**CÁC MỐI QUAN HỆ TRONG CƠ SỞ DỮ LIỆU**

**@ManyToOne**

* Đây là một annotation được sử dụng để thiết lập mối quan hệ nhiều-đến-một (many-to-one) giữa hai entity trong mô hình quan hệ cơ sở dữ liệu
* Khi sử dụng **@ManyToOne**, chúng ta đánh dấu một trường trong entity hiện tại là một liên kết đến một entity khác. Điều này thường được ánh xạ vào một khóa ngoại (foreign key) trong cơ sở dữ liệu, để thể hiện mối quan hệ giữa hai bảng trong cơ sở dữ liệu

A screenshot of a computer program

Description automatically generated

**@JoinColumn**

* **@ManyToOne** không bắt buộc phải đi kèm với **@JoinColumn**. Khi sử dụng **@ManyToOne** mà không chỉ định **@JoinColumn**, Hibernate (hoặc JPA provider khác) sẽ tự động tạo ra một cột khóa ngoại với tên mặc định dựa trên tên trường và tên khóa chính của entity liên kết. Thông thường, tên cột khóa ngoại được tạo theo mẫu: "tenTruong\_tenKhoaChinhEntityLienKet"
* Tuy nhiên, chúng ta có thể sử dụng **@JoinColumn** để tùy chỉnh tên cột khóa ngoại hoặc thêm các thuộc tính khác

A picture containing text, font, screenshot, line

Description automatically generated

**@OneToMany**

**@OneToMany** là một annotation được sử dụng để thiết lập mối quan hệ một-nhiều giữa hai entity. Điều này có nghĩa là một instance của entity gốc có thể có nhiều instances của entity liên kết, trong khi mỗi instance của entity liên kết chỉ thuộc về một instance của entity gốc duy nhất

**@OneToOne**

* **@OneToOne** là một annotation được sử dụng để thiết lập một mối quan hệ một-một (one-to-one) giữa hai entity

A screenshot of a computer program

Description automatically generated with low confidence

* Nếu chúng ta không đánh dấu thuộc tính liên kết là duy nhất (*unique* = true), thì quan hệ **@OneToOne** không có sự khác biệt rõ ràng với quan hệ **@ManyToOne**. Điều này là do mặc định của **@OneToOne** là không yêu cầu thuộc tính liên kết là duy nhất. Tuy nhiên, có một số khía cạnh quan trọng cần lưu ý:
  + Từ góc độ khái niệm, quan hệ **@OneToOne** vẫn thể hiện một quan hệ một-một. Dù có hoặc không có thuộc tính liên kết duy nhất, quan hệ vẫn giới hạn mỗi đối tượng chỉ có một đối tượng liên kết tương ứng và ngược lại
  + Mặc dù không yêu cầu thuộc tính liên kết duy nhất, việc sử dụng **@OneToOne** có thể mang lại sự rõ ràng hơn trong thiết kế và giúp mô hình dữ liệu trở nên dễ hiểu hơn. Nếu chúng ta đã định nghĩa rõ ràng quan hệ một-một giữa hai entity, việc sử dụng **@OneToOne** sẽ giúp người đọc mã hiểu rõ hơn về ý định của chúng ta

**@ManyToMany**

* **@ManyToMany** được sử dụng để thiết lập một mối quan hệ nhiều-nhiều giữa hai entity. Điều này có nghĩa là mỗi instance của entity gốc có thể có nhiều instances của entity liên kết, và ngược lại, mỗi instance của entity liên kết cũng có thể thuộc về nhiều instances của entity gốc
* Thông thường, quan hệ **@ManyToMany** được thể hiện bằng việc sử dụng một bảng trung gian (join table) để lưu trữ thông tin về mối quan hệ. Bảng trung gian này chỉ chứa hai cột là các khóa ngoại để liên kết với các bảng gốc, và không có khóa chính riêng cho bảng trung gian

A screenshot of a computer program

Description automatically generated with low confidence

* **@JoinTable**
  + **@JoinTable** được sử dụng để tạo một bảng trung gian (join table) trong quan hệ **@ManyToMany** giữa hai entity và thiết lập mối quan hệ giữa chúng thông qua bảng trung gian đó
  + **@JoinTable** không bắt buộc khi thiết lập mối quan hệ **@ManyToMany**. Khi chúng ta không định nghĩa **@JoinTable**, JPA sẽ tự động tạo một bảng trung gian với các tên và cấu trúc mặc định dựa trên các thông tin của các entity và mối quan hệ

**Thuộc tính cascade và fetch trong @ManyToOne, @OneToMany, @OneToOne và @ManyToMany**

* Thuộc tính *cascade*: Được sử dụng để xác định các hoạt động CRUD (Create, Read, Update, Delete) trên entity chủ (entity gốc) sẽ được chuyển tiếp cho entity liên kết. Nó cho phép tự động áp dụng các thay đổi từ entity gốc cho entity liên kết. Ví dụ: khi xóa một UserEntity, các OrderEntity liên quan cũng sẽ bị xóa theo
* Thuộc tính *fetch*: Xác định cách dữ liệu của entity liên kết sẽ được lấy từ cơ sở dữ liệu khi truy vấn entity gốc. Có hai giá trị phổ biến cho thuộc tính fetch:
  + FetchType.LAZY: Entity liên kết sẽ chỉ được tải từ cơ sở dữ liệu khi nó được truy cập thực sự. Điều này giúp tối ưu hiệu suất truy vấn. Khi sử dụng FetchType.LAZY, một proxy sẽ được tạo ra cho entity liên kết và dữ liệu sẽ được tải khi truy cập vào trường này
  + FetchType.EAGER: Entity liên kết sẽ được tự động tải từ cơ sở dữ liệu ngay khi entity gốc được truy vấn. Điều này có thể dẫn đến việc tải nhiều dữ liệu không cần thiết và làm giảm hiệu suất truy vấn
  + FetchType mặc định:
    - Với annotation **@ManyToOne** và **@OneToOne** thì fetchType mặc định là EAGER
    - Với annotation **@ManyToMany** và **@OneToMany** thì fetchType mặc định là LAZY

**Proxy**

* Khi sử dụng FetchType.LAZY trong mối quan hệ **@ManyToOne** hoặc **@OneToMany**, Hibernate (hoặc JPA provider khác) sẽ tạo ra một proxy object để đại diện cho entity liên kết
* Proxy object là một đối tượng giả mạo (fake object) được tạo ra bởi framework ORM như Hibernate. Nó giúp trì hoãn việc tải dữ liệu từ cơ sở dữ liệu cho đến khi thực sự cần thiết. Khi truy cập vào trường liên kết, proxy object sẽ gửi yêu cầu tải dữ liệu từ cơ sở dữ liệu và trả về dữ liệu thực tế
* Cơ chế proxy hoạt động bằng cách tạo một lớp con kế thừa từ entity liên kết và ghi đè các phương thức truy cập. Khi gọi phương thức truy cập, proxy object sẽ thực hiện việc tải dữ liệu từ cơ sở dữ liệu và truyền nó cho đối tượng gốc
* Ví dụ, khi sử dụng FetchType.LAZY với một trường **@ManyToOne**, Hibernate sẽ tạo ra một proxy object cho entity liên kết. Khi truy cập vào trường liên kết, proxy object sẽ gửi yêu cầu tải dữ liệu từ cơ sở dữ liệu và trả về entity thực tế
* Điều này giúp tối ưu hiệu suất truy vấn bằng cách trì hoãn việc tải dữ liệu không cần thiết. Proxy object chỉ tải dữ liệu khi thực sự cần thiết, giúp giảm tải cho cơ sở dữ liệu và tăng tốc độ truy vấn

**Thuộc tính mappedBy trong @OneToMany, @OneToOne và @ManyToMany**

Thuộc tính *mappedBy* không thể bị bỏ qua khi sử dụng **@OneToMany** và **@OneToOne**. Thuộc tính này xác định trường trong entity liên kết mà mối quan hệ được ánh xạ tới. Nó cho biết rằng trong entity liên kết, trường nào sẽ giữ tham chiếu đến entity gốc. Nếu bỏ qua *mappedBy*, JPA không biết cách ánh xạ mối quan hệ và sẽ gây ra lỗi

A screenshot of a computer program

Description automatically generated with medium confidence

Khi JPA truy cập vào orders trong UserEntity, nó sẽ sử dụng reflection để truy cập vào trường user trong OrderEntity, từ đó truy xuất thông tin liên quan đến mối quan hệ, bao gồm cả **@JoinColumn**(name="user\_id") để lấy giá trị khóa ngoại. Điều này cho phép chúng ta truy cập đến các thuộc tính liên quan và thực hiện các thao tác trên mối quan hệ một-nhiều giữa UserEntity và OrderEntity

**Thiết lập mối quan hệ một chiều giữa các entity**

* Khi thiết lập các mối quan hệ giữa các entity, không nhất thiết phải thiết lập quan hệ hai chiều. Điều này có nghĩa là chúng ta không cần phải sử dụng *mappedBy* để thiết lập quan hệ ngược lại từ entity liên kết với entity gốc
* Trong một mối quan hệ hai chiều, chúng ta có thể sử dụng *mappedBy* để chỉ định tên của thuộc tính trong entity liên kết mà sẽ được ánh xạ tới quan hệ ngược lại. Điều này giúp quan hệ hai chiều trở nên rõ ràng hơn và hỗ trợ việc truy vấn dữ liệu
* Tuy nhiên, nếu chúng ta chỉ quan tâm đến mối quan hệ một chiều và không cần truy vấn dữ liệu từ entity liên kết về entity gốc, chúng ta có thể bỏ qua *mappedBy* và không cần thiết lập quan hệ hai chiều. Trong trường hợp này, chúng ta chỉ cần thiết lập mối quan hệ một chiều từ entity gốc đến entity liên kết bằng cách sử dụng các annotation như **@ManyToOne**, **@OneToMany**, **@OneToOne**, **@ManyToMany** mà không cần *mappedBy*

**Composite key**

Composite key được sử dụng khi bảng trong cơ sở dữ liệu có một khóa chính bao gồm nhiều cột thay vì chỉ một cột duy nhất

**A picture containing text, screenshot, font, line

Description automatically generated**

**A screenshot of a computer program

Description automatically generated with low confidence**

**A picture containing text, screenshot, font, number

Description automatically generated**

**A picture containing text, screenshot, font, number

Description automatically generated**

**Các trường hợp của quan hệ n-n có thể được phân loại**

* Trường hợp 1: Bảng trung gian chỉ cần duy nhất 2 khóa ngoại => **@ManyToMany**
* Trường hợp 2: Bảng trung gian có hai khóa ngoại là tổ hợp khóa chính => Composite key
* Trường hợp 3: Bảng trung gian có hai khóa ngoại không phải là khóa chính composite và cần một khóa chính đại diện. Trong trường hợp này, chúng ta cần sử dụng một entity trung gian để đại diện cho bảng trung gian và thiết lập hai quan hệ 1-n => Tách ra thành hai quan hệ 1-n