Hibernate Mapping

Mudassar Hakim
mudassar.trainer@gmail.com

Unidirectional Mapping Cardinality Relationship: One-To-Many

One-To-Many Relationship Mapping

1. <set> - Set (no duplicate)
2. <list> - List (order/index)
3. <array> - Array (order/index)
4. <bag> - (duplicate allowed/no order)

Unidirectional Mapping Cardinality Relationship: One-To-Many: Using <set>

1. One-to-Many relationship: Using <set> in mapping file (1)

- An event has many speakers
- Mapping file of Parent class - *Event.hbm.xml*

```
<class name="Event" table="events">
      <id name="id" column="uid" type="long" unsaved-value="null"
          <generator class="increment"/>
      </id>
      <property name="name" type="string" length="100"/>

      <set name="speakers" cascade="all">
          <key column="event_id"/>
          <one-to-many class="Speaker"/>
      </set>
</class>
```

Name of field in the parent class

Foreign key in the child table

Child class

1. One-to-Many relationship: Using <set> in mapping file (2)

- An event has many speakers
- Mapping file of Child class - *Speaker.hbm.xml*

```
<class name="Speaker" table="speakers">
    <id name="id" column="uid" type="long">
        <generator class="increment"/>
    </id>
    <property name="firstName" type="string" length="20"/>
    <property name="lastName" type="string" length="20"/>
</class>
```

One to Many relationship: Using Set in Domain Class

- An event has many speakers and attendees

```
public class Event {

  private Long id;
  private String name;
  private Date startDate;
  private int duration;

  // Event has one-to-many relationship with Speaker
  private Set speakers;

  // ...
```

One to Many relationship: Creating Object Instance

- An event has many speakers and attendees

```
// Create an Event object which has one to many relationship
// with Speaker objects.

Event event = new Event();
event.setName("Spring Conference");
event.setSpeakers(new HashSet());
event.getSpeakers().add(new Speaker("Mudassar", "Hakim"));
event.getSpeakers().add(new Speaker("Dave", "Smith"));
event.getSpeakers().add(new Speaker("Rod", "Johnson"));

session.saveOrUpdate(event);
```

One to Many relationship: Using <set> in the mapping file

- Tables

```
******** Table: events *******
+-------------+----------------------+------------+-------------+-------------+
|     UID     |         NAME         | START_DATE |   DURATION  | LOCATION_ID |
+-------------+----------------------+------------+-------------+-------------+
| 1           | Spring Conference    |             0           |
| 2           | Hibernate Conference |             0           |
+-------------+----------------------+------------+-------------+-------------+
```

Foreign key

```
******** Table: speakers *******
+-------------+-------------+-----------------------+-----------------------+
|     UID     |   EVENT_ID  |       FIRSTNAME       |       LASTNAME        |
+-------------+-------------+-----------------------+-----------------------+
| 1           | 1           | Mudassar              | Hakim                 |
| 2           | 1           | Dave                  | Smith                 |
| 3           | 2           | Rod                   | Johnson               |
| 4           | 2           | Daniel                | Jones                 |
| 5           | 2           | James                 | Gosling               |
+-------------+-------------+-----------------------+-----------------------+
```

Unidirectional Mapping Cardinality Relationship: One-To-Many: Using <list>

2. One to Many relationship: Using <list> in mapping file

- Group has many stories
- Mapping file of parent class - *Group.hbm.xml*

```
<class name="Group" table="grouptable">
     <id name="id" unsaved-value="0">
         <generator class="increment" />
     </id>

     <list name="stories" cascade="all">
         <key column="parent_id" />
         <!-- index in a single list -->
         <index column="idx" />
         <one-to-many class="Story" />
     </list>
     <property name="name" type="string" />
 </class>
```

2. One to Many relationship: Using <list> in mapping file (2)

- Group has many stories
- Mapping file of child class - *Story.hbm.xml*

```
<class name="Story" table="story">
    <id name="id" unsaved-value="0">
        <generator class="increment" />
    </id>
    <property name="info" />
</class>
```

One to Many relationship: Using List in Domain Class

- Group has many stories

```
public class Group {

    private int id;
    private String name;

    private List stories;

    public void setStories(List l) {
        stories = l;
    }

    public List getStories() {
        return stories;
    }
    // ...
```

One to Many relationship: Creating Object Instances

- Group has many stories

```
ArrayList list = new ArrayList();
list.add(new Story("Tom Jones"));
list.add(new Story("Beatles"));
list.add(new Story("Elvis"));

Group sp = new Group("Singers");
sp.setStories(list);

ArrayList list2 = new ArrayList();
list2.add(new Story("Bill Clinton"));
list2.add(new Story("Ronald Reagan"));

Group sp2 = new Group("Politicians");
sp2.setStories(list2);
```

One to Many relationship: Using <list> in the mapping file

- Tables

```
******** Table: grouptable *******
+-------------+----------------------+
|     ID      |         NAME         |
+-------------+----------------------+
| 1           | Singers              |
| 2           | Politicians          |
+-------------+----------------------+

******** Table: story *******
+-------------+---------------------------------+-------------+-------------+
|     ID      |              INFO               |     IDX     |  PARENT_ID  |
+-------------+---------------------------------+-------------+-------------+
| 1           | Tom Jones                       | 0           | 1           |
| 2           | Beatles                         | 1           | 1           |
| 3           | Elvis                           | 2           | 1           |
| 4           | Bill Clinton                    | 0           | 2           |
| 5           | Ronald Reagan                   | 1           | 2           |
+-------------+---------------------------------+-------------+-------------+
```

Unidirectional Mapping Cardinality Relationship: One-To-Many: Using <array>

3. One to Many relationship: Using <array> in mapping file

- Group has many stories
- Mapping file of parent class - *Group.hbm.xml*

```
<class name="Group" table="grouptable">
    <id name="id" unsaved-value="0">
        <generator class="increment"/>
    </id>

    <array name="stories" cascade="all">
        <key column="parent_id"/>
        <index column="idx"/>
        <one-to-many class="Story"/>
    </array>
    <property name="name" type="string"/>
</class>
```

3. One to Many relationship: Using <array> in mapping file (2)

- Group has many stories
- Mapping file of child class - *Story.hbm.xml*

```
<class name="Story" table="story">
    <id name="id" unsaved-value="0">
        <generator class="increment"/>
    </id>
    <property name="info"/>
</class>
```

One to Many relationship: Using an array in Domain Class

- Group has many stories

```
public class Group {

    private int id;
    private String name;

    // Group object has an array of Story objects
    private Story[] stories;

    public void setStories(Story[] l) {
        stories = l;
    }

    public Story[] getStories() {
        return stories;
    }

    // ...
```

One to Many relationship: Creating an Object Instance

- Group has many stories

```
// Create an Group object which has one to many relationship
// with Story objects.

Group sp = new Group("Group Name");
sp.setStories(new Story[]{new Story("Story Name 1"),
                          new Story("Story Name 2")});
```

One to Many relationship: Using <array> in the mapping file

- Tables

```
******** Table: grouptable *******
+-------------+----------------------+
|      ID     |         NAME         |
+-------------+----------------------+
| 1           | Singers              |
| 2           | Politicians          |
+-------------+----------------------+

******** Table: story *******
+-------------+---------------------------------+-------------+-------------+
|      ID     |              INFO               |     IDX     |  PARENT_ID  |
+-------------+---------------------------------+-------------+-------------+
| 1           | Tom Jones                       | 0           | 1           |
| 2           | Beatles                         | 1           | 1           |
| 3           | Bill Clinton                    | 0           | 2           |
| 4           | Ronald Reagan                   | 1           | 2           |
+-------------+---------------------------------+-------------+-------------+
```

Unidirectional Mapping Cardinality Relationship: One-To-Many: Using <bag>

4. One to Many relationship: Using <bag> in mapping file

- Using bag for association mapping, the domain objects in the collection could be duplicate (non-set) and without order (non-list)
- *Group.hbm.xml*

```
<class name="Group" table="grouptable">

    <id name="id" unsaved-value="0">
        <generator class="increment"/>
    </id>

    <bag name="stories" cascade="all">
        <key column="parent_id"/>
        <one-to-many class="Story"/>
    </bag>

    <property name="name" type="string"/>

</class>
```

One to Many relationship: Using an List in Domain Class

- Group has many stories

```
public class Group {

    private int id;
    private String name;
    private List stories;

    public void setStories(List l) {
        stories = l;
    }

    public List getStories() {
        return stories;
    }

    // ...
```

One to Many relationship: Creating an Object Instance

- Group has many stories

```
// Create an Group object which has one to many relationship
// with Story objects.

ArrayList list = new ArrayList();
list.add(new Story("Story Name 1"));
list.add(new Story("Story Name 2"));
Group sp = new Group("Group Name");
sp.setStories(list);
```

One to Many relationship: Using <bag> in the mapping file

- Tables

```
******** Table: grouptable *******
+-------------+----------------------+
|      ID     |         NAME         |
+-------------+----------------------+
| 1           | Singers              |
| 2           | Polticians           |
+-------------+----------------------+

******** Table: story *******
+-------------+--------------------------------+-------------+
|      ID     |              INFO              |  PARENT_ID  |
+-------------+--------------------------------+-------------+
| 1           | Tom Jones                      | 1           |
| 2           | Beatles                        | 1           |
| 3           | Ronald Reagan                  | 2           |
+-------------+--------------------------------+-------------+
```

Unidirectional Mapping Cardinality Relationship: One-To-One

One to One Relationship

- Expresses a relationship between two classes where each instance of the first class is related to a single instance of the second class
- One student living at single address
- Relationship mapping is only given in Student.hbm.xml

One-to-One relationship:mapping file (1)

```
<class name="com.mudassar.myexample.Student" table="student">
        <id name="studentId" type="long">
            <column name="STUDENT_ID" />
            <generator class="native" />
        </id>
        <property name="studentName" type="java.lang.String">
            <column name="STUDENT_NAME" />
        </property>
        <many-to-one name="studentAddress"
class="com.mudassar.myexample.Address" cascade="all"
unique="true">
            <column name="STUDENT_ADDRESS" />
        </many-to-one>
    </class>
```

One-to-One relationship:mapping file (2)

```
<class name="com.mudassar.myexample.Address" table="ADDRESS">
       <id name="addressId" type="long">
           <column name="ADDRESSID" /> <generator class="native" />
       </id>
       <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>
   </class>
```

One to One relationship: Tables

```
******** Table: address *******
+----------+--------------+--------------+--------+------------------------+
|ADDRESS_ID|   STREET     |      CITY    |  STATE |        ZIPCODE         |
+----------+--------------+--------------+--------+------------------------+
| 1        | OMR Road     | Mumbai       | MH     | 400703                 |
| 2        | Kings Road   | Delhi        | HM     | 400023                 |
+----------+--------------+--------------+--------+------------------------+

******** Table: student *******
+------------+-----------------------------+-----------------------------+
|  STUDENT_ID |          STUDENT_NAME       |         STUDENT_ADDRESS     |
+------------+-----------------------------+-----------------------------+
| 1          | Mudassar                    | 1                           |
| 2          | Ritesh                      | 2                           |
+------------+-----------------------------+-----------------------------+
```

Bidirectional Mapping Cardinality Relationship: One-To-One

One-to-One bidirectional relationship: mapping file (1)

```
<class name="com.myexample.domain.Student" table="STUDENT">
        <id name="studentId" type="long">
            <column name="STUDENT_ID" />
            <generator class="native" />
        </id>
        <property name="studentName" type="java.lang.String">
            <column name="STUDENT_NAME" />
        </property>
        <one-to-one name="studentAddress"
class="com.myexample.domain.Address" cascade="save-update">
        </one-to-one>
    </class>
```

One-to-One bidirectional relationship: mapping file (2)

```
<class name="com.myexample.domain.Address" table="address">
        <id name="addressId" type="long"> <column name="ADDRESS_ID" />
            <generator class="foreign" >
               <param name="property">stud</param>
            </generator>
        </id>
        .
        .
        <property name="zipcode" type="java.lang.String">
            <column name="ZIPCODE" />
        </property>
        <one-to-one name="stud" class="com.myexample.domain.Student"
constrained="true"></one-to-one>
    </class>
```

One to One bidirectional relationship: Tables

```
******** Table: address *******
+----------+--------------+--------------+-------+------------------------+
|ADDRESS_ID|   STREET     |      CITY    | STATE |       ZIPCODE          |
+----------+--------------+--------------+-------+------------------------+
| 1        | OMR Road     | Mumbai       | MH    | 400703                 |
| 2        | Kings Road   | Delhi        | HM    | 400023                 |
+----------+--------------+--------------+-------+------------------------+

******** Table: student *******
+------------+------------------------------+
|  STUDENT_ID |          STUDENT_NAME       |
+------------+------------------------------+
| 1          | Mudassar                     |
| 2          | Ritesh                       |
+------------+------------------------------+
```

Bidirectional Mapping Cardinality Relationship: One-To-Many

One-to-Many bidirectional relationship: mapping file (1)

```
<hibernate-mapping>
    <class name="com.myexample.domain.Employee" table="employee">
        <id name="employeeId" type="long">
            <column name="EMPLOYEE_ID" /> <generator class="native" />
        </id>
        <property name="employeeName" type="java.lang.String">
            <column name="EMPLOYEE_NAME" />
        </property>
        <set name="empPhNos" table="employee_phone1" inverse="true">
            <key> <column name="EMPLOYEE_ID" /> </key>
            <one-to-many class="com.myexample.domain.Phone" />
        </set>
    </class>
</hibernate-mapping>
```

One-to-Many bidirectional relationship: mapping file (2)

```
<hibernate-mapping>
    <class name="com.myexample.domain.Phone" table="employee_phone1">
        <id name="phoneId" type="long">
            <column name="PHONE_ID" /> <generator class="native" />
        </id>
        <property name="phoneType" type="java.lang.String">
            <column name="PHONE_TYPE" />
        </property>
        <property name="phoneNumber" type="java.lang.String">
            <column name="PHONE_NUMBER" />
        </property>
        <many-to-one name="emp" class="com.myexample.domain.Employee" >
            <column name="EMPLOYEE_ID" />
        </many-to-one>
    </class>
</hibernate-mapping>
```

One to Many bidirectional relationship: 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                      |
+-------------+-------------------------------+
```

Hibernate Association: Cascade & Inverse

A simple case of parent – child relationship. Here are the Parent mapping file.

- Parent.hbm.xml

```
 <hibernate-mapping>
   <class name="com.mudassar.myexample.Parent"  table="parent">
      <id name="id" type="long"  column="pid">
        <generator class="native"  />
      </id>
      <set name="children" inverse="false"  cascade="none">
         <key column="pid" />
         <one-to-many  class="com.mudassar.myexample.Child"  />
     </set>
  </class>
</hibernate>
```

Corresponding Java file:

- Parent.java

```
import java.util.Set;
public class Parent {
   private long id;
   private Set Children;
   public Set getChildren() {return Children;}
   public void setChildren(Set children) { Children children;}
   public long getId() {return id; }
   public void setId(long id) {this.id = id;}
}
```

Here are the Child mapping file.

- Child.hbm.xml

```
<hibernate-mapping>
   <class name="com.mudassar.myexample.Child"  table="child">
      <id name="id" type="long"  column="cid">
          <generator class="native" />
      </id>
      <property name="firstName" column="firstName" />
      <property name="lastName" column="lastName"  />
  </class>
</hibernate>
```

Corresponding Java file:

- Child.java

```
public class Child {
  private long id;
  private String firstName;
  private String lastName;
  public String getFirstName() {return firstName; }
  public void setFirstName(String firstName) {this.firstName =  firstName; }
  public long getId() {return id; }
  public void setId(long id) {this.id = id;}
  public String getLastName() {return lastName;}
  public void setLastName(String lastName) {this.lastName = lastName;}
}
```

DDL for parent & child

```
create table child (
   cid bigint not null  auto_increment,
   firstName varchar(12),
   lastName varchar(12),
   pid  bigint,
  primary key (cid)
);

create table parent (
   pid bigint not  null auto_increment,
   primary key (pid)
);
```

1. Uni directional one-to-many association

- This is the simplest possible case I could think of. As mentioned it’s uni-directional mapping. The parent object knows about the child object but the child object doesn’t know anything about the parent object.
- In this case inverse is ‘false’. Simply stated it means that the onus of setting up the relationship is with the parent object. As we move along the test cases the relevance of inverse attribute would become clearer. So hang on as we move along the test cases.

1.1 Both parent and child are persisted

- **Source code:**

```
Parent parent = new Parent();
Child child = new  Child();  child.setFirstName("John");  child.setLastName("Smith");
Set  children = new  HashSet();
children.add(child);   parent.setChildren(children);
session.saveOrUpdate(child);
session.saveOrUpdate(parent);
```

- **Query**

```
Hibernate: insert into child (firstName, lastName) values (?,  ?)
Hibernate: insert into parent values ( )
Hibernate: update child set  pid=? where cid=?
```

- **Analysis**

  The result is an insertion of a record in both the parent and the child table. Pay special attention to the sequence of events here. First the child record is inserted, then the parent record is inserted and then the primary key of the parent record is put into the child record.

1.2 Only parent is persisted

- **Source code:**

```
Parent parent = new Parent();
Child child = new  Child();  child.setFirstName("John");  child.setLastName("Smith");
Set  children = new  HashSet();
children.add(child);   parent.setChildren(children);
// session.saveOrUpdate(child);
session.saveOrUpdate(parent);
```

- **Query**

~~Hibernate: insert into child (firstName, lastName) values (?, ?)~~

```
Hibernate: insert into parent values ( )
Hibernate: update child set  pid=? where cid=?
```

- **Error**

```
SEVERE: Could not synchronize database state with  session
net.sf.hibernate.TransientObjectException: object references an  unsaved transient
instance - save the transient instance before flushing:  example.Child at
net.sf.hibernate.impl.SessionImpl.throwTransientObjectException(SessionImpl.java:2788)
```

1.2 Only parent is persisted

- **Analysis**

  Hibernate inserts a record in the parent table.Then Hibernate realizes that there is a relationship with a child. Since inverse is false , the onus of implementing the association falls on the shoulders of the parent object . Now hibernate attempts to put the value of the primary key of the parent record on the child object. But realizes that the child object is still not persisted. Hence the exception. Look at the exception message. Hibernate is saying that the child object is still transient and it can’t establish the relationship. What do you think will happen to the parent record because of the exception. Will that inserted record be rolled back ? .The answer is No. The parent record will be there.

1.3 Only child is persisted

- **Source code:**

  Parent parent = new Parent();

  Child child = new Child(); child.setFirstName("John"); child.setLastName("Smith");

  Set children = new HashSet();

  children.add(child);   parent.setChildren(children);

  session.saveOrUpdate(child);

  // session.saveOrUpdate(parent);

- **Query**

  Hibernate: insert into child (firstName, lastName) values (?, ?)

  ~~Hibernate: insert into parent values ( )~~

  ~~Hibernate: update child set pid=? where cid=?~~

- **Analysis**

  In this case only a child record is inserted. No record was inserted in the parent table. Why no exception? Because child doesn’t even know that there is any one out there trying to have an association with it. Inverse is false- it means the onus of setting the relationship falls on the parent object. Child object gets persisted oblivious to any association issue.

2. Uni directional one-to-many association with cascade.

- In this case everything remains same except that the cascade option has been changed from none to all in the parent.hbm.xml. Now that cascade is turned on if a parent record is persisted then the child record will automatically be persisted by the virtue of cascade.

```
<hibernate-mapping>
   <class name="com.mudassar.myexample.Parent"  table="parent">
      <id name="id" type="long"  column="pid">
        <generator class="native"  />
      </id>
      <set name="children" inverse="false"  cascade="all">
         <key column="pid" />
         <one-to-many  class="com.mudassar.myexample.Child"  />
     </set>
  </class>
</hibernate>
```

2.1 Parent has reference to the child. Both are persisted.

- **Source code:**

  ```
  Parent parent = new Parent();
  Child child = new  Child();  child.setFirstName("John");  child.setLastName("Smith");
  Set  children = new  HashSet();
  children.add(child);
  parent.setChildren(children);
  session.saveOrUpdate(child);
  session.saveOrUpdate(parent);
  ```

- **Query**

  ```
  Hibernate: insert into child (firstName, lastName) values (?,  ?)
  Hibernate: insert into parent values ( )
  Hibernate: update child set  pid=? where cid=?
  ```

- **Analysis**

  Both the parent and the child record are successfully persisted. This test doesn’t prove much since we are persisting both the parent and the child.

2.2 Only parent is persisted

- **Source code:**

```
Parent parent = new Parent();
Child child = new  Child();  child.setFirstName("John"); child.setLastName("Smith");
Set  children = new  HashSet();
children.add(child);
parent.setChildren(children);
// session.saveOrUpdate(child);
session.saveOrUpdate(parent);
```

- **Query**

```
Hibernate: insert into parent values ( )
Hibernate: insert into  child (firstName, lastName) values (?, ?)
Hibernate: update child set pid=?  where cid=?
```

- **Analysis**:

  The parent record is persisted. The child record is persisted with the foreign key. It’s all good. This is interesting. In this case we are not explicitly persisting the child and still the child record got persisted. Last time we did that case 1.2 we had an error. This shows what ‘cascade’ does. It persists all the child records anytime it is asked to be persisted. It takes so much work off the shoulders of the developers.

  Pay attention to the query. It’s the same as case 1.1. It simply means that instead of developers doing the job, some of the tasks are now being done by Hibernate.

2.3 Only child is persisted

- **Source code:**

  Parent parent = new Parent();

  Child child = new Child(); child.setFirstName("John"); child.setLastName("Smith");

  Set children = new HashSet();

  children.add(child);   parent.setChildren(children);

  session.saveOrUpdate(child);

  // session.saveOrUpdate(parent);

- **Query**

  ~~Hibernate: insert into parent values ( )~~

  Hibernate: insert into child (firstName, lastName) values (?, ?)

  ~~Hibernate: update child set pid=? where cid=?~~

- **Analysis**

  In this case the result is exactly similar to case case 1.3 for the same reason. Child object has no idea that it is part of association and is happily persisted. In this case cascade had no effect since the parent was not even persisted.

3. Bi directional one-to-many association.

- Now we will make the association bi-directional. No changes will be done to the parent.hbm.xml. The child.hbm.xml will be changed to have a reference to the parent. It means the POJO for Child will also change. The cascade setting for the parent.hbm.xml is reset to none.

```
<hibernate-mapping>
   <class name="com.mudassar.myexample.Parent"  table="parent">
      <id name="id" type="long"  column="pid">
        <generator class="native"  />
      </id>
      <set name="children" inverse="false"  cascade="none">
         <key column="pid" />
         <one-to-many  class="com.mudassar.myexample.Child"  />
     </set>
  </class>
</hibernate>
```

Here are the Child mapping file.

- Child.hbm.xml

```
<hibernate-mapping>
   <class name="com.mudassar.myexample.Child"  table="child">
      <id name="id" type="long"  column="cid">
         <generator class="native" />
      </id>
      <property name="firstName" column="firstName" />
      <property name="lastName" column="lastName" />
      <many-to-one name="parent"
              class="com.mudassar.myexample.Parent" column="pid"  />
    </class>
</hibernate>
```

Corresponding Java file:

- Child.java

```
public class Child {
  private Parent parent;
  private long id;
  private String firstName;
  private String lastName;
  public Parent getParent() {return parent;}
  public void setParent(Parent parent) {this.parent = parent;}
  public String getFirstName() {return firstName; }
  public void setFirstName(String firstName) {this.firstName =  firstName; }
  public long getId() {return id; }
  public void setId(long id) {this.id = id;}
  public String getLastName() {return lastName;}
  public void setLastName(String lastName) {this.lastName = lastName;}
}
```

3.1 Child has reference to parent. Parent has reference to the child. Both are persisted.

- **Source code:**

```
Parent parent = new  Parent();
Child child = new  Child();  child.setFirstName("John");  child.setLastName("Smith");
child.setParent(parent);  //set parent to the child
Set children = new  HashSet();
children.add(child);  parent.setChildren(children);
session.saveOrUpdate(child); session.saveOrUpdate(parent);
```

- **Query**

```
Hibernate: insert into child (firstName, lastName, pid) values (?,  ?, ?)
Hibernate: insert into parent values ( )
Hibernate: update child set  firstName=?, lastName=?, pid=? where cid=?
Hibernate: update child set pid=?  where cid=?
```

- **Analysis**

  The first query is easy. Note how the first query here is different from the first query of the case1.1 where we had only uni-directional mapping. In that case the query didn't have any column named pid because, you guessed it, the mapping in case 1.1 is uni-directional. The second query is generated because the child has a relationship with the parent. The third query is for the parent and in the fourth query the parent establishes a relationship with the child. Please note that although in the first query pid is being passed , it is null until the fourth query is executed.

3.2 Child has reference to parent. Parent has reference to the child. Only parent is persisted.

- **Source code:**

  Parent parent = new Parent();

  Child child = new Child(); child.setFirstName("John"); child.setLastName("Smith");

  child.setParent(parent); //set parent to the child

  Set children = new HashSet();

  children.add(child); parent.setChildren(children);

  // session.saveOrUpdate(child);

  session.saveOrUpdate(parent);

- **Query**

  Hibernate: insert into parent values ( )

  ~~Hibernate: insert into child (firstName, lastName) values (?, ?)~~

  Hibernate: update child set pid=? where cid=?

- **Error**

  SEVERE: Could not synchronize database state with session net.sf.hibernate.TransientObjectException: object references an unsaved transient instance - save the transient instance before flushing: example.Child

- **Analysis**: Hibernate successfully persisted a record in the parent table. Look at the second query. Hibernate rightly attempted to establish a relationship. “Inverse” is set at ‘false’ and hence the parent object is trying to establish the relationship. But Hibernate realizes that the child record is still not persisted (remember cascade is ‘none’ now).

3.3 Child has reference to parent. Parent has reference to the child. Only child is persisted.

- **Source code:**

  Parent parent = new Parent();

  Child child = new Child(); child.setFirstName("John"); child.setLastName("Smith");

  child.setParent(parent); //set parent to the child

  Set children = new HashSet();

  children.add(child); parent.setChildren(children);

  session.saveOrUpdate(child);

  // session.saveOrUpdate(parent);

- **Query**

  ~~Hibernate: insert into parent values ( )~~

  Hibernate: insert into child (firstName, lastName) values (?, ?)

  Hibernate: update child set pid=? where cid=?

- **Error:**

  net.sf.hibernate.TransientObjectException: object references an unsaved transient instance - save the transient instance before flushing: example.Parent

- **Analysis:** A record is inserted in the child table. But this record has the ‘pid’ as null. Hibernate inserts a record in the child table. Now it needs to update the value of ‘pid’ in the child table. Hibernate attempts to find the pid by looking at the parent object. It finds that the parent object is still transient and it fails.

4. Bi directional one-to-many association with inverse set as “true”.

- This case is an extension of case discussed in step 3. In this case we are going to set inverse as ‘true’. It means now the onus of setting up the relationship is NOT with the parent.

```
<hibernate-mapping>
   <class name="com.mudassar.myexample.Parent"  table="parent">
      <id name="id" type="long"  column="pid">
        <generator class="native"  />
      </id>
      <set name="children" inverse="true"  cascade="none">
         <key column="pid" />
         <one-to-many  class="com.mudassar.myexample.Child"  />
     </set>
  </class>
</hibernate>
```

4.1 Child has reference to parent. Parent has reference to the child. Both are persisted.

- **Source code:**

```
Parent parent = new  Parent();
Child child = new  Child();  child.setFirstName("John");  child.setLastName("Smith");
child.setParent(parent);  //set parent to the child
Set children = new  HashSet();
children.add(child);  parent.setChildren(children);
session.saveOrUpdate(child); session.saveOrUpdate(parent);
```

- **Query**

```
Hibernate: insert into child (firstName, lastName, pid) values (?,  ?, ?)
Hibernate: insert into parent values ( )
Hibernate: update child set  firstName=?, lastName=?, pid=? where cid=?
```

~~Hibernate: update child set pid=? where cid=?~~

- **Analysis**

  Result is same as case 3.1 . So what difference 'inverse' made. Look at the query. Compare these queries with the queries generated by case 3.1 and the power of 'inverse' will come to you. Inverse maps the two bidirectional mappings so that unnecessary updates will not be taking place. In stead of both the parties trying to setup the association by setting inverse as 'true' one party clearly dictates that the onus of maintaining the relationship is someone else's. This results in one less query.

4.2 Child has reference to parent. Parent has reference to the child. Only parent is persisted.

- **Source code:**

  Parent parent = new Parent();

  Child child = new Child(); child.setFirstName("John"); child.setLastName("Smith");

  child.setParent(parent); //set parent to the child

  Set children = new HashSet();

  children.add(child); parent.setChildren(children);

  // session.saveOrUpdate(child);

  session.saveOrUpdate(parent);

- **Query**

  ~~Hibernate: insert into child (firstName, lastName, pid) values (?, ?, ?)~~

  Hibernate: insert into parent values ( )

  ~~Hibernate: update child set firstName=?, lastName=?, pid=? where cid=?~~

- **Analysis**

  After looking at the query if there are any doubts please revisit the result of step 3.2. There we had got an exception. No exception here. But no record is inserted in the child table. This is the effect of setting inverse to “true”. By doing that the parent says that the association is child’s responsibility and I can persist myself without worrying about the association. Hence in this case parent gets persisted irrespective of whether child is persisted.

4.3 Child has reference to parent. Parent has reference to the child. Only child is persisted.

- **Source code:**

```
Parent parent = new  Parent();
Child child = new  Child();  child.setFirstName("John");  child.setLastName("Smith");
child.setParent(parent);  //set parent to the child
Set children = new  HashSet();
children.add(child);  parent.setChildren(children);
session.saveOrUpdate(child);
// session.saveOrUpdate(parent);
```

- **Query**

  Hibernate: insert into child (firstName, lastName, pid) values (?, ?, ?)

  ~~Hibernate: insert into parent values ( )~~

  Hibernate: update child set firstName=?, lastName=?, pid=? where cid=?

- **Error:**

  net.sf.hibernate.TransientObjectException: object references an unsaved transient instance - save the transient instance before flushing: example.Parent

- **Analysis:** A record is inserted in the child table. But this record has the 'pid' as null. Hibernate inserts a record in the child table. Now it needs to update the value of 'pid' in the child table. Hibernate attempts to find the pid by looking at the parent object. It finds that the parent object is still transient and it fails.

Mapping Cardinality Relationship: Many-To-Many

Many to Many relationship

- *Speakers and Events relationship*
  - A Speaker speaks in many events and an Event has many speakers
- *EventManyToMany.hbm.xml*

```
<class name="EventManyToMany" table="m_events">
   <id name="id" column="uid" type="long" unsaved-value="null">
        <generator class="increment"/>
   </id>
   <property name="name" type="string" length="100"/>
   <property name="startDate" column="start_date" type="date"/>
   <property name="duration" type="integer"/>

   <!-- events_speakers is a join table -->
   <set name="speakers" table="events_speakers" cascade="all">
       <key column="event_id"/>
       <many-to-many column="speaker_id" class="SpeakerManyToMany"/>
   </set>
</class>
```

Many to Many relationship

- *SpeakerManyToMany.hbm.xml*

```
<class name="SpeakerManyToMany" table="m_speakers">
   <id name="id" column="uid" type="long">
       <generator class="increment"/>
   </id>
   <property name="firstName" type="string" length="20"/>
   <property name="lastName" type="string" length="20"/>

   <!-- events_speakers is a join table -->
  <set name="events" table="events_speakers" cascade="all"
inverse="true">
       <key column="speaker_id"/>
       <many-to-many column="event_id" class="EventManyToMany"/>
   </set>

</class>
```

Many to Many relationship:

- Event has many speakers

```
public class EventManyToMany {

    private Long id;
    private String name;
    private Date startDate;
    private int duration;
    private Set speakers;
    private Set attendees;

    public void setSpeakers(Set speakers) {
        this.speakers = speakers;
    }

    public Set getSpeakers() {
        return speakers;
    }

    // ...
```

Many to Many relationship:

- A speaker speaks in many events

```
public class SpeakerManyToMany {

    private Long id;
    private String firstName;
    private String lastName;
    private Set events;

    public Set getEvents() {
        return this.events;
    }

    public void setEvents(Set events) {
        this.events = events;
    }

    // ...
```

Many to Many relationship: Creating object instances

```
        EventManyToMany event = new EventManyToMany();
        event.setName("Spring conference");
        SpeakerManyToMany speaker1=new SpeakerManyToMany("John", "Smith", event);
        SpeakerManyToMany speaker2=new SpeakerManyToMany("Joe", "Smith", event);
        SpeakerManyToMany speaker3=new SpeakerManyToMany("Mudassar", "Hakim",
event);
        HashSet set=new HashSet();
        set.add(speaker1); set.add(speaker2); set.add(speaker3);
        event.setSpeakers(set);
        session.save(event);
        EventManyToMany event2 = new EventManyToMany();
        event2.setName("Hibernate Conference");
        SpeakerManyToMany speaker4=new SpeakerManyToMany("Diane", "Woon", event2)
        HashSet set1=new HashSet();
        set1.add(speaker4); set1.add(speaker3);
        event2.setSpeakers(set1);
        session.save(event2);
```

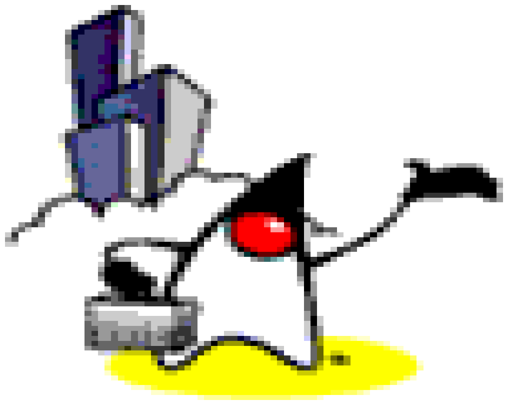
Many to Many relationship

```
******** Table: m_events *******
+-------------+----------------------+------------+-------------+-------------+
|     UID     |         NAME         | START_DATE |   DURATION  | LOCATION_ID |
+-------------+----------------------+------------+-------------+-------------+
| 1           | JavaOne conference   |              0           |
| 2           | Passion Conference   |              0           |
+-------------+----------------------+------------+-------------+-------------+

******** Table: m_speakers *******
+-------------+----------------------+----------------------+
|     UID     |       FIRSTNAME      |       LASTNAME       |
+-------------+----------------------+----------------------+
| 1           | Joe                  | Smith                |
| 2           | John                 | Smith                |
| 3           | Mudassar             | Hakim                |
| 4           | Diane                | Woon                 |
+-------------+----------------------+----------------------+

******** Table: events_speakers *******
+-------------+-------------+
|   EVENT_ID  |  SPEAKER_ID |
+-------------+-------------+
| 1           | 1           |
| 1           | 2           |
| 1           | 3           |
| 2           | 3           |
| 2           | 4           |
+-------------+-------------+
```

Join table



Mapping Cardinality Relationship: Using <map>

One-Has-Collection relationship: Using <map> in mapping file

- SupportProperty class has a field whose type is Map
- *SupportProperty.hbm.xml*

```
<class name="SupportProperty" table="supportproperty">
    <id name="id">
        <generator class="increment"/>
    </id>

    <map name="properties">
        <key column="id"/>
        <index column="property_name" type="string"/>
        <element column="property_value" type="string"/>
    </map>

    <property name="name" type="string"/>
</class>
```

One-Has-Collection relationship: Domain Class

- SupportProperty class has a field whose type is Map

```
public class SupportProperty {

    private int id;
    private String name;
    private Map properties;

    public void setProperties(Map m) {
        properties = m;
    }

    public Map getProperties() {
        return properties;
    }

    // ...
```

One-Has-Collection relationship: Creating an Object Instance

- SupportProperty class has a field whose type is Map

```
// Create Domain object, SupportProperty object has a Map object.
SupportProperty sp = new SupportProperty();
sp.setName("MyProperties");

HashMap h = new HashMap();
h.put("car", "ford");
h.put("house", "lexington");
sp.setProperties(h);

session.save(sp);

// Create another object instance
SupportProperty sp2 = new SupportProperty();
sp2.setName("YourProperties");
HashMap h2 = new HashMap();
h2.put("tv", "samsung");
h2.put("house", "lexington");
sp2.setProperties(h2);

session.save(sp2);
```

One to Many relationship: Using <map> in the mapping file

- Tables

```
******** Table: supportproperty *******
+-------------+----------------------+
|     ID      |         NAME         |
+-------------+----------------------+
| 1           | MyProperties         |
| 2           | YourProperties       |
+-------------+----------------------+

******** Table: properties *******
+-------------+----------------------+----------------------+
|     ID      |    PROPERTY_NAME     |    PROPERTY_VALUE    |
+-------------+----------------------+----------------------+
| 1           | car                  | ford                 |
| 1           | house                | lexington            |
| 2           | tv                   | samsung              |
| 2           | house                | lexington            |
+-------------+----------------------+----------------------+
```

Thank you!

Mudassar Hakim
mudassar.trainer@gmail.com