Hibernate Mapping - 1

Mudassar Hakim
mudassar.trainer@gmail.com

Mapping Cardinality Relationship using Annotation : One-To-One

One-to-One: Annotated Domain class (1)

```
@Entity
@Table(name="student1",catalog="test")
public class Student1 {
private long studentId;
private String studentName;
private StudentLocation
studentLocation;
public Student1(String studentName) {
super();
this.studentName = studentName;
}
@Id
@GeneratedValue(strategy = IDENTITY)
@Column(name = "STUDENT_ID", nullable =
false)
public long getStudentId() {
return studentId;
}public void setStudentId(long
studentId) {
this.studentId = studentId;
}
```

```
@Column(name="STUDENT_NAME", length=45)
public String getStudentName() {
return studentName;
}
public void setStudentName(String
studentName) {
this.studentName = studentName;
}
@OneToOne(fetch = FetchType.LAZY,
mappedBy = "stud", cascade =
CascadeType.ALL)
public StudentLocation
getStudentLocation() {
return studentLocation;
}
public void
setStudentLocation(StudentLocation
studentLocation) {
this.studentLocation = studentLocation;
}}
```

One-to-One: Annotated Domain class (2)

```
@Entity
@Table(name="student_location",catalog=
"test")
public class StudentLocation {

private Student1 stud;
private long studentId;
private String street;
private String city;
private String state;
private String zipcode;
private long indexStudentId;

public StudentLocation(String string,
String string2, String string3, String
string4) {
// TODO Auto-generated constructor stub
setStreet(string); setCity(string2);
setState(string3); setZipcode(string4);
}
```

```
@OneToOne(fetch = FetchType.LAZY)
@JoinColumn(name="STUDENT_ID")
public Student1 getStud() {
return stud;
}
public void setStud(Student1 stud) {
this.stud = stud;
}
@GenericGenerator(name = "generator",
strategy = "foreign",
parameters = @Parameter( name =
"property" , value = "stud"))
@Id
@GeneratedValue(generator =
"generator")
@Column(name = "STUDENT_ID", unique =
true, nullable = false)
public long getStudentId() {
return studentId;
}
```

One-to-One: Annotated Domain class (2)

```
public void setStudentId(long
studentId) {
this.studentId = studentId;
}
@Column(name = "STREET", length = 45)
public String getStreet() {
return street;
}
public void setStreet(String street) {
this.street = street;
}
@Column(name = "CITY", length = 45)
public String getCity() {
return city;
}
public void setCity(String city) {
this.city = city;
}
```

```
@Column(name = "STATE", length = 45)
public String getState() {
return state;
}
public void setState(String state) {
this.state = state;
}
@Column(name = "ZIPCODE", length = 45)
public String getZipcode() {
return zipcode;
}
public void setZipcode(String zipcode){
this.zipcode = zipcode;
}
public void setIndexStudentId(long
indexStudentId) {
this.indexStudentId = indexStudentId;
}}
```

One-to-One Annotation: output tables

```
******** Table: student1 *******
+-------------+---------------------------------+
|  STUDENT_ID |           STUDENT_NAME          |
+-------------+---------------------------------+
| 1           | Mudassar                        |
+-------------+---------------------------------+

******** Table: student_location *******
+-------------+------------------+---------------+-------+------------+
|  STUDENT_ID |    STREET        |     CITY      | STATE |  ZIPCODE   |
+-------------+------------------+---------------+-------+------------+
| 1           | OMR Road         | Mumbai        | MH    | 400703     |
+-------------+------------------+---------------+-------+------------+
```

Mapping Cardinality Relationship using Annotation : One-To-Many

One-to-Many:Annotated Domain class1

```
@Entity
@Table(name = "employee", catalog =
"test")
public class Employee {
private long employeeId;
private String employeeName;
private Set<Phone> empPhNos=new
HashSet<Phone>();
@Id
@GeneratedValue(strategy = IDENTITY)
@Column(name = "EMPLOYEE_ID")
public long getEmployeeId() {
return employeeId;
}
public void setEmployeeId(long
employeeId) {
this.employeeId = employeeId;
}public void setEmployeeName(String
employeeName) {
this.employeeName = employeeName; }
```

```
@Column(name = "EMPLOYEE_NAME", length
= 45)
public String getEmployeeName() {
return employeeName;
}
@OneToMany(fetch = FetchType.LAZY,
mappedBy = "employee")
@Column(name="phoneId")
public Set<Phone> getEmpPhNos() {
return empPhNos;
}
public void setEmpPhNos(Set<Phone>
empPhNos) {
this.empPhNos = empPhNos;
}
public Employee(String employeeName) {
super();
this.employeeName = employeeName;
}}
```

One-to-Many:Annotated Domain class2

```
@Entity
@Table(name = "employee_phone1",
catalog = "test")
public class Phone {
private long phoneId;
private String phoneType;
private String phoneNumber;
private Employee employee;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "EMPLOYEE_ID")
public Employee getEmployee() {
return employee;
}
public void setEmployee(Employee
employee) {
this.employee = employee;
}public void setPhoneId(long phoneId) {
this.phoneId = phoneId; }
```

```
public Phone(String phoneType, String
phoneNumber) {
super();
this.phoneType = phoneType;
this.phoneNumber = phoneNumber;
}
@Id
@GeneratedValue(strategy = IDENTITY)
@Column(name = "PHONE_ID")
public long getPhoneId() {
return phoneId;
}
@Column(name = "PHONE_TYPE", length =
45)
public String getPhoneType() {
return phoneType;
}
.
.
.
```

One-to-Many Annotation: output tables

```
******** Table: employee_phone1 *******
+-------------+--------------+--------------+-------------+
|   PHONE_ID  | PHONE_TYPE   | PHONE_NUMBER | EMPLOYEE_ID |
+-------------+--------------+--------------+-------------+
| 1           | Aircel       | 9768888700   | 1           |
| 2           | Airtel       | 9768888777   | 1           |
+-------------+--------------+--------------+-------------+

******** Table: employee *******
+-------------+---------------------------------+
| EMPLOYEE_ID |          EMPLOYEE_NAME          |
+-------------+---------------------------------+
| 1           | Mudassar                        |
+-------------+---------------------------------+
```

Mapping Cardinality Relationship : Component

Component Mapping

- Since the House and Address entities are strongly related (composition relation), it is better to store them in a single table. The relational model is shown below.
- House.hbm.xml is used to create the House table.
- The component element is used to map all the Address entity fields to the House table. In Hibernate terms the Address entity is called the component and it cannot have its own primary key, it uses the primary key of the enclosing House entity.

Component Mapping: hbm.xml

```
<hibernate-mapping>
    <class name="com.mudassar.myexample.domain.House" table="HOUSE">
        <id name="houseId" type="int">
            <column name="HOUSE_ID" /> <generator class="native" />
        </id>
        <property name="name" type="java.lang.String">
            <column name="NAME" />
        </property>
        <component name="houseAddress"
class="com.mudassar.myexample.domain.Address">
          <property name="street" type="java.lang.String">
              <column name="STREET" />    </property>
          <property name="city" type="java.lang.String">
              <column name="CITY" />     </property>
          <property name="state" type="java.lang.String">
              <column name="STATE" />   </property>
          <property name="zipcode" type="java.lang.String">
              <column name="ZIPCODE" />   </property>
        </component>
    </class> </hibernate-mapping>
```

Component Mapping: Annotations

```
@Entity
@Table(name = "STUDENT")
public class Student {
private long studentId;.
private String studentName;
private Address studentAddress;
public Student() {}
public Student(String studentName,
Address studentAddress) {
this.studentName = studentName;
this.studentAddress = studentAddress;
}
@Id
@GeneratedValue
@Column(name = "STUDENT_ID")
public long getStudentId() {
return this.studentId;
}
public void setStudentId(long studentId) {
this.studentId = studentId;
}
@Column(name = "STUDENT_NAME",
nullable = false, length = 100)
public String getStudentName() {
return this.studentName;
}
public void setStudentName(String
studentName) {
this.studentName = studentName;
}
@Embedded
public Address getStudentAddress() {
return this.studentAddress;
}
public void setStudentAddress(Address
studentAddress) {
this.studentAddress = studentAddress;
}}
```

Mapping Inheritance: 3 Different Ways

Inheritance Relationship Table Representations

- 3 different ways
  - One table for each class hierarchy
  - One table for each subclass
  - One table per each concrete class implementation
- Each of these techniques has different costs and benefits

Example Class Hierarchy

- Book class is parent class

```
public class Book {
    int id;
    String title;
    String artist;
    Date purchaseDate;
    double cost;
    ..
}
```

- SpecialEditionBook is a child class of Book class

```
public class SpecialEditionBook extends Book {
    private String newfeatures;
    ..
}
```

- InternationalBook is a child class of Book class

```
public class InternationalBook extends Book {
    private String languages;
    private int region;
    ..
}
```

Mapping Inheritance:
1 Table for the Class Hierarchy

One Table per Class Hierarchy

- A single table for the whole class hierarchy
  - The table contains fields of all classes
- Discriminator column contains key to identify the base type
  - The column indicates which type/subtype a row belongs
- Advantages
  - Offers best performance even for in the deep hierarchy since single select may suffice
- Disadvantages
  - Change to any class in the hierarchy requires a change of the table
  - Wasted space

One Table per Class Hierarchy

- How to define the mapping
  - Use <*subclass*> element with *extends* and *discriminator-value* attributes

One Table per Class Hierarchy: Parent (Book.hbm.xml)

```
<class name="Book" table="Book"
       discriminator-value="Book">
    <id name="id" type="integer"  unsaved-value="0">
        <generator class="increment"/>
    </id>

    <!-- Parent class mapping file specifies the
         discriminator column -->
    <discriminator column="Book_type" type= "string"/>

    <property name="title"/>
    <property name="artist"/>
    <property name="purchasedate" type="date"/>
    <property name="cost" type="double"/>

</class>
```

One Table per Class Hierarchy: Child (SpecialEditionBook.hbm.xml)

```
<hibernate-mapping>

   <subclass name="SpecialEditionBook"
          extends="Book"
          discriminator-value="SpecialEditionBook">
      <property name="newfeatures" type="string" />
   </subclass>

</hibernate-mapping>
```

One Table per Class Hierarchy: Child (InternationalBook.hbm.xml)

```
<hibernate-mapping>

   <subclass name="InternationalBook"
          extends="Book"
          discriminator-value="InternationalBook">
      <property name="languages"/>
      <property name="region" />
   </subclass>

</hibernate-mapping>
```

One Table per Class Hierarchy

Discriminator column

```
******** Table: Book *******
+-------------+----------------------+----------------------+--------------+------------------------+----------------------+----------------------+-------------+-----------------------+
|      ID     |        TITLE         |        ARTIST        | PURCHASEDATE |          COST          |      NEWFEATURES     |       LANGUAGES      |    REGION   |       BOOK_TYPE       |
+-------------+----------------------+----------------------+--------------+------------------------+----------------------+----------------------+-------------+-----------------------+
| 1           | Book                 | R                    | 2008-04-11   | 9.99                   |                                                             Book                  |
| 2           | sBook                | R                    | 2008-04-11   | 9.99                   | W                    |                                      SpecialEditionBook    |
| 3           | IBook                | R                    | 2008-04-11   | 9.99                   |                        S                    | 4           | InternationalBook     |
+-------------+----------------------+----------------------+--------------+------------------------+----------------------+----------------------+-------------+-----------------------+
```

Discriminator value

Mapping Inheritance:
1 Table for Subclass

One Table per Subclass

- One table for each class in the hierarchy
  - Common fields (fields of the parent class) are maintained in the parent table
  - Subclass table maintain only the subclass specific fields
  - Foreign key relationship exists between common table and subclass tables
- Advantages
  - Does not require complex changes to the schema when a class is modified
  - Works well with shallow hierarchy
- Disadvantages
  - Can result in poor performance with deep hierarchy – as hierarchy grows, the number of joins required to construct a leaf class also grows

One Table per Subclass

- How to define the mapping
  - Use *<joined-subclass>* element with *extends* attribute in the mapping file of the subclass

One Table per Subclass: Parent (Book.hbm.xml)

```
<!-- Mapping file of Parent class has no hierarchical
     relationship specific element -->
<class name="Book" table="Book">
    <id name="id" type="integer" unsaved-value="0">
        <generator class="increment" />
    </id>

    <property name="title" />
    <property name="artist" />
    <property name="purchasedate" type="date" />
    <property name="cost" type="double" />

</class>
```

One Table per Subclass - Child (SpecialEditionBook.hbm.xml)

```
<joined-subclass name="SpecialEditionBook"
                 extends="Book"
                 table="SpecialEditionBook">
    <key column="id" />
    <property name="newfeatures" type="string" />
</joined-subclass>
```

One Table per Subclass - Child (InternationalBook.hbm.xml)

```
<joined-subclass name="InternationalBook"
                 extends="Book"
                 table="InternationalBook">
    <key column="id" />
    <property name="languages"/>
    <property name="region"/>
</joined-subclass>
```

One Table per Subclass

```
******** Table: Book *******
+-------------+----------------------+----------------------+--------------+------------------------+
|     ID      |        TITLE         |        ARTIST        | PURCHASEDATE |          COST          |
+-------------+----------------------+----------------------+--------------+------------------------+
| 1           | Book                 | R                    | 2008-04-11   | 9.99                   |
| 2           | sBook                | R                    | 2008-04-11   | 9.99                   |
| 3           | IBook                | R                    | 2008-04-11   | 9.99                   |
+-------------+----------------------+----------------------+--------------+------------------------+

******** Table: SpecialEditionBook *******
+-------------+--------------------------------+
|     ID      |          NEWFEATURES           |
+-------------+--------------------------------+
| 2           | W                              |
+-------------+--------------------------------+

******** Table: InternationalBook *******
+-------------+--------------------------------+-------------+
|     ID      |           LANGUAGES            |   REGION    |
+-------------+--------------------------------+-------------+
| 3           | S                              | 4           |
+-------------+--------------------------------+-------------+
```

Common fields are maintained in the parent table

ID of the child table points to the row of the the parent table.

Mapping Inheritance: 1 Table for Concrete Class

One Table per Concrete Class

- Map each of the concrete classes as normal persistent class
- Pros
  - Easiest to implement
- Cons
  - Data belonging to a parent class is scattered across a number of different tables, which represent concrete classes
  - A query couched in terms of parent class is likely to cause a large number of select operations
  - Changes to a parent class can touch large number of tables
  - This scheme is not recommended for most cases

One Table per Concrete Class

- How to define the mapping
  - The mapping of the subclass repeats the properties of the parent class

One Table per Concrete Class

```
<hibernate-mapping>
    <class name="Book" table="cd" discriminator-value="cd">
        <id name="id" type="integer" unsaved-value="0">
            <generator  class="increment"/>
        </id>
        <property name="title"/>
        <property name="artist"/>
        <property name="purchasedate" type="date"/>
        <property name="cost" type="double"/>
    </class>

    <class  name="SpecialEditionBook" table="secd">
        <id name="id" type="integer" unsaved-value="0">
            <generator  class="increment"/>
        </id>
        <property name="title"/>
        <property name="artist"/>
        <property name="purchasedate" type="date"/>
        <property name="cost" type="double"/>
        <property name="newfeatures" type="string"/>
    </class>


</hibernate-mapping>
```

The mapping of the subclass repeats the properties of the parent class

One Table per Concrete Class

```
******** Table: Book *******
+-------------+----------------------+----------------------+--------------+-------------------------+
|      ID     |        TITLE         |        ARTIST        | PURCHASEDATE |          COST           |
+-------------+----------------------+----------------------+--------------+-------------------------+
| 1           | Book                 | R                    | 2008-04-11   | 9.99                    |
+-------------+----------------------+----------------------+--------------+-------------------------+

******** Table: SpecialEditionBook *******
+-------------+----------------------+----------------------+--------------+-------------------------+----------------------+
|      ID     |        TITLE         |        ARTIST        | PURCHASEDATE |          COST           |     NEWFEATURES      |
+-------------+----------------------+----------------------+--------------+-------------------------+----------------------+
| 1           | sBook                | R                    | 2008-04-11   | 9.99                    | W                    |
+-------------+----------------------+----------------------+--------------+-------------------------+----------------------+

******** Table: InternationalBook *******
+-------------+----------------------+----------------------+--------------+-------------------------+----------------------+-------------+
|      ID     |        TITLE         |        ARTIST        | PURCHASEDATE |          COST           |      LANGUAGES       |   REGION    |
+-------------+----------------------+----------------------+--------------+-------------------------+----------------------+-------------+
| 1           | IBook                | R                    | 2008-04-11   | 9.99                    | S                    | 4           |
| 2           | IBook                | R                    | 2008-04-11   | 100.9                   | T                    | 3           |
+-------------+----------------------+----------------------+--------------+-------------------------+----------------------+-------------+
```

Thank you!

Mudassar Hakim
mudassar.trainer@gmail.com