Plus loin avec le Repository

Démonstration 11 du module 5

Les objectifs de cette démonstration sont

- Création de requête par mot clef dans Repository
- Manipulation avec JPQL
- Utilisation des requêtes nommées et natives

Contexte

- Continuer dans le projet précédent
- Reprenons les personnes et leur civilité
- Renommer le package des entités précédentes en com.
- Mettre l'annotation @Profil(«Demo») sur le bean dans la classe d'exécution
- Dupliquer les classes de la démonstration ManyToOne

Déroulement

1. Query Methods

- Dans la classe Personne :
    - Modifier le nom de l'entité et de la table
    - Ajouter un attribut email

```
package fr.eni.demo.repository;

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 = "EntitePersonne")
@Table(name = "personne_repo")
public class Personne {

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

	private String nom;
	private String prenom;
	private String email;
```

```
	@ManyToOne
	private Civilite civilite;

	public Personne() {
	}

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

	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;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

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

}
```

- Dans la classe Civilite :
    - Modifier le nom de l'entité et de la table

```
@Entity(name = "EntiteCivilite")
@Table(name = "civilite_repo")
public class Civilite {
```

- Créer un Repository pour chaque entité
    - Ajouter ces méthodes à celui de Personne :

```
package fr.eni.demo.repository;

import java.util.List;

import org.springframework.data.repository.CrudRepository;

public interface PersonneRepository extends CrudRepository<Personne, Integer> {
	//Déclarer des filtre complexes
	//Déclarer des méthodes avec les mots de JPQL
	List<Personne> findByEmailAndNom(String email, String nom);

	List<Personne> findByNomAndPrenom(String nom, String prenom);

	List<Personne> findDistinctByNomOrPrenom(String nom, String prenom);
}
```

- Dans la classe d'exécution de l'application
    - Copier le code du nouveau bean :

```
@Bean
public CommandLineRunner demoRepository(PersonneRepository persRepository, CiviliteRepository civRepository) {
	return (args) -> {

	fr.eni.demo.repository.Civilite monsieur = new fr.eni.demo.repository.Civilite("M", "Monsieur");
	fr.eni.demo.repository.Civilite madame = new fr.eni.demo.repository.Civilite("Mme", "Madame");
	civRepository.save(monsieur);
	civRepository.save(madame);

	fr.eni.demo.repository.Personne albert = new fr.eni.demo.repository.Personne("Dupontel", "Albert",
				"adupontel@eni.fr", monsieur);
	fr.eni.demo.repository.Personne jack = new fr.eni.demo.repository.Personne("Lemmon", "Jack",
				"jlemmon@eni.fr", monsieur);
	fr.eni.demo.repository.Personne sophie = new fr.eni.demo.repository.Personne("Marceau", "Sophie",
				"smarceau@eni.fr", madame);
	fr.eni.demo.repository.Personne chris = new fr.eni.demo.repository.Personne("Lemmon", "Chris",
				"clemmon@eni.fr", monsieur);

	persRepository.save(albert);
	persRepository.save(jack);
	persRepository.save(sophie);
	persRepository.save(chris);

	System.out.println("Liste des personnes");
	System.out.println("-------------------------");
	for (fr.eni.demo.repository.Personne p : persRepository.findAll()) {
		System.out.println(p);
	}

	System.out.println("\nfindByEmailAndNom : email : jlemmon@eni.fr -  nom : Lemmon" );
	System.out.println("-------------------------");
	for (fr.eni.demo.repository.Personne p : persRepository.findByEmailAndNom("jlemmon@eni.fr","Lemmon")) {
			System.out.println(p);
		}
```

```
        System.out.println("\nfindByNomAndPrenom : Dupontel Albert");
        System.out.println("-------------------------");
        for (fr.eni.demo.repository.Personne p : persRepository.findByNomAndPrenom("Dupontel","Albert")) {
                        System.out.println(p);
        }

        System.out.println("\nfindDistinctByNomOrPrenom : Lemmon Jack");
        System.out.println("-------------------------");
        for (fr.eni.demo.repository.Personne p : persRepository.findDistinctByNomOrPrenom("Lemmon","Jack")) {
                System.out.println(p);
        }
        };
}
```

- Exécution, Traces attendues :

```
...

Liste des personnes
------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.civilite_cle as civilite5_1_,
        personne0_.email as email2_1_,
        personne0_.nom as nom3_1_,
        personne0_.prenom as prenom4_1_
    from
        personne_repo personne0_
Hibernate:
    select
        civilite0_.cle as cle1_0_0_,
        civilite0_.libelle as libelle2_0_0_
    from
        civilite_repo civilite0_
    where
        civilite0_.cle=?
Hibernate:
    select
        civilite0_.cle as cle1_0_0_,
        civilite0_.libelle as libelle2_0_0_
    from
        civilite_repo civilite0_
    where
        civilite0_.cle=?
Personne [id=1, nom=Dupontel, prenom=Albert, email=adupontel@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]
Personne [id=2, nom=Lemmon, prenom=Jack, email=jlemmon@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]
Personne [id=3, nom=Marceau, prenom=Sophie, email=smarceau@eni.fr, civilite=Civilite [cle=Mme,
libelle=Madame]]
Personne [id=4, nom=Lemmon, prenom=Chris, email=clemmon@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]

findByEmailAndNom : email : jlemmon@eni.fr -  nom : Lemmon
------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.civilite_cle as civilite5_1_,
        personne0_.email as email2_1_,
        personne0_.nom as nom3_1_,
        personne0_.prenom as prenom4_1_
    from
```

```
        personne_repo personne0_
    where
        personne0_.email=?
        and personne0_.nom=?
Hibernate:
    select
        civilite0_.cle as cle1_0_0_,
        civilite0_.libelle as libelle2_0_0_
    from
        civilite_repo civilite0_
    where
        civilite0_.cle=?
Personne [id=2, nom=Lemmon, prenom=Jack, email=jlemmon@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]

findByNomAndPrenom : Dupontel Albert
------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.civilite_cle as civilite5_1_,
        personne0_.email as email2_1_,
        personne0_.nom as nom3_1_,
        personne0_.prenom as prenom4_1_
    from
        personne_repo personne0_
    where
        personne0_.nom=?
        and personne0_.prenom=?
Hibernate:
    select
        civilite0_.cle as cle1_0_0_,
        civilite0_.libelle as libelle2_0_0_
    from
        civilite_repo civilite0_
    where
        civilite0_.cle=?
Personne [id=1, nom=Dupontel, prenom=Albert, email=adupontel@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]

findDistinctByNomOrPrenom : Lemmon Jack
------------------------
Hibernate:
    select
        distinct personne0_.id as id1_1_,
        personne0_.civilite_cle as civilite5_1_,
        personne0_.email as email2_1_,
        personne0_.nom as nom3_1_,
        personne0_.prenom as prenom4_1_
    from
        personne_repo personne0_
    where
        personne0_.nom=?
        or personne0_.prenom=?
Hibernate:
    select
        civilite0_.cle as cle1_0_0_,
        civilite0_.libelle as libelle2_0_0_
    from
        civilite_repo civilite0_
    where
        civilite0_.cle=?
Personne [id=2, nom=Lemmon, prenom=Jack, email=jlemmon@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]
Personne [id=4, nom=Lemmon, prenom=Chris, email=clemmon@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]
```

```
…
```

- Au niveau base de données, il n'y a rien d'ajouter au niveau structure.
- Spring transforme le nom de la méthode en requête SQL

2. Repository, JPQL et requête native

- Bien faire attention, que nous travaillons sur les entités et non sur les tables :
  - C'est pour cela, que nous avons posé des noms différents entre les entités et les tables pour une fois.
- Dans le Repository de Personne
  - Ajouter la requête en JPQL suivante :

```
//Requete en JPQL - travaille sur l'entité
@Query("select p from EntitePersonne p where p.prenom = ?1")
List<Personne> trouverPersonnesParPrenom(String prenom);
```

    - L'entité de la classe a été appelée : EntitePersonne donc dans la requête JPQL, il faut indiquer ce nom
    - ?1 représente le premier paramètre (chaque ? est indexé par le nombre de paramètres et dans l'ordre de la requête)
- Il est possible de faire des requêtes en SQL (gestion de cas complexes).
  - Cette solution s'utilise pour des requêtes complexes sur de l'existant principalement
  - Il faut privilégier le JPQL pour l'ORM
  - Ajouter la requête en SQL suivante :

```
//Requête en SQL - travaille sur les tables
@Query(value="select * from personne_repo p where p.prenom = ?1", nativeQuery=true)
List<Personne> trouverPersonnesParPrenomSQL(String nom);
```

    - La table a été appelée : personne_repo donc dans la requête SQL, il faut indiquer ce nom

- Dans la classe d'exécution de l'application
  - Compléter le bean précédent :

```
System.out.println("\nJPQL trouverPersonnesParPrenom : jack");
System.out.println("-------------------------");
for (fr.eni.demo.repository.Personne p : persRepository.trouverPersonnesParPrenom("Jack")) {
	System.out.println(p);
}

System.out.println("\nSQL trouverPersonnesParPrenomSQL : chris");
System.out.println("-------------------------");
for (fr.eni.demo.repository.Personne p : persRepository.trouverPersonnesParPrenomSQL("Chris")) {
	System.out.println(p);
}
```

- Exécution, Traces attendues :

```
JPQL trouverPersonnesParPrenom : jack
-------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.civilite_cle as civilite5_1_,
```

```
        personne0_.email as email2_1_,
        personne0_.nom as nom3_1_,
        personne0_.prenom as prenom4_1_
    from
        personne_repo personne0_
    where
        personne0_.prenom=?
Hibernate:
    select
        civilite0_.cle as cle1_0_0_,
        civilite0_.libelle as libelle2_0_0_
    from
        civilite_repo civilite0_
    where
        civilite0_.cle=?
Personne [id=2, nom=Lemmon, prenom=Jack, email=jlemmon@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]

SQL trouverPersonnesParPrenomSQL : chris
------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.civilite_cle as civilite5_1_,
        personne0_.email as email2_1_,
        personne0_.nom as nom3_1_,
        personne0_.prenom as prenom4_1_
    from
        personne_repo personne0_
    where
        personne0_.prenom=?
Hibernate:
    select
        civilite0_.cle as cle1_0_0_,
        civilite0_.libelle as libelle2_0_0_
    from
        civilite_repo civilite0_
    where
        civilite0_.cle=?
Personne [id=4, nom=Lemmon, prenom=Chris, email=clemmon@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]
```

- Constater, qu'Hibernate génère des requêtes sur la même table au final.

Aller plus loin :

- Vous pouvez tester de mettre Personne dans le cas de l'entité :

```
@Query("select p from Personne p where p.prenom = ?1")
List<Personne> trouverPersonnesParPrenomJPQL(String prenom);
```

- constater l'erreur produite :

```
Caused by: java.lang.IllegalArgumentException: org.hibernate.hql.internal.ast.QuerySyntaxException:
Personne is not mapped [select p from Personne p where p.prenom = ?1]
```

- Spring vous indique bien, qu'il n'y a pas d'entité Personne qui soit mappée.
- Elle s'appelle EntitePersonne

- Remettre le code comme précédemment.

3. Repository, JPQL et requête nommée

- Sur la classe Personne :

```
package fr.eni.demo.repository;

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

@Entity(name = "EntitePersonne")
@Table(name = "personne_repo")

@NamedQueries({
		@NamedQuery(name = "Personne.findNomCommencePar", query = "select p from EntitePersonne p
where p.nom like CONCAT(:var,'%')"),
		@NamedQuery(name = "Personne.findMessieurs", query = "select p from EntitePersonne p where
p.civilite.cle = 'M'") })
public class Personne {
```

  - Ajout de 2 requêtes nommées
  - JPQL : impose utilisation du nom de l'entité
  - like : impose la concaténation d'un % au paramètre déclaré :var
  - Utilisation de paramètre déclaré pour être plus simple que ?1

- Dans le Repository de Personne
  - Ajouter les 2 méthodes pour appeler les requêtes nommées :

```
	// Requêtes nommées
	List<Personne> findNomCommencePar(@Param("var")String var);//paramètre déclaré :var

	List<Personne> findMessieurs();
}
```

  - Pour passer le paramètre déclaré :var, utilisation de l'annotation @Param(«var»)

- Dans la classe d'exécution de l'application
  - Compléter le bean précédent :

```
System.out.println("\n@NamedQuery findNomCommencePar : Lem");
System.out.println("-------------------------");
for (fr.eni.demo.repository.Personne p : persRepository.findNomCommencePar("Lem")) {
	System.out.println(p);
}

System.out.println("\n@NamedQuery findMessieurs");
System.out.println("-------------------------");
for (fr.eni.demo.repository.Personne p : persRepository.findMessieurs()) {
	System.out.println(p);
}
```

- Exécution, Traces attendues :

```
…

@NamedQuery findNomCommencePar : Lem
------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.civilite_cle as civilite5_1_,
        personne0_.email as email2_1_,
        personne0_.nom as nom3_1_,
        personne0_.prenom as prenom4_1_
    from
        personne_repo personne0_
    where
        personne0_.nom like concat(?, '%')
Hibernate:
    select
        civilite0_.cle as cle1_0_0_,
        civilite0_.libelle as libelle2_0_0_
    from
        civilite_repo civilite0_
    where
        civilite0_.cle=?
Personne [id=2, nom=Lemmon, prenom=Jack, email=jlemmon@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]
Personne [id=4, nom=Lemmon, prenom=Chris, email=clemmon@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]

@NamedQuery findMessieurs
------------------------
Hibernate:
    select
        personne0_.id as id1_1_,
        personne0_.civilite_cle as civilite5_1_,
        personne0_.email as email2_1_,
        personne0_.nom as nom3_1_,
        personne0_.prenom as prenom4_1_
    from
        personne_repo personne0_
    where
        personne0_.civilite_cle='M'
Hibernate:
    select
        civilite0_.cle as cle1_0_0_,
        civilite0_.libelle as libelle2_0_0_
    from
        civilite_repo civilite0_
    where
        civilite0_.cle=?
Personne [id=1, nom=Dupontel, prenom=Albert, email=adupontel@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]
Personne [id=2, nom=Lemmon, prenom=Jack, email=jlemmon@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]
Personne [id=4, nom=Lemmon, prenom=Chris, email=clemmon@eni.fr, civilite=Civilite [cle=M,
libelle=Monsieur]]
```

Aller plus loin :

- Pour mettre en avant que les requêtes nommées sont chargées et validées dès le chargement de l'entité :
    - Créer une faute dans la requête nommée de la classe Personne, retirer le p :

```
@Entity(name = "EntitePersonne")
@Table(name = "personne_repo")
@NamedQueries({
        @NamedQuery(name="Personne.findNomCommencePar",
                        query="select p from EntitePersonne where p.nom like CONCAT(:var,'%') "),
        @NamedQuery(name="Personne.findMessieurs",
        query="select p from EntitePersonne p where p.civilite.cle = 'M'")
})
public class Personne {
```

- Le chargement des requêtes nommées se fait après la création des tables, et elles sont testées
    - constater l'erreur produite :

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

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

    alter table personne_repo
       add constraint FKhygv753xf89qfdwfnsqt3414x
       foreign key (civilite_cle)
       references civilite_repo (cle)
…

org.hibernate.QueryException: Unable to resolve path [p.nom], unexpected token [p] [select p from
fr.eni.demo.repository.Personne where p.nom like CONCAT(:var,'%') ]
        …
```

    - Spring vous indique bien qu'il n'existe pas de paramètre p
- Remettre le code comme précédemment.