JPA

Menu

[1) Mapping 1](#_Toc135916528)

* 1. [@**ManyToOne, @ManyToOne, @ManyToMany** 1](#_Toc135916529)
  2. [**Fetch** 1](#_Toc135916530)
  3. [**@JoinColum** 2](#_Toc135916531)
  4. [**@Column** 2](#_Toc135916532)
  5. [**@Lob** 2](#_Toc135916533)
  6. [**@Id** 2](#_Toc135916534)
  7. [**@GeneratedValue(strategy = GenerationType.IDENTITY)** 2](#_Toc135916535)
  8. [**Cascade** 3](#_Toc135916536)
  9. [**@Embeddable** 3](#_Toc135916537)
  10. [**@Embedded** 3](#_Toc135916538)
  11. [**@AttributeOverrides** và **@AttibuteOverride** 3](#_Toc135916539)
  12. [**@EmbeddedId** 4](#_Toc135916540)

# Mapping

One to many

**@ManyToOne(fetch = FetchType.*LAZY*, optional = false)  
@JoinColumn(name = "one\_id", nullable = false)  
private One one;**

## @**ManyToOne, @ManyToOne, @ManyToMany**

Để đánh dấu cho loại liên kết

## **Fetch**

là 1 phương thức bên trong , dùng để đĩnh nghĩa phương thức lấy các phương thức liên quan ( mặc định sẽ là Lazy

fetch = FetchType.LAZY tức là mặc định không lấy ra các đối tượng liên quan nhưng bên trong transaction

**Ưu điểm:** tiết kiệm thời gian và bộ nhớ khi select

**Nhược điểm**: gây ra lỗi LazyInitializationException, khi muốn lấy các đối tượng liên quan phải mở transaction 1 lần nữa để query

fetch = FetchType.EAGER thì khi lấy đối tượng Bảng chính là nó mặc định query luôn các đối tượng Bảng liên quan và lưu vào list của bảng chính

**Ưu điểm:** có thể lấy luôn các đối tượng liên quan, xử lý đơn giản, tiện lợi

**Nhược điểm**: tốn nhiều thời gian và bộ nhớ khi select, dữ liệu lấy ra bị thừa, không cần thiết.

## **@JoinColum**

Chỉ định tên khóa ngoại tham chiếu đến bảng sở hữu thuộc tính ánh xạ.

**mappedBy**

Được dùng để định nghĩa bên tham chiếu bên trong **one to many**

## **@Column**

dùng để đánh dấu column bên trong sẽ có nhiều method hỗ trợ

* + - **Name** : dùng để định nghĩa tên của trường
    - **Unique** : nếu là true thì chỉ có thể là giá trị đọc lập không trùng
    - **Precision** : giới hạn số trước dâu phẩy (kiểu generic(0,0)
    - **Scale** : giới hạn số sau dấu phẩy kiểu dữ liệu (generic(0,0)
    - **Nullable**: nếu là true thì cho phép null (mặc định là true)
    - **Length** : dùng cho kiểu dữ liệu chuỗi (nó sẽ giới hạn kí tự ) vd: length = 50 thì nó sẽ là varchar(50)
    - **columnDefinition** : dùng để định nghĩa các kiễu dữ liệu thường

## **@Lob**

Anotation dùng để tạo các kiểu siêu dữ liệu dữ liệu VD: text, Longlob..

## **@Id**

Dùng để đánh dấu primery key của bảng

## **@GeneratedValue(strategy = GenerationType.IDENTITY)**

Dùng để tạo kiểu số tự tăng

## **Cascade**

| **CASCADE** | **MÔT TẢ** |
| --- | --- |
| ALL | Tương ứng với tất cả các loại cascade. cascade={DETACH, MERGE, PERSIST, REFRESH, REMOVE} |
| DETACH | Nếu đối tượng cha bị detached khỏi persistence context thì các đối tượng tham chiếu tới nó cũng bị detached. |
| MERGE | Nếu đối tượng cha được merged vào persistence context, thì các đối tượng tham chiếu tới nó cũng được merged. |
| PERSIST | Nếu đối tượng cha được persisted vào persistence context, thì các đối tượng tham chiếu tới nó cũng được persisted. |
| REFRESH | Nếu đối tượng cha được refreshed ở persistence context hiện tại, thì các đối tượng tham chiếu tới nó cũng được refreshed. |
| REMOVE | Nếu đối tượng cha bị removed khỏi persistence context, thì các đối tượng tham chiếu tới nó cũng được removed. |

## **@Embeddable**

để khai báo rằng một lớp sẽ được nhúng bởi các entity(Thực thể) khác.

## **@Embedded**

được sử dụng để khai báo rằng nhúng thực thể được đánh dấu @Embeddable ở phía trên vào thực thể ở dưới đây.

## **@AttributeOverrides** và **@AttibuteOverride**

để ghi đè các thuộc tính cột của kiểu embedded của ta.

**@Embedded**

**@AttributeOverrides**({

**@AttributeOverride**( **name** = "firstName", **column** = **@Column**(**name** = "contact\_first\_name")),

**@AttributeOverride**( **name** = "lastName", **column** = **@Column**(**name** = "contact\_last\_name")),

**@AttributeOverride**( **name** = "phone", **column** = **@Column**(**name** = "contact\_phone"))

})

## **@EmbeddedId**

được sử dụng để khai báo thuộc tính primary key cho những đối tượng phức tạp gồm nhiều primary key

## **@MapsId**

The @MapsId annotation in JPA/Hibernate is used to map the primary key of a child entity to the primary key of a parent entity. This is useful when the two entities have a one-to-one relationship.

# Repository

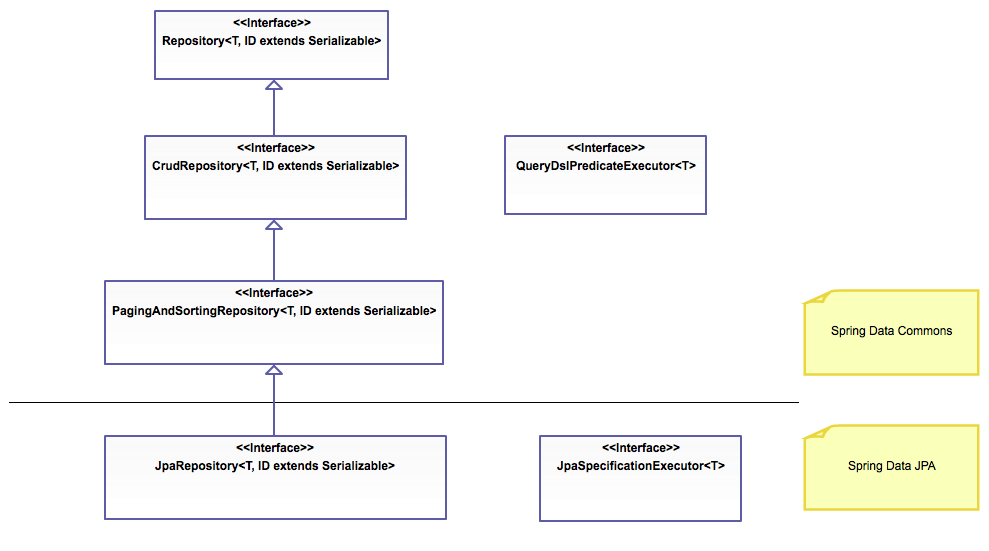
**Lý do:** Việc xây dựng và triển khai tầng truy cập dữ liệu trong các ứng dụng đôi khi gây ra cho lập trình viên khá nhiều phiền toái do sự cồng kềnh và phức tạp của nó

Vì vậy để giảm thiểu ảnh hưởng từ vần đề trên, spring cung cấp cho ta giải pháp đó là các Abstract repository,

Repository là gì ?

* Nó là 1 interface được hỗ trợ bởi spring data jpa để giảm thiểu việc viết code quá nhiều để thao tác với cơ sở dữ liệu

Phần cốt lỗi của spring data abstract repository là interface có tên là repository



Các cú pháp đặc biêt:

* Query method
* Update bằng @Query
* Update bằng native query
  + Cần phải có @Transacional và @ Modifying
  + Transactional dùng để đánh dấu vào tạo 1 transaction cho native query vì nó sẽ không hiểu đây là method gì của interface nên cần phải đánh dấu và tạo 1 transaction
* @Transactional là từ khóa dùng để đánh dấu hiểu đơn giản khi có thay đỗi dữ liệu nếu thất bài thì nó sẽ rollback dữ liệu và bảo toàn dữ liệu cũ

# Keywork

* Có hỗ trợ Querydsl support
* Repository interfaces
* Querydsl support
* Auditing support
* Pagination support
* Transaction management
* Spring Boot integration
* Customization
* Caching
* Batching
* Event sourcing