Relation 1-N

Démonstration 6 du module 5

Les objectifs de cette démonstration sont

- Mise en place d'une relation 1-N unidirectionnelle entre une personne et ses adresses
- Mise en avant de la différence entre LAZY et EAGER
- Mise en avant de l'attribut orphanRemoval
- Mise en place d'une relation 1-N bidirectionnelle entre une personne et ses adresses

Contexte

- Continuer dans le projet précédent
- Dans cette démonstration, nous allons manipuler des entités détachées.
  - Nous avons déjà vu qu'il y a des erreurs [org.hibernate.LazyInitializationException]
  - Pour les éviter, ajoutons dans un premier temps dans application.properties :

```
#Pour éviter les soucis avec l'association entre Trainer et Course
spring.jpa.properties.hibernate.enable_lazy_load_no_trans=true
```

Déroulement

1. Relation 1-N unidirectionnelle – En Lazy

- Dupliquer les classes unidirectionnelles
- Renommer le package des bidirectionnelles en com
- Dans la classe Personne
  - Modifier le nom de l'entité et de la table
  - Supprimer les attributs dateNaissance et age pour simplifier la classe
  - Modifier l'attribut adresse en une liste d'adresse et initialiser la liste dans les constructeurs.
  - Ajouter une méthode pour gérer l'ajout d'une adresse
  - Placer l'annotation @OneToMany
    - CascadeType.ALL → pour que l'ORM gère la mise à jour dans tous les cas (save, update, delete,…)
    - FetchType.LAZY → pour charger uniquement l'id de l'adresse associée
    - orphanRemoval=true → si les adresses ne sont plus associées à une personne ;

l'ORM les supprimera.

```java
package fr.eni.demo.otm.uni;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

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

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

	private String nom;
	private String prenom;

	@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, orphanRemoval = true)
	@JoinColumn(name = "personne_id")
	private List<Adresse> adresses;

	public Personne() {
		adresses = new ArrayList<Adresse>();
	}

	public Personne(String nom, String prenom) {
		this();
		this.nom = nom;
		this.prenom = prenom;
	}

	public void addAdresse(Adresse a) {
		adresses.add(a);
	}

	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 List<Adresse> getAdresses() {
		return adresses;
	}

	public void setAdresses(List<Adresse> adresses) {
		this.adresses = adresses;
	}

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

- Dans la classe Adresse
  - Modifier le nom de l'entité et de la table uniquement

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

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

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

- Création des Repository pour gérer les 2 entités

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

import org.springframework.data.repository.CrudRepository;

public interface PersonneOTMURepository extends CrudRepository<Personne, Long>{

}
```

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

import org.springframework.data.repository.CrudRepository;

public interface AdresseOTMURepository extends CrudRepository<Adresse, Integer>{

}
```

- 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 demoOneToManyUni(PersonneOTMURepository persDAO,
		AdresseOTMURepository adresseDAO) {
	return (args) -> {
		// save a few customers
		fr.eni.demo.otm.uni.Adresse a1 = new fr.eni.demo.otm.uni.Adresse("44000", "Nantes");
```

```
		fr.eni.demo.otm.uni.Adresse a2 = new fr.eni.demo.otm.uni.Adresse("33000", "Bordeaux");
		fr.eni.demo.otm.uni.Adresse a3 = new fr.eni.demo.otm.uni.Adresse("29000", "Brest");
		fr.eni.demo.otm.uni.Adresse a4 = new fr.eni.demo.otm.uni.Adresse("74000", "Chamonix");

		fr.eni.demo.otm.uni.Personne albert = new fr.eni.demo.otm.uni.Personne("Dupontel",
"Albert");
		fr.eni.demo.otm.uni.Personne sophie = new fr.eni.demo.otm.uni.Personne("Marceau",
"Sophie");

			albert.addAdresse(a1);
			albert.addAdresse(a2);
			sophie.addAdresse(a3);
			sophie.addAdresse(a4);

			persDAO.save(albert);
			persDAO.save(sophie);

			System.out.println("Liste des personnes : ");
			System.out.println("-------------------------------");
			for (fr.eni.demo.otm.uni.Personne personne : persDAO.findAll()) {
				System.out.println(personne.toString());
			}
	};
}
```

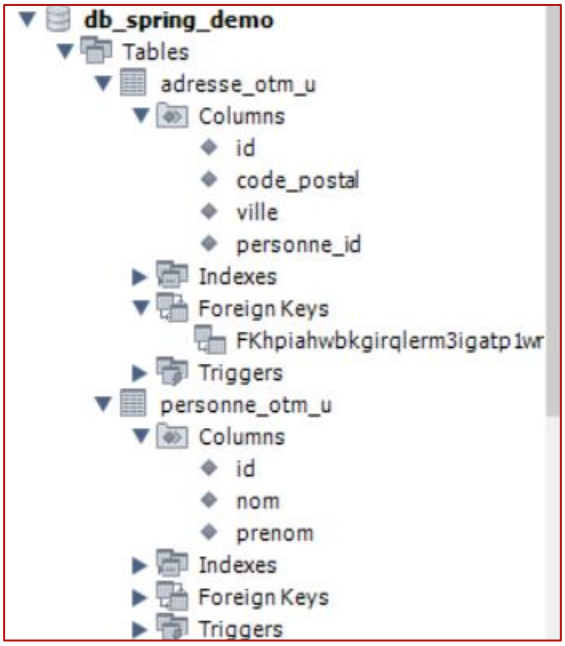
- Traces d'exécution :

```
…
Hibernate:

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

    create table personne_otm_u (
       id bigint not null auto_increment,
        nom varchar(255),
        prenom varchar(255),
        primary key (id)
    ) engine=InnoDB
Hibernate:

    alter table adresse_otm_u
       add constraint FKhpiahwbkgirqlerm3igatp1wn
       foreign key (personne_id)
       references personne_otm_u (id)
…
Hibernate:
    insert
    into
        personne_otm_u
        (nom, prenom)
    values
        (?, ?)
Hibernate:
    insert
    into
        adresse_otm_u
        (code_postal, ville)
    values
        (?, ?)
```

```
Hibernate:
    insert
    into
        adresse_otm_u
        (code_postal, ville)
    values
        (?, ?)
Hibernate:
    update
        adresse_otm_u
    set
        personne_id=?
    where
        id=?
Hibernate:
    update
        adresse_otm_u
    set
        personne_id=?
    where
        id=?
Hibernate:
    insert
    into
        personne_otm_u
        (nom, prenom)
    values
        (?, ?)
Hibernate:
    insert
    into
        adresse_otm_u
        (code_postal, ville)
    values
        (?, ?)
Hibernate:
    insert
    into
        adresse_otm_u
        (code_postal, ville)
    values
        (?, ?)
Hibernate:
    update
        adresse_otm_u
    set
        personne_id=?
    where
        id=?
Hibernate:
    update
        adresse_otm_u
    set
        personne_id=?
    where
        id=?
Liste des personnes :
-------------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.nom as nom2_1_,
        personne0_.prenom as prenom3_1_
    from
        personne_otm_u personne0_
Hibernate:
```

```
    select
        adresses0_.personne_id as personne4_0_0_,
        adresses0_.id as id1_0_0_,
        adresses0_.id as id1_0_1_,
        adresses0_.code_postal as code_pos2_0_1_,
        adresses0_.ville as ville3_0_1_
    from
        adresse_otm_u adresses0_
    where
        adresses0_.personne_id=?
Personne [id=1, nom=Dupontel, prenom=Albert, adresses=[Adresse [id=1, codePostal=44000, ville=Nantes],
Adresse [id=2, codePostal=33000, ville=Bordeaux]]]
Hibernate:
    select
        adresses0_.personne_id as personne4_0_0_,
        adresses0_.id as id1_0_0_,
        adresses0_.id as id1_0_1_,
        adresses0_.code_postal as code_pos2_0_1_,
        adresses0_.ville as ville3_0_1_
    from
        adresse_otm_u adresses0_
    where
        adresses0_.personne_id=?
Personne [id=2, nom=Marceau, prenom=Sophie, adresses=[Adresse [id=3, codePostal=29000, ville=Brest],
Adresse [id=4, codePostal=74000, ville=Chamonix]]]
```

- Création 2 tables avec une clef de jointure
    - En base, on retrouve bien cela :



- Par contre, il faut aussi constater qu'avant d'afficher de chaque personne et son adresse, Hibernate a dû refaire une requête pour récupérer chaque adresse.
    - LAZY ➔ il n'y avait par défaut que l'id de l'adresse et pas tous ses attributs
    - C'est aussi pour cela, qu'il nous fallait mettre l'option « enable_lazy_load_no_trans » dans notre fichier application.properties.
    - A ce moment-là, les adresses sont détachés de leur personne.

2. Relation 1-N unidirectionnelle – En Eager

- Modifier la classe Personne ; placer l'attribut fetch de l'association en EAGER

```
@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, orphanRemoval = true)
@JoinColumn(name = "personne_id")
private List<Adresse> adresses;
```

- Mettre en commentaires dans application.properties, l'option « enable_lazy_load_no_trans »
- Exécuter, Traces attendues :

```
Hibernate:

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

    create table personne_otm_u (
       id bigint not null auto_increment,
        nom varchar(255),
        prenom varchar(255),
        primary key (id)
    ) engine=InnoDB
Hibernate:

    alter table adresse_otm_u
       add constraint FKhpiahwbkgirqlerm3igatp1wn
       foreign key (personne_id)
       references personne_otm_u (id)
2022-03-25 08:52:53.491  INFO 19780 --- [  restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator       :
HHH000490: Using JtaPlatform implementation:
[org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2022-03-25 08:52:53.499  INFO 19780 --- [  restartedMain] j.LocalContainerEntityManagerFactoryBean :
Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-03-25 08:52:53.853  INFO 19780 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       :
LiveReload server is running on port 35729
2022-03-25 08:52:53.871  INFO 19780 --- [  restartedMain] fr.eni.demo.DemoJpaApplication           :
Started DemoJpaApplication in 3.157 seconds (JVM running for 3.677)
Hibernate:
    insert
    into
        personne_otm_u
        (nom, prenom)
    values
        (?, ?)
Hibernate:
    insert
    into
        adresse_otm_u
        (code_postal, ville)
    values
        (?, ?)
Hibernate:
    insert
    into
        adresse_otm_u
        (code_postal, ville)
    values
        (?, ?)
Hibernate:
```

```
    update
        adresse_otm_u
    set
        personne_id=?
    where
        id=?
Hibernate:
    update
        adresse_otm_u
    set
        personne_id=?
    where
        id=?
Hibernate:
    insert
    into
        personne_otm_u
        (nom, prenom)
    values
        (?, ?)
Hibernate:
    insert
    into
        adresse_otm_u
        (code_postal, ville)
    values
        (?, ?)
Hibernate:
    insert
    into
        adresse_otm_u
        (code_postal, ville)
    values
        (?, ?)
Hibernate:
    update
        adresse_otm_u
    set
        personne_id=?
    where
        id=?
Hibernate:
    update
        adresse_otm_u
    set
        personne_id=?
    where
        id=?
Liste des personnes :
------------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.nom as nom2_1_,
        personne0_.prenom as prenom3_1_
    from
        personne_otm_u personne0_
Hibernate:
    select
        adresses0_.personne_id as personne4_0_0_,
        adresses0_.id as id1_0_0_,
        adresses0_.id as id1_0_1_,
        adresses0_.code_postal as code_pos2_0_1_,
        adresses0_.ville as ville3_0_1_
    from
        adresse_otm_u adresses0_
```

```
    where
        adresses0_.personne_id=?
Hibernate:
    select
        adresses0_.personne_id as personne4_0_0_,
        adresses0_.id as id1_0_0_,
        adresses0_.id as id1_0_1_,
        adresses0_.code_postal as code_pos2_0_1_,
        adresses0_.ville as ville3_0_1_
    from
        adresse_otm_u adresses0_
    where
        adresses0_.personne_id=?
Personne [id=1, nom=Dupontel, prenom=Albert, adresses=[Adresse [id=1, codePostal=44000, ville=Nantes],
Adresse [id=2, codePostal=33000, ville=Bordeaux]]]
Personne [id=2, nom=Marceau, prenom=Sophie, adresses=[Adresse [id=3, codePostal=29000, ville=Brest],
Adresse [id=4, codePostal=74000, ville=Chamonix]]]
```

- La récupération des éléments ; se fait avant le parcours de la liste
    - Voilà la différence entre Eager et Lazy
    - Attention, que vos relations n'aient pas elles-mêmes des relations Eager. Sinon vous finirez pas remonter toute la base de données
- On constate que la requête pourrait être optimisée avec une jointure. Nous y reviendrons plus tard.

3. Relation 1-N unidirectionnelle – orphanRemoval

- Mettons en avant l'attribut orphanRemoval.
- Ajouter dans la méthode `demoOneToManyUni` du bean :

```
//Tester l'attribut : orphan
albert.setAdresses(new ArrayList< fr.eni.demo.otm.uni.Adresse>());
persDAO.save(albert);

System.out.println("Détachement des adresses de albert : ");
        System.out.println("-------------------------------");
        for (fr.eni.demo.otm.uni.Adresse adr : adresseDAO.findAll()) {
                System.out.println(adr.toString());
}
```

- Exécuter, Traces attendues :

```
Hibernate:
    update
        adresse_otm_u
    set
        personne_id=null
    where
        personne_id=?
Hibernate:
    delete
    from
        adresse_otm_u
    where
        id=?
Hibernate:
    delete
    from
        adresse_otm_u
    where
        id=?
```

```
Détachement des adresses de albert :
------------------------------
Hibernate:
    select
        adresse0_.id as id1_0_,
        adresse0_.code_postal as code_pos2_0_,
        adresse0_.ville as ville3_0_
    from
        adresse_otm_u adresse0_
Adresse [id=3, codePostal=29000, ville=Brest]
Adresse [id=4, codePostal=74000, ville=Chamonix]
```

- Le fait de remplacer la liste des adresses d'albert par une nouvelle liste
  - à détacher l'ensemble des adresses précédentes d'albert : a1 et a2
  - Hibernate à mis à jour en la clef de jointure entre Personne et Adresse en mettant à null pour chaque adresse la clef de jointure : personne_id=null
- L'attribut orphanRemoval=true a forcé Hibernate a supprimé les 2 adresses en plus.
  - Et maintenant en base, il y a uniquement les adresses de sophie.

Aller plus loin

- Mettre en avant que si l'attribut orphanRemoval est à false
- Les 2 adresses restent en base de données
- Modifier l'attribut orphanRemoval

```
@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, orphanRemoval = false)
@JoinColumn(name = "personne_id")
private List<Adresse> adresses;
```

- Exécuter, Traces attendues :

```
Hibernate:
    update
        adresse_otm_u
    set
        personne_id=null
    where
        personne_id=?
Détachement des adresses de albert :
------------------------------
Hibernate:
    select
        adresse0_.id as id1_0_,
        adresse0_.code_postal as code_pos2_0_,
        adresse0_.ville as ville3_0_
    from
        adresse_otm_u adresse0_
Adresse [id=1, codePostal=44000, ville=Nantes]
Adresse [id=2, codePostal=33000, ville=Bordeaux]
Adresse [id=3, codePostal=29000, ville=Brest]
Adresse [id=4, codePostal=74000, ville=Chamonix]
```

- Le fait de remplacer la liste des adresses d'albert par une nouvelle liste ;
  - à détacher l'ensemble des adresses précédentes d'albert : a1 et a2 comme précédemment
  - Hibernate à mis à jour en la clef de jointure entre Personne et Adresse en mettant à null pour chaque adresse la clef de jointure : personne_id=null.

- Par contre, il n'a pas supprimé les 2 adresses.
    - Il y a toujours 4 adresses en base.
    - Ainsi, nous avons bien mis en avant l'intérêt de l'attribut orphanRemoval sur une association 1-N
    - Remettre l'attribut à true.

4. Relation 1-N 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. Cet attribut se place sur le côté N
    - Il faut aussi ajouter le comportement pour gérer la bidirectionnalité en Java (comme pour le cas 1-1)
- Dupliquez les classes précédentes
- Dans la classe Adresse :
    - Changer le nom de la table et de l'entité
    - Ajouter le nouvel attribut dans la classe Adresse et son annotation (Getter/Setter)

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

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

	private String codePostal;
	private String ville;

	@ManyToOne(cascade = { CascadeType.PERSIST, CascadeType.MERGE })
	private Personne personne;
…

	public Personne getPersonne() {
		return personne;
	}

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

- Dans la classe Personne
    - Changer le nom de la table et de l'entité
    - On doit ajouter l'attribut mappedBy
    - et retirer l'annotation @JoinColumn car c'est dans la classe Adresse que le traitement de la colonne sera gérée par Hibernate
    - Et ajouter le code pour gérer la bidirectionnalité

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

import java.util.ArrayList;
import java.util.List;

import javax.persistence.*;

@Entity(name = "personne_otm_bi")
@Tablr(name = "personne_otm_bi")
public class Personne {

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

	private String nom;
	private String prenom;

	@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, orphanRemoval = true, mappedBy =
"personne")
	private List<Adresse> adresses;

	public Personne() {
		adresses = new ArrayList<Adresse>();
	}

	public Personne(String nom, String prenom) {
		this();
		this.nom = nom;
		this.prenom = prenom;
	}

	public void addAdresse(Adresse a) {
		adresses.add(a);
		//Gestion bidirectionnelle
		a.setPersonne(this);
	}

	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 List<Adresse> getAdresses() {
		return adresses;
	}
```

```
	public void setAdresses(List<Adresse> adresses) {
		System.out.println("setAdresses");
		this.adresses = adresses;
		//Gestion bidirectionnelle
		for (Adresse a : this.adresses) {
			a.setPersonne(this);
		}
	}

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

}
```

- Création des repository

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

import org.springframework.data.repository.CrudRepository;

public interface PersonneOTMBiRepository extends CrudRepository<Personne, Long>{

}
```

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

import org.springframework.data.repository.CrudRepository;

public interface AdresseOTMBiRepository extends CrudRepository<Adresse, Integer>{

}
```

- 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 demoOneToManyBi(PersonneOTMBiRepository persDAO, AdresseOTMBiRepository
adresseDAO) {
	return (args) -> {
		// save a few customers
		fr.eni.demo.otm.bi.Adresse a1 = new fr.eni.demo.otm.bi.Adresse("44000", "Nantes");
		fr.eni.demo.otm.bi.Adresse a2 = new fr.eni.demo.otm.bi.Adresse("33000", "Bordeaux");
		fr.eni.demo.otm.bi.Adresse a3 = new fr.eni.demo.otm.bi.Adresse("29000", "Brest");
		fr.eni.demo.otm.bi.Adresse a4 = new fr.eni.demo.otm.bi.Adresse("74000", "Chamonix");

		fr.eni.demo.otm.bi.Personne albert = new fr.eni.demo.otm.bi.Personne("Dupontel", "Albert");
		fr.eni.demo.otm.bi.Personne sophie = new fr.eni.demo.otm.bi.Personne("Marceau", "Sophie");

			albert.addAdresse(a1);
			albert.addAdresse(a2);
			sophie.addAdresse(a3);
			sophie.addAdresse(a4);

			persDAO.save(albert);
			persDAO.save(sophie);

			System.out.println("Liste des personnes : ");
```

```
                System.out.println("------------------------------");
                for (fr.eni.demo.otm.bi.Personne personne : persDAO.findAll()) {
                        System.out.println(personne.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());

                // Tester l'attribut : orphan
                albert.setAdresses(new ArrayList<fr.eni.demo.otm.bi.Adresse>());
                persDAO.save(albert);

                System.out.println("Détachement des adresses de albert : ");
                System.out.println("------------------------------");
                for (fr.eni.demo.otm.bi.Adresse adr : adresseDAO.findAll()) {
                        System.out.println(adr.toString());
                }
            };
        }
```

- Traces attendues :

```
…
Hibernate:

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

    create table personne_otm_bi (
       id bigint not null auto_increment,
        nom varchar(255),
        prenom varchar(255),
        primary key (id)
    ) engine=InnoDB
Hibernate:

    alter table adresse_otm_bi
       add constraint FKfmnsba4cn67qyd427eqq5kii0
       foreign key (personne_id)
       references personne_otm_bi (id)
…
Liste des personnes :
------------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.nom as nom2_1_,
        personne0_.prenom as prenom3_1_
    from
        personne_otm_bi personne0_
Hibernate:
    select
        adresses0_.personne_id as personne4_0_0_,
        adresses0_.id as id1_0_0_,
        adresses0_.id as id1_0_1_,
        adresses0_.code_postal as code_pos2_0_1_,
        adresses0_.personne_id as personne4_0_1_,
```

```
        adresses0_.ville as ville3_0_1_
    from
        adresse_otm_bi adresses0_
    where
        adresses0_.personne_id=?
Hibernate:
    select
        adresses0_.personne_id as personne4_0_0_,
        adresses0_.id as id1_0_0_,
        adresses0_.id as id1_0_1_,
        adresses0_.code_postal as code_pos2_0_1_,
        adresses0_.personne_id as personne4_0_1_,
        adresses0_.ville as ville3_0_1_
    from
        adresse_otm_bi adresses0_
    where
        adresses0_.personne_id=?
Personne [id=1, nom=Dupontel, prenom=Albert, adresses=[Adresse [id=1, codePostal=44000, ville=Nantes],
Adresse [id=2, codePostal=33000, ville=Bordeaux]]]
Personne [id=2, nom=Marceau, prenom=Sophie, adresses=[Adresse [id=3, codePostal=29000, ville=Brest],
Adresse [id=4, codePostal=74000, ville=Chamonix]]]

Testons la relation bidirectionnelle -- Affichage a2 et de sa personne
------------------------------
Adresse [id=2, codePostal=33000, ville=Bordeaux]
Personne [id=1, nom=Dupontel, prenom=Albert, adresses=[Adresse [id=1, codePostal=44000, ville=Nantes],
Adresse [id=2, codePostal=33000, ville=Bordeaux]]]
setAdresses
Hibernate:
    select
        personne0_.id as id1_1_1_,
        personne0_.nom as nom2_1_1_,
        personne0_.prenom as prenom3_1_1_,
        adresses1_.personne_id as personne4_0_3_,
        adresses1_.id as id1_0_3_,
        adresses1_.id as id1_0_0_,
        adresses1_.code_postal as code_pos2_0_0_,
        adresses1_.personne_id as personne4_0_0_,
        adresses1_.ville as ville3_0_0_
    from
        personne_otm_bi personne0_
    left outer join
        adresse_otm_bi adresses1_
            on personne0_.id=adresses1_.personne_id
    where
        personne0_.id=?
Hibernate:
    delete
    from
        adresse_otm_bi
    where
        id=?
Hibernate:
    delete
    from
        adresse_otm_bi
    where
        id=?
Détachement des adresses de albert :
------------------------------
Hibernate:
    select
        adresse0_.id as id1_0_,
        adresse0_.code_postal as code_pos2_0_,
        adresse0_.personne_id as personne4_0_,
        adresse0_.ville as ville3_0_
```

```
    from
        adresse_otm_bi adresse0_
Hibernate:
    select
        personne0_.id as id1_1_0_,
        personne0_.nom as nom2_1_0_,
        personne0_.prenom as prenom3_1_0_,
        adresses1_.personne_id as personne4_0_1_,
        adresses1_.id as id1_0_1_,
        adresses1_.id as id1_0_2_,
        adresses1_.code_postal as code_pos2_0_2_,
        adresses1_.personne_id as personne4_0_2_,
        adresses1_.ville as ville3_0_2_
    from
        personne_otm_bi personne0_
    left outer join
        adresse_otm_bi adresses1_
            on personne0_.id=adresses1_.personne_id
    where
        personne0_.id=?
Adresse [id=3, codePostal=29000, ville=Brest]
Adresse [id=4, codePostal=74000, ville=Chamonix]

Hibernate:
    insert
    into
        personne_otm_bi
        (nom, prenom)
    values
        (?, ?)
Hibernate:
    insert
    into
        adresse_otm_bi
        (code_postal, personne_id, ville)
    values
        (?, ?, ?)
…
```

- Création des 2 tables avec jointure.
- Gestion du orphanRemoval
- Et grâce au code nous avons l'association qui fonctionne bien dans les 2 sens.
- Hibernate gère très bien la manipulation des 2
    - Il n'y a pas de différence au niveau des tables entre les 2 versions