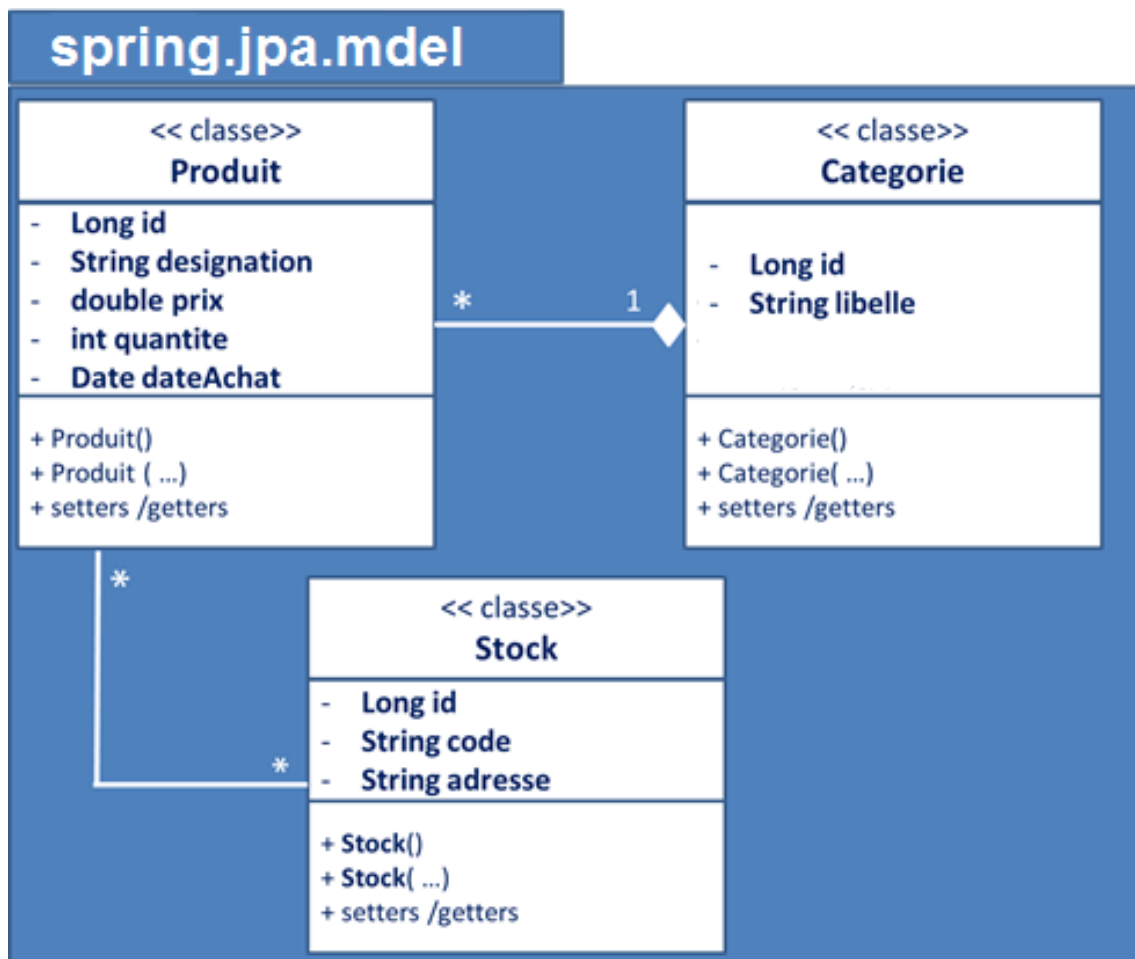


Atelier Spring -03- JPA –Partie07

Objectifs

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

Voici un modèle conceptuel qui modélise une relation **N-N** une classe « **Produit** » et une classe « **Stock** » :



1. Prendre une copie du projet «**jpa-spring-data-spring-boot-2-relations**» et le nommer «**jpa-spring-data-spring-boot-2-relations-2**»

2. Créer une nouvelle entité «**Stock**» 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.ManyToMany;

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

    @Column(length = 50)
    private String code;

    @Column(length = 50)
    private String adresse;

    @ManyToMany (mappedBy = "stocks")
    private Collection<Produit> produits = new ArrayList<Produit>();

    public String getCode() {
        return code;
    }

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

    public Long getId() {
        return id;
    }

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

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

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

    @Override
    public String toString() {
        return "Stock [id=" + id + ", code=" + code + ", adresse=" + adresse + "];";
    }
}
```

```

    }

    public Stock() {
        super();
    }

    public Stock(String code, String adresse) {
        super();
        this.code = code;
        this.adresse = adresse;
    }

    public String getAdresse() {
        return adresse;
    }

    public void setAdresse(String adresse) {
        this.adresse = adresse;
    }
}

```

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

```

package spring.jpa.model;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Date;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToMany;
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;
```

@ManyToMany

```
private Collection<Stock> stocks = new ArrayList<Stock>();
```

```
public Collection<Stock> getStocks() {  
    return stocks;  
}
```

```
public void setStocks(Collection<Stock> stocks) {  
    this.stocks = stocks;  
}
```

```
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 **N-N** via l'instruction suivante dans l'entité « **Produit** » :

```

@ManyToMany
private Collection<Stock> stocks = new ArrayList<Stock>();

```

- L'attribut « **stocks** » est un attribut de relation avec l'entité « **Stock** »
- De l'autre côté, une relation **N-N** est déclarée dans l'entité « **Stock** » via l'instruction :

```

@ManyToMany (mappedBy = "stocks")
private Collection<Produit> produits = new ArrayList<Produit>();

```

- Il s'agit bien d'une relation bidirectionnelle
 - L'attribut « **mappedBy** » dans l'annotation « **@ManyToMany** » permet de référencer la relation dans l'entité « **Produit** ».
4. Lancer l'exécution du projet et remarquer la génération d'une table intermédiaire « **produit_stocks** » contenant deux clés étrangères « **produits_id** » et « **stocks_id** ». Ces deux colonnes présentent une clé primaire composée pour cette table intermédiaire.



5. Passer maintenant pour réaliser des traitements sur les données. Créer une interface « **StockRepository** » 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.Stock;

public interface StockRepository extends
JpaRepository<Stock, Long> { }

```

6. 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.model.Stock;
import spring.jpa.repository.CategorieRepository;
import spring.jpa.repository.ProduitRepository;
import spring.jpa.repository.StockRepository;

@SpringBootApplication
public class JpaSpringBootApplication {

    // Récupérer une implémentation de l'interface "ProduitRepository" par injection de dépendance
    static ProduitRepository produitRepos ;

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

    // Récupérer une implémentation de l'interface "StockRepository" par injection de dépendance
    static StockRepository stockRepos;

    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);

        // Récupérer une implémentation de l'interface "StockRepository" par injection de dépendance
        stockRepos =contexte.getBean(StockRepository.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();
}

// créer 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);

// créer deux stocks;
Stock s1 = new Stock("1", "Sfax");
Stock s2 = new Stock("2", "Tunis");

//Ajouter le produit p1 aux deux stocks s1 et s2
p1.getStocks().add(s1);
p1.getStocks().add(s2);

// enregistrement dans la BD en utilisant "produitRepository"
produitRepos.save(p1);
produitRepos.save(p2);
produitRepos.save(p3);

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

static void afficherTousLesProduits()
{
    System.out.println("*****");
    // Lister l'ensemble des produits

```

```

        System.out.println("Afficher tous les 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("*****");
        // récupérer l'entité "Catégorie" ayant l'id en paramètres
        Categorie cD = categorieRepos.getOne(id);

        if (cD !=null)
        {
            // Lister l'ensemble des produits
            System.out.println("Afficher tous les produits de la catégorie ["+id+"]");
            Collection <Produit> lC = cD.getProduits();
            for (Produit p : lC)
            {
                System.out.println(p);
            }
        }
        else
        {
            System.out.println("catégorie non existante...");
        }
        System.out.println("*****");
    }
}

```

- Remarquer les deux instructions qui permettent d'affecter un stock à un produit :

```

//Ajouter le produit p1 aux deux stocks s1 et s2
    p1.getStocks().add(s1); // affecter s1 à p1
    p1.getStocks().add(s2); // affecter s2 à p2

```

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

object references an unsaved transient instance - save the transient instance before flushing: spring.jpa.model.Stock

8. Ajouter dans l'annotation **@ManyToMany** , dans l'entité «**Produit**» le type de cascade en mode «**PERSIST**».
9. Relancer l'exécution et remarquer le succès des insertions comme suit:

Table: Stock

				id	adresse	code
<input type="checkbox"/>		Éditer		Copier		Supprimer
	4	Sfax	1			
<input type="checkbox"/>		Éditer		Copier		Supprimer
	5	Tunis	2			

Table: Produit










<div><div>←T→</div><div>▼</div></div>			id	date_achat	designation	prix	quantite	categorie_id	
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	3	2022-04-15	Yahourt	0.4	20	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	6	2022-02-15	Chocolat	2000	5	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	7	2022-05-15	Panier	1.2	30	2

Table: produit_stocks

produits_id	stocks_id
3	4
3	5

10. Ajouter, maintenant, le code qui permet d'affecter le produit «**p2**» au stock «**s1**» et affecter le produit «**p3**» au stock «**s2**». Prendre le code en gras ci-dessous :

```
// créer deux stocks;
Stock s1 = new Stock("1", "Sfax");
Stock s2 = new Stock("2", "Tunis");

//Affecter les stocks aux produits
p1.getStocks().add(s1);
p1.getStocks().add(s2);

// enregsitrement dans la BD en utilisant "produitRepository"
produitRepos.save(p1);
produitRepos.save(p2);
produitRepos.save(p3);

p2.getStocks().add(s1);
p3.getStocks().add(s2);
```

11. Lancer l'exécution et remarquer que l'affectation n'est pas réalisée. Il est nécessaire de synchroniser la modification avec la base de données. Donc, ajouter les deux lignes, en gras, suivantes :

```
p2.getStocks().add(s1);
p3.getStocks().add(s2);

// enregistrement dans la BD en utilisant "produitRepos"
produitRepos.save(p2);
produitRepos.save(p3);
```

- Relancer l'exécution et remarquer l'affectation dans la table intermédiaire « produit-stocks » :

Table: produit_stocks

produits_id	stocks_id
3	4
3	5
6	4
7	5

12. Supprimer le stock de « sfax » en utilisant l'instruction suivante à la fin de la méthode « main » puis relancer l'exécution.

```
// Supprimer le stock de "sfax"
stockRepos.deleteById(s1.getId());
```

13. Remarquer la génération de l'erreur suivante :

```
Cannot delete or update a parent row: a foreign key constraint fails (`spring-jpa`.`produit_stocks`, CONSTRAINT `FK5s4f144o4178avq5fofeqhkti` FOREIGN KEY (`stocks_id`) REFERENCES `stock` (`id`))
```

14. Cette erreur est due à la suppression d'une entité référencée par une clé étrangère dans une autre entité. Pour remédier à ce problème, ajoute le type de cascade « DELETE » dans l'annotation « @ManyToMany » dans l'entité « Stock » :

```
@ManyToMany (mappedBy = "stocks", cascade = CascadeType.REMOVE)
private Collection<Produit> produits = new ArrayList<Produit>();
```