**Many to One Mapping**

* Consider we live in the world that one **USER** can have Multiple **vehicle**.

**Key points to remember when you do the relational mapping**

* Always take a look from the current table side when U write an annotation.
* Always do the **BI-DIRECTIONAL** mapping, so that U can fetch the details from entity to another.
* Better to do the LAZY fetch, so if U fetch data from one table entity other table data doesn’t get selected automatically by HB. Unless U need both the table data for UR logic U are supposed to apply

@ManyToOne (fetch = FetchType.LAZY) LAZY fetch.

* **Foreign key relation between two Entity can be in below ways**

1. **Either U can have it in separate table which holds the Foreign key relation by using @JoinTable annotation.**

If U look from user table one row of user is mapped to Multiple rows (Many vehicle) of Vehicle table.

@OneToMany to vehicle Table

@JoinTable(name="USER\_VEHICLE",joinColumns=@JoinColumn(name="USER\_ID"),inverseJoinColumns=@JoinColumn(name="VEHICLE\_ID"))

|  |  |  |
| --- | --- | --- |
| **USER Table** | | |
| **User\_ID** | **Name** | **Email** |
| 111 | ILAN | Gmail |
| 212 | KUMARAN | Yahoo |

If U look from Vehicle Table Multiple rows of Vehicle is mapped to one row (one user) of user table.

Join table name is mentioned in @JoinTable annotation and JoinColumn name and inverse JoinColumn name is also mentioned in same @JoinTable annotation.

@ManyToOne to User Table

|  |  |
| --- | --- |
| **VEHICLE TABLE** | |
| **Vehicle\_No** | **Vehicle\_name** |
| 101 | Yamaha |
| 102 | Honda |

Below is the separate JOIN Table that contains the mapping between user and vehicle table

|  |  |
| --- | --- |
| **USER\_VEHICLE** | |
| **VEHICLE\_ID** | **USER\_ID** |
| 101 | 111 |
| 102 | 212 |

1. **Or U can have the Foreign key relation in any one of the Entity by using @JoinColumn annotation.**

**Points to remember using the @JoinColumn annotation**

ManyToOne mapping we can have the JoinColumn only in ManyToOne table not in OneToMany mapping table bcz of the reason below

So one user can have multiple vehicle i.e. multiple vehicles are mapped to one user.

User table that contain Multiple user row can’t hold all the primary keys of multiple vehicles.

Even Vehicle table that contain Multiple vehicle rows can’t hold all the primary keys of multiple User.

@OneToMany(mappedBy=" **userID**") to vehicle Table

|  |  |  |
| --- | --- | --- |
| **USER Table** | | |
| **User\_ID** | **Name** | **Email** |
| 111 | ILAN | Gmail |
| 212 | KUMARAN | Yahoo |

Below is the vehicle table that has the JOIN column contains the primary key of user table.

@ManyToOne to User Table

@JoinColumn(name="**User\_ID**")

|  |  |  |
| --- | --- | --- |
| **Vehicle** | | |
| **Vehicle\_No** | **Vehicle\_name** | **@JoinColumn User\_ID (Foreign Key)**  **private USER userID;** |
| 101 | Yamaha | 212 |
| 102 | Honda | 111 |
| 103 | Jeep | 212 |
| 104 | Audi | 111 |
| 105 | BMW | 111 |
| 106 | Benz | 111 |

**MappedBy is used to inject the PK values to Foreign Key entity instance variable**

MappedBy signals hibernate that the key for the relationship is on the other side.

This means that although you link 2 tables together, only 1 of those tables has a foreign key constraint to the other one. MappedBy allows you to still link from the table not containing the constraint to the other table.

MappedBy always be in the opposite entity to JoinColumn.

MappedBy property takes Child entity instance variable parameter and looks for the mentioned variable, then supplies the Primary key to it.

USER table is the parent table.

VEHICLE table is the child table.

So USER table has primary key which is going to be a foreign key for VEHICLE table.

How will the USER table give its primary key to JoinColumn of VEHICLE table?

It’s basically by mappedBy=" **userID**", since we use mappedBy we will mention the VEHICLE entity instance (class)variable object name as parameter to mappedBy, so USER entity identify the instance variable in VEHICLE entity and handover to it,

So rule apply here mappedBy parameter value and JoinColumn variable name should be the same. MappedBy should always be available in primary key provider entity.

Save the Primary key entity first only then HB will know what is the PK that has to be provided to child table JoinColumn.

**MappedBy is used to inject the PK values to Foreign Key entity instance variable**

MappedBy signals hibernate that the key for the relationship is on the other side.

This means that although you link 2 tables together, only 1 of those tables has a foreign key constraint to the other one. MappedBy allows you to still link from the table not containing the constraint to the other table.

MappedBy always be in the opposite entity to JoinColumn.

MappedBy property takes Child entity instance variable parameter and looks for the mentioned variable, then supplies the Primary key to it.

USER table is the parent table.

VEHICLE table is the child table.

So USER table has primary key which is going to be a foreign key for VEHICLE table.

How will the USER table give its primary key to JoinColumn of VEHICLE table?

It’s basically by mappedBy=" **userID**", since we use mappedBy we will mention the VEHICLE entity instance (class)variable object name as parameter to mappedBy, so USER entity identify the instance variable in VEHICLE entity and handover to it,

So rule apply here mappedBy parameter value and JoinColumn variable name should be the same. MappedBy should always be available in primary key provider entity.

Save the Primary key entity first only then HB will know what is the PK that has to be provided to child table JoinColumn.

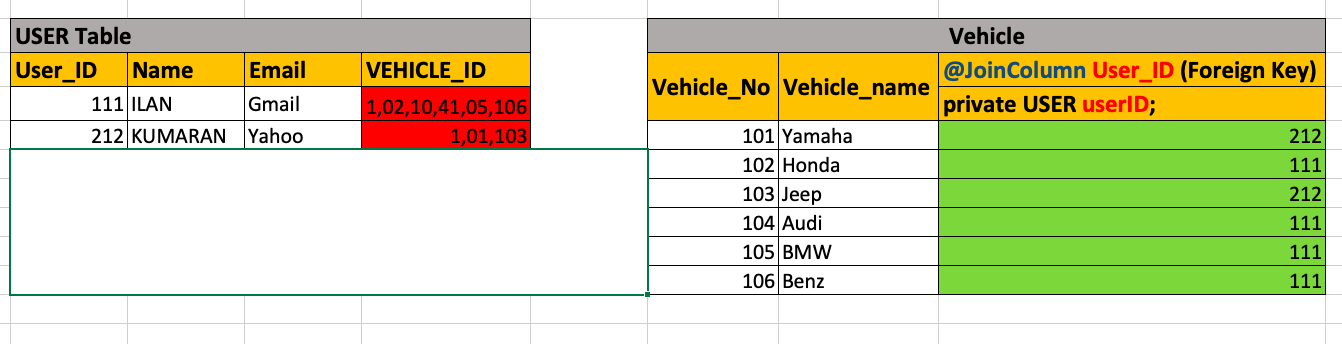
**BI-Directional Mapping**

**ManyToOne (one user and multiple vehicle)**

You can have JoinColumn or JoinTable when it comes to Bi-directional mapping.

If you are doing Bi-Directional mapping on ManyToOne, then you can have the JoinColumn Only Child Table i.e. Vehicle Table, since User ILAN in User Table can’t hold his multiple Vehicle ID Primary key in single row of User Table. But Vehicle table can hold each user mapped to the Vehicle.

Below picture says the graphical representation of the above scenario.



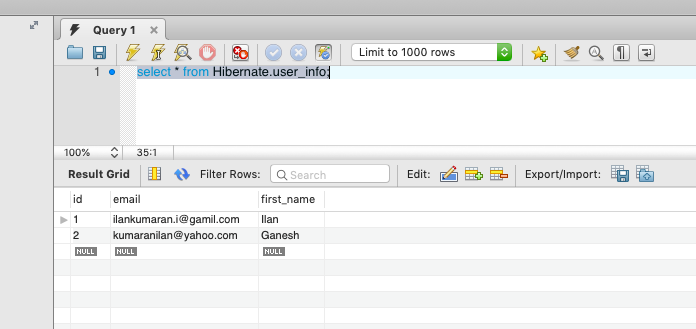
**OneToOne (one user and one vehicle)**

If you are doing Bi-Directional mapping on OneToOne, then you can have the JoinColumn either in Parent or in Child Table or you can maintain the relation in JoinTable.

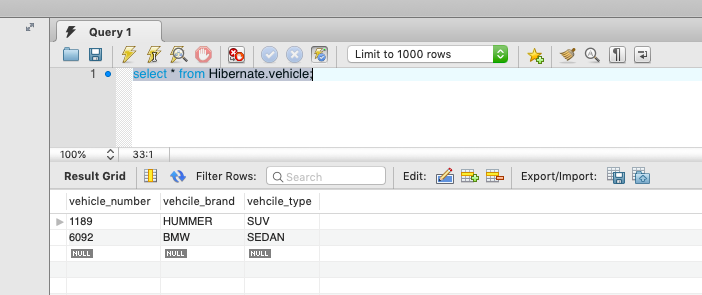
**ManyToMany (Multiple user and multiple vehicle)**

Either way User or Vehicle can’t hold the relational mapping foreign on any of its table, so HB always create the JoinTable when U do ManyToMany relations on Bi-directional.

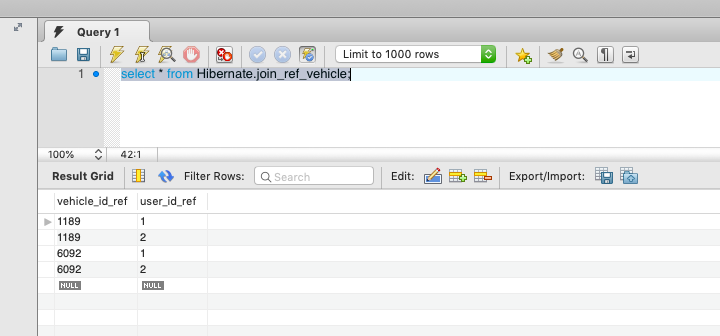
USER



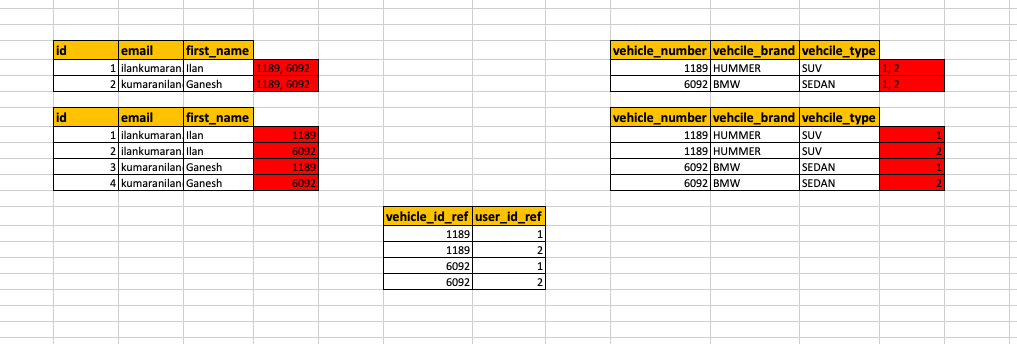
VEHICLE



USER & VEHICLE JoinTable



Reason we can’t hold the relation in any one of its table is represented below diagram.



**Scenario 1:**

We can’t hold foreign key in same row of its relation mapping.

**Scenario 2:**

We can’t hold foreign key in multiple rows of its relation mapping, since the same date of the row is duplicated and the foreign is only difference.

**Single directional mapping**

Consider U are doing the single directional mapping, then save the object first where u did the single directional mapping.

**ManyToOne (one user and multiple vehicle)**

Consider you have done single directional(OneToMany) mapping **ONLY** on User side that is Parent side. U didn’t use @JoinTable AND @JoinColumn annotation, then HB automatically creates a JoinTable, since it can’t accommodate Multiple Vehicle rows on USER Table. Note: save the USER object first.

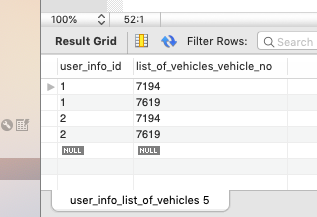
Consider you have done single directional(ManyToOne) mapping **ONLY** on vehicle side that is child side. U didn’t use @JoinTable AND @JoinColumn annotation, then HB automatically adds the JoinColumn on vehicle table and maps the User primary as foreign key in Vehicle Table. Note: save the VEHICLE’s object first.

**OneToOne (one user and one vehicle)**

If U are not using @JoinTable AND @JoinColumn annotation, then HB doesn’t know what to do with OneToOne Mapping on both the side entity. So it doesn’t create JoinTable and JoinColumn when it comes to OneToOne.

**ManyToMany (Multiple user and multiple vehicle)**

Consider you have done single directional(ManyToMany) mapping **ONLY** on User side. U didn’t use @JoinTable AND @JoinColumn annotation, then HB automatically creates a JoinTable that has the relational mapping between User and Vehicle with help of USER Table. Note: save the USER object first.

****

Consider you have done single directional(ManyToMany) mapping **ONLY** on vehicle side. U didn’t use @JoinTable AND @JoinColumn annotation, then HB automatically creates a JoinTable that has the relational mapping between User and Vehicle with help of VEHICLE Table. Note: save the VEHICLE’s object first.

