Association ManyToOne

# Démonstration 9 du module 2

|  |
| --- |
| L’objectif de cette démonstration :   * Déclaration d’une association ManyToOne |

Déroulement

# Contexte

* Continuer l’application précédente
* Dans cette itération, nous voulons ajouter la notion de civilité ; les possibilités sont :
  + Monsieur (M)
  + Madame (Mme)
  + Mix (Mx)
* Pour cela nous allons ajouter la relation (N:1) entre Employe et Civilite :
* Voici le diagrammes des entités

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

Description générée automatiquement

# Création de la classe Civilite :

* Utilisation des annotations de JPA pour en faire une entité
  + Sa clef primaire est gérée par le code donc par de générateur de clef
  + Le libellé doit être unique

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

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

**import** lombok.\*;

@NoArgsConstructor

@AllArgsConstructor

@Data

@Builder

@Entity

@Table(name = "CIVILITY")

**public** **class** Civilite {

@Id

@Column(name = "CIVILITY\_ID", length = 3)

**private** String clef;

@Column(name = "LABEL", unique = **true**, length = 50)

**private** String libelle;

}

# Modification de la classe Employe :

* Ajout d’un attribut Civilite avec l’annotation @ManyToOne
  + Préciser la colonne de jointure
* Les civilités sont gérées par le code ou un script en base
  + Il existe : Monsieur ou Madame
  + Il n’y a aucune autre possibilité pour le moment
* Il ne faut pas de cascade sur cette association
* Et par défaut c’est un fetch EAGER qui sera mis en place par l’ORM cela nous convient

//Ajout de l'association ManyToOne

@ManyToOne

@JoinColumn(name = "CIVILITY\_ID")

**private** Civilite civilite;

# Création de CiviliteRepository :

* Créer l’interface CiviliteRepository qui hérite de JpaRepository<Civilite, String>

# Utilisation des tests unitaires

* Vérifions le bon fonctionnement de notre association ManyToOne
* Créer une classe TestManyToOne
  + Se faire injecter TestEntityManager, EmployeRepository et CiviliteRepository
* Créer une méthode de pré-chargement des civilités en base avant chaque test
* Créer 3 tests :
  + Persister un employé et sa civilité
  + Récupérer un ensemble d’employés avec des civilités différentes
  + Supprimer un employé et valider que les civilités sont toujours présentes 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.\*;

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

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

@Slf4j

@DataJpaTest

**public** **class** TestManyToOne {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

CiviliteRepository civiliteRepository;

@Autowired

EmployeRepository employeRepository;

**private** Civilite monsieur;

**private** Civilite madame;

**private** Civilite mx;//Non binaire

@BeforeEach

**public** **void** initCivilite() {

monsieur = Civilite

.*builder*()

.clef("M")

.libelle("Monsieur")

.build();

madame = Civilite

.*builder*()

.clef("Mme")

.libelle("Madame")

.build();

mx = Civilite

.*builder*()

.clef("Mx")

.libelle("Mix")

.build();

civiliteRepository.save(monsieur);

civiliteRepository.save(madame);

civiliteRepository.save(mx);

}

@Test

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

**final** Employe employe = Employe

.*builder*()

.nom("DELACHESNAIS")

.prenom("Frédéric")

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

.immatriculation("ENI\_ECOLE\_14398")

.numDom("02XXXXXXXX")

.numPortable("06XXXXXXXX")

.build();

// Association ManyToOne

employe.setCivilite(mx);

// Appel du comportement

**final** Employe employeDB = employeRepository.save(employe);

// Vérification de l'identifiant de l'employé

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

// Vérification de sa civilité

*assertThat*(employeDB.getCivilite()).isNotNull();

*assertThat*(employeDB.getCivilite()).isEqualTo(mx);

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

}

@Test

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

List<Employe> employes = jeuDeDonnees();

// sauver le jeu de données en base

employes.forEach(emp ->{

entityManager.persist(emp);

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

});

// Appel du comportement

**final** List<Employe> employesDB = employeRepository.findAll();

// Vérification de l'identifiant des employés

employesDB.forEach(emp ->{

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

// Vérification de la civilité

*assertThat*(emp.getCivilite()).isNotNull();

});

}

@Test

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

**final** Employe employe = Employe

.*builder*()

.nom("DELACHESNAIS")

.prenom("Frédéric")

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

.immatriculation("ENI\_ECOLE\_14398")

.numDom("02XXXXXXXX")

.numPortable("06XXXXXXXX")

.build();

// Association ManyToOne

employe.setCivilite(mx);

// Contexte de la DB

**final** Employe employeDB = entityManager.persist(employe);

entityManager.flush();

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

*assertThat*(employeDB.getCivilite()).isNotNull();

*assertThat*(employeDB.getCivilite()).isEqualTo(mx);

// Appel du comportement

employeRepository.delete(employeDB);

// Vérification que l'entité a été supprimée

**final** Employe employeDB2 = entityManager.find(Employe.**class**, employeDB.getId());

*assertNull*(employeDB2);

// Vérifier que tous les civilités sont toujours présentes - PAS de cascade

**final** List<Civilite> civilites = civiliteRepository.findAll();

*assertThat*(civilites).isNotNull();

*assertThat*(civilites).isNotEmpty();

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

*assertThat*(civilites.size()).isEqualTo(3);

}

**private** List<Employe> jeuDeDonnees() {

List<Employe> employes = **new** ArrayList<>();

employes.add(Employe

.*builder*()

.civilite(madame)

.nom("BAILLE")

.prenom("Anne-Lise")

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

.immatriculation("ENI\_ECOLE\_12398")

.numDom("02XXXXXXXX")

.build());

employes.add(Employe

.*builder*()

.civilite(mx)

.nom("DELACHESNAIS")

.prenom("Frédéric")

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

.immatriculation("ENI\_ECOLE\_14398")

.numDom("02XXXXXXXX")

.numPortable("06XXXXXXXX")

.build());

employes.add(Employe

.*builder*()

.civilite(monsieur)

.nom("NICOLAS")

.prenom("Cédric")

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

.immatriculation("ENI\_ECOLE\_10100")

.numDom("02XXXXXXXX")

.numPortable("06XXXXXXXX")

.build());

**return** employes;

}

}

* Tous les tests doivent être verts