Cave à vin (partie 02)

# TP02 du module 02 – Spring Data

|  |
| --- |
| Ce TP sera un TP fil rouge du cours   * Création d’une association OneToOne |

|  |
| --- |
| **Durée estimée** |
| 30 minutes et 1 heure |

Solution

# Créer ou modifier les entités

## Voici le diagramme des entités à réaliser :

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

Description générée automatiquement

Voici le diagramme des tables en base de données :

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

Description générée automatiquement

* Le package des classes est fr.eni.cave.bo.client
* Utiliser les annotations de JPA
  + Appliquer les annotations pour l’association OneToOne
* Utiliser les annotations de Lombok

## Création de l’entité Adresse :

**package** fr.eni.cave.bo.client;

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

**import** lombok.\*;

@NoArgsConstructor

@AllArgsConstructor

@Data

@Builder

@Entity

@Table(name="CAV\_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 l’entité Client :

* Ajout de l’attribut adresse pour gérer l’association OneToOne
  + Elle est unidirectionnelle
  + Il faut imposer la contrainte d’une composition. Les 2 entités ont le même cycle de vie à cascade = CascadeType.ALL
  + Il faut que si l’adresse devient orpheline, elle soit supprimée à orphanRemoval = true
  + Et par défaut les 2 entités sont chargées à fetch = FetchType.EAGER

**package** fr.eni.cave.bo.client;

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

**import** lombok.\*;

@NoArgsConstructor

@AllArgsConstructor

@Getter

@Setter

@ToString(of = { "pseudo", "nom", "prenom" })

@EqualsAndHashCode(of = { "pseudo" })

@Entity

@Table(name = "CAV\_CLIENT")

@Builder

**public** **class** Client {

…

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

@JoinColumn(name = "ADDRESS\_ID")

**private** Adresse adresse;

}

# Créer AdresseRepository

**package** fr.eni.cave.dal;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** fr.eni.cave.bo.client.Adresse;

**public** **interface** AdressRepository **extends** JpaRepository<Adresse, Integer> {

}

# Créer des tests unitaires

* En vous aidant des tests unitaires pour valider l’association OneToOne de la démonstration
* Créer les tests unitaires pour cette association

**package** fr.eni.cave.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.cave.bo.client.Adresse;

**import** fr.eni.cave.bo.client.Client;

**import** fr.eni.cave.dal.ClientRepository;

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

@Slf4j

@DataJpaTest

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

@Autowired

**private** TestEntityManager entityManager;

@Autowired

ClientRepository repository;

@Test

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

**final** Adresse adresse = Adresse

.*builder*()

.rue("15 rue de Paris")

.codePostal("35000")

.ville("Rennes")

.build();

**final** Client client = Client

.*builder*()

.pseudo("bobeponge@email.fr")

.password("carré")

.nom("Eponge")

.prenom("Bob")

.build();

// Association

client.setAdresse(adresse);

// Appel du comportement

**final** Client clientDB = repository.save(client);

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

// Vérification de la cascade de l'association

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

*assertThat*(clientDB.getAdresse().getId()).isGreaterThan(0);

}

@Test

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

**final** Adresse adresse = Adresse

.*builder*()

.rue("15 rue de Paris")

.codePostal("35000")

.ville("Rennes")

.build();

**final** Client client = Client

.*builder*()

.pseudo("bobeponge@email.fr")

.password("carré")

.nom("Eponge")

.prenom("Bob")

.build();

// Association

client.setAdresse(adresse);

// Contexte de la DB

entityManager.persist(client);

entityManager.flush();

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

// Appel du comportement

repository.delete(client);

// Vérification que l'entité a été supprimée

Client clientDB = entityManager.find(Client.**class**, client.getPseudo());

*assertNull*(clientDB);

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

*assertNull*(adresseDB);

}

@Test

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

**final** Adresse adresse = Adresse

.*builder*()

.rue("15 rue de Paris")

.codePostal("35000")

.ville("Rennes")

.build();

**final** Client client = Client

.*builder*()

.pseudo("bobeponge@email.fr")

.password("carré")

.nom("Eponge")

.prenom("Bob")

.build();

// Association

client.setAdresse(adresse);

// Contexte de la DB

entityManager.persist(client);

entityManager.flush();

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

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

client.setAdresse(**null**);

// Appel du comportement

repository.delete(client);

// Vérification que l'entité a été supprimée

Client clientDB = entityManager.find(Client.**class**, client.getPseudo());

*assertNull*(clientDB);

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

*assertNull*(adresseDB);

}

}

* Les 3 tests unitaires valident
  + La création, la suppression et la mise à jour des entités en cascade
  + Le dernier test permet de valider que si l’entité associée devient orpheline, elle sera supprimée. Les entités détachées sont supprimées
* Les tables créées pour l’association Client-Adresse sont :

create table cav\_client (address\_id integer unique, passwpord varchar(68) not null, last\_name varchar(90) not null, first\_name varchar(150) not null, login varchar(255) not null, primary key (login))

create table cav\_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))

* + Voici la clef étrangère (jointure) ajoutée :

alter table if exists cav\_client add constraint FKo8rjcgcw5ak4bheqeya5jxtio foreign key (address\_id) references cav\_address

* Voici le type de trace obtenu :

Client(pseudo=bobeponge@email.fr, nom=Eponge, prenom=Bob)

Adresse(id=1, rue=15 rue de Paris, codePostal=35000, ville=Rennes)