# Leçon 2 : Relation ManyToMany entre entités

**Enoncé du TP2 :**

L'objectif de ce TP est de relier l'entité `Film` existante avec une nouvelle entité nommée `Acteur` par une relation plusieurs à plusieurs (modélisée par l'annotation `@ManyToMany`). La règle de gestion étant : un ou plusieurs films sont tournés par un ou plusieurs acteurs et un ou plusieurs acteurs jouent dans un ou plusieurs films.

Cette relation sera modélisée dans la BD avec une table intermédiaire (nommé `film_acteurs`) dont la clé primaire est la jointure des clés primaires des tables film et acteur. On va considérer que l'entité `Acteur` est l'entité esclave dans cette relation (représenté par l'attribut `mappedBy` contenant l'instance de l'entité esclave).

## Etape 1 : Modification et ajout des entités

Dans le package `com.gestion.filmotheque.entities` on va Modifier l'entité Film en lui ajoutant un attribut de type liste des acteurs.

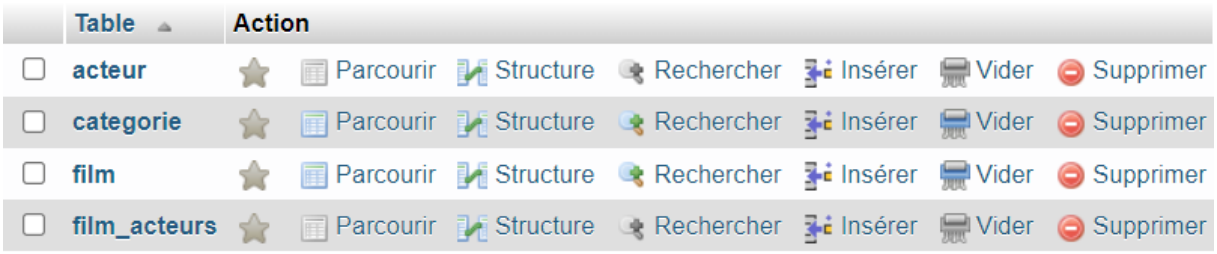
```
@ManyToMany
private List<Acteur> acteurs;
```

On va créer une nouvelle entité nommé `Acteur` comme suit :

```
package com.gestion.filmotheque.entities;

import java.util.List;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
public class Acteur {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String nom;
    private String prenom;
    @ManyToMany(mappedBy = "acteurs")
    private List<Film> films;

}
```

Démarrer l'application Spring Boot, les 2 tables `acteur` et `film_acteurs` sont crées dans la BD, comme le montre le log suivant :

```
Hibernate: create table acteur (id integer not null auto_increment, nom varchar(255), prenom varchar(25
Hibernate: create table film_acteurs (films_id integer not null, acteurs_id integer not null) engine=In
Hibernate: alter table film_acteurs add constraint FKne9plolj4btuawch1i3awynv9 foreign key (acteurs_id)
Hibernate: alter table film_acteurs add constraint FK3e4us7hqw0p65w8icfq2vj54a foreign key (films_id) r
```




## Etape 2 : Gestion des acteurs

Créer dans leurs packages respectifs tous les composants nécessaires pour la gestion des acteurs (indépendamment des films) à savoir les interfaces `ActeurRepository`, `IServiceActeur`, et les classes `ServiceActeur`, `ActeurController` et les vues associées.

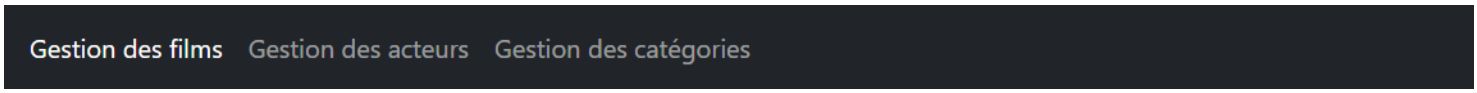
## Etape 3 : Modification du formulaire d'ajout d'un film

L'association entre film et acteurs sera faite pendant l'ajout d'un nouveau film.

Ajouter dans le formulaire d'ajout d'un nouveau film un champ de type select multiple nommé acteurs pour sélectionner les acteurs qui ont joués dans ce film. Les acteurs devrait être ajoutés préalablement dans la table.

## Etape 4 : Ajouter une barre de navigation entre les 3 gestions

Ajouter une barre de navigation permettant d'accéder à chacune des 3 options suivantes :




## Etape 5 : Effectuer la suppression d'un film par la méthode delete

Remplace le lien hypertexte de suppression par ce formulaire dont l'action correspond à l'URL du lien et la méthode à delete :

```
<form th:action="@{/film/delete/{id} (id=${f.id})}" th:method="delete">
    <input type="submit" value="Supprimer" class="btn btn-outline-danger">
</form>
```

Au fait, le protocole HTTP n'admet que 2 méthodes d'envoi `get` et `post`. Pour envoyer une requête HTTP avec une autre méthode (ici c'est `delete`) Spring Boot ajoute un champs masqué dans le formulaire contenant la méthode d'envoi delete et définit la méthode post par défaut. Pour autoriser la réception de ce champ masqué, il faut ajouter la propriété suivante dans `application.properties` :

```
spring.mvc.hiddenmethod.filter.enabled=true
```

Finalement dans le FilmController, changer le mapping de l'action de suppression qui était `@GetMapping` par `@DeleteMapping`.