Spring Data JPA : Associations

I. Table des matières

II. Association one to many

1. Diagramme de classe

Une catégorie peut avoir plusieurs livres.

Un livre peut correspondre à une catégorie.

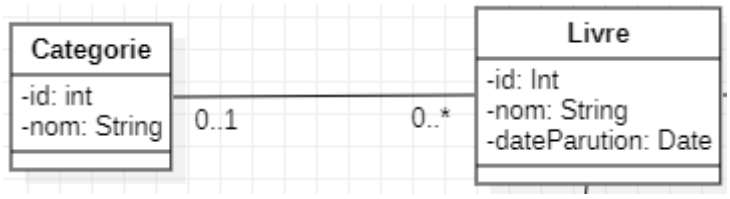


Figure 1 Diagramme de classe

2. Diagramme ER (BDD)

L'entité Livre a une clé étrangère idCategorie qui fait référence à la clé primaire id de l'entité Catégorie.

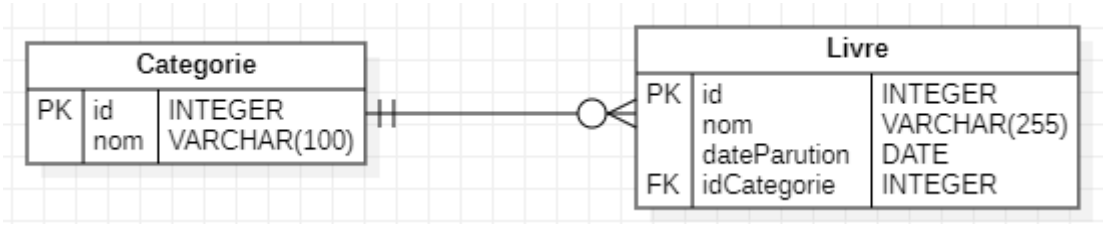


Figure 2 Diagramme ER

3. Classes

a. Classe Livre

La classe Livre contient une référence à la classe Categorie via l'annotation **@ManyToOne**. Cela signifie qu'un livre appartient à une seule catégorie.

La colonne de jointure est spécifiée avec **@JoinColumn(name = "categorie_id")** la classe qui porte l'annotation **@JoinColumn** est la classe propriétaire de la relation c'est-à-dire que quand on sauvegarde un livre on sauvegarde aussi le fait que le livre soit associé à sa catégorie .

```
@Entity
class Livre(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Long? = null,

    var nom: String? = null,

    var dateDeParution: LocalDate? = null,

    @ManyToOne
    @JoinColumn(name = "categorie_id")
    var categorie: Categorie? = null
)
```

b. Classe Categorie

La classe Categorie a une liste de livres avec l'annotation **@OneToMany**. La propriété mappedBy spécifie le nom de la propriété dans la classe Livre qui gère cette relation.

```
@Entity
class Categorie(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Long? = null,

    var nom: String? = null,

    @OneToMany(mappedBy = "categorie")
    var livres: List<Livre>? = null
)
```

III. Association many to many

4. Diagramme de classe

Une personne peut avoir plusieurs rôles.

Un rôle peut correspondre à plusieurs personnes.

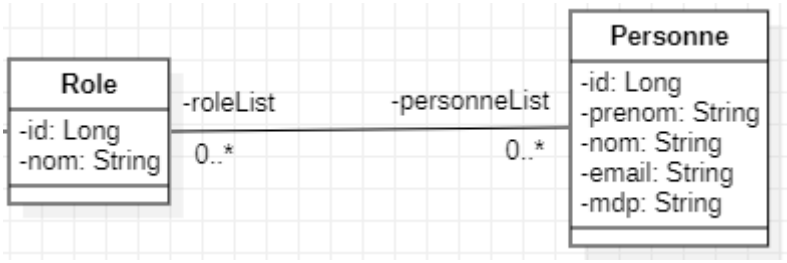
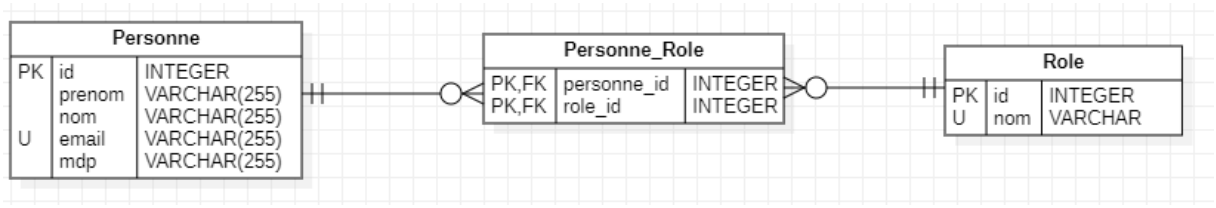


Figure 3 Diagrame de classe

5. Diagramme ER

Une association many to many dans un diagramme de classe est traduite par une table associative dans la BDD.



6. Classes

c. Classe Personne

La classe Personne a une liste de rôles avec l'annotation **@ManyToMany**.

La table associative est spécifiée avec l'annotation **@JoinTable**.

Les colonnes joinColumns et inverseJoinColumns spécifient les colonnes qui serviront de clés étrangères pour lier les tables Personne et Role, cela indique aussi que c'est cette classe (Personne) qui est le propriétaire de la relation.

```
@Entity
class Personne(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Long? = null,

    var nom: String,

    var email: String,

    var mdp: String,

    @ManyToMany
    @JoinTable(
        name = "personne_role",
        joinColumns = [JoinColumn(name = "personne_id")],
        inverseJoinColumns = [JoinColumn(name = "role_id")]
    )
    var roles: List<Role>? = null
)
```

d. Classe Rôle

La classe Role à une liste d'utilisateurs avec l'annotation **@ManyToMany(mappedBy = "roles")**.

Cela spécifie que la relation est déjà gérée du côté de la classe Utilisateur et la colonne correspondante dans la table d'association est spécifiée avec mappedBy.

```
@Entity
class Role(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Long? = null,

    var nom: String,

    @ManyToMany(mappedBy = "roles")
    var personnes: List<Personne>? = null
)
```

IV. Classe associative

7. Diagramme de classe

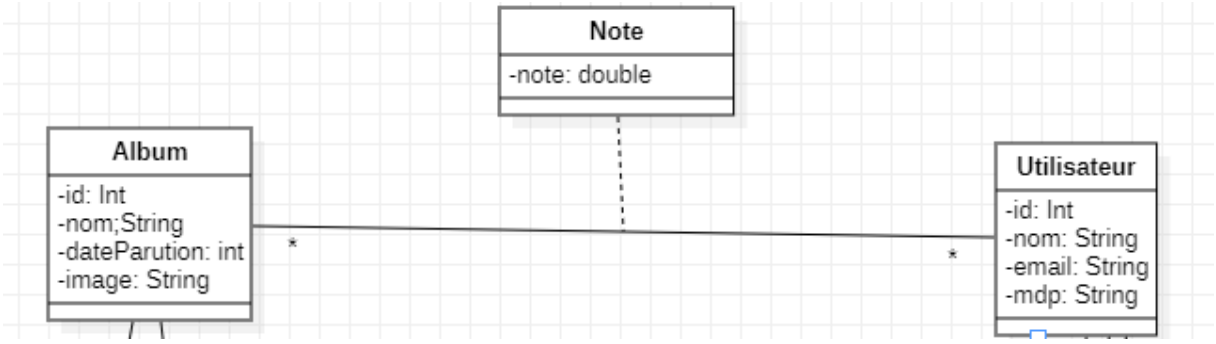


Figure 4 Diagramme classe (théorique)

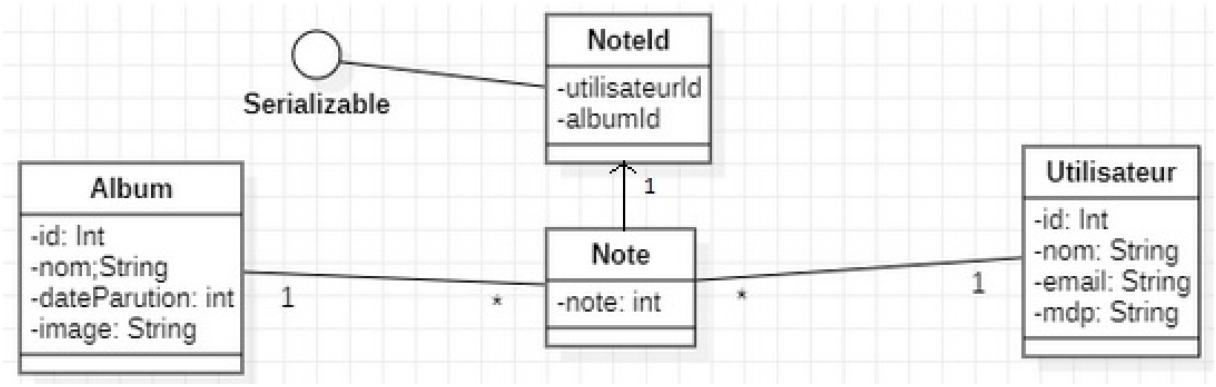
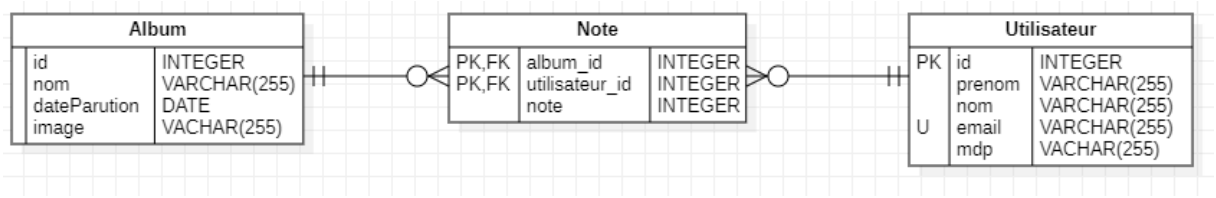


Figure 5 Diagramme de classe pratique

8. Diagramme ER



9. Classes

e. Classe Album

```
@Entity
class Album(
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    val id: Long? = null,

    val nom: String,
    val dateParution: LocalDate,
    val image: String,

    @OneToMany(mappedBy = "album")
    val notes: List<Note> = mutableListOf()
)
```

f. Classe Utilisateur

```
@Entity
class Utilisateur(
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    val id: Long? = null,

    val nom: String,
    val email: String,
    val mdp: String,

    @OneToMany(mappedBy = "utilisateur")
    val notes: List<Note> = mutableListOf()
)
```

g. Classe Note

L'identifiant de la classe Note est composé de deux id (albumId et utilisateurId) on ne peut pas indiquer 2 annotations **@Id** dans une classe on est donc obligé d'utiliser une classe « embarqué » (embeded) spécifiquement pour indiquer l'identifiant de la classe Note.

Chaque composante de l'id (albumId et utilisateurId) et lier à une association manyToOne.

On indique au niveau de **@MapsId** le nom de l'attribut de NoteId qui est lié à l'association.

```
@Entity
class Note(
    @EmbeddedId
    var noteId: NoteId? = null,

    @MapsId("albumId")
    @ManyToOne
    @JoinColumn(name = "album_id")
    var album: Album? = null,

    @MapsId("utilisateurId")
    @ManyToOne
    @JoinColumn(name = "utilisateur_id")
    var utilisateur: Utilisateur? = null,

    val note: Int
) {
    // autres propriétés de la note
}
```

h. Classe NoteId

L'annotation **@Embedable** indique que la classe NoteId peut être « embarqué » (embeded)

Elle doit implémenter l'interface Serializable. Elle comporte les 2 parties de la clé primaire (attributs utilisateurId et albumId).

```
@Embeddable
class NoteId(
    val utilisateurId: Long,
    val albumId: Long
) : Serializable
```