HIBERNATE - MAP MAPPINGS

A **Map** is a java collection that stores elements in key-value pairs and does not allow duplicate elements in the list. The Map interface provides three collection views, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings.

A Map is mapped with a <map> element in the mapping table and an unordered map can be initialized with java.util.HashMap.

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. We will store certificate related information in a separate table which has following structure:

```
create table CERTIFICATE (
   id INT NOT NULL auto_increment,
   certificate_type VARCHAR(40) default NULL,
   certificate_name VARCHAR(30) 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 Map 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 Map getCertificates() {
      return certificates;
   }
   public void setCertificates( Map 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 <map> element will be used to define the rule for the Map used.

```
<?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>
```

```
      <map name="certificates" cascade="all">
         <key column="employee_id"/>
         <index column="certificate_type" type="string"/>
         <one-to-many />
      </map>
      <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 **<map>** element is used to set the relationship between Certificate and Employee classes. We used the **cascade** attribute in the <map> 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 **Map** variable in the parent class, in our case it is *certificates*.
- The **<index>** element is used to represents the key parts of the key/value map pair. The key will be stored in the column certificate_type using a type of string.
- The **<key>** element is the column in the CERTIFICATE table that holds the foreign key to the parent object ie. table EMPLOYEE.
- 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.

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 an Employee record alongwith a list of certificates and then we will apply CRUD operations on that record.

```
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  */
      HashMap set = new HashMap();
      set.put("ComputerScience", new Certificate("MCA"));
      set.put("BusinessManagement", new Certificate("MBA"));
      set.put("ProjectManagement", new Certificate("PMP"));

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

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

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

      /* 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, HashMap 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());
            Map ec = employee.getCertificates();
            System.out.println("Certificate: " +
              (((Certificate)ec.get("ComputerScience")).getName()));
            System.out.println("Certificate: " +
              (((Certificate)ec.get("BusinessManagement")).getName()));
            System.out.println("Certificate: " +
              (((Certificate)ec.get("ProjectManagement")).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.

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

First Name: Manoj  Last Name: Kumar  Salary: 4000
Certificate: MCA
Certificate: MBA
Certificate: PMP
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 |
+----+------------+-----------+--------+
| 60 | Manoj      | Kumar     |   5000 |
+----+------------+-----------+--------+
1 row in set (0.00 sec)

mysql>select * from CERTIFICATE;
+----+--------------------+------------------+-------------+
| id | certificate_type   | certificate_name | employee_id |
+----+--------------------+------------------+-------------+
| 16 | ProjectManagement  | PMP              |          60 |
| 17 | BusinessManagement | MBA              |          60 |
| 18 | ComputerScience    | MCA              |          60 |
+----+--------------------+------------------+-------------+
3 rows in set (0.00 sec)

mysql>
```