADDRESS tables, they should have following records:

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+---------+
| id | first_name | last_name | salary | address |
+----+------------+-----------+--------+---------+
|  1 | Manoj      | Kumar     |   5000 | 5       |
+----+------------+-----------+--------+---------+
1 row in set (0.00 sec)

mysql> select * from ADDRESS;
+----+-------------+-----------+------------+---------+
| id | street_name | city_name | state_name | zipcode |
+----+-------------+-----------+------------+---------+
|  1 | Kondapur    | Hyderabad | AP         | 532     |
+----+-------------+-----------+------------+---------+
1 row in set (0.00 sec)

mysql>
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js