Relation N-1

Démonstration 7 du module 5

Les objectifs de cette démonstration sont

- Mise en place d'une relation N-1 unidirectionnelle entre des personnes et leur civilité
- Montrer qu'il n'y a pas de suppression de civilité quand il y a suppression des personnes associées.

Contexte

- Continuer dans le projet précédent

Déroulement

Relation N-1

- Créer un package fr.eni.demo.mto
- Renommer le package des précédents en com

Classe Civilite :

- Elle a un identifiant de type String qui sera géré par le code.
- Pas d'annotation @GeneratedValue

```
package fr.eni.demo.mto;

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

@Entity(name = "civilite_mto")
@Table(name = "civilite_mto")
public class Civilite {

	@Id
	private String cle;

	private String libelle;

	public Civilite() {
	}

	public Civilite(String cle, String libelle) {
		this.cle = cle;
		this.libelle = libelle;
	}
```

```
	public String getCle() {
		return cle;
	}

	public void setCle(String cle) {
		this.cle = cle;
	}

	public String getLibelle() {
		return libelle;
	}

	public void setLibelle(String libelle) {
		this.libelle = libelle;
	}

	@Override
	public String toString() {
		return "Civilite [cle=" + cle + ", libelle=" + libelle + "]";
	}

}
```

Classe Personne :

- Nous retirons pour le moment l'association avec les adresses.
- Nous nous concentrons sur l'association ManyToOne entre Personne et Civilite

```
package fr.eni.demo.mto;

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

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

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

	private String nom;
	private String prenom;

	@ManyToOne
	private Civilite civilite;

	public Personne() {
	}

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

	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 prenom) {
		this.prenom = prenom;
	}

	public Civilite getCivilite() {
		return civilite;
	}

	public void setCivilite(Civilite civilite) {
		this.civilite = civilite;
	}

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

2 Repository

```
package fr.eni.demo.mto;

import org.springframework.data.repository.CrudRepository;

public interface CiviliteMTORepository extends CrudRepository<Civilite, String>{

}
```

```
package fr.eni.demo.mto;

import org.springframework.data.repository.CrudRepository;

public interface PersonneMTORepository extends CrudRepository<Personne, Long>{

}
```

Exécution

- 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 demoManyToOne(PersonneMTORepository persDAO,
		CiviliteMTORepository civiliteDAO) {
	return (args) -> {
		fr.eni.demo.mto.Civilite c1 = new fr.eni.demo.mto.Civilite("M", "Monsieur");
```

```
		fr.eni.demo.mto.Civilite c2 = new fr.eni.demo.mto.Civilite("Mme", "Madame");
		civiliteDAO.save(c1);
		civiliteDAO.save(c2);

		fr.eni.demo.mto.Personne albert = new fr.eni.demo.mto.Personne("Dupontel", "Albert");
		fr.eni.demo.mto.Personne jack = new fr.eni.demo.mto.Personne("Lemmon", "Jack");
		fr.eni.demo.mto.Personne sophie = new fr.eni.demo.mto.Personne("Marceau", "Sophie");

			albert.setCivilite(c1);
			sophie.setCivilite(c2);
			jack.setCivilite(c1);

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

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

			persDAO.delete(sophie);
			System.out.println("Liste des personnes après suppression de sophie : ");
			System.out.println("-------------------------------");
			for (fr.eni.demo.mto.Personne personne : persDAO.findAll()) {
				System.out.println(personne.toString());
			}

			System.out.println("Liste des civilités : ");
			System.out.println("-------------------------------");
			for (fr.eni.demo.mto.Civilite c : civiliteDAO.findAll()) {
				System.out.println(c.toString());
			}
		};
	}
```

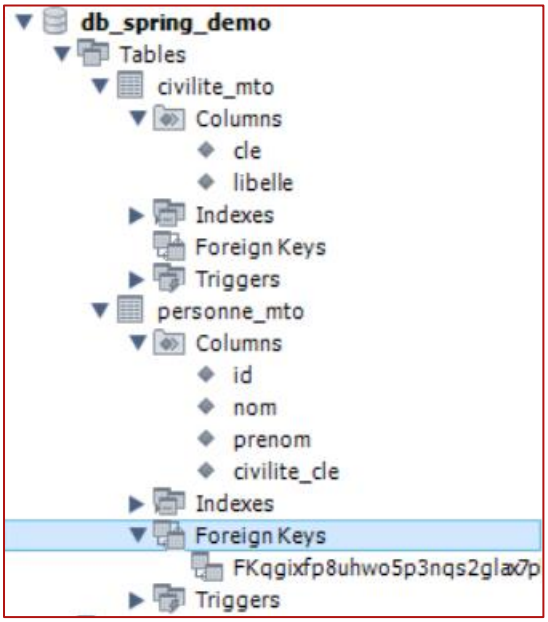
- Traces d'exécution :

```
…
Hibernate:

    create table civilite_mto (
       cle varchar(255) not null,
        libelle varchar(255),
        primary key (cle)
    ) engine=InnoDB
Hibernate:

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

    alter table personne_mto
       add constraint FKqgixfp8uhwo5p3nqs2glax7pb
       foreign key (civilite_cle)
       references civilite_mto (cle)
…
Hibernate:
    select
        civilite0_.cle as cle1_0_0_,
```

```
        civilite0_.libelle as libelle2_0_0_
    from
        civilite_mto civilite0_
    where
        civilite0_.cle=?
Hibernate:
    insert
    into
        civilite_mto
        (libelle, cle)
    values
        (?, ?)
…
Hibernate:
    insert
    into
        personne_mto
        (civilite_cle, nom, prenom)
    values
        (?, ?, ?)
Hibernate:
    select
        civilite_.cle,
        civilite_.libelle as libelle2_0_
    from
        civilite_mto civilite_
    where
        civilite_.cle=?
…
Liste des personnes :
-----------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.civilite_cle as civilite4_1_,
        personne0_.nom as nom2_1_,
        personne0_.prenom as prenom3_1_
    from
        personne_mto personne0_
Hibernate:
    select
        civilite0_.cle as cle1_0_0_,
        civilite0_.libelle as libelle2_0_0_
    from
        civilite_mto civilite0_
    where
        civilite0_.cle=?
Hibernate:
    select
        civilite0_.cle as cle1_0_0_,
        civilite0_.libelle as libelle2_0_0_
    from
        civilite_mto civilite0_
    where
        civilite0_.cle=?
Personne [id=1, nom=Dupontel, prenom=Albert, civilite=Civilite [cle=M, libelle=Monsieur]]
Personne [id=2, nom=Marceau, prenom=Sophie, civilite=Civilite [cle=Mme, libelle=Madame]]
Personne [id=3, nom=Lemmon, prenom=Jack, civilite=Civilite [cle=M, libelle=Monsieur]]

…
Hibernate:
    delete
    from
        personne_mto
    where
        id=?
```

```
Liste des personnes après suppression de sophie :
------------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.civilite_cle as civilite4_1_,
        personne0_.nom as nom2_1_,
        personne0_.prenom as prenom3_1_
    from
        personne_mto personne0_
Hibernate:
    select
        civilite0_.cle as cle1_0_0_,
        civilite0_.libelle as libelle2_0_0_
    from
        civilite_mto civilite0_
    where
        civilite0_.cle=?
Personne [id=1, nom=Dupontel, prenom=Albert, civilite=Civilite [cle=M, libelle=Monsieur]]
Personne [id=3, nom=Lemmon, prenom=Jack, civilite=Civilite [cle=M, libelle=Monsieur]]
Liste des civilités :
------------------------------
Hibernate:
    select
        civilite0_.cle as cle1_0_,
        civilite0_.libelle as libelle2_0_
    from
        civilite_mto civilite0_
Civilite [cle=M, libelle=Monsieur]
Civilite [cle=Mme, libelle=Madame]
```

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



- Hibernate associe bien à chaque personne sa civilité
- Et lors de la suppression de sophie, la civilité Madame est toujours présente en base.
- Tout fonctionne parfaitement