Many-To-Many

Spring Data JPA

Take a glance about **Many-to-Many relationships** in Spring Boot Projects.

@mauricioperez

@mauricioperez

01

Many to Many Relationship

When you have a scenario where you want to relate a group of data from one table to another there's called a **Many to Many relationship**.

Hint: A Many-to-Many relationship takes into consideration two tables (the ones which are going to relate) and a middle table negotiating between both.

@mauricioperez

02

How to identify a Many-to-Many relationship?

Real-life example

Let’s suppose given the scenario that you want **many products are into many orders** that customers do for a particular business.

Key Concept

Let’s think about a scenario: “Okay, so if I order one specifical product (for example a shampoo) and my girlfriend tries to buy the same shampoo from her account to give a gift to her mother and also a cup, then in her order will be two products, and mine’s one product.”

Ah! the shampoo belongs to two orders and my girlfriend’s order contains more than one product.

@mauricioperez

03 why do we need a third table?

The Situation

The reason why we need to implement a middle table when dealing with many-to-many data, it's because data is going to be fixed on both the two tables, therefore **we can't go on saving data randomly on one of them (or both) to save the data dealing/transaction**.

Without a middle table, we'd be infringing norms and principles about database scalation.

Our middle entity will **hold the negotiation records between the two tables** which are dealing data.

@mauricioperez

04

Many-To-Many Relationship components

Key differentiations ahead!

When working with these kind of relationships we got to manually incorporate our middle table in the Java code with a specific annotation: **@JoinTable**.

You are going to incoporate this table on either one of the tables that'll be entering into negotiation of the Many-to-Many relationship.

In order to maintain the continuity's example, let's say I want to declare the middle table inside my orders entity, and my products entity will be mapped by the orders entity side.

@mauricioperez

05

Many-To-Many Relationship components

From the Orders entity.

So let's say we are in the Orders entity, and we are going to relate to products table. **Now, we need two annotations here: @ManyToMany and @JoinTable annotation.**

This is unique only for this entity, because I just wanted to be like that, **and I chose this to be the owner of the relation**, so that's why it contains these two annotations.

The order would be:

```
@ManyToMany(//some properties)
@JoinTable(// Some more properties)
private Set<Products> prods = new HashSet<>();
```

@mauricioperez

06

Many-To-Many Relationship components

From the Orders entity.

What's inside @ManyToMany?

1. **fetch**: Here you'll put the fetching mode, it's recommended to put it on EAGER mode.

2. **cascade**: Here you'll put the type of the database operation that you want it to execute for your saved related records to be affected in. The recommended mode is to put it on PERSIST.

@mauricioperez

07

Many-To-Many Relationship components

From the Orders entity.

What's inside @JoinTable?

1. **name**: Here you'll put the name of the middle-table's name you put in the database.
2. **joinColumns**: This will be owner relationship's id, in this case the Orders table. The params are name and referencedColumnName respectively.
3. **inverseJoinColumns**: Here's where you put the foreign table's id you want to relate (products) and the name of the id property you gave it into the table entity. The params are name and referencedColumnName respectively.

@mauricioperez

08

Many-To-Many Relationship components

From the Orders entity.

One last thing to highlight here is that the property that you define as type of the other table's entity as to be of type of a Java Collection either Lists or Sets.

In the example I will demonstrate I am using Set, and is of type of the other entity:

```
private Set<Products> prods = new HashSet<>();
```

@mauricioperez

09

Many-To-Many Relationship components

From the Products Entity.

This relationship's side will only hold one annotation: @ManyToMany.

Yes! Simple as that.

And here you'll be defining three params inside the annotation, no more.

The form that will get this property of the relation:

```
@ManyToMany(// Params here)
private Set<Orders> orders = new HashSet<>();
```

@mauricioperez

10

Many-To-Many Relationship components

From the Products Entity.

Inside the @ManyToMany annotation here, will be:

```
@ManyToMany (
    mappedBy = “prods”,
    fetch = FetchType.EAGER,
    cascade = CascadeType.PERSIST
)
```

The last params I already have explained them on a slides ago. Now let’s focus on what “mappedBy” means.

@mauricioperez

11

Many-To-Many Relationship components

From the Products Entity.

The question here is: What the hell does the mappedBy property means?

So to answer that question: Basically is just a property that you put inside the non-owner entity where you are telling it that **it’s already being owned by another entity in the relationship**, in this case because it’s Orders entity the owner, **Products entity has to declare who have already included it into the relationship by telling the name of the property that has been placed in the owner entity**, in that case was prod, of type Products.

@mauricioperez

Code Demonstration

12

@mauricioperez

Another example

In courses entity

```java
@Entity
public class Person extends BaseEntity{

    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.PERSIST)
    @JoinTable(name = "person_courses",
            joinColumns = {
                    @JoinColumn(name = "person_id", referencedColumnName = "personId")},
            inverseJoinColumns = {
                    @JoinColumn(name = "course_id", referencedColumnName = "courseId")})
    private Set<Courses> courses = new HashSet<>();
}
```

In instructor entity

```java
@Entity
public class Courses extends BaseEntity{

    @ManyToMany(mappedBy = "courses", fetch = FetchType.EAGER
            ,cascade = CascadeType.PERSIST)
    private Set<Person> persons = new HashSet<>();
}
```

Thank you!

@mauricioperez

Don’t forget about who’s going to be your entity relationship owner.