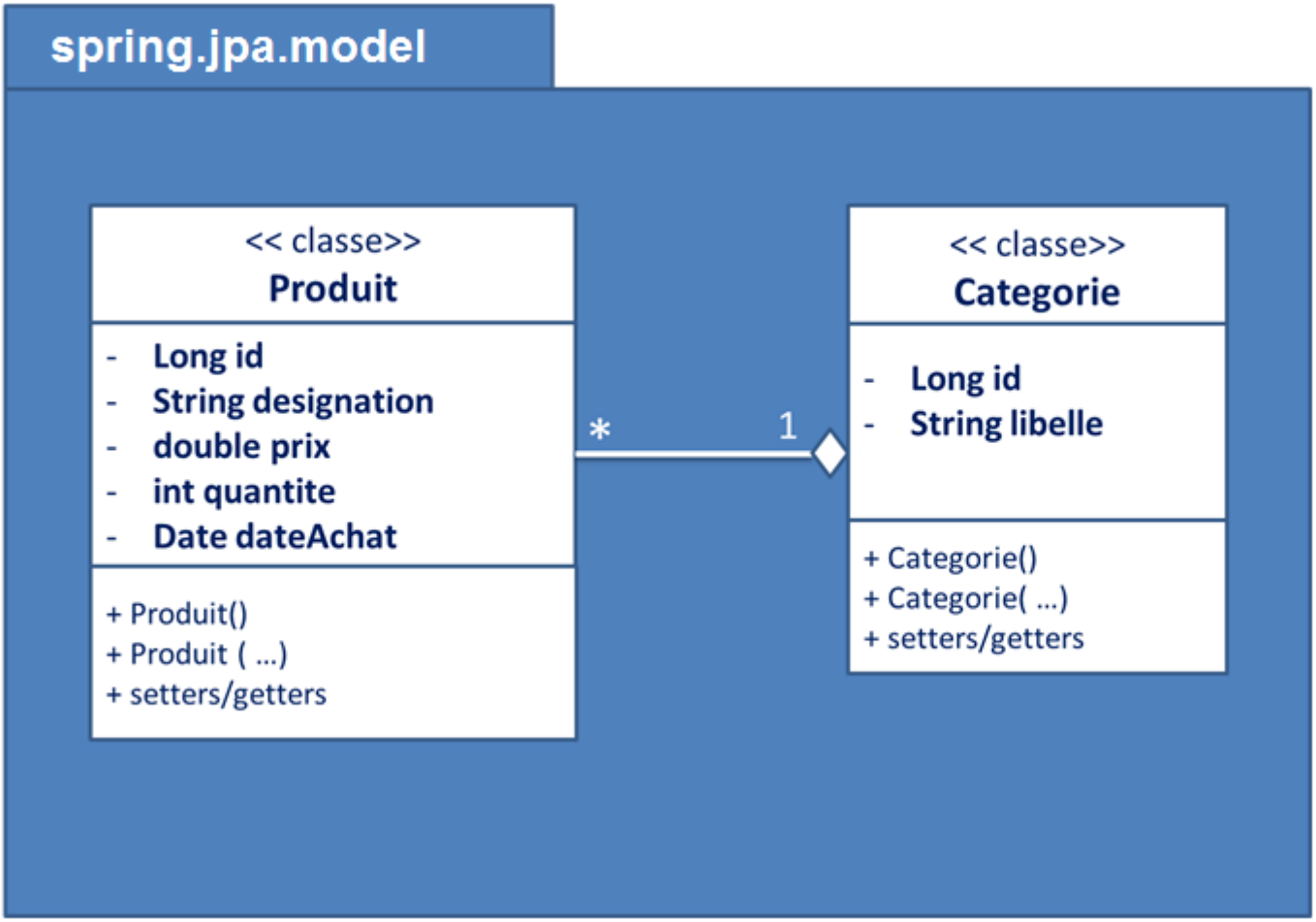
Atelier Spring -03-
JPA –Partie06

Objectifs

- **JPA et Spring Data**
  - Manipuler les relations entre les tables
  - @OneToMany, @ManyToOne
  - Comportement de cascade
  - Mode de récupération de collection

Voici un modèle conceptuel qui modélise une relation **1-N** d'un côté et **N-1** de l'autre côté entre une classe « **Categorie** » et une classe « **Produit** » :



1. Prendre une copie du projet «**jpa-spring-data-spring-boot**» et la nommer «**jpa-spring-data-spring-boot-2-relations**».
2. Créer une nouvelle entité «**Categorie**» ayant le code suivant :

```
package spring.jpa.model;

import java.util.ArrayList;
import java.util.Collection;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.persistence.OneToMany;

@Entity
public class Categorie
{
	@Id
	@GeneratedValue
	private Long id;

	@Column(length = 50)
	private String code;

	@Column(length = 50)
	private String libelle;


	@OneToMany (mappedBy = "categorie" )
    private Collection <Produit> produits = new ArrayList<Produit>();


	public Collection<Produit> getProduits() {
		return produits;
	}

	public void setProduits(Collection<Produit> produits) {
		this.produits = produits;
	}

	public Categorie(String code, String libelle) {
		super();
		this.code = code;
		this.libelle = libelle;
	}

	@Override
	public String toString() {
		return "Categorie [id=" + id + ", code=" + code + ", libelle=" + libelle +
"]";
	}

	public Categorie() {}
```

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

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

	public String getCode() {
		return code;
	}

	public void setCode(String code) {
		this.code = code;
	}

	public String getLibelle() {
		return libelle;
	}

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

3. Remplacer le code de l'entité «**Produit**» par le code suivant :

```
package spring.jpa.model;

import java.util.Date;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Temporal;
import jakarta.persistence.TemporalType;

@Entity
public class Produit {
	@Id
	@GeneratedValue
	private Long id;

	@Column(length = 50)
	private String designation;

	private double prix;
	private int quantite;

	@Temporal(TemporalType.DATE)
	java.util.Date dateAchat;

	@ManyToOne
```

```
    private Categorie categorie;

    public Produit(String designation, double prix, int quantite, Date dateAchat,
Categorie categorie) {
        super();
        this.designation = designation;
        this.prix = prix;
        this.quantite = quantite;
        this.dateAchat = dateAchat;
        this.categorie = categorie;
    }
    public Categorie getCategorie() {
        return categorie;
    }
    public void setCategorie(Categorie categorie) {
        this.categorie = categorie;
    }
    public java.util.Date getDateAchat() {
        return dateAchat;
    }

    public void setDateAchat(java.util.Date dateAchat) {
        this.dateAchat = dateAchat;
    }

    public Produit(String designation, double prix, int quantite, Date dateAchat) {
        super();
        this.designation = designation;
        this.prix = prix;
        this.quantite = quantite;
        this.dateAchat = dateAchat;
    }

    public Long getId() {
        return id;
    }

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

    public String getDesignation() {
        return designation;
    }

    public void setDesignation(String designation) {
        this.designation = designation;
    }

    public double getPrix() {
        return prix;
    }
    public void setPrix(double prix) {
        this.prix = prix;
    }
    public int getQuantite() {
```

```
		return quantite;
	}
	public void setQuantite(int quantite) {
		this.quantite = quantite;
	}
	public Produit() {
		super();
	}

	public Produit(String designation, double prix, int quantite) {
		super();
		this.designation = designation;
		this.prix = prix;
		this.quantite = quantite;
	}
	@Override
	public String toString() {
		return "Produit [id=" + id + ", designation=" + designation + ", prix=" +
prix + ", quantite=" + quantite
				+ ", dateAchat=" + dateAchat + ", categorie=" + categorie + "]";
	}

	public Produit(Long id, String designation, double prix, int quantite) {
		super();
		this.id = id;
		this.designation = designation;
		this.prix = prix;
		this.quantite = quantite;
	}
}
```

- Remarquer la mise en place d'une relation **1-N** via l'instruction dans l'entité « **Produit** »

```
@ManyToOne
    private Categorie categorie;
```

- L'attribut « **categorie** » est un attribut de relation avec l'entité « **Categorie** ».
- Cet attribut est transformé en une colonne « **categorie_id** » dans la table « **Produit** » qui représente une clé étrangère sur la colonne « **id** » de la table « **Categorie** ».

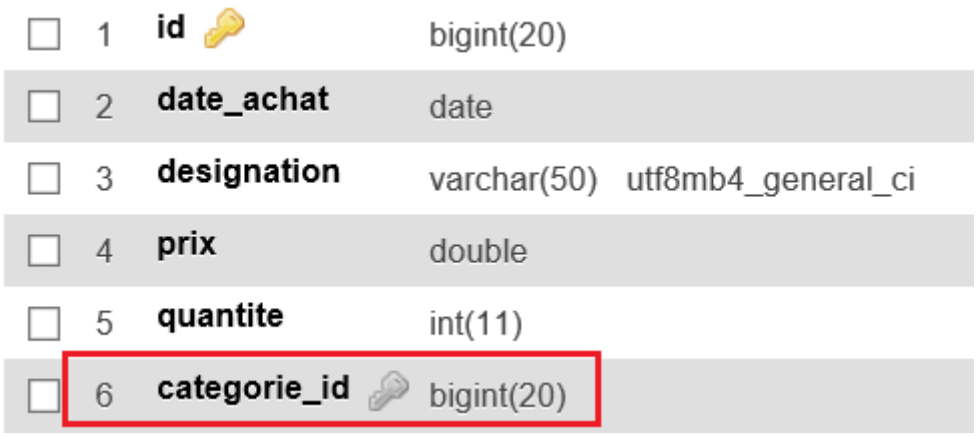
- De l'autre côté, une relation **N-1** est déclarée avec l'entité « **Produit** » via l'instruction :

```
@OneToMany (mappedBy = "categorie" )
    private Collection <Produit> produits = new ArrayList<Produit>();
```

- Il s'agit bien d'une relation **bidirectionnelle**
- L'attribut « **mappedBy** » dans l'annotation « **@OneToMany** » permet de référencer la relation dans l'entité « **Produit** ».

4. Rendre la propriété « spring.jpa.hibernate.ddl-auto » à la valeur « **create** » dans le fichier «**application.properties**» (Pour restructurer les tables)

5. Lancer l'exécution du projet et remarquer la génération de clé étrangère « **categorie_id** » dans la table « **Produit** »



6. Passer maintenant pour réaliser des traitements sur les données. Créer une interface «**CategorieRepository**» qui hérite de l'interface «**JpaRepository**» ayant le code suivant :

```
package spring.jpa.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import spring.jpa.model.Categorie;

public interface CategorieRepository extends JpaRepository<Categorie, Long> { }
```

7. Prendre la nouvelle version de la classe « **JpaSpringBootApplication** » :

```
package spring.jpa;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Collection;
import java.util.List;


import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import spring.jpa.model.Categorie;
import spring.jpa.model.Produit;
import spring.jpa.repository.CategorieRepository;
import spring.jpa.repository.ProduitRepository;

@SpringBootApplication
public class JpaSpringBootApplication {

      static ProduitRepository produitRepos ;
```

```
// Récupérer une implémentation de l'interface "CategorieRepository" par injection de dépendance
	static CategorieRepository categorieRepos;



	public static void main(String[] args) {

// référencer le contexte
		ApplicationContext contexte =
SpringApplication.run(JpaSpringBootApplication.class, args);
// Récupérer une implémentation de l'interface "ProduitRepository" par injection de dépendance
		produitRepos =contexte.getBean(ProduitRepository.class);

// Récupérer une implémentation de l'interface "CategorieRepository" par injection de dépendance
		categorieRepos =contexte.getBean(CategorieRepository.class);

		// créer deux catégories;
		Categorie cat1 = new Categorie("AL", "Alimentaire");
		Categorie cat2 = new Categorie("PL", "Plastique");

		//Attacher les deux catégories à la BD (insertion)
		categorieRepos.save(cat1);
		categorieRepos.save(cat2);


		SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
		java.util.Date date1 = null;
		java.util.Date date2 = null;
		java.util.Date date3 = null;

		try {
			date1 = sdf.parse("2022-04-15");
			date2 = sdf.parse("2022-02-15");
			date3 = sdf.parse("2022-05-15");
		} catch (ParseException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		// Insérer 3 produits
		Produit p1 =new Produit("Yahourt", 0.400, 20, date1 , cat1);
		Produit p2 =new Produit("Chocolat", 2000.0, 5, date2, cat1);
		Produit p3 =new Produit("Panier", 1.200, 30, date3, cat2);
		produitRepos.save(p1);
		produitRepos.save(p2);
		produitRepos.save(p3);

		//Afficher la liste des produits
		afficherTousLesProduits();

		//Afficher la liste des produits d'une catégorie donnée
		afficherTousLesProduitsDeLaCategorie(1L);

		//Modifier la catégorie d'un produit
		p1.setCategorie(cat2);
		//Synchroniser avec la BD
		produitRepos.saveAndFlush(p1);
		//Afficher la liste des produits d'une catégorie donnée

		//afficherTousLesProduitsDeLaCategorie(1L);
		afficherTousLesProduitsDeLaCategorie(1L);
```

```
	}


	static void afficherTousLesProduits()
	{
		System.out.println("****************************************");
		System.out.println("***    Liste de tous les produits    ***");
		System.out.println("****************************************");
		// Lister l'ensemble des produits
		List<Produit> lp = produitRepos.findAll();
		for (Produit p : lp)
		{
			System.out.println(p);
		}
		System.out.println("****************************************");
	}

	static void afficherTousLesProduitsDeLaCategorie(Long id)
	{
	   System.out.println("*********************************************************");
	   System.out.println("***    Liste de tous les produits de la catégorie["+id+"]   ***");
	   System.out.println("*********************************************************");

	  // récupérer l'entité "Catégorie" ayant l'id en paramètres
		Categorie cD = categorieRepos.getOne(id);

			if (cD !=null)
			{
				// Lister l'ensemble des produits
			Collection <Produit> lC = cD.getProduits();
				for (Produit p : lC)
				{
					System.out.println(p);
				}
			}
			else
			{
				System.out.println("catégorie non existante...");
			}
		System.out.println("****************************************");
	}
}
```

8. Lancer l'exécution. Remarquer l'affichage d'une erreur :

```
could not initialize proxy [spring.jpa.model.Categorie#1] - no Session
```

9. Ajouter à la fin du fichier « **application.properties** » la ligne suivante :

```
spring.jpa.properties.hibernate.enable_lazy_load_no_trans=true
```

10. Relancer l'exécution et remarquer l'insertion dans la base de données :

+ Options

	id	date_achat	designation	prix	quantite	categorie_id
Éditer Copier Supprimer	3	2022-04-15	Yahourt	0.4	20	2
Éditer Copier Supprimer	4	2022-02-15	Chocolat	2000	5	1
Éditer Copier Supprimer	5	2022-05-15	Panier	1.2	30	2

11. Mettre l'instruction suivante en commentaire et relancer l'exécution (ligne 77 dans le code de la classe « **JpaSpringBootApplication** » :

```
produitRepos.saveAndFlush(p1);
```

- Remarquer que la modification de la catégorie n'est pas réalisée au niveau de la base de données :

+ Options

	id	date_achat	designation	prix	quantite	categorie_id
Éditer Copier Supprimer	3	2022-04-15	Yahourt	0.4	20	1
Éditer Copier Supprimer	4	2022-02-15	Chocolat	2000	5	1
Éditer Copier Supprimer	5	2022-05-15	Panier	1.2	30	2

- En fait, la méthode **saveAndFlush** permet de synchroniser les entités chargées en mémoire avec l'état des tables dans la BD
- Décommenter l'instruction pour enregistrer les modifications dans la BD.

12. Passons maintenant à la suppression d'une entité «**Categorie**». Ajouter le code suivant à la fin de la méthode «**main**» dans la classe « **JpaSpringBootApplication** » :

```
        //Supprimer la catégorie ayant l'id :2
System.out.println("*********************************");
System.out.println("Suppression de la categorie[2]...");
Categorie cs = categorieRepos.getOne(2L);
if (cs!=null) categorieRepos.delete(cs);
System.out.println("*********************************");
//Afficher la liste des produits
 afficherTousLesProduits();
```

13. Lancer l'exécution et remarquer la génération de l'erreur suivante :

```
Cannot delete or update a parent row: a foreign key constraint fails (`spring-jpa-0`.`produit`, CONSTRAINT `FK52xhp55kbbl6u4rbluxm3g9hw` FOREIGN KEY (`categorie_id`) REFERENCES `categorie` (`id`))
```

14. Cette erreur est due à la suppression d'une catégorie liée par une contrainte de clé étrangère dans la table « **Produit** ».
    - ✓ Pour remédier à ce problème, indiquer à Spring Data de réaliser une suppression en cascade lors de la suppression d'une catégorie (le cascade « **REMOVE** ») : càd supprimer aussi tous les produits qui y sont liés. Modifier l'annotation « **@OneToMany** » dans la classe « **Categorie** » comme suit :

```
@OneToMany (mappedBy = "categorie" ,cascade = {CascadeType.REMOVE})
private Collection <Produit> produits = new ArrayList<Produit>();
```

    - ✓ Relancer l'exécution et remarquer la suppression du produit attaché à la catégorie supprimée :

+ Options

	id	date_achat	designation	prix	quantite	categorie_id
Éditer Copier Supprimer	3	2022-04-15	Yahourt	0.4	20	1
Éditer Copier Supprimer	4	2022-02-15	Chocolat	2000	5	1

15. Maintenant, modifier une catégorie en lui associant un nouveau produit, ajouter le code suivant à la fin de la méthode « **main** » :

```
// Modifier une catégorie existante par l'ajout d'un nouveu produit
        Produit p4 =new Produit("ADOL", 7.400, 20, date1 );
        p4.setCategorie(cat1);
        cat1.getProduits().add(p4);
        categorieRepos.save(cat1);

       //Afficher la liste des produits
        afficherTousLesProduits();
```

16. Lancer l'exécution et remarquer que rien ne change dans la BD.
    - ✓ Il est nécessaire d'indiquer à Spring Data de réaliser une mise à jour en cascade lors de la modification d'une catégorie.
    - ✓ Ajouter, alors, dans l'annotation « **@OneToMany** » le cascade « **MERGE** » comme suit:

```
@OneToMany (mappedBy = "categorie" ,cascade = {CascadeType.REMOVE  ,
CascadeType.MERGE  })
    private Collection <Produit> produits = new ArrayList<Produit>();
```

✓ Lancer l'exécution et remarquer l'ajout du nouveau produit affecté à la catégorie ayant l'id 1 :

+ Options

	id	date_achat	designation	prix	quantite	categorie_id
Éditer Copier Supprimer	3	2022-04-15	Yahourt	0.4	20	1
Éditer Copier Supprimer	4	2022-02-15	Chocolat	2000	5	1
Éditer Copier Supprimer	6	2022-04-15	ADOL	7.4	20	1

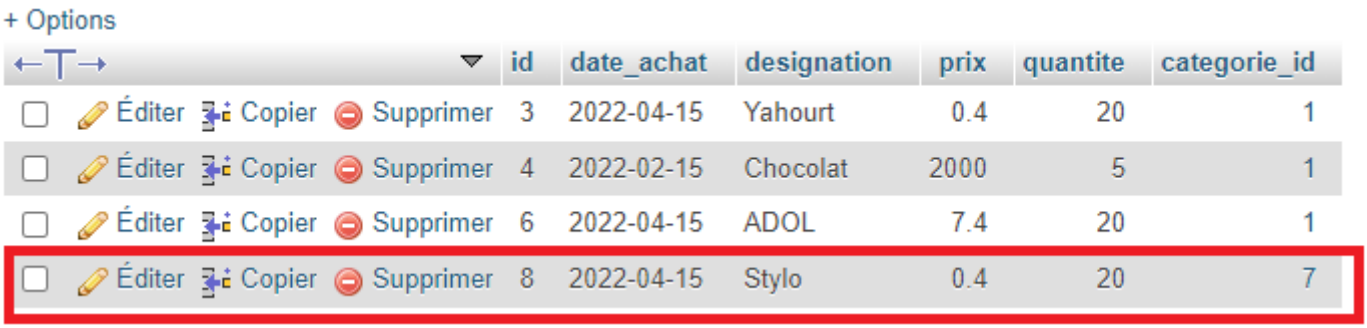
17. Maintenant, insérer une nouvelle catégorie avec un nouveau produit :
    ✓ ajouter le code suivant à la fin de la méthode « **main** » :

```
// Insérer une nouvelle  catégorie  avec l'ajout d'un nouveau produit
        Produit p5 =new Produit("Stylo", 0.400, 20, date1 );
         Categorie cat3 = new Categorie("BR", "Bureautique");
         p5.setCategorie(cat3);
         cat3.getProduits().add(p5);
         categorieRepos.save(cat3);
    //Afficher la liste des produits
       afficherTousLesProduits();
```

18. Lancer l'exécution et remarquer qu'une catégorie est ajoutée alors que le produit n'est pas ajouté dans la BD.
    ✓ Il est nécessaire d'indiquer à Spring Data de réaliser une mise à jour en cascade lors de l'insertion d'une catégorie.
    ✓ Ajouter alors dans l'annotation « **@OneToMany** » le cascade « **PERSIST** » comme suit :

```
@OneToMany (mappedBy = "categorie" ,cascade = {CascadeType.REMOVE  ,
CascadeType.MERGE , CascadeType.PERSIST})
    private Collection <Produit> produits = new ArrayList<Produit>();
```

✓ Lancer l'exécution et remarquer l'insertion du nouveau produit :



+ Options

	id	date_achat	designation	prix	quantite	categorie_id
Éditer Copier Supprimer	3	2022-04-15	Yahourt	0.4	20	1
Éditer Copier Supprimer	4	2022-02-15	Chocolat	2000	5	1
Éditer Copier Supprimer	6	2022-04-15	ADOL	7.4	20	1
Éditer Copier Supprimer	8	2022-04-15	Stylo	0.4	20	7

19. Il est possible de réécrire l'instruction précédente autrement en utilisant le type de cascade « **ALL** »

```
@OneToMany (mappedBy = "categorie" ,cascade = CascadeType.ALL  )
 private Collection <Produit> produits = new ArrayList<Produit>();
```

20. Lancer le test