JPA class relations notes

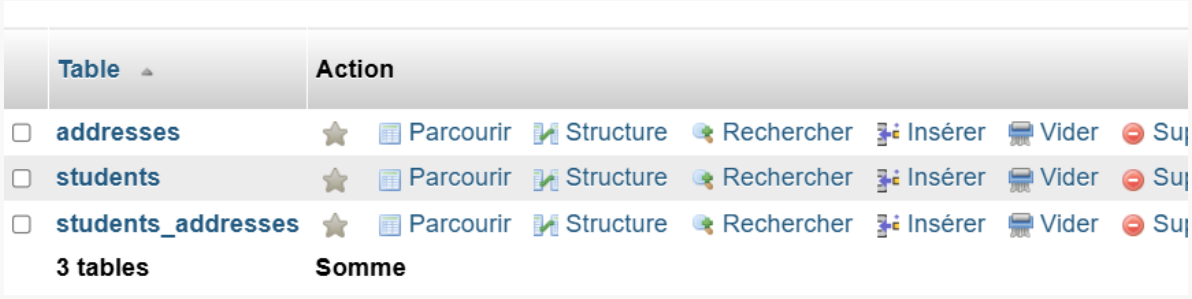
`@OneToMany` and `@ManyToOne` Unidirectionnel Relation :

The importance of the use @joinColumn in order to avoid having a join table that is created automatically in the owning side, in this case the Student Entity. The @joinColumn annotation specifies the foreign key of the inverse side entity, and therefore creates the owning side entity with the column representing the foreign key of the other entity, if not specified JPA will create a join table student_addresses automatically.

Throught the application of the classes relations, I've also observed the automatic creation of the join table even with the use of the @joinColumn , After some debugging realised that there's a difference between specifying the relation above the Collection representing the inverse side entity and specifying the relation above the getter of that collection. When specifying the relation above the collections as in the image below (figure 1), the join table was created automatically with other tables, however when specifying above the collection (figure 2), only the tables (students and addresses) are created.

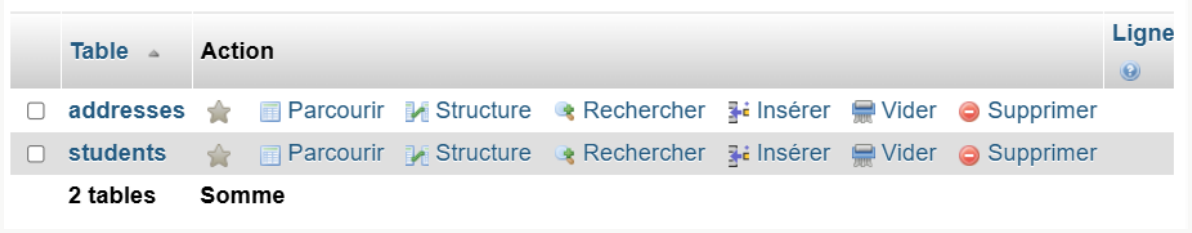
```
// Getters & Setters
@OneToMany(cascade = {CascadeType.ALL}, fetch = FetchType.EAGER)  2 usages
@JoinColumn(name = "student_id")
public Collection<Address> getAddresses() { return addresses; }
```

Figure 1 : Above getter method




```
@OneToMany(cascade = {CascadeType.ALL}, fetch = FetchType.EAGER)  2 usages
@JoinColumn(name = "student_id")
private Collection<Address> addresses = new ArrayList<Address>();
```

Figure 2 : Above the Collection representing the inverse entity



Bidirectionnel Relations :

When specifying the mappedBy attribute, we are creating a biderectionnel relation between two entities, mappedBy can only be found in the inverse side and it specifies the field representing the inverse entity in the owning side. Not specifying correctly the name of the attribute may lead to errors in the console and database not being persisted.

- The foreign key is always stored in the child table (Address) for both unidirectional and bidirectional @OneToMany and @ManyToOne relationships
- The @ManyToOne side is the owning side, while the @OneToMany side is the inverse side