HIBERNATE - LIST MAPPINGS

A **List** is a java collection that stores elements in sequence and allow duplicate elements. The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index, and search for elements in the list. More formally, lists typically allow pairs of elements e1 and e2 such that e1.equals*e2*, and they typically allow multiple null elements if they allow null elements at all.

A List is mapped with a <list> element in the mapping table and initialized with java.util.ArrayList.

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,
   PRIMARY KEY (id)
);
```

Further, assume each employee can have one or more certificate associated with him/her. A List collection mapping needs an index column in the collection table. The index column defines the position of the element in the collection. So we will store certificate related information in a separate table which has following structure:

```
create table CERTIFICATE (
   id INT NOT NULL auto_increment,
   certificate_name VARCHAR(30) default NULL,
   idx INT default NULL,
   employee_id INT default NULL,
   PRIMARY KEY (id)
);
```

There will be **one-to-many** relationship between EMPLOYEE and CERTIFICATE objects.

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 collection of certificates in **List** variable.

```
import java.util.*;

public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;
   private List certificates;

   public Employee() {}
   public Employee(String fname, String lname, int salary) {
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
   }
   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 List getCertificates() {
      return certificates;
   }
   public void setCertificates( List certificates ) {
      this.certificates = certificates;
   }
}
```

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

```
public class Certificate{
   private int id;
   private String name;

   public Certificate() {}
   public Certificate(String name) {
      this.name = name;
   }
   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;
   }
}
```

Define Hibernate Mapping File:

Let us develop our mapping file which instructs Hibernate how to map the defined classes to the database tables. The <list> element will be used to define the rule for List collection used. The index of list is always of type integer and is mapped using the <list-index> element.

```
<?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>
      <list name="certificates" cascade="all">
         <key column="employee_id"/>
         <list-index column="idx"/>
         <one-to-many />
      </list>
      <property name="firstName" column="first_name" type="string"/>
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>

   <class name="Certificate" table="CERTIFICATE">
      <meta attribute="class-description">
         This class contains the certificate records.
      </meta>
      <id name="id" type="int" column="id">
         <generator />
      </id>
      <property name="name" column="certificate_name" 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 **<list>** element is used to set the relationship between Certificate and Employee classes. We used the **cascade** attribute in the <list> element to tell Hibernate to persist the Certificate objects at the same time as the Employee objects. The **name** attribute is set to the defined **List** variable in the parent class, in our case it is *certificates*.
- The **<key>** element is the column in the CERTIFICATE table that holds the foreign key to the parent object ie. table EMPLOYEE.
- The **<list-index>** element is used to keep the position of the element and map with the index column in the collection table. The index of the persistent list starts at zero. You could change this, for example, with <list-index base="1".../> in your mapping.
- The **<one-to-many>** element indicates that one Employee object relates to many

Certificate objects and, as such, the Certificate object must have a Employee parent associated with it. You can use either **<one-to-one>**, **<many-to-one>** or **<many-to-many>** elements based on your requirement. If we changed this example to use a many-to-many relationship, we would need an association table to map between the parent and the child objects.

Create Application Class:

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 certificates 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 a set of certificates for the first employee  */
      ArrayList set1 = new ArrayList();
      set1.add(new Certificate("MCA"));
      set1.add(new Certificate("MBA"));
      set1.add(new Certificate("PMP"));

      /* Add employee records in the database */
      Integer empID1 = ME.addEmployee("Manoj", "Kumar", 4000, set1);

      /* Another set of certificates for the second employee  */
      ArrayList set2 = new ArrayList();
      set2.add(new Certificate("BCA"));
      set2.add(new Certificate("BA"));

      /* Add another employee record in the database */
      Integer empID2 = ME.addEmployee("Dilip", "Kumar", 3000, set2);

      /* 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 employee record in the database */
   public Integer addEmployee(String fname, String lname,
                              int salary, ArrayList cert){
      Session session = factory.openSession();
      Transaction tx = null;
      Integer employeeID = null;
      try{
```

```
         tx = session.beginTransaction();
         Employee employee = new Employee(fname, lname, salary);
         employee.setCertificates(cert);
         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 iterator1 =
                           employees.iterator(); iterator1.hasNext();){
            Employee employee = (Employee) iterator1.next();
            System.out.print("First Name: " + employee.getFirstName());
            System.out.print("  Last Name: " + employee.getLastName());
            System.out.println("  Salary: " + employee.getSalary());
            List certificates = employee.getCertificates();
            for (Iterator iterator2 =
                        certificates.iterator(); iterator2.hasNext();){
               Certificate certName = (Certificate) iterator2.next();
               System.out.println("Certificate: " + certName.getName());
            }
         }
         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 Certificate.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 CERTIFICATE tables. You can see certificates has been sorted in reverse order. You can try by changing your mapping file, simply set **sort="natural"** and execute your program and compare the results.

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

First Name: Manoj  Last Name: Kumar  Salary: 4000
Certificate: MCA
Certificate: MBA
Certificate: PMP
First Name: Dilip  Last Name: Kumar  Salary: 3000
Certificate: BCA
Certificate: BA
First Name: Manoj  Last Name: Kumar  Salary: 5000
Certificate: MCA
Certificate: MBA
Certificate: PMP
```

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

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
| 51 | Manoj      | Kumar     |   5000 |
+----+------------+-----------+--------+
1 row in set (0.00 sec)

mysql> select * from CERTIFICATE;
+----+------------------+------+-------------+
| id | certificate_name | idx  | employee_id |
+----+------------------+------+-------------+
|  6 | MCA              |    0 |          51 |
|  7 | MBA              |    1 |          51 |
|  8 | PMP              |    2 |          51 |
+----+------------------+------+-------------+
3 rows in set (0.00 sec
```

```
mysql>
```

Alternatively, you could map a Java array instead of a list. A array mapping is virtually identical to the previous example, except with different element and attribute names
$< array > and < array - index >$. However, for reasons explained earlier, Hibernate applications rarely use arrays.