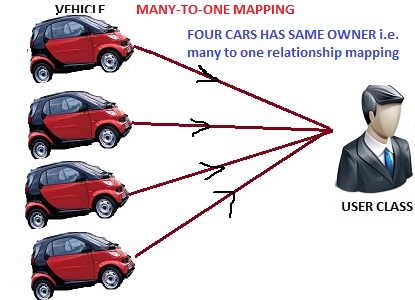
Many to One Mapping in Hibernate Example

In [the previous tutorial,](https://www.dineshonjava.com/p/hibernate-one-to-many-mapping-tutorial.html)we look that what is One to Many Mapping and also discussed some examples about that.

In this tutorial of Many to one mapping in hibernate we will discuss about the Many To One Relationship Mapping. Actually Many To One is the reverse of the One To Many(USER has many Vehicles means one user related to the many vehicles in reverse we can say that many vehicles related to the one user i.e. Many To One relationship mapping).

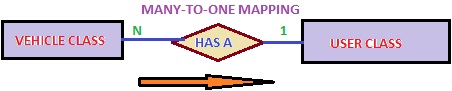
**Many-to-One Relationships**

A many-to-one relationship is where one entity contains values that refer to another entity (a column or set of columns) that has unique values. In relational databases, these many-to-one relationships are often enforced by foreign key/primary key relationships, and the relationships typically are between fact and dimension tables and between levels in a hierarchy.



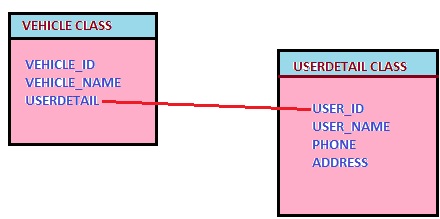
In this example, multiple vehicles (BMW Car, AUDI Car, Maruti Car and Mahindra etc.) are linked to the same User (whose primary key is 1).

Class diagram for that is given below.



According to the relationship, many vehicles can have the same owner.

To create this relationship you need to have a *USER* and *VEHICLE* table. The relational model is shown below.



For that, we will use the following annotation.  
**@ManyToOne** :  
**Target:**  
Fields (including property get methods)Defines a single-valued association to another entity class that has many-to-one multiplicity. It is not normally necessary to specify the target entity explicitly since it can usually be inferred from the type of the object being referenced. If the relationship is bidirectional, the non-owning **OneToMany**entity side must use the mappedByelement to specify the relationship field or property of the entity that is the owner of the relationship.  The **ManyToOne** annotation may be used within an embeddable class to specify a relationship from the embeddable class to an entity class. If the relationship is bidirectional, the non-owning **OneToMany**entity side must use the **mappedBy**element of the **OneToMany**annotation to specify the relationship field or property of the embeddable field or property on the owning side of the relationship. The dot (“.”) notation syntax must be used in the **mappedBy**element to indicate the relationship attribute within the embedded attribute. The value of each identifier used with the dot notation is the name of the respective embedded field or property.

Now we look the following Example related to the **One to Many** mapping.

**UserDetails.java**

**package** com.sdnext.hibernate.tutorial.dto;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table (name=”USER”)

**public** **class** UserDetails

{

@Id

@Column(name=”USER\_ID”)

@GeneratedValue(strategy=GenerationType.AUTO)

**private** **int** userId;

@Column(name=”USER\_NAME”)

**private** String userName;

**public** **int** getUserId() {

**return** userId;

}

**public** **void** setUserId(**int** userId) {

**this**.userId = userId;

}

**public** String getUserName() {

**return** userName;

}

**public** **void** setUserName(String userName) {

**this**.userName = userName;

}

}

**Vehicle.java**

**package** com.sdnext.hibernate.tutorial.dto;

**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.ManyToOne;

**import** javax.persistence.Table;

@Entity

@Table(name=”VEHICLE”)

**public** **class** Vehicle

{

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

@Column(name=”VEHICLE\_ID”)

**private** **int** vehicleId;

@Column(name=”VEHICLE\_NAME”)

**private** String vehicleName;

@ManyToOne

@JoinColumn(name =”USER\_ID”)

**private** UserDetails user;

**public** UserDetails getUser() {

**return** user;

}

**public** **void** setUser(UserDetails user) {

**this**.user = user;

}

**public** **int** getVehicleId() {

**return** vehicleId;

}

**public** **void** setVehicleId(**int** vehicleId) {

**this**.vehicleId = vehicleId;

}

**public** String getVehicleName() {

**return** vehicleName;

}

**public** **void** setVehicleName(String vehicleName) {

**this**.vehicleName = vehicleName;

}

}

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>

<!– Database connection settings –>

<property name=”connection.driver\_class”>com.mysql.jdbc.Driver</property>

<property name=”connection.url”>jdbc:mysql://localhost:3306/hibernateDB</property>

<property name=”connection.username”>root</property>

<property name=”connection.password”>root</property>

<!– JDBC connection pool (use the built-in) –>

<property name=”connection.pool\_size”>100</property>

<!– SQL dialect –>

<property name=”hibernate.dialect”>org.hibernate.dialect.MySQL5Dialect</property>

<!– Enable Hibernate’s automatic session context management –>

<property name=”current\_session\_context\_class”>thread</property>

<!– Disable the second-level cache –>

<property name=”cache.provider\_class”>org.hibernate.cache.NoCacheProvider</property>

Advertisements

<!– Echo all executed SQL to stdout –>

<property name=”show\_sql”>**true**</property>

<!– Drop and re-create the database schema on startup –>

<property name=”hbm2ddl.auto”>create</property>

<mapping **class**=”com.sdnext.hibernate.tutorial.dto.UserDetails”/>

<mapping **class**=”com.sdnext.hibernate.tutorial.dto.Vehicle”/>

</session-factory>

</hibernate-configuration>

**HibernateTestDemo.java**

**package** com.sdnext.hibernate.tutorial;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.cfg.AnnotationConfiguration;

**import** com.sdnext.hibernate.tutorial.dto.UserDetails;

**import** com.sdnext.hibernate.tutorial.dto.Vehicle;

**public** **class** HibernateTestDemo {

/\*\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args)

{

UserDetails user = **new** UserDetails(); //create an user entity

Vehicle vehicle = **new** Vehicle(); //create a vehicle entity

Vehicle vehicle2 = **new** Vehicle(); //create second vehicle entity

vehicle.setVehicleName(“BMW Car”); //set BMW car

vehicle.setUser(user); //set user for that car

vehicle2.setVehicleName(“AUDI Car”); //set second car Audi

vehicle2.setUser(user);//set user for that car

user.setUserName(“Dinesh Rajput”); //set user property

SessionFactory sessionFactory = **new** AnnotationConfiguration().configure().buildSessionFactory(); //create the session factory object

Session session = sessionFactory.openSession(); //create the session object

session.beginTransaction(); //create the transaction object

session.save(vehicle);

session.save(vehicle2);

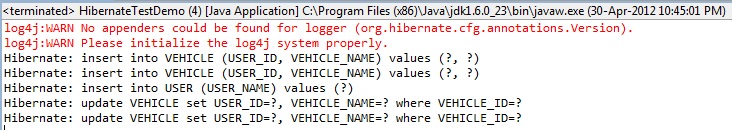
session.save(user);

session.getTransaction().commit();

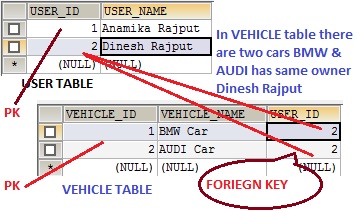
session.close();

}

}



Now we look at the table structure about this example.



Now how can implement this mapping through mapping file(**.hbm.xml**) instead of the **annotations**?

For user class.  
**UserDetails.hbm.xml**

<?xml version=”1.0″?>

<!DOCTYPE hibernate-mapping PUBLIC

“-//Hibernate/Hibernate Mapping DTD 3.0//EN”

“http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd”>

<hibernate-mapping>

<**class** name=”com.sdnext.hibernate.tutorial.dto.UserDetails” table=”USER”>

<id name=”userId” type=”**long**” column=”ID” >

<generator **class**=”assigned”/>

</id>

<property name=”userName”>

<column name=”UserName” />

</property>

</**class**>

</hibernate-mapping>

Mapping File For Vehicle  Class…  
**vehicle.hbm.xml**

<?xml version=”1.0″?>

<!DOCTYPE hibernate-mapping PUBLIC

“-//Hibernate/Hibernate Mapping DTD 3.0//EN”

“http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd”>

<hibernate-mapping>

<**class** name=”com.sdnext.hibernate.tutorial.dto.Vehicle” table=”VEHICLE”>

<id name=”userId” type=”**long**” column=”ID” >

<generator **class**=”assigned”/>

</id>

<property name=”vehicleName” column=”VEHICLE\_NAME”> </property>

The *many-to-one* element is used to create the many-to-one relationship between the Vehicle and UserDetail entities. The *cascade* option is used to cascade the required operations to the associated entity. If the *cascade* option is set to all then all the operations will be cascaded. For instance, when you save a *Vehicle*object, the associated UserDetail object will also be saved automatically.