Association OneToOne unidirectionnelle

# Démonstration 5 du module 2

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

Déroulement

# Contexte

* Continuer l’application précédente
* Dans cette itération, nous allons ajouter la relation (1:1) entre Employe et Adresse :
  + Voici le diagrammes des entités

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

Description générée automatiquement

# Entité Adresse :

* Il n’y a pas de modification à apporter à cette entité

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

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

**import** lombok.\*;

@NoArgsConstructor

@AllArgsConstructor

@Data

@Builder

@Entity

@Table(name="ADDRESS")

**public** **class** Adresse {

@Id

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

@Column(name = "ADDRESS\_ID")

**private** Integer id;

@Column(name = "STREET",nullable = **false**, length = 250)

**private** String rue;

@Column(name = "POSTAL\_CODE",nullable = **false**, length = 5)

**private** String codePostal;

@Column(name = "CITY",nullable = **false**, length = 150)

**private** String ville;

}

# Modification de la classe d’entité Employe :

* Ajout d’un attribut d’Adresse
* Utilisation des annotations @OneToOne et @JoinColumn

**package** fr.eni.demo.bo.pk2;

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

**import** lombok.\*;

@NoArgsConstructor

@AllArgsConstructor

@Getter

@Setter

@EqualsAndHashCode(of= {"immatriculation"})

@ToString

@Builder

/\*\*

\*

\* **@author** Eni Ecole

\*/

@Entity

@Table(name="EMPLOYEE")

**public** **class** Employe {

…

@OneToOne

@JoinColumn(name = "ADDRESS\_ID")

**private** Adresse adresse;

}

# Utilisation des tests unitaires

* Pour le moment, nous n’avons pas préciser les paramètres :
  + cascade,
  + orphanRemoval
  + Ou fetch
* Testons le comportement par défaut de l’annotation @OneToOne
* Relancer la classe de tests unitaires des employés : TestEmployeRepository
  + Tout est vert
  + Spring Data Jpa a créé :
    - Une table pour l’entité Adresse
    - Une table pour l’entité Employé
    - Et placer une clef de jointure

create table address (

address\_id integer generated by default as identity,

postal\_code varchar(5) not null,

city varchar(150) not null,

street varchar(250) not null,

primary key (address\_id))

create table employee (

employee\_id integer generated by default as identity,

address\_id integer unique,

cell\_number varchar(12),

home\_phone\_number varchar(12),

last\_name varchar(90) not null,

employee\_registration varchar(100) not null unique,

first\_name varchar(150) not null,

email varchar(255) not null unique,

primary key (employee\_id))

alter table if exists employee add constraint FKjg6213xja82kb6egh7ohv2xc9 foreign key (address\_id) references address

Cependant, l’adresse des employés est nulle pour le moment

Les tests que nous allons réaliser sortent des tests de base sur les Repository. Pour cela, nous allons créer des classes de tests indépendantes

## Test du paramètre cascade

### Ajout d’un employé avec une adresse

* Créer une nouvelle classe de tests unitaires : TestOneToOneUni
* Nous allons travailler en mode TDD (Test Driven Development)
  + Créer un test qui sera en échec la première fois
  + Modifier les paramètres de la classe pour valider le test
* Créer un test pour sauvegarder un employé et son adresse en persistant l’employé
  + Vérifier les identifiants des 2 entités

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

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

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

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

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

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

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

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

@Slf4j

@DataJpaTest

**public** **class** TestOneToOneUni {

@Autowired

EmployeRepository repository;

@Test

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

**final** Adresse adresse = Adresse

.*builder*()

.rue("15 rue de Paris")

.codePostal("35000")

.ville("Rennes")

.build();

**final** Employe employeASauver = Employe

.*builder*()

.nom("BAILLE")

.prenom("Anne-Lise")

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

.immatriculation("ENI\_ECOLE\_12398")

.numDom("02XXXXXXXX")

.build();

// Association

employeASauver.setAdresse(adresse);

// Appel du comportement

**final** Employe employeDB = repository.save(employeASauver);

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

// Vérification de l'identifiant de l'employé

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

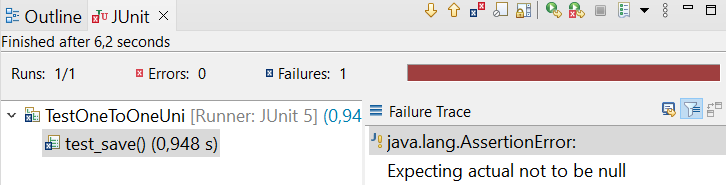
// Vérification de la cascade de l'association

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

}

}

* La ou les lignes correspondent dans nos tests aux associations uni ou bi directionnelles seront à ajouter dans les méthodes des services (couche BLL)
* Lancer le test, il est en échec, il indique que l’identifiant d’adresse est nul



* + L’employé a été persisté mais pas l’adresse associée
* Ajouter le paramètre cascade à ALL
  + Les 2 entités ont un cycle de vie identique, il faut donc les gérer ensemble.

**…**

**import** jakarta.persistence.CascadeType;

…

@OneToOne(cascade = CascadeType.***ALL***)

@JoinColumn(name = "ADDRESS\_ID")

**private** Adresse adresse;

}

* Relancer le test
  + Il est vert maintenant, les 2 entités ont été persistées
  + Dans les traces nous avons les 2 INSERT

insert into address (postal\_code,street,city,address\_id) values (?,?,?,default)

insert into employee (address\_id,email,employee\_registration,last\_name,home\_phone\_number,cell\_number,first\_name,employee\_id) values (?,?,?,?,?,?,?,default)

* + Et dans la trace l’employé et son adresse ont un identifiant

Employe(id=1, nom=BAILLE, prenom=Anne-Lise, email=abaille@campus-eni.fr, immatriculation=ENI\_ECOLE\_12398, numDom=02XXXXXXXX, numPortable=null, adresse=Adresse(id=1, rue=15 rue de Paris, codePostal=35000, ville=Rennes))

### Suppression d’un employé avec une adresse

* Le paramètre cascade est à ALL
  + Il gère : PERSIST, MERGE, REFRESH, REMOVE et DETACH
  + Donc si nous supprimons l’entité principale, il doit supprimer l’entité associée
* Créer un test pour valider cela
* Ajout d’un la classe de tests de TestEtityManager et du test test\_delete

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

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

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

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

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

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

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

**import** fr.eni.demo.bo.Adresse;

**import** fr.eni.demo.bo.Employe;

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

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

@Slf4j

@DataJpaTest

**public** **class** TestOneToOneUni {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

EmployeRepository repository;

@Test

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

**final** Adresse adresse = Adresse

.*builder*()

.rue("15 rue de Paris")

.codePostal("35000")

.ville("Rennes")

.build();

**final** Employe employeASauver = Employe

.*builder*()

.nom("BAILLE")

.prenom("Anne-Lise")

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

.immatriculation("ENI\_ECOLE\_12398")

.numDom("02XXXXXXXX")

.build();

// Association

employeASauver.setAdresse(adresse);

// Appel du comportement

**final** Employe employeDB = repository.save(employeASauver);

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

// Vérification de l'identifiant de l'employé

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

// Vérification de la cascade de l'association

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

}

@Test

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

**final** Adresse adresse = Adresse

.*builder*()

.rue("15 rue de Paris")

.codePostal("35000")

.ville("Rennes")

.build();

**final** Employe employe = Employe

.*builder*()

.nom("BAILLE")

.prenom("Anne-Lise")

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

.immatriculation("ENI\_ECOLE\_12398")

.numDom("02XXXXXXXX")

.build();

// Association

employe.setAdresse(adresse);

// Contexte de la DB

entityManager.persist(employe);

entityManager.flush();

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

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

// Appel du comportement

repository.delete(employe);

// Vérification que l'entité a été supprimée

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

*assertNull*(employeDB);

Adresse adresseDB = entityManager.find(Adresse.**class**, adresse.getId());

*assertNull*(adresseDB);

}

}

* Les 2 tests passent sans souci

## Test du paramètre orphanRemoval

C’est un paramètre pour supprimer des entités orphelines.

* Dans notre exemple, nous positionnons un CascadeType.ALL
* Si notre employé est supprimé, automatiquement notre adresse l’est
* Cependant, si nous utilisons setAdresse(null), alors cela ne supprimera pas l’adresse qui était associée
  + Créer un test pour valider cette remarque, nous fonctionnons encore en TDD :

…

@Test

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

**final** Adresse adresse = Adresse

.*builder*()

.rue("15 rue de Paris")

.codePostal("35000")

.ville("Rennes")

.build();

**final** Employe employe = Employe

.*builder*()

.nom("BAILLE")

.prenom("Anne-Lise")

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

.immatriculation("ENI\_ECOLE\_12398")

.numDom("02XXXXXXXX")

.build();

// Association

employe.setAdresse(adresse);

// Contexte de la DB

entityManager.persist(employe);

entityManager.flush();

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

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

//Supprimer le lien entre l'entité Employe et l'entité Adresse

employe.setAdresse(**null**);

// Appel du comportement

repository.delete(employe);

// Vérification que l'entité a été supprimée

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

*assertNull*(employeDB);

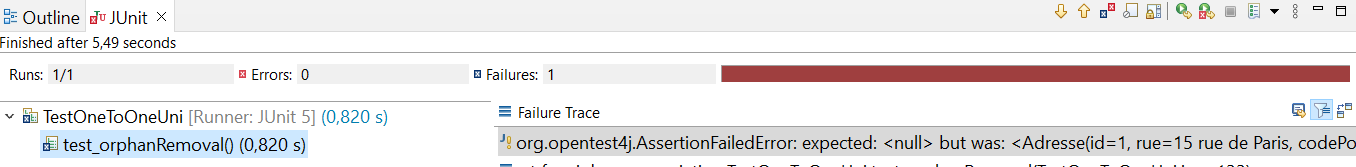
Adresse adresseDB = entityManager.find(Adresse.**class**, adresse.getId());

*assertNull*(adresseDB);

}

}

* Exécuter le test, il est en échec



* + L’adresse n’a pas été supprimée de la base
* Pour régler ce phénomène, il faut ajouter le paramètre orphanRemoval à true

…

@OneToOne(cascade = CascadeType.***ALL***, orphanRemoval = **true**)

@JoinColumn(name = "ADDRESS\_ID")

**private** Adresse adresse;

}

* Relancer le test, il passe maintenant sans souci.

## Test du paramètre fetch

### EAGER par défaut sur OneToOne

@DataJpaTest induit pour chaque test la création d’un contexte de transaction appelée Session. Pour montrer le comportement du paramètre **fetch**, il faut être dans 2 contextes transactionnels différents

* Un pour sauvegarder les données
* Un pour récupérer les données
* Sinon, tout est dans le cache de l’ORM et on ne peut rien prouver. C’est le but de l’ORM moins de requêtes vers la base, conserver les entités en cache.

Nous allons utiliser le contexte de Spring Boot directement pour ces tests

* + @SpringBootTest
  + Nous serons donc sur la vraie base de données : Microsoft SQL Server
  + Nous créons une méthode annotée @BeforeEach qui sauvegardera l’employé et son adresse
  + Puis un test pour valider le comportement du fetch
* Création d’une nouvelle classe de test : TestOneToOneUniFetch
* Sur une association OneToOne, par défaut le comportement de l’ORM est en EAGER
  + Il remonte l’entité et l’entité associée en 1 requête
  + Vérifier cela avec le test suivant :

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

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

**import** java.util.Optional;

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

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

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

**import** org.springframework.boot.test.context.SpringBootTest;

**import** fr.eni.demo.bo.Adresse;

**import** fr.eni.demo.bo.Employe;

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

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

@SpringBootTest

@Slf4j

**public** **class** TestOneToOneUniFetch {

@Autowired

EmployeRepository repository;

Adresse adresse ;

Employe employe ;

@BeforeEach

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

adresse = Adresse

.*builder*()

.rue("15 rue de Paris")

.codePostal("35000")

.ville("Rennes")

.build();

employe = Employe

.*builder*()

.nom("BAILLE")

.prenom("Anne-Lise")

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

.immatriculation("ENI\_ECOLE\_12398")

.numDom("02XXXXXXXX")

.build();

//Association

employe.setAdresse(adresse);

// Appel du comportement

**final** Employe employe = repository.save(employe);

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

// Vérification de l'identifiant de l'employé

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

//Vérification de la cascade de l'association

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

}

@Test

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

// Appel du traitement

**int** id = employe.getId();

**final** Optional<Employe> op = repository.findById(id);

// Vérification que l'Optional contienne une entité

*assertThat*(op.isPresent()).isTrue();

// Récupération de l'entité

**final** Employe employeDB = op.get();

// Validation de l'entité

*assertThat*(employeDB.getId()).isEqualTo(id);

*assertThat*(employeDB.getImmatriculation()).isEqualTo(employe.getImmatriculation());

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

Adresse adresseDB = employeDB.getAdresse();

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

*assertThat*(adresseDB.getCodePostal()).isNotNull();

*assertThat*(adresseDB.getCodePostal()).isEqualTo(adresse.getCodePostal());

}

}

* Lancer ce test
  + Dans les traces, il y a bien un SELECT LEFT JOIN pour associer les 2 tables

select e1\_0.employee\_id,a1\_0.address\_id,a1\_0.postal\_code,a1\_0.street,a1\_0.city,e1\_0.email,e1\_0.employee\_registration,e1\_0.last\_name,e1\_0.home\_phone\_number,e1\_0.cell\_number,e1\_0.first\_name from employee e1\_0

left join address a1\_0 on a1\_0.address\_id=e1\_0.address\_id where e1\_0.employee\_id=?

* + L’ORM a bien chargé en 1 requête les 2 entités (EAGER)

### fetch a LAZY

* Ajouter le paramètre fetch et le mettre à LAZY

**…**

**import** jakarta.persistence.FetchType;

…

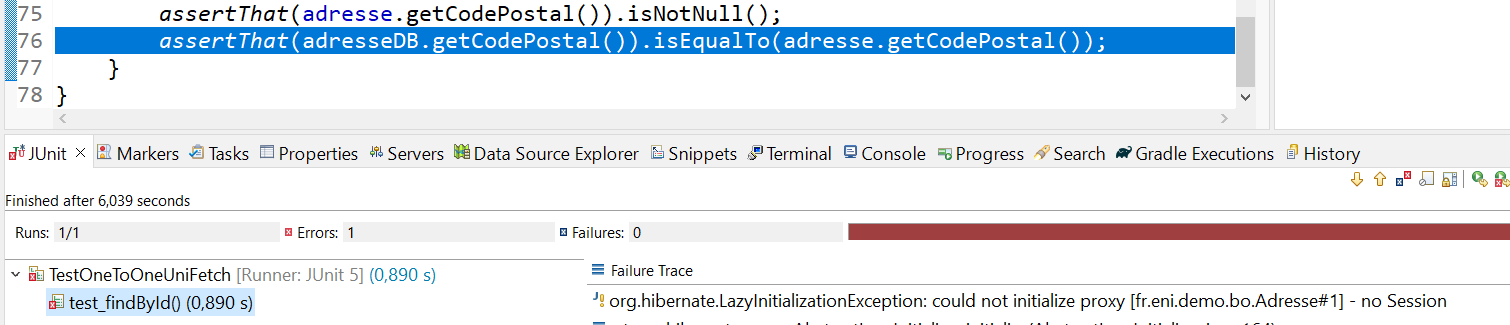
@OneToOne(cascade = CascadeType.***ALL***, orphanRemoval = **true** , fetch = FetchType.***LAZY***)

@JoinColumn(name = "ADDRESS\_ID")

**private** Adresse adresse;

}

* + Le but est de charger que le premier niveau de l’association c’est-à-dire l’employé dans notre cas
  + L’adresse sera chargée que lorsqu’il faudra la manipuler
* Le but est d’éviter de saturer le cache de l’ORM pour des données peu utilisées
* Relancer Le test
  + Il est en erreur :



* + Il ne peut pas accéder aux attributs d’adresse hormis l’identifiant
* Le principe est que l’ORM remonte uniquement l’identifiant de l’association pour pouvoir aller chercher l’entité si besoin
  + Vous pouvez le vérifier par rapport à la requête réalisée :

select e1\_0.employee\_id,e1\_0.address\_id,e1\_0.email,e1\_0.employee\_registration,e1\_0.last\_name,e1\_0.home\_phone\_number,e1\_0.cell\_number,e1\_0.first\_name from employee e1\_0 where e1\_0.employee\_id=?

* + Il y a bien la clef étrangère qui est l’identifiant de l’adresse de l’employé
  + Mais pas le LEFT JOIN avec la table ADDRESS

Remarques :

* Dans un projet en exécution, nous serions avec une notion de session et de transaction active qui permettrait que l’ORM aille en base de données lors de l’accès à l’adresse
* Dans les tests unitaires, on a un contexte de transaction séparée.

### fetch a EAGER

* Activer le paramètre fetch a EAGER pour vérifier que c’était le comportement par défaut

…

@OneToOne(cascade = CascadeType.***ALL***, orphanRemoval = **true**, fetch = FetchType.***EAGER***)

@JoinColumn(name = "ADDRESS\_ID")

**private** Adresse adresse;

}

* Relancer le test
  + Il est de nouveau vert
  + Et nous retrouvons la requête avec jointure