Relations JPA

Les Types de Relations

@OneToMany / @ManyToOne

Cas d'usage : Un auteur a plusieurs livres, un livre a un seul auteur.

Configuration Unidirectionnelle (Rarement recommandée)

```
@Entity
public class Author {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @OneToMany
    @JoinColumn(name = "author_id") // Crée une FK dans Book
    private List<Book> books = new ArrayList<>();
}

@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    // Pas de référence à Author
}
```

Problème : Génère des UPDATE supplémentaires !

Lancés automatiquement par Hibernate pour mettre à jour la clé étrangère.

Configuration Bidirectionnelle (Recommandée)

```
@Entity
public class Author {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @OneToMany(
        mappedBy = "author",          // Référence au champ dans Book
        cascade = CascadeType.ALL,
        orphanRemoval = true,
```

```
        fetch = FetchType.LAZY         // Par défaut
    )
    private List<Book> books = new ArrayList<>();
    // Méthodes helper pour synchroniser les deux côtés
    public void addBook(Book book) {
        books.add(book);
        book.setAuthor(this);
    }
    public void removeBook(Book book) {
        books.remove(book);
        book.setAuthor(null);
    }
}

@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToOne(
        fetch = FetchType.LAZY,       // Recommandé
        optional = false              // NOT NULL en base
    )
    @JoinColumn(name = "author_id")   // Nom de la FK
    private Author author;
}
```

Le côté **@ManyToOne** est TOUJOURS le propriétaire (owner) de la relation.

Toutes les modifications portant sur cette relation devront se faire côté **owner**.

@ManyToMany

Cas d'usage : Un étudiant suit plusieurs cours, un cours a plusieurs étudiants.

Configuration Unidirectionnelle

```
@Entity
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToMany
    @JoinTable(
        name = "student_course",
        joinColumns = @JoinColumn(name = "student_id"),
        inverseJoinColumns = @JoinColumn(name = "course_id")
    )
```

```
    private Set<Course> courses = new HashSet<>();
}

@Entity
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    // Pas de référence à Student
}
```

Configuration Bidirectionnelle (Recommandée)

```
@Entity
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToMany(
        cascade = {CascadeType.PERSIST, CascadeType.MERGE},
        fetch = FetchType.LAZY
    )
    @JoinTable(
        name = "student_course",
        joinColumns = @JoinColumn(name = "student_id"),
        inverseJoinColumns = @JoinColumn(name = "course_id")
    )
    private Set<Course> courses = new HashSet<>();
    public void enrollCourse(Course course) {
        courses.add(course);
        course.getStudents().add(this);
    }
    public void unenrollCourse(Course course) {
        courses.remove(course);
        course.getStudents().remove(this);
    }
}

@Entity
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToMany(mappedBy = "courses")
    private Set<Student> students = new HashSet<>();
}
```

ManyToMany avec attributs

Ce n'est pas vraiment une ManyToMany...

Avec EmbeddedId

```
@Entity
public class Enrollment {
    @EmbeddedId
    private EnrollmentId id;
    @ManyToOne
    @MapsId("studentId")
    private Student student;
    @ManyToOne
    @MapsId("courseId")
    private Course course;
    private LocalDate enrollmentDate;
    private Integer grade;
}

@Embeddable
public class EnrollmentId implements Serializable {
    private Long studentId;
    private Long courseId;
    // equals() et hashCode()
}

@Entity
public class Student {
    @OneToMany(mappedBy = "student", cascade = CascadeType.ALL, orphanRemoval =
true)
    private Set<Enrollment> enrollments = new HashSet<>();
}

@Entity
public class Course {
    @OneToMany(mappedBy = "course", cascade = CascadeType.ALL, orphanRemoval =
true)
    private Set<Enrollment> enrollments = new HashSet<>();
}
```

Sans , mais avec Id supplémentaire

```
@Entity
@Table(
    name = "enrollment",
    indexes = {
        @Index(
            name = "idx_enrollment_pk",
            columnList = "student_id, course_id",
            unique = true
        )
    }
)
```

```
public class Enrollment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;  // ← Clé primaire auto-générée simple
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "student_id", nullable = false)
    private Student student;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "course_id", nullable = false)
    private Course course;
    private Integer grade;
    private LocalDate enrollmentDate;
    // Constructeurs
    // Getters/Setters
    // equals et hashCode
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Enrollment)) return false;
        Enrollment that = (Enrollment) o;
        return id != null && id.equals(that.getId());
    }
    @Override
    public int hashCode() {
        return getClass().hashCode();
    }
    @Override
    public String toString() {
        return "Enrollment{" +
                "id=" + id +
                ", student=" + (student != null ? student.getName() : "null") +
                ", course=" + (course != null ? course.getName() : "null") +
                ", grade=" + grade +
                ", enrollmentDate=" + enrollmentDate +
                '}';
    }
}
```