Relation 1-1

Démonstration 5 du module 5

Les objectifs de cette démonstration sont

- Mise en place d'une relation 1-1 unidirectionnelle entre une personne et son adresse
- Mise en place d'une relation 1-1 bidirectionnelle entre une personne et son adresse

Contexte

- Continuer dans le projet précédent

Déroulement

1. Relation 1-1 unidirectionnelle

- Nous reprenons la classe Personne mais avec une simple clef (IDENTITY)

```
package fr.eni.demo.oneone.uni;

import java.time.LocalDate;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Transient;

@Entity(name = "personne_OTO_u")
@Table(name = "personne_OTO_u")
public class Personne {

	@Id
	private long id;

	private String nom;
	private String prenom;
	private LocalDate dateNaissance;

	@Transient
	private int age;

	@OneToOne(cascade = CascadeType.PERSIST)
	private Adresse adresse;

	public Personne() {
	}

	public long getId() {
		return id;
	}

	public void setId(long id) {
```

```
		this.id = id;
	}

	public String getNom() {
		return nom;
	}

	public void setNom(String nom) {
		this.nom = nom;
	}

	public String getPrenom() {
		return prenom;
	}

	public void setPrenom(String prénom) {
		this.prenom = prénom;
	}

	public LocalDate getDateNaissance() {
		return dateNaissance;
	}

	public void setDateNaissance(LocalDate dateNaissance) {
		this.dateNaissance = dateNaissance;
	}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

}
```

- Nous créons une classe Adresse :

```
package fr.eni.demo.oneone.uni;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity(name = "adresse_OTO_u")
@Table(name = "adresse_OTO_u")
public class Adresse {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;

	private String codePostal;
	private String ville;

	public Adresse() {
	}

	public Adresse(String codePostal, String ville) {
		this.codePostal = codePostal;
		this.ville = ville;
	}
```

```
	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public String getCodePostal() {
		return codePostal;
	}

	public void setCodePostal(String codePostal) {
		this.codePostal = codePostal;
	}

	public String getVille() {
		return ville;
	}

	public void setVille(String ville) {
		this.ville = ville;
	}

	@Override
	public String toString() {
		return "Adresse [id=" + id + ", codePostal=" + codePostal + ", ville=" + ville + "]";
	}
}
```

- Association l'adresse de la personne
  - Ajouter l'attribut adresse et utiliser l'annotation @OneToOne
  - Un constructeur avec les paramètres
  - Ajouter le Getter/Setter
  - Redéfinir toString

```
package fr.eni.demo.oneone.uni;

import java.time.LocalDate;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;
import javax.persistence.Transient;

@Entity(name = "personne_OTO_u")
@Table(name = "personne_OTO_u")
public class Personne {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private long id;

	private String nom;
	private String prenom;
	private LocalDate dateNaissance;

	@Transient
	private int age;
```

```
	@OneToOne(cascade = CascadeType.PERSIST)
	private Adresse adresse;
…

	public Personne(String nom, String prenom, LocalDate dateNaissance, Adresse adresse) {
		this.nom = nom;
		this.prenom = prenom;
		this.dateNaissance = dateNaissance;
		this.adresse = adresse;
	}
…

	public Adresse getAdresse() {
		return adresse;
	}

	public void setAdresse(Adresse adresse) {
		this.adresse = adresse;
	}

	@Override
	public String toString() {
		return "Personne [id=" + id + ", nom=" + nom + ", prenom=" + prenom + ", dateNaissance=" +
dateNaissance
				+ ", age=" + age + ", adresse=" + adresse + "]";
	}

}
```

- L'association OneToOne avec cascade = PERSIST; va permettre que lors de la sauvegarde de la personne, l'adresse le soit aussi.

- Création d'un Repository pour gérer l'entité

```
package fr.eni.demo.oneone.uni;

import org.springframework.data.repository.CrudRepository;

public interface PersonneOTOURepository extends CrudRepository<Personne, Long>{

}
```

- Dans la classe d'exécution de l'application
  - Positionner l'annotation @Profile(«demo») sur le bean précédent et renommer le package com des entités
  - Copier le code du nouveau bean :

```
@Bean
public CommandLineRunner demoOneToOneUni(PersonneOTOURepository repository) {
	return (args) -> {
		fr.eni.demo.oneone.uni.Adresse a1 = new fr.eni.demo.oneone.uni.Adresse("75000", "Paris");
		fr.eni.demo.oneone.uni.Adresse a2 = new fr.eni.demo.oneone.uni.Adresse("35000", "Rennes");

		fr.eni.demo.oneone.uni.Personne p1 = new fr.eni.demo.oneone.uni.Personne ("Legrand", "Lucie",
				LocalDate.parse("2008-06-18"), a1);
		fr.eni.demo.oneone.uni.Personne p2 = new fr.eni.demo.oneone.uni.Personne ("Legrand", "Lucie2",
				LocalDate.parse("2006-04-03"), a2);

		repository.save(p1);
		repository.save(p2);

		// fetch all
```

```
            System.out.println("Liste des personnes : ");
            System.out.println("------------------------------");
            for (fr.eni.demo.oneone.uni.People p : repository.findAll()) {
                  System.out.println(p.toString());
            }
      };}
```

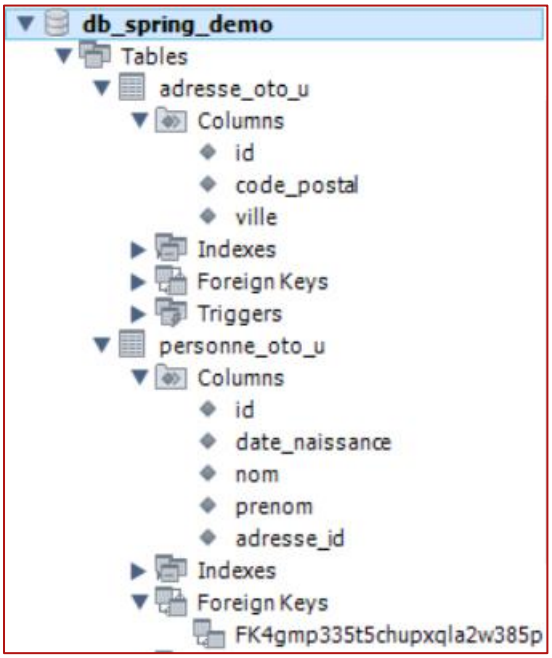
- Traces d'exécution :

```
…

Hibernate:

    create table adresse_oto_u (
       id integer not null auto_increment,
        code_postal varchar(255),
        ville varchar(255),
        primary key (id)
    ) engine=InnoDB
Hibernate:

    create table personne_oto_u (
       id bigint not null auto_increment,
        date_naissance date,
        nom varchar(255),
        prenom varchar(255),
        adresse_id integer,
        primary key (id)
    ) engine=InnoDB
Hibernate:

    alter table personne_oto_u
       add constraint FK4gmp335t5chupxqla2w385pob
       foreign key (adresse_id)
       references adresse_oto_u (id)

…
Liste des personnes :
------------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.adresse_id as adresse_5_1_,
        personne0_.date_naissance as date_nai2_1_,
        personne0_.nom as nom3_1_,
        personne0_.prenom as prenom4_1_
    from
        personne_oto_u personne0_
Hibernate:
    select
        adresse0_.id as id1_0_0_,
        adresse0_.code_postal as code_pos2_0_0_,
        adresse0_.ville as ville3_0_0_
    from
        adresse_oto_u adresse0_
    where
        adresse0_.id=?
Hibernate:
    select
        adresse0_.id as id1_0_0_,
        adresse0_.code_postal as code_pos2_0_0_,
        adresse0_.ville as ville3_0_0_
    from
        adresse_oto_u adresse0_
    where
        adresse0_.id=?
```

```
Personne [id=1, nom=Legrand, prenom=Lucie, dateNaissance=2008-06-18, age=0, adresse=Adresse [id=1,
codePostal=75000, ville=Paris]]
Personne [id=2, nom=Legrand, prenom=Lucie2, dateNaissance=2006-04-03, age=0, adresse=Adresse [id=2,
codePostal=35000, ville=Rennes]]
```

- Création 2 table avec une clef de jointure

- En base, on retrouve bien cela :



2. Relation 1-1 bidirectionnelle

- Dans ce cas, il y a dans chaque classe, un attribut de l'autre classe
- Pour éviter que l'ORM ne passe son temps à aller d'association en association (boucler)
    - Il faut préciser que l'ORM ne regarde qu'un côté de l'association.
    - Pour cela, il faut utiliser l'attribut mappedBy
    - Le choix est arbitraire. En général, on garde en direct celui qui sera le plus utilisé au niveau vue. Dans notre cas, personne est plus importante qu'adresse
- Dupliquez les classes précédentes
    - Ajouter l'attribut personne avec @OneToOne dans la classe Adresse et Getter/Setter :

```
package fr.eni.demo.oneone.bi;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;

@Entity(name = "adresse_oto_bi")
@Table(name = "adresse_oto_bi")
public class Adresse {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;

	private String codePostal;
	private String ville;

	@OneToOne(mappedBy = "adresse")
	private Personne personne;
…
	public Personne getPersonne() {
		return personne;
	}

	public void setPersonne(Personne personne) {
		this.personne = personne;
	}
…
```

- Pour le moment, dans la classe Personne, nous changeons seulement le nom de la table et de l'entité

```
package fr.eni.demo.oneone.bi;

import java.time.LocalDate;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;
import javax.persistence.Transient;

@Entity(name = "personne_oto_bi")
```

```
@Table(name = "personne_oto_bi")
public class Personne {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private long id;

	private String nom;
	private String prenom;
	private LocalDate dateNaissance;

	@Transient
	private int age;

	@OneToOne(cascade = CascadeType.PERSIST)
	private Adresse adresse;
```

- Création du repository

```
package fr.eni.demo.oneone.bi;

import org.springframework.data.repository.CrudRepository;

public interface PersonneOTOBiRepository extends CrudRepository<Personne, Long>{

}
```

- Dans la classe d'exécution de l'application
  - Positionner l'annotation @Profile(«demo») sur le bean précédent et renommer le package com des entités
  - Copier le code du nouveau bean :

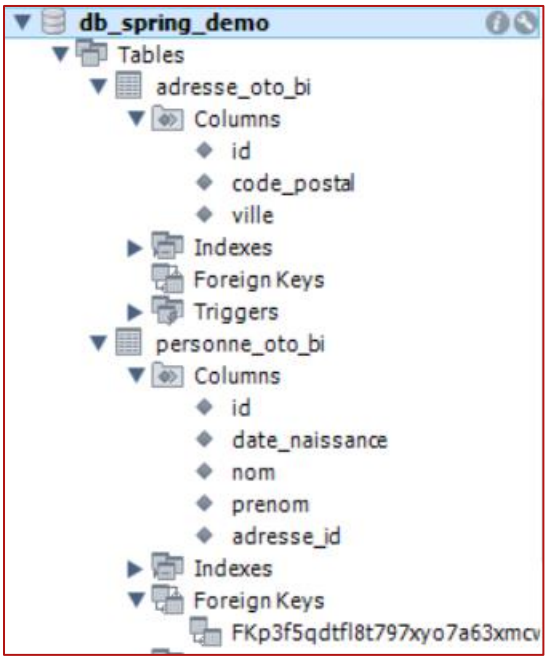
```
@Bean
public CommandLineRunner demoOneToOneBi(PersonneOTOBiRepository repository) {
	return (args) -> {
		fr.eni.demo.oneone.bi.Adresse a1 = new fr.eni.demo.oneone.bi.Adresse("75000", "Paris");
		fr.eni.demo.oneone.bi.Adresse a2 = new fr.eni.demo.oneone.bi.Adresse("35000", "Rennes");

		fr.eni.demo.oneone.bi.Personne p1 = new fr.eni.demo.oneone.bi.Personne("Legrand", "Lucie",
				LocalDate.parse("2008-06-18"), a1);
		fr.eni.demo.oneone.bi.Personne p2 = new fr.eni.demo.oneone.bi.Personne("Legrand", "Lucie2",
				LocalDate.parse("2006-04-03"), a2);

		repository.save(p1);
		repository.save(p2);

		// fetch all
		System.out.println("Liste des personnes : ");
		System.out.println("-------------------------------");
		for (fr.eni.demo.oneone.bi.Personne p : repository.findAll()) {
			System.out.println(p.toString());
		}

		System.out.println("\nTestons la relation bidirectionnelle -- Affichage a2 et de sa
personne");
		System.out.println("-------------------------------");System.out.println(a2);
		System.out.println(a2.getPersonne());

}
```

- Traces :

```
…
Hibernate:

    create table adresse_oto_bi (
       id integer not null auto_increment,
        code_postal varchar(255),
        ville varchar(255),
        primary key (id)
    ) engine=InnoDB
Hibernate:

    create table personne_oto_bi (
       id bigint not null auto_increment,
        date_naissance date,
        nom varchar(255),
        prenom varchar(255),
        adresse_id integer,
        primary key (id)
    ) engine=InnoDB
Hibernate:

    alter table personne_oto_bi
       add constraint FKp3f5qdtfl8t797xyo7a63xmcw
       foreign key (adresse_id)
       references adresse_oto_bi (id)
…
Hibernate:
    insert
    into
        adresse_oto_bi
        (code_postal, ville)
    values
        (?, ?)
Hibernate:
    insert
    into
        personne_oto_bi
        (adresse_id, date_naissance, nom, prenom)
    values
        (?, ?, ?, ?)
…
Liste des personnes :
-------------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.adresse_id as adresse_5_1_,
        personne0_.date_naissance as date_nai2_1_,
        personne0_.nom as nom3_1_,
        personne0_.prenom as prenom4_1_
    from
        personne_oto_bi personne0_
Hibernate:
    select
        adresse0_.id as id1_0_0_,
        adresse0_.code_postal as code_pos2_0_0_,
        adresse0_.ville as ville3_0_0_,
        personne1_.id as id1_1_1_,
        personne1_.adresse_id as adresse_5_1_1_,
        personne1_.date_naissance as date_nai2_1_1_,
        personne1_.nom as nom3_1_1_,
        personne1_.prenom as prenom4_1_1_
    from
        adresse_oto_bi adresse0_
```

```
    left outer join
        personne_oto_bi personne1_
            on adresse0_.id=personne1_.adresse_id
    where
        adresse0_.id=?
Hibernate:
    select
        personne0_.id as id1_1_1_,
        personne0_.adresse_id as adresse_5_1_1_,
        personne0_.date_naissance as date_nai2_1_1_,
        personne0_.nom as nom3_1_1_,
        personne0_.prenom as prenom4_1_1_,
        adresse1_.id as id1_0_0_,
        adresse1_.code_postal as code_pos2_0_0_,
        adresse1_.ville as ville3_0_0_
    from
        personne_oto_bi personne0_
    left outer join
        adresse_oto_bi adresse1_
            on personne0_.adresse_id=adresse1_.id
    where
        personne0_.adresse_id=?
Hibernate:
    select
        adresse0_.id as id1_0_0_,
        adresse0_.code_postal as code_pos2_0_0_,
        adresse0_.ville as ville3_0_0_,
        personne1_.id as id1_1_1_,
        personne1_.adresse_id as adresse_5_1_1_,
        personne1_.date_naissance as date_nai2_1_1_,
        personne1_.nom as nom3_1_1_,
        personne1_.prenom as prenom4_1_1_
    from
        adresse_oto_bi adresse0_
    left outer join
        personne_oto_bi personne1_
            on adresse0_.id=personne1_.adresse_id
    where
        adresse0_.id=?
Hibernate:
    select
        personne0_.id as id1_1_1_,
        personne0_.adresse_id as adresse_5_1_1_,
        personne0_.date_naissance as date_nai2_1_1_,
        personne0_.nom as nom3_1_1_,
        personne0_.prenom as prenom4_1_1_,
        adresse1_.id as id1_0_0_,
        adresse1_.code_postal as code_pos2_0_0_,
        adresse1_.ville as ville3_0_0_
    from
        personne_oto_bi personne0_
    left outer join
        adresse_oto_bi adresse1_
            on personne0_.adresse_id=adresse1_.id
    where
        personne0_.adresse_id=?
Personne [id=1, nom=Legrand, prenom=Lucie, dateNaissance=2008-06-18, age=0, adresse=Adresse [id=1,
codePostal=75000, ville=Paris]]
Personne [id=2, nom=Legrand, prenom=Lucie2, dateNaissance=2006-04-03, age=0, adresse=Adresse [id=2,
codePostal=35000, ville=Rennes]]

Testons la relation bidirectionnelle -- Affichage a2 et de sa personne
-------------------------------
Adresse [id=2, codePostal=35000, ville=Rennes]
null
```

- Création des 2 tables avec jointure.
- Hibernate gère très bien la manipulation des 2
    - Il n'y a pas de différence au niveau des tables entre les 2 versions



- Dans le cas de la remontée des personnes, il y a bien les informations de l'adresse.
- Dans le cas ; où on veut récupérer les informations depuis l'adresse, on constate qu'il manque la personne.
    - Quand en Java, nous devons gérer une association bidirectionnelle. Il faut ajouter le code pour celle-ci :
    - Ajouter dans la classe Personne dans le mutateur d'adresse, la mise à jour du lien bidirectionnelle et appeler cette méthode dans son constructeur avec paramètres.

```
public Personne(String nom, String prenom, LocalDate dateNaissance, Adresse adresse) {
        this.nom = nom;
        this.prenom = prenom;
        this.dateNaissance = dateNaissance;
        //Gestion de la bidirectionnalité
        //this.adresse = adresse;
        setAdresse(adresse);
}
public void setAdresse(Adresse adresse) {
        this.adresse = adresse;
        //Gestion de la bidirectionnalité
        adresse.setPersonne(this);
}
```

- Retester :

```
…
Testons la relation bidirectionnelle -- Affichage a2 et de sa personne
------------------------------
Adresse [id=2, codePostal=35000, ville=Rennes]
Personne [id=2, nom=Legrand, prenom=Lucie2, dateNaissance=2006-04-03, age=0, adresse=Adresse
[id=2, codePostal=35000, ville=Rennes]]
```

    - Cette fois, les informations de la personne sont bien chargées dans a2.