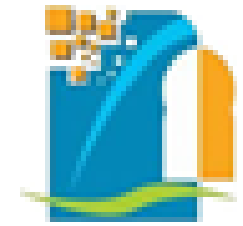




Université Sultan Moulay Slimane
Faculté Des Sciences Et Techniques
- Béni Mellal -



Les relations entre entités

Youssef Saadi

Master Informatique Décisionnelle

Faculté Des Sciences Et Techniques
Université Sultan Moulay Slimane Béni-Mellal

AU: 2019/2020

Objets Inclus : @Embeddable

```
@Embeddable
public class Address implements Serializable {
    @Column(length=30)
    private String address1;

    @Column(length=30)
    private String address2;

    @Column(length=12)
    private String zipCode;

    public Address() {}

    public String getAddress1() { return address1; }
    public void setAddress1(String address1) { this.address1 = address1; }

    public String getAddress2() { return address2; }
    public void setAddress2(String address2) { this.address2 = address2; }

    public String getZipCode() { return zipCode; }
    public void setZipCode(String zipCode) { this.zipCode = zipCode;}
}
```

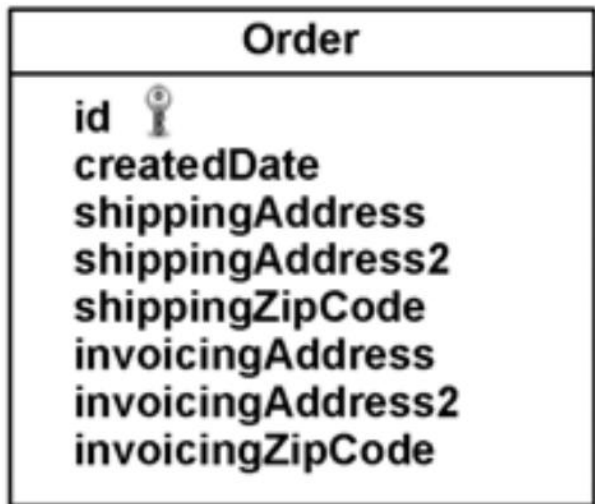
Objets Inclus : @Embedded

```
@Entity
public class Person {
    @Id
    private String firstName;

    @Embedded
    private Address address;

    public Person() { }
    public Address getAddress() { return address; }
    public void setAddress(Address address) { this.address = address; }
}
```

Objets inclus : Plusieurs inclusions



```
@Entity
@Table(name="ORDERS")
public class Order {
    private @Id int id;
    private @Temporal(TemporalType.DATE) Date createdDate;

    @Embedded

    @AttributeOverrides({
        @AttributeOverride(name="address1",
            column=@Column(name="shippingAddress1")),
        @AttributeOverride(name="address2",
            column=@Column(name="shippingAddress2")),
        @AttributeOverride(name="zipCode", column=@Column(name="shippingZipCode"))
    })
    private Address shippingAddress;
```

One to One : relation unidirectionnelle

- Chaque commune possède un maire, et un maire ne peut être maire que d'une seule commune.
- Dans cet exemple :
 - Ce sont les communes qui tiennent la relation entre les communes et les maires.

Entité Esclave

```
@Entity
public class Maire implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(length=40)
    private String nom ;

    // suite de la classe
}
```

L'entité « maître » : Commune

@Entity

```
public class Commune implements Serializable {
```

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

```
private Long id;
```

@Column(length=40)

```
private String nom ;
```

@OneToOne

```
private Maire maire ;
```








```
// suite de la classe
```

```
}
```

Se traduit au niveau BD par ajout d'une colonne



Modification de la BD

 Quick access	Nom		Type
	Index (2)		
	 Index primaire		id
	 FK8yi2t83o5x9on2osqgvhoke47		maire_id
	Clé secondaire (1)		
	 FK8yi2t83o5x9on2osqgvhoke47		maire_id -> maire.id
	Champs (3)		
	 id		int(11)
	 name		varchar(255)
	 maire_id		int(11)

One to One : relation bidirectionnelle

```
@Entity
public class Maire implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(length=40)
    private String nom ;

    @OneToOne(mappedBy="maire") // référence la relation dans la classe Commune
    private Commune commune ;

    // suite de la classe
}
```


Modification de la BD

The screenshot shows a database management tool interface. On the left, there is a sidebar with a 'Quick access' section and a list of databases under 'localhost'. The 'maire' database is selected. The main area displays the table structure for 'maire' with columns: 'id' (int(11), No), 'name' (varchar(255), Ou), and 'commune' (tinyblob, Ou). A primary index is also shown on the 'id' column.

Nom	Type	Nu
Index (1)		
Index primaire	id	
Champs (3)		
id	int(11)	No
name	varchar(255)	Ou
commune	tinyblob	Ou

Relation 1:N cas : unidirectionnel

- JPA crée une table de jointure entre les deux tables associées aux deux entités
- Cette table de jointure porte une clé étrangère vers la clé primaire de la première table, et une clé étrangère vers la clé primaire de la deuxième table.
- Exemple :
 - Un marin appartient à l'équipage d'un seul bateau,
 - Un bateau peut faire naviguer plusieurs Marins.

La classe Marin

```
@Entity
public class Marin implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    // reste de la classe
}
```

L'entité Bateau

```
@Entity
public class Bateau implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    int id;

    String libelle;

    @OneToMany
    Collection<Marin> marins;

    //reste de la classe

}
```

Modification de la BD : la table Bateau et Marin

The image shows two screenshots from phpMyAdmin. The top screenshot shows the 'bateau' table structure, and the bottom screenshot shows the 'marin' table structure. Both tables have a primary key 'id' of type 'int(11)'. The 'bateau' table has a 'libelle' field of type 'varchar(255)'. The 'marin' table has a 'name' field of type 'varchar(255)'. The left sidebar shows the database navigation tree with 'springdb' expanded to show 'bateau' and 'marin'.

Nom	Type
Index (1)	
Index primaire	id
Champs (2)	
id	int(11)
libelle	varchar(255)

Nom	Type
Index (1)	
Index primaire	id
Champs (2)	
id	int(11)
name	varchar(255)

La table de jonction

★ Quick access	Nom	Type	Nul	P...	Extras
localhost booksdb information_schema mysql performance_schema phpmyadmin productdb springdb bateau bateau_marin	Index (2)				
	FKiosvc6vwyfj3qdhmgtv94gmx1	Bateau_id			
	UK_f80rsvfvr3vf4od1hmp1iahs7	marins_id			unique
	Clé secondaire (2)				
	FK2cac71n97tf19493riqwuune7	marins_id -> marin.id			
	FKiosvc6vwyfj3qdhmgtv94gmx1	Bateau_id -> bateau.id			
	Champs (2)				
	Bateau_id	int(11)		N...	
	marins_id	int(11)		N...	

Relation 1:N cas : bidirectionnel

- Une relation 1:p bidirectionnelle doit correspondre à une relation p:1 dans la classe destination de la relation.
- Le caractère bidirectionnel d'une relation 1:p est marqué en définissant l'attribut **mappedBy** sur la relation.
- JPA nous pose une contrainte ici : l'attribut `mappedBy` est défini pour l'annotation `@OneToMany`, mais pas pour l'annotation `@ManyToOne`.

Les deux entités Marin et Bateau

@Entity

```
public class Marin implements Serializable {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
    private Long id;
```

```
    @ManyToOne
```

```
    private Bateau bateau ;
```

```
    // reste de la classe
```

```
}
```

@Entity

```
public class Bateau implements Serializable {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
    private Long id;
```

```
    @OneToMany(mappedBy="bateau")
```

```
    private Collection<Marin> marins ;
```

```
    // reste de la classe
```

```
}
```


Modification de la BD

★ Quick access	Nom	Type
<ul style="list-style-type: none">localhost<ul style="list-style-type: none">booksdbinformation_schemamysqlperformance_schemaphpmyadminproductdbspringdb<ul style="list-style-type: none">bateaucommunecomptesmairemarin	Index (2)	
	🔑 Index primaire	id
	🔑 FKoh44a3munaboqpx59ytgguvyb	bateau_id
	Clé secondaire (1)	
	🔗 FKoh44a3munaboqpx59ytgguvyb	bateau_id -> bateau.id
	Champs (3)	
	📄 id	int(11)
	📄 name	varchar(255)
	📄 bateau_id	int(11)

Relation N:P cas : unidirectionnel

- La façon classique d'enregistrer ce modèle en base consiste à créer une table de jointure, et c'est ce que fait JPA.
- Une table de jointure est créé entre les deux tables qui portent les entités. Cette table référence les deux clés primaires des deux entités au travers de clés étrangères.
- Exemple :
 - Un musicien joue sur plusieurs instruments
 - Un instrument peut être utilisé par plusieurs musiciens

Les deux entités

@Entity

```
public class Musicien implements Serializable {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
    private Long id;
```

```
    @ManyToMany
```

```
    private Collection<Instrument> instruments ;
```

```
    // reste de la classe
```

```
}
```

@Entity

```
public class Instrument implements Serializable {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
    private Long id;
```

```
    // reste de la classe
```

```
}
```

Modification de la BD

★ Quick access	Nom	Type	Nul
localhost	Index (2)		
<ul style="list-style-type: none"> > booksdb > information_schema > mysql > performance_schema > phpmyadmin > productdb ▼ springdb <ul style="list-style-type: none"> > bateau > commune > comptes > instrument > maire > marin > musicien > musicien_instrument 	<ul style="list-style-type: none"> 🔑 FK9uu6h7349sfh5ubnf7qkc15hs 🔑 FKcragm41tuunlij56mki2jwq7i 	<ul style="list-style-type: none"> instruments_id Musicien_id 	
	Clé secondaire (2)		
	<ul style="list-style-type: none"> 🔗 FK9uu6h7349sfh5ubnf7qkc15hs 🔗 FKcragm41tuunlij56mki2jwq7i 	<ul style="list-style-type: none"> instruments_id -> instrument.id Musicien_id -> musicien.id 	
	Champs (2)		
	<ul style="list-style-type: none"> 📄 Musicien_id 📄 instruments_id 	<ul style="list-style-type: none"> int(11) int(11) 	<ul style="list-style-type: none"> Non Non

Relation N:P cas : bidirectionnel

- Dans le cas bidirectionnel, l'entité cible porte également une relation **@ManyToMany** vers l'entité maître.
- Cette relation doit comporter un attribut **mappedBy**, qui indique le nom de la relation correspondante dans l'entité maître.

```
@Entity
public class Musicien implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToMany
    private Collection<Instrument> instruments ;

    // reste de la classe
}
```

L'entité Instrument

@Entity

```
public class Instrument implements Serializable {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
    private Long id;
```

```
    @ManyToMany(mappedBy="instruments")
```

```
    private Collection<Musicien> musiciens ;
```

```
    // reste de la classe
```

```
}
```

Comportement cascade

- L'Entity Manager permet de mener à bien cinq opérations sur une entité : DETACH, MERGE, PERSIST, REMOVE, REFRESH.
- Le comportement *cascade* consiste à spécifier ce qui se passe pour une entité en relation d'une entité père lorsque cette entité père subit une des opérations définies ci-dessus.
- Exemple :

```
@Entity
public class Commune implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @OneToOne(cascade={CascadeType.PERSIST, CascadeType.REMOVE})
    private Maire maire ;

    // reste de la classe
}
```