Atelier 02 Spring Boot : Persister et interroger les données avec

Spring Data JPA

**Objectifs :**

1. Créer une association *OneToMany* entre deux entités,
2. Utilisation de Lombok,
3. Interroger les entités en fournissant un attribut non clé,
4. Ecrire des requêtes @Query en utilisant le langage *JPQL,*
5. Ecrire des requêtes @Query en passant des entités en paramètre,
6. Interroger les produits selon l’id de leur catégorie,
7. Trier les données,
8. Ajouter les méthodes du *Repository* à la couche *Service*.

# Créer une association *OneToMany* entre deux entités

|  |  |  |
| --- | --- | --- |
| Categorie | 1..1 1..\* | Produit |
| - IdCat : Long | - IdProduit : Long |
| * nomCat : String * descriptionCat : String | categorie produits | * nomProduit : String * prixProduit : double |
|  |  |

1. Sachant qu’un produit possède une et une seule catégorie, et une catégorie contient plusieurs produits. Créer, L’entité *Categorie* :

**package** com.example.demo.entities;

**import** java.util.List;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue; **import** jakarta.persistence.GenerationType; **import** jakarta.persistence.Id;

**import** jakarta.persistence.OneToMany;

@Entity

**public class** Categorie { @Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long idCat;

**private** String nomCat;

**private** String descriptionCat;

@OneToMany(mappedBy = "categorie")

**private** List<Produit> produits;

**public** Categorie() {}

**public** Categorie(String nomCat, String descriptionCat, List<Produit> produits)

{

**super**();

**this**.nomCat = nomCat; **this**.descriptionCat = descriptionCat; **this**.produits = produits;

}

**public** Long getIdCat() {

**return** idCat;

}

**public void** setIdCat(Long idCat) {

**this**.idCat = idCat;

}

**public** String getNomCat() {

**return** nomCat;

}

**public void** setNomCat(String nomCat) {

**this**.nomCat = nomCat;

}

**public** String getDescriptionCat() {

**return** descriptionCat;

}

**public void** setDescriptionCat(String descriptionCat) {

**this**.descriptionCat = descriptionCat;

}

**public** List<Produit> getProduits() {

**return** produits;

}

**public void** setProduits(List<Produit> produits) {

**this**.produits = produits;

}

}

1. Ajouter l’attribut *categorie* à l’entité *Produit* :

@ManyToOne

**private** Categorie categorie;

Générer les getters et les setters pour l’attribut *categorie*

**public** Categorie getCategorie() {

**return** categorie;

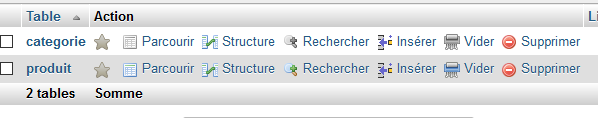
}

**public void** setCategorie(Categorie categorie) {

**this**.categorie = categorie;

}

1. Démarrer l’application et voir les changements dans la base données



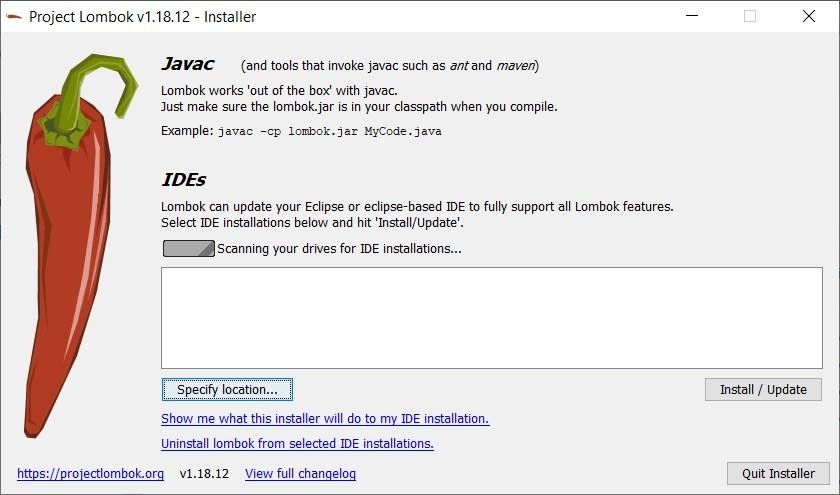
# Utilisation de Lombok

Lombok est une bibliothèque java permettant de simplifier le code des entités en épargnant au développeur l’écriture des méthodes getters, setters, equals,..

Lombok injecte automatiquement ces méthode dans le byte code lors de la compilation.

# Installation de Lombok

1. Télécharger Lombok.jar à partir de : <https://projectlombok.org/download> puis exécuter le :



1. Spécifier l’emplacement de l’exécutable de votre IDE : SpringToolSuite4.exe
2. Ajouter la dépendance Lombok au fichier pom.xml :

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

1. Modifier le code de l’entité *Categorie* comme suit :

**package** com.example.demo.entities;

**import** java.util.List;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue; **import** jakarta.persistence.GenerationType; **import** jakarta.persistence.Id;

**import** jakarta.persistence.OneToMany;

**import** com.fasterxml.jackson.annotation.JsonIgnore;

**import** lombok.AllArgsConstructor;

**import** lombok.Data;

**import** lombok.NoArgsConstructor;

@Data @NoArgsConstructor @AllArgsConstructor @Entity

**public class** Categorie { @Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long idCat;

**private** String nomCat;

**private** String descriptionCat;

@JsonIgnore

@OneToMany(mappedBy = "categorie")

**private** List<Produit> produits;

}

# Interroger les entités en fournissant un attribut non clé

On se propose à présent de créer les méthodes nécessaires pour interroger les données des produits en fournissant le nom du produit comme critère de recherche

1. Créer la méthode *findByNomProduit* dans l’interface *ProduitRepository* :

**public interface** ProduitRepository **extends** JpaRepository<Produit, Long> {

List<Produit> findByNomProduit(String nom); List<Produit> findByNomProduitContains(String nom);

}

1. Tester les méthodes *findByNomProduit et findByNomProduitContains* en ajoutant les méthodes *testFindByNomProduit et testFindByNomProduitContains* à la classe *ProduitsApplicationTests* :

@Test

**public void** testFindByNomProduit()

{

List<Produit> prods = produitRepository.findByNomProduit("iphone X");

**for** (Produit p : prods)

{

System.***out***.println(p);

}

}

@Test

**public void** testFindByNomProduitContains ()

{

List<Produit> prods=produitRepository.findByNomProduitContains("iphone");

**for** (Produit p : prods)

{

System.***out***.println(p);

} }

# Ecrire des requêtes @Query en utilisant le langage *JPQL*

On se propose à présent de créer les méthodes nécessaires pour interroger les données des produits en fournissant plusieurs critères de recherche. Pour ce faire on va utiliser l’annotation @Query et le langage JPQL.

1. Créer la méthode *findByNomPrix* dans l’interface *ProduitRepository*, qui retourne les produits dont le nom contient un texte donné et le prix est supérieur à une valeur donnée :

@Query("select p from Produit p where p.nomProduit like %?1 and p.prixProduit > ?2")

List<Produit> findByNomPrix (String nom, Double prix);

On peut nommer les paramètres avec l’annotation @Param :

@Query("select p from Produit p where p.nomProduit like %:nom and p.prixProduit > :prix") List<Produit> findByNomPrix (@Param("nom") String nom,@Param("prix") Double prix);

1. Tester la méthode *findByNomPrix* en ajoutant la méthode *testFindByNomPrix*

à la classe *ProduitsApplicationTests* :

@Test

**public void** testfindByNomPrix()

{

List<Produit> prods = produitRepository.findByNomPrix("iphone X", 1000.0);

**for** (Produit p : prods)

{

System.***out***.println(p);

}

}

# Ecrire des requêtes @Query en passant des entités en paramètre

On se propose, ici, de récupérer les produits qui ont une catégorie donnée.

1. Créer la méthode *findByCategorie* dans l’interface *ProduitRepository*, qui retourne les produits ayant une catégorie donnée :

@Query("select p from Produit p where p.categorie = ?1") List<Produit> findByCategorie (Categorie categorie);

1. Tester la méthode *findByCategorie* en ajoutant la méthode

*testFindByCategorie à* la classe *ProduitsApplicationTests* :

@Test

**public void** testfindByCategorie()

{

Categorie cat = **new** Categorie(); cat.setIdCat(1L);

List<Produit> prods = produitRepository.findByCategorie(cat);

**for** (Produit p : prods)

{

System.***out***.println(p);

}

}

# Interroger les produits selon l’id de leur catégorie

On se propose, ici, de récupérer les produits. Et ce en passant en paramètre l’Id de leur catégorie.

1. Créer la méthode *findByCategorieIdCat* dans l’interface *ProduitRepository*, qui retourne les produits ayant une catégorie donnée, en fournissant l’id de la catégorie:

List<Produit> findByCategorieIdCat(Long id);

1. Tester la méthode *findByCategorie* en ajoutant la méthode

*testFindByCategorie à* la classe *ProduitsApplicationTests* :

@Test

**public void** findByCategorieIdCat()

{

List<Produit> prods = produitRepository.findByCategorieIdCat(1L);

**for** (Produit p : prods)

{

System.***out***.println(p);

}

}

# Trier les données.

Trier les produits selon leurs noms

1. Créer la méthode *findByOrderByNomProduitAsc* dans l’interface *ProduitRepository* :

List<Produit> findByOrderByNomProduitAsc();

1. Tester la méthode *findByOrderByNomProduitAsc* en ajoutant la méthode

*testfindByOrderByNomProduitAsc à* la classe *ProduitsApplicationTests* :

@Test

**public void** testfindByOrderByNomProduitAsc()

{

List<Produit> prods = produitRepository.findByOrderByNomProduitAsc();

**for** (Produit p : prods)

{

System.***out***.println(p);

}

}

Trier les produits selon leurs noms et leurs prix

1. Créer la méthode *trierProduitsNomsPrix* dans l’interface *ProduitRepository* :

@Query("select p from Produit p order by p.nomProduit ASC, p.prixProduit DESC") List<Produit> trierProduitsNomsPrix ();

1. Tester la méthode *trierProduitsNomsPrix* en ajoutant la méthode

*testrierProduitsNomsPrix à* la classe *ProduitsApplicationTests* :

@Test

**public void** testTrierProduitsNomsPrix()

{

List<Produit> prods = produitRepository.trierProduitsNomsPrix();

**for** (Produit p : prods)

{

System.***out***.println(p);

}

}

# Ajouter les méthodes du Repository à la couche Service

Une fois on a développé et tester les différentes méthodes ci-dessus, on peut les ajouter à la couche service, on modifiant l’interface ProduitService comme suit :

**public interface** ProduitService { Produit saveProduit(Produit p); Produit updateProduit(Produit p); **void** deleteProduit(Produit p); **void** deleteProduitById(Long id); Produit getProduit(Long id); List<Produit> getAllProduits();

List<Produit> findByNomProduit(String nom); List<Produit> findByNomProduitContains(String nom); List<Produit> findByNomPrix (String nom, Double prix); List<Produit> findByCategorie (Categorie categorie); List<Produit> findByCategorieIdCat(Long id); List<Produit> findByOrderByNomProduitAsc(); List<Produit> trierProduitsNomsPrix();

}

Puis en implémentant ces méthodes dans la classe ProduitServiceImpl :

@Override

**public** List<Produit> findByNomProduit(String nom) {

**return** produitRepository.findByNomProduit(nom);

}

@Override

**public** List<Produit> findByNomProduitContains(String nom) {

**return** produitRepository.findByNomProduitContains(nom);

}

@Override

**public** List<Produit> findByNomPrix(String nom, Double prix) {

**return** produitRepository.findByNomPrix(nom, prix);

}

@Override

**public** List<Produit> findByCategorie(Categorie categorie) {

**return** produitRepository.findByCategorie(categorie);

}

@Override

**public** List<Produit> findByCategorieIdCat(Long id) {

**return** produitRepository.findByCategorieIdCat(id);

}

@Override

**public** List<Produit> findByOrderByNomProduitAsc() {

**return** produitRepository.findByOrderByNomProduitAsc();

}

@Override

**public** List<Produit> trierProduitsNomsPrix() {

**return** produitRepository.trierProduitsNomsPrix();

}