Association ManyToMany unidirectionnelle

# Démonstration 10 du module 2

|  |
| --- |
| Les objectifs de cette démonstration :   * Déclaration d’une association ManyToMany unidirectionnelle * Manipulation des paramètres (cascade, orphanRemoval et fetch) * Utilisation de l’annotation @JoinColumn |

Déroulement

# Contexte

* Continuer l’application précédente
* Dans cette itération, notre application gère le métier de formateur :
  + Nous voulons pouvoir associer des formateurs et les cours qu’ils dispensent
  + Un formateur est un employé dans notre projet pour le moment, nous ne savons pas gérer l’héritage donc pour simplifier
    - Notre formateur a tous les attributs d’un employé
    - Mais pas ses associations, pour le moment
    - Tout cela se fera dans une itération future (avec la notion d’héritage)
* Pour cela nous allons ajouter la relation (M:N) entre Formateur et Cours :
* Voici le diagrammes des entités

Une image contenant texte, capture d’écran, Police, diagramme

Description générée automatiquement

# Création de la classe Cours :

* Création d’un package au package fr.eni.demo.bo : fr.eni.demo.bo.formation
* Utilisation des annotations de JPA pour en faire une entité

**package** fr.eni.demo.bo.formation;

**import** jakarta.persistence.\*;

**import** lombok.\*;

@NoArgsConstructor

@AllArgsConstructor

@Data

@Builder

@Entity

@Table(name = "COMPUTER\_COURSE")

**public** **class** Cours {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "COMPUTER\_COURSE\_ID")

**private** Integer id;

@Column(name = "COMPUTER\_SCIENCE\_COURSE", length = 150)

**private** String filiere;

@Column(name = "REFERENCE", unique = **true**, length = 15)

**private** String reference;

@Column(name = "TITLE", length = 250)

**private** String titre;

@Column(name = "DURATION")

**private** **int** duree;

}

# Création de la classe Formateur :

* Utilisation des annotations de JPA pour en faire une entité
* L’association ManyToMany doit préciser la table de jointure et les colonnes
  + Il faut placer le paramètre fetch = FetchType.LAZY
  + Il ne faut pas placer le paramètre cascade, car nous voulons conserver les cours en base même si un formateur est supprimé
  + La gestion de l’ajout, la modification ou la suppression des cours est indépendante des formateurs
  + Nous initialisons la liste pour éviter les NullPointerException
    - Avec Lombok, il faut utiliser l’annotation @Builder.Default et mettre = new ArrayList<>() ;

**package** fr.eni.demo.bo.formation;

**import** java.util.\*;

**import** jakarta.persistence.\*;

**import** lombok.\*;

@NoArgsConstructor

@AllArgsConstructor

@Getter

@Setter

@EqualsAndHashCode(of = { "immatriculation" })

@ToString

@Builder

/\*\*

\*

\* **@author** Eni Ecole

\*/

@Entity

@Table(name = "TRAINER")

**public** **class** Formateur {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "EMPLOYE\_ID")

**private** Integer id;

@Column(name = "LAST\_NAME", nullable = **false**, length = 90)

**private** String nom;

@Column(name = "FIRST\_NAME", nullable = **false**, length = 150)

**private** String prenom;

@Column(nullable = **false**, unique = **true**, length = 255)

**private** String email;

@Column(name = "EMPLOYEE\_REGISTRATION", nullable = **false**, unique = **true**, length = 100)

**private** String immatriculation;

@Column(name = "HOME\_PHONE\_NUMBER", length = 12)

**private** String numDom;

@Column(name = "CELL\_NUMBER", length = 12)

**private** String numPortable;

@Column(name = "COMPUTER\_SCIENCE\_COURSE", length = 150)

**private** String filiere;

//Association ManyToMany

@ManyToMany(fetch = FetchType.***LAZY***)

@JoinTable(name = "COMPUTER\_COURSES\_PROVIDED",

joinColumns = { @JoinColumn(name = "TRAINER\_ID") },

inverseJoinColumns = {@JoinColumn(name = "COMPUTER\_COURSE\_ID") })

@ToString.Exclude

**private** @Builder.Default List<Cours> coursDispenses = **new** ArrayList<>();

}

# Création de CoursRepository et FormateurRepository :

* Créer l’interface CoursRepository qui hérite de JpaRepository<Cours, Integer>
* Créer l’interface FormateurRepository qui hérite de JpaRepository<Formateur, Integer>

# Utilisation des tests unitaires

* Vérifions le bon fonctionnement de notre association ManyToMany unidirectionnelle
* Créer une classe TestManyToManyUni
  + Se faire injecter TestEntityManager, FormateurRepository et CoursRepository
* Créer une méthode de pré-chargement de plusieurs cours en base avant chaque test
* Créer 2 tests :
  + Persister un formateur et ses cours
  + Supprimer un formateur et valider que les cours sont toujours présents en base

**package** fr.eni.demo.association;

**import** **static** org.assertj.core.api.Assertions.*assertThat*;

**import** **static** org.junit.jupiter.api.Assertions.*assertNull*;

**import** java.util.\*;

**import** org.junit.jupiter.api.\*;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.test.autoconfigure.orm.jpa.\*;

**import** fr.eni.demo.bo.formation.\*;

**import** fr.eni.demo.dal.formation.\*;

**import** lombok.extern.slf4j.Slf4j;

@Slf4j

@DataJpaTest

**public** **class** TestManyToManyUni {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

FormateurRepository formateurRepository;

@Autowired

CoursRepository coursRepository;

**private** List<Cours> listeCoursDB = **new** ArrayList<>();

@BeforeEach

**public** **void** initCours() {

listeCoursDB.add(Cours

.*builder*()

.filiere("Développement")

.reference("DEV2023-M030")

.titre("Web Client HTML CSS")

.duree(5)

.build());

listeCoursDB.add(Cours

.*builder*()

.filiere("Développement")

.reference("DEV2023-M360")

.titre("Java Frameworks - APIs Web")

.duree(10)

.build());

listeCoursDB.add(Cours

.*builder*()

.filiere("Système et Réseau")

.reference("TSRR-120")

.titre("Bases des réseaux")

.duree(5)

.build());

listeCoursDB.add(Cours

.*builder*()

.filiere("Système et Réseau")

.reference("TSRR-170")

.titre("GLPI")

.duree(5)

.build());

listeCoursDB.forEach(c -> {

coursRepository.save(c);

// Vérification de l'identifiant

*assertThat*(c.getId()).isGreaterThan(0);

});

}

@Test

**public** **void** test\_save() {

**final** Formateur formateur = Formateur

.*builder*()

.nom("BAILLE")

.prenom("Anne-Lise")

.email("abaille@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_12398")

.numDom("02XXXXXXXX")

.filiere("Développement")

.build();

// Association ManyToMany

**final** List<Cours> coursDev = listeCoursDB

.stream()

.filter(item -> item.getFiliere().equals("Développement"))

.toList();

*assertThat*(coursDev).isNotNull();

*assertThat*(coursDev).isNotEmpty();

*assertThat*(coursDev.size()).isEqualTo(2);

formateur.setCoursDispenses(coursDev);

// Appel du comportement

**final** Formateur formateurDB = formateurRepository.save(formateur);

// Vérification de l'identifiant du formateur

*assertThat*(formateurDB.getId()).isGreaterThan(0);

// Vérification de la cascade de l'association

*assertThat*(formateurDB.getCoursDispenses()).isNotNull();

*assertThat*(formateurDB.getCoursDispenses()).isNotEmpty();

*assertThat*(formateurDB.getCoursDispenses().size()).isEqualTo(2);

***log***.info(formateurDB.toString());

}

@Test

**public** **void** test\_delete() {

**final** Formateur formateur = Formateur

.*builder*()

.nom("MATHIEU")

.prenom("Wilfrid")

.email("wmathieu@campus-eni.fr")

.immatriculation("ENI\_ECOLE\_14388")

.numDom("02XXXXXXXX")

.numPortable("06XXXXXXXX")

.filiere("Système et Réseau")

.build();

// Association ManyToMany

**final** List<Cours> coursRes = listeCoursDB

.stream()

.filter(item -> item.getFiliere().equals("Système et Réseau"))

.toList();

*assertThat*(coursRes).isNotNull();

*assertThat*(coursRes).isNotEmpty();

*assertThat*(coursRes.size()).isEqualTo(2);

formateur.setCoursDispenses(coursRes);

// Contexte de la DB

**final** Formateur formateurDB = entityManager.persist(formateur);

entityManager.flush();

*assertThat*(formateurDB.getId()).isGreaterThan(0);

*assertThat*(formateurDB.getCoursDispenses()).isNotNull();

*assertThat*(formateurDB.getCoursDispenses()).isNotEmpty();

List<Cours> coursDispensesDB = formateurDB.getCoursDispenses();

*assertThat*(coursDispensesDB).isNotNull();

*assertThat*(coursDispensesDB).isNotEmpty();

*assertThat*(coursDispensesDB.size()).isEqualTo(2);

// Appel du comportement

formateurRepository.delete(formateurDB);

// Vérification que l'entité a été supprimée

**final** Formateur formateurDB2 = entityManager.find(Formateur.**class**, formateur.getId());

*assertNull*(formateurDB2);

// Vérifier que tous les cours associés sont toujours en base - pas de suppressions de la table des cours

List<Cours> lstCoursDB = coursRepository.findAll();

*assertThat*(lstCoursDB).isNotNull();

*assertThat*(lstCoursDB).isNotEmpty();

*assertThat*(lstCoursDB.size()).isEqualTo(4);

}

}

* Tous les tests doivent être verts
  + Les tables créées sont :

create table computer\_course (

computer\_course\_id integer generated by default as identity,

duration integer,

reference varchar(15) unique,

computer\_science\_course varchar(150),

title varchar(250),

primary key (computer\_course\_id))

create table computer\_courses\_provided (

computer\_course\_id integer not null,

trainer\_id integer not null)

create table trainer (

employe\_id integer generated by default as identity,

cell\_number varchar(12),

home\_phone\_number varchar(12),

last\_name varchar(90) not null,

employee\_registration varchar(100) not null unique,

computer\_science\_course varchar(150),

first\_name varchar(150) not null,

email varchar(255) not null unique,

primary key (employe\_id))

alter table if exists computer\_courses\_provided

add constraint FKniemapf5aqoksngeksybdahes

foreign key (computer\_course\_id) references computer\_course

alter table if exists computer\_courses\_provided

add constraint FKio5431wa60hy2m8mex37pssdh

foreign key (trainer\_id) references trainer

* + 3 tables dont une table de jointure