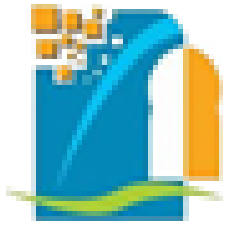

Université Sultan Moulay Slimane
Faculté Des Sciences Et Techniques
- Béni Mellal -



# Les relations entre entités

***Youssef Saadi***

Master Informatique Décisionnelle

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

AU: 2019/2020

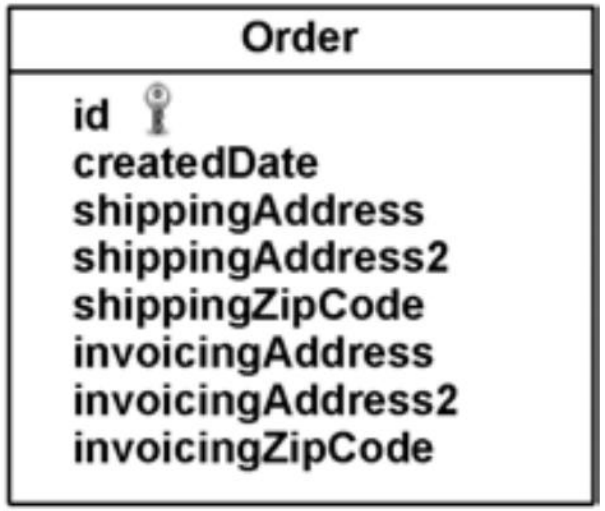
# Objets Inclus : @Embeddable

```
@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="shipppingAddress1")),
    @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

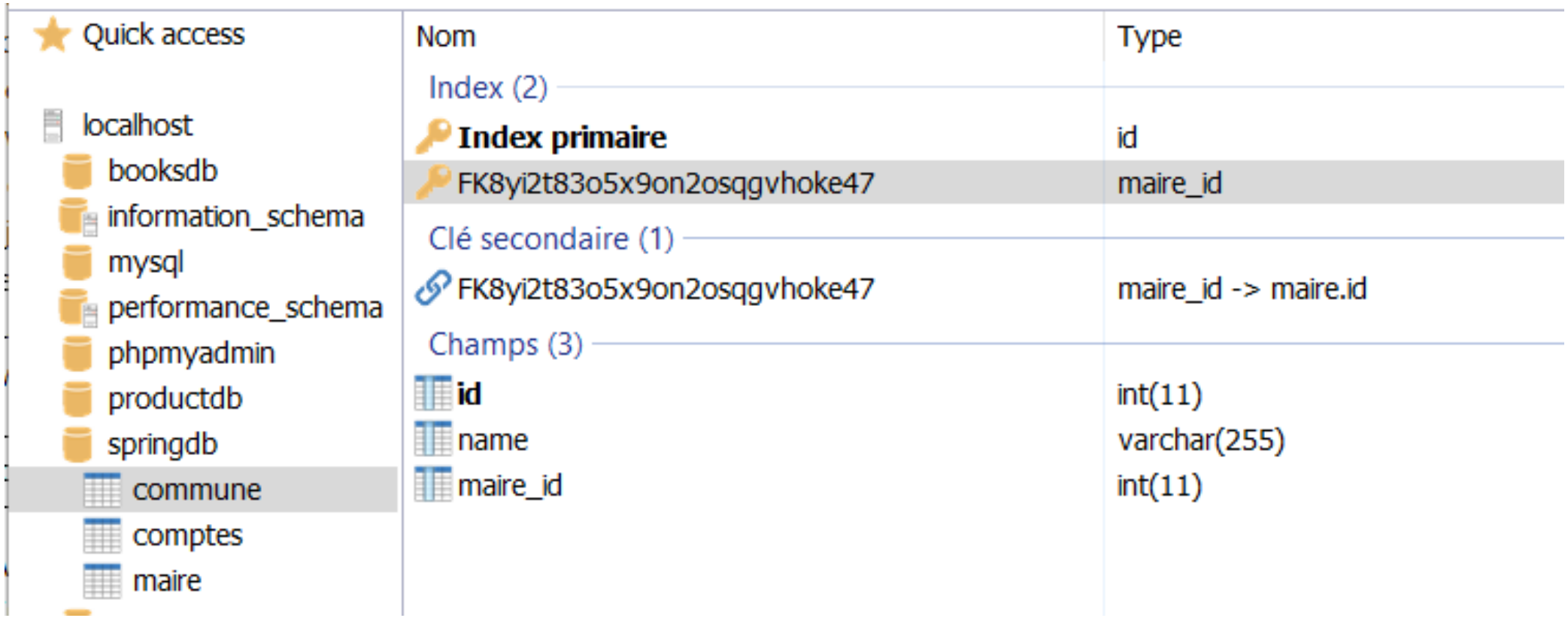
```
@Entity
public  class Maire  implements Serializable {

    @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

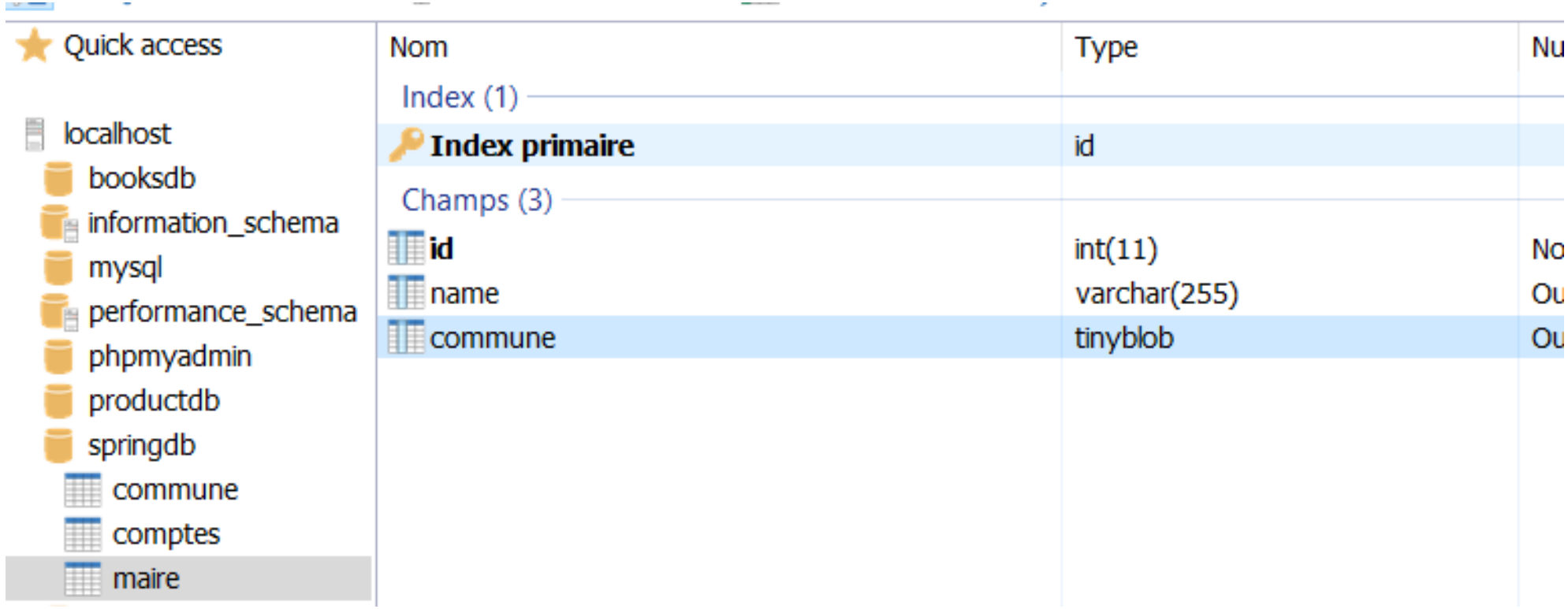
# Modification de la BD



# One to One : relation bidirectionelle

```
@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



# 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.

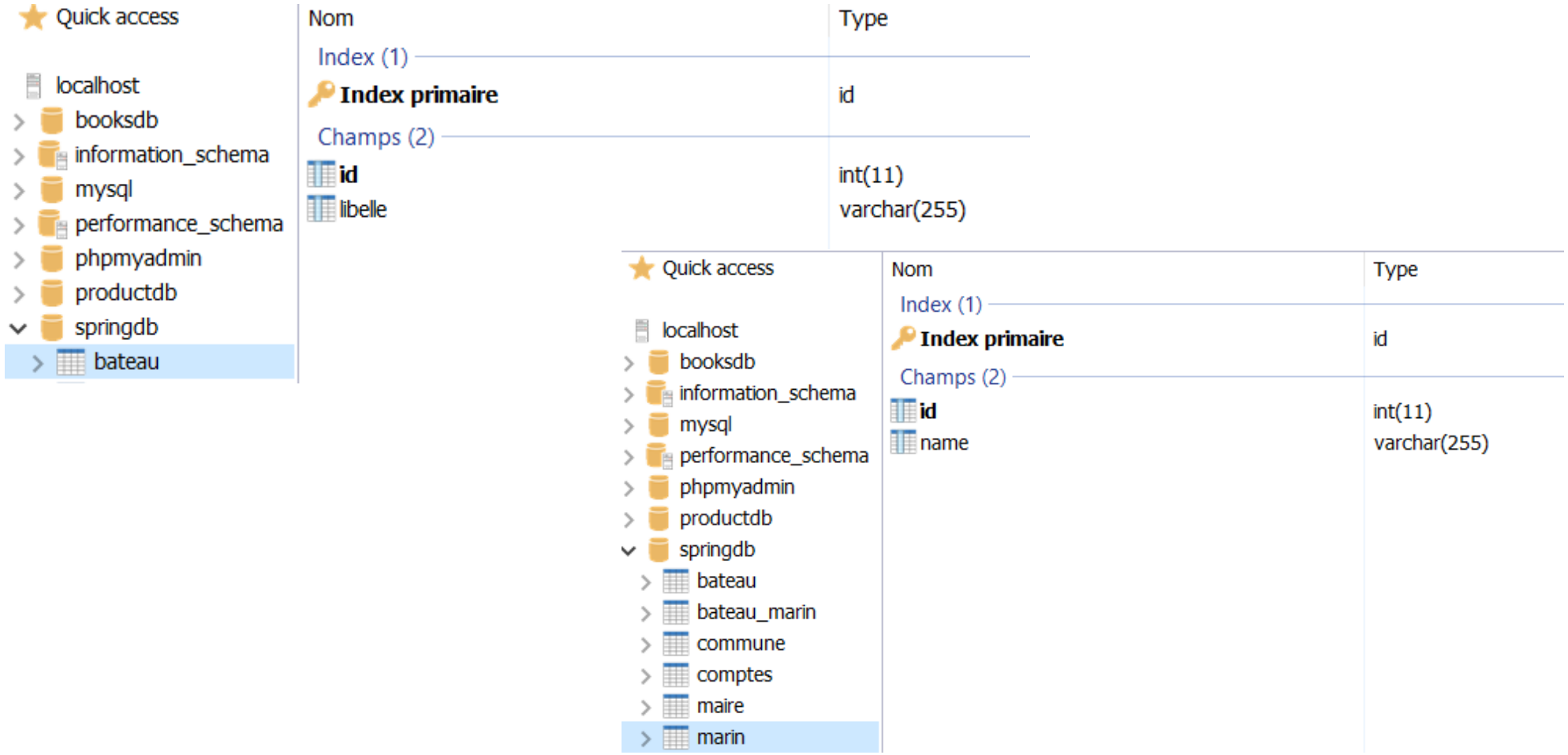
# La classe Marin

```
@Entity
public  class Marin  implements Serializable {

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

    // reste de la classe
}
```

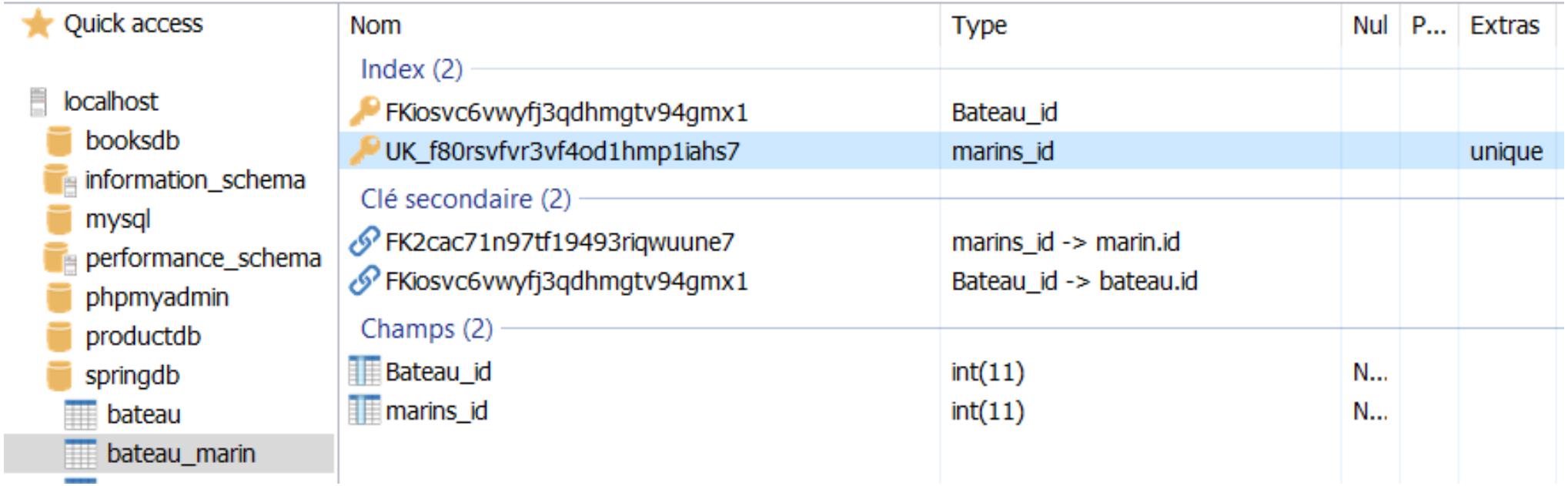
# L'entité Bateau

```java
@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



# La table de jonction



# Relation 1:N  cas : bidirectionnel

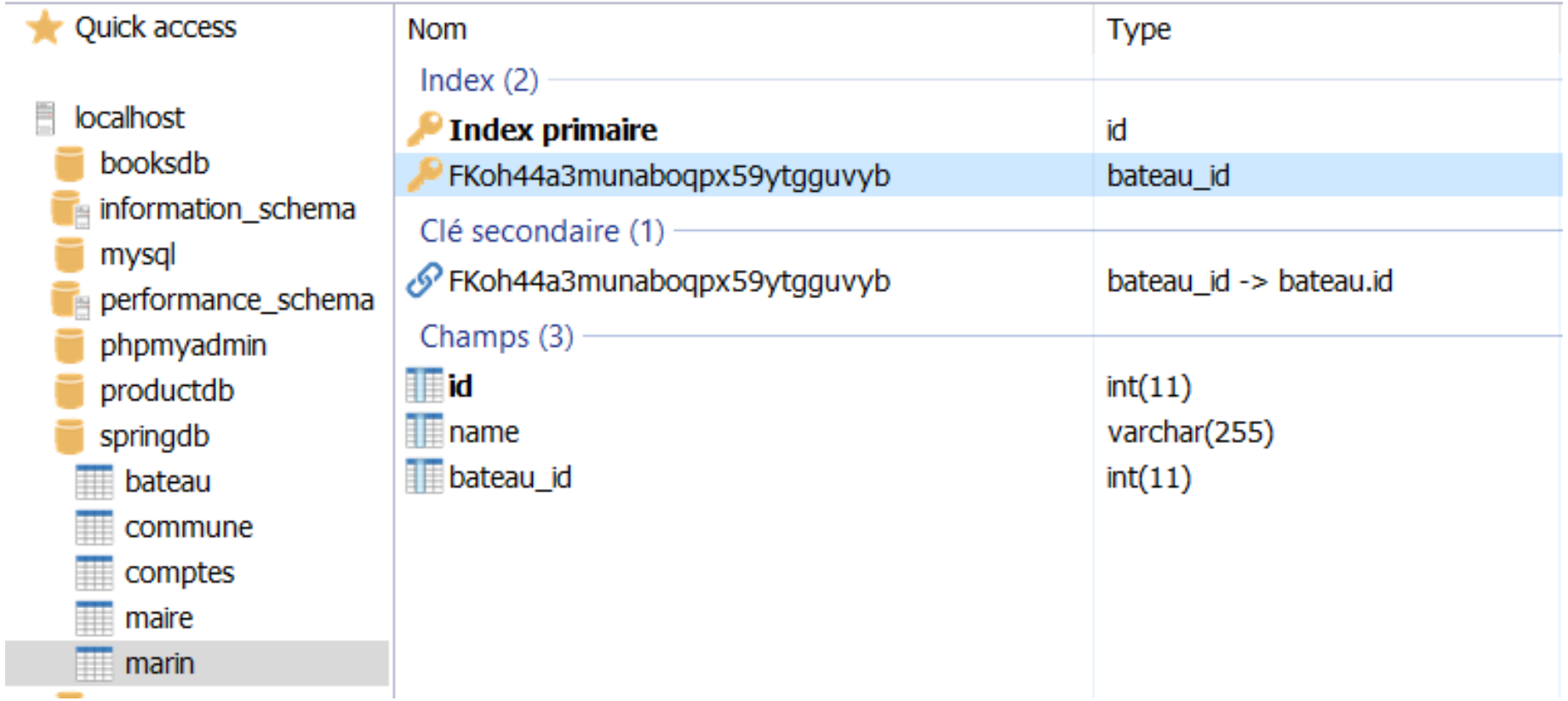
- Une relation 1:p bidirectionnelle doit correspondre à une relation p:1 dans la classe destination de la relation.
- Le caractère bidirectionnel d'une relation 1:p est marqué en définissant l'attribut **mappedBy** sur la relation.
- JPA nous pose une contrainte ici : l'attribut mappedBy est défini pour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

# Les deux entités Marin et Bateau

```java
@Entity
public  class Marin  implements Serializable {

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

    @ManyToOne
    private Bateau bateau ;

    // reste de la classe
}
```

```java
@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



# Relation N:P cas : unidirectionnel

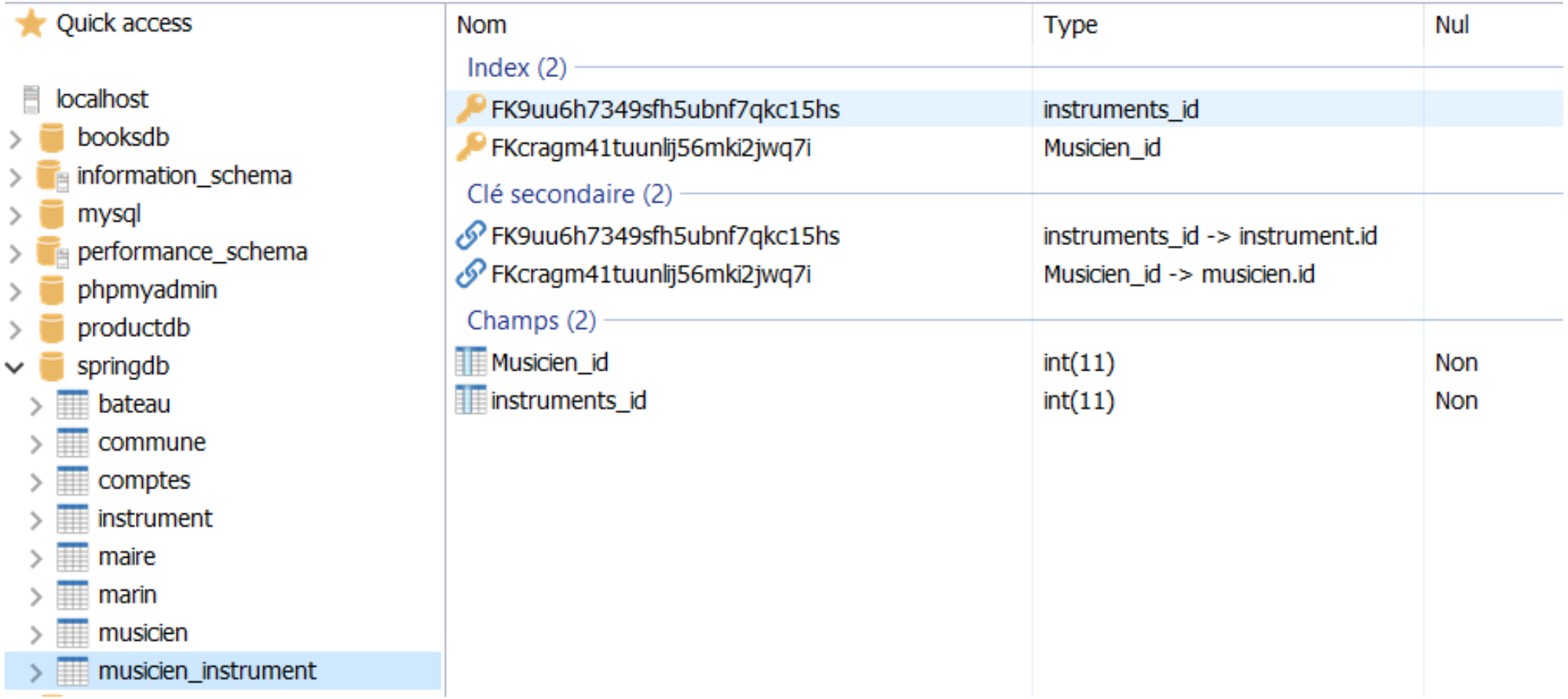
- La façon classique d'enregistrer ce modèle en base consiste à créer une table de jointure, et c'est ce que fait JPA.
- Une table de jointure est créé entre les deux tables qui portent les entités. Cette table référence les deux clés primaires des deux entités au travers de clés étrangères.
- Exemple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

# Les deux entités

```java
@Entity
public  class Musicien  implements Serializable {

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

    @ManyToMany
    private Collection<Instrument> instruments ;

    // reste de la classe
}
```

```java
@Entity
public  class Instrument  implements Serializable {

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

    // reste de la classe
}
```

# Modification de la BD



# 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.

```java
@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
}
```