<https://www.devglan.com/hibernate/hibernate-many-to-many-mapping-example>

# Hibernate many to many mapping annotation example

By Lokesh Gupta | Filed Under: [Hibernate](https://howtodoinjava.com/hibernate/)

**Hibernate many to many mapping** is made between two entities where one can have relation with multiple other entity instances. For example, for a subscription service SubscriptionEntity and ReaderEntity can be two type of entities. Any subscription can have multiple readers, where a reader can subscribe to multiple subscriptions.

In this hibernate tutorial, we will learn to create many to many mapping in database using hibernate.

Table of contents

Hibernate many to many mapping design

Owner entity

Mapped entity

Configure entities in hibernate config file

Demo

## 1. Hibernate many to many mapping design

To demonstrate many to many mapping using hibernate annotations, we will associate two entities i.e. ReaderEntity and SubscriptionEntity.

Their database schema should look like this. Using these tables, any application can save multiple associations between readers and subscriptions.



## 2. Owner entity

Owner entity is the entity which is **responsible make making the association and maintaining it**. In our case, I am making ReaderEntity the owner entity. **@JoinTable** annotation has been used to make this association.

|  |
| --- |
| ReaderEntity.java |
| package hibernate.test.manyToMany.joinTable;    import java.io.Serializable;  import java.util.Set;    import javax.persistence.CascadeType;  import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.GenerationType;  import javax.persistence.Id;  import javax.persistence.JoinColumn;  import javax.persistence.JoinTable;  import javax.persistence.ManyToMany;  import javax.persistence.Table;  import javax.persistence.UniqueConstraint;    @Entity(name = "ReaderEntity")  @Table(name = "READER", uniqueConstraints = {          @UniqueConstraint(columnNames = "ID"),          @UniqueConstraint(columnNames = "EMAIL") })    public class ReaderEntity implements Serializable  {      private static final long serialVersionUID = -1798070786993154676L;        @Id      @GeneratedValue(strategy = GenerationType.IDENTITY)      @Column(name = "ID", unique = true, nullable = false)      private Integer readerId;        @Column(name = "EMAIL", unique = true, nullable = false, length = 100)      private String email;        @Column(name = "FIRST\_NAME", unique = false, nullable = false, length = 100)      private String firstName;        @Column(name = "LAST\_NAME", unique = false, nullable = false, length = 100)      private String lastName;        @ManyToMany(cascade=CascadeType.ALL)      @JoinTable(name="READER\_SUBSCRIPTIONS", joinColumns={@JoinColumn(referencedColumnName="ID")}                                          , inverseJoinColumns={@JoinColumn(referencedColumnName="ID")})      private Set<SubscriptionEntity> subscriptions;        //Getters and setters  } |

## 3. Mapped entity

Our mapped entity is SubscriptionEntity which is mapped to ReaderEntity using “**mappedBy**” attribute.

|  |
| --- |
| SubscriptionEntity.java |
| package hibernate.test.manyToMany.joinTable;    import java.io.Serializable;  import java.util.Set;    import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.GenerationType;  import javax.persistence.Id;  import javax.persistence.ManyToMany;  import javax.persistence.Table;  import javax.persistence.UniqueConstraint;    @Entity(name = "SubscriptionEntity")  @Table(name = "SUBSCRIPTION", uniqueConstraints = {          @UniqueConstraint(columnNames = "ID")})    public class SubscriptionEntity implements Serializable  {      private static final long serialVersionUID = -6790693372846798580L;        @Id      @GeneratedValue(strategy = GenerationType.IDENTITY)      @Column(name = "ID", unique = true, nullable = false)      private Integer subscriptionId;        @Column(name = "SUBS\_NAME", unique = true, nullable = false, length = 100)      private String subscriptionName;        @ManyToMany(mappedBy="subscriptions")      private Set<ReaderEntity> readers;        //Getters and setters  } |

## 4. Configuring entities in hibernate config file

We have make available both entities to runtime. To do so, we have to add them in **hibernate.cfg.xml** file.

|  |
| --- |
| hibernate.cfg.xml |
| <?xml version="1.0" encoding="utf-8"?>  <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "<http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd>">  <hibernate-configuration>      <session-factory>          <property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>          <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernatetest</property>          <property name="hibernate.connection.password">XXXXXX</property>          <property name="hibernate.connection.username">root</property>          <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>          <property name="show\_sql">true</property>          <property name="hbm2ddl.auto">create</property>          <mapping class="hibernate.test.manyToMany.joinTable.ReaderEntity"/>          <mapping class="hibernate.test.manyToMany.joinTable.SubscriptionEntity"/>      </session-factory>  </hibernate-configuration> |

## 5. Hibernate many to many annotation mapping example

Now, its time to test the code. I have written following code to test above entities and their many to many relationship.

|  |
| --- |
| TestJoinTable.java |
| package hibernate.test.manyToMany;    import hibernate.test.HibernateUtil;  import hibernate.test.manyToMany.joinTable.\*;    import java.util.HashSet;  import java.util.Set;    import org.hibernate.Session;    public class TestJoinTable  {      public static void main(String[] args)      {          Session session = HibernateUtil.getSessionFactory().openSession();          session.beginTransaction();            //Add subscription          SubscriptionEntity subOne = new SubscriptionEntity();          subOne.setSubscriptionName("Entertainment");            SubscriptionEntity subTwo = new SubscriptionEntity();          subTwo.setSubscriptionName("Horror");            Set<SubscriptionEntity> subs = new HashSet<SubscriptionEntity>();          subs.add(subOne);          subs.add(subTwo);            //Add readers          ReaderEntity readerOne = new ReaderEntity();          readerOne.setEmail("demo-user1@mail.com");          readerOne.setFirstName("demo");          readerOne.setLastName("user");            ReaderEntity readerTwo = new ReaderEntity();          readerTwo.setEmail("demo-user2@mail.com");          readerTwo.setFirstName("demo");          readerTwo.setLastName("user");            Set<ReaderEntity> readers = new HashSet<ReaderEntity>();          readers.add(readerOne);          readers.add(readerTwo);            readerOne.setSubscriptions(subs);          readerTwo.setSubscriptions(subs);            session.save(readerOne);          session.save(readerTwo);            session.getTransaction().commit();          HibernateUtil.shutdown();      }  } |

Program Output:

|  |
| --- |
| Console |
| Hibernate: insert into READER (EMAIL, FIRST\_NAME, LAST\_NAME) values (?, ?, ?)  Hibernate: insert into SUBSCRIPTION (SUBS\_NAME) values (?)  Hibernate: insert into SUBSCRIPTION (SUBS\_NAME) values (?)  Hibernate: insert into READER (EMAIL, FIRST\_NAME, LAST\_NAME) values (?, ?, ?)  Hibernate: insert into READER\_SUBSCRIPTIONS (readers\_ID, subscriptions\_ID) values (?, ?)  Hibernate: insert into READER\_SUBSCRIPTIONS (readers\_ID, subscriptions\_ID) values (?, ?)  Hibernate: insert into READER\_SUBSCRIPTIONS (readers\_ID, subscriptions\_ID) values (?, ?)  Hibernate: insert into READER\_SUBSCRIPTIONS (readers\_ID, subscriptions\_ID) values (?, ?) |

**Hibernate Many To Many Mapping Example – Annotation**

Let’s take the Example of **Employee** and **Department**, one **Employee** can be part of many **Departments** and similarly, one **Department** can have many **Employees**. Let’s dig into the code.

In Many-to-Many relationship a mediator table is mandatory, this table stores the primary key of both tables (**EMPLOYEE** and **DEPARTMENT**) as a **foreign key**.

### Creating table

Create **EMPLOYEE, DEPARTMENT**and**EMPLOYEE\_DEPARTMENT**Tables, simply Copy and Paste the following SQL query in the query editor to get the table created.

 CREATE TABLE "EMPLOYEE"

(

"EMP\_ID" NUMBER(10,0) NOT NULL ENABLE,

"EMP\_NAME" VARCHAR2(255 CHAR),

PRIMARY KEY ("EMP\_ID")

);

CREATE TABLE "DEPARTMENT"

(

"DEP\_ID" NUMBER(10,0) NOT NULL ENABLE,

"DEP\_NAME" VARCHAR2(255 CHAR),

PRIMARY KEY ("DEP\_ID")

);

CREATE TABLE "EMPLOYEE\_DEPARTMENT"

(

"EMP\_ID" NUMBER(10,0) NOT NULL ENABLE,

"DEP\_ID" NUMBER(10,0) NOT NULL ENABLE,

PRIMARY KEY (EMP\_ID , DEP\_ID),

CONSTRAINT FK\_EMP\_ID FOREIGN KEY (EMP\_ID) REFERENCES EMPLOYEE (EMP\_ID),

CONSTRAINT FK\_DEP\_ID FOREIGN KEY (DEP\_ID) REFERENCES DEPARTMENT (DEP\_ID)

);

### ****Employee.java****

Create a new Java file **Employee.java** under the package **com.javainterviewpoint** and add the following code

package com.javainterviewpoint;

import java.util.Set;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.JoinTable;

import javax.persistence.ManyToMany;

import javax.persistence.Table;

@Entity

@Table(name="EMPLOYEE")

public class Employee

{

@Id

@Column(name="EMP\_ID")

@GeneratedValue

private int empId;

@Column(name="EMP\_NAME")

private String empName;

@ManyToMany(cascade = {CascadeType.ALL})

@JoinTable(name="EMPLOYEE\_DEPARTMENT",

joinColumns={@JoinColumn(name="EMP\_ID")},

inverseJoinColumns={@JoinColumn(name="DEP\_ID")})

private Set<Department> department;

public Employee()

{

super();

}

public Employee(String empName)

{

super();

this.empName = empName;

}

public int getEmpId()

{

return empId;

}

public void setEmpId(int empId)

{

this.empId = empId;

}

public String getEmpName()

{

return empName;

}

public void setEmpName(String empName)

{

this.empName = empName;

}

public Set<Department> getDepartment()

{

return department;

}

public void setDepartment(Set<Department> department)

{

this.department = department;

}

@Override

public String toString()

{

return "Employee [empId=" + empId + ", empName=" + empName + "]";

}

}

### ****Department.java****

Create a new Java file **Department.java** under the package **com.javainterviewpoint** and add the following code

package com.javainterviewpoint;

import java.util.Set;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.ManyToMany;

import javax.persistence.Table;

@Entity

@Table(name="DEPARTMENT")

public class Department

{

@Id

@Column(name="DEP\_ID")

@GeneratedValue

private int depId;

@Column(name="DEP\_NAME")

private String depName;

@ManyToMany(mappedBy="department")

private Set<Employee> employee;

public Department()

{

super();

}

public Department(int depId, String depName, Set employee)

{

super();

this.depId = depId;

this.depName = depName;

this.employee = employee;

}

public int getDepId()

{

return depId;

}

public void setDepId(int depId)

{

this.depId = depId;

}

public String getDepName()

{

return depName;

}

public void setDepName(String depName)

{

this.depName = depName;

}

public Set<Employee> getEmployee()

{

return employee;

}

public void setEmployee(Set<Employee> employee)

{

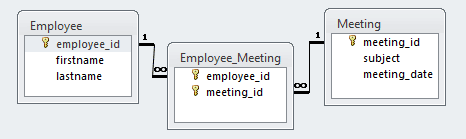
this.employee = employee;

}

}

We are using Employee-Meeting relationship as a many to many relationship example.

Each Employee can attain more than one meetings and each meetings can have more than one employee



public class Employee

@ManyToMany(cascade = {CascadeType.ALL})

@JoinTable(name="EMPLOYEE\_MEETING", joinColumns={@JoinColumn(name="EMPLOYEE\_ID")},

inverseJoinColumns={@JoinColumn(name="MEETING\_ID")})

private Set<Meeting> meetings = new HashSet<Meeting>();

public class Meeting

@ManyToMany(mappedBy="meetings")

private Set<Employee> employees = new HashSet<Employee>();

