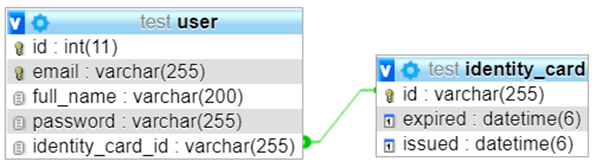
**@OneToOne**

Cách biểu thị quan hệ 1-1 trong cơ sở dữ liệu rất phổ biến. Ví dụ mỗi user chỉ có một chứng minh nhân dân (identity card). Bình thường, khi các bạn tạo table trong csdl để biểu thị mối quan hệ này, thì sẽ có một bảng chứa khóa ngoại của bảng còn lại.



Thể hiện mỗi quan hệ này trong code bằng Hibernate thì chúng ta sẽ dùng annotaton @OneToOne

📃 Class IdentityCard

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Entity

@Table(name="identity\_card")

public class IdentityCard {

@Id

private String id;

@Column(name="expired")

private Date expired;

@Column(name="issued")

private Date issued;

}

📃 Class User

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Entity

@Table(name="user")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int id;

@NotNull

@Column(name="email", unique = true)

private String email;

@Column(name="full\_name")

private String fullName;

@Column(name="password")

private String password;

@OneToOne // Đánh dấu có mỗi quan hệ 1-1 với IdentityCard

@JoinColumn(name = "identity\_card\_id") // Liên kết với nhau qua khóa ngoại identity\_card\_id

private IdentityCard identityCard;

}

Hai cơ chế lấy đối tượng từ databse của Spring JPA:

* fetch = FetchType.LAZY: Khi bạn find, select một đối tượng từ database thì JPA sẽ không lấy ra các các đối tượng phụ thuộc, chỉ lấy ra các thông tin có trong bảng ứng với entity.
* fetch = FetchType.EAGER: Khi bạn find, select một đối tượng từ database thì JPA sẽ lấy ra luôn ra cả các đối tượng liên quan phụ thuộc.

FetchType mặc định của @OneToOne là EAGER

## @ManyToOne vs @OneToMany

Cách biểu thị quan hệ 1-n trong cơ sở dữ liệu là rất phổ biến. Ví dụ một user có thể tạo ra nhiều đơn hàng (order). Bình thường, khi các bạn tạo table trong csdl để biểu thị mối quan hệ này, thì bảng đại diện phía nhiều (phía n trong câu 1-n) sẽ chứa id của bảng tham chiếu (phía 1 trong câu 1-n)



Thể hiện mỗi quan hệ này trong code bằng Hibernate thì chúng ta sẽ dùng annotaton @OneToMany và @ManyToOne

📃 Class Order

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Entity

@Table(name="orders")

public class Order {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int id;

@Column(name="total\_money")

private float totalMoney;

@ManyToOne

@JoinColumn(name="user\_id")

private User user;

}

📃 Class User

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Entity

@Table(name="user")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int id;

@NotNull

@Column(name="email", unique = true)

private String email;

@Column(name="full\_name")

private String fullName;

@Column(name="password")

private String password;

@OneToMany(mappedBy = "user")

private List<Order> orders;

}

FetchType mặc định của @ManyToOne là EAGER, của @OneToMany là LAZY

## @ManyToMany

Cách biểu thị quan hệ n-n trong cơ sở dữ liệu là rất phổ biến. Ví dụ một người có thể viết nhiều cuốn sách và một cuốn sách có thể do nhiều người cùng viết. Bình thường, khi các bạn tạo table trong csdl để biểu thị mối quan hệ này, chúng ta sẽ tạo ra một bảng mới, tham chiếu tới cả bảng này.



Thể hiện mỗi quan hệ này trong code bằng Hibernate thì chúng ta sẽ dùng annotaton @ManyToMany

📃 Class Book

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Entity

@Table(name="book")

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int id;

@Column(name="title")

private String title;

@ManyToMany

@JoinTable(

name = "author",

joinColumns = @JoinColumn(name = "user\_id"),

inverseJoinColumns = @JoinColumn(name = "book\_id")

)

private List<User> authors;

}

Class User

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@Entity

@Table(name="user")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int id;

@NotNull

@Column(name="email", unique = true)

private String email;

@Column(name="full\_name")

private String fullName;

@Column(name="password")

private String password;

@OneToOne

@JoinColumn(name = "identity\_card\_id")

private IdentityCard identityCard;

@ManyToMany(mappedBy = "authors")

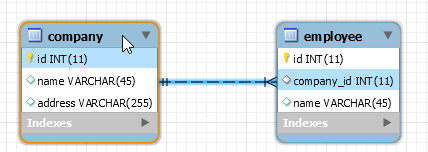
private List<Book> books;

}

### Cascade là gì?

Cascade là một tính năng giúp quản lý trạng thái của các đối tượng trong một mối quan hệ một cách tự động.

Ví dụ ta có mối quan hệ giữa bảng company và bảng employee như sau:



Mối quan hệ giữa company và employee là 1-n (một company chứa nhiều employee).

Khi xóa hoặc update id của một row trong table company sẽ có các trường hợp sau:

* **Trường hợp 1**: chưa có employee nào tham chiếu tới company đó (company đó chưa có employee nào) -> xóa / update bình thường.
* **Trường hợp 2**: đã có employee tham chiếu tới company đó (company đó đã có employee):
  + **2.1**: Mặc định sẽ không có sửa, update vì nó sẽ ảnh hưởng tới các employee đang tham chiếu tới. (ON UPDATE/DELETE = NO ACTION, RESTRICT)
  + **2.2**: set company\_id của các employee đang tham chiếu tới bằng null (ON UPDATE/ON DELETE = SET NULL) (Tạm hiểu là công ty bị xóa, thay đổi thì các nhân viên sẽ không thuộc công ty nào cả)
  + **2.3**: set company\_id của các employee bằng id của company sau khi update (cho trường hợp update) . Và xóa tất cả các employee có company\_id tham chiếu tới company bị xóa (ON UPDATE/ON DELETE = CASCADE)

Cascade chính là tính năng trong trường hợp 2.3. Khi một bản ghi thay đổi thì nó sẽ tự động update các bản ghi đang tham chiếu tới nó.

### Khi nào nên sử dụng cascade

Lưu ý rằng chính việc tự động update các bản ghi đang tham chiếu tới khiến cho hiệu năng bị giảm xuống cho nên tùy trường hợp mà ta áp dụng nó.

Ví dụ theo logic thông thường, khi xóa 1 company thì ta sẽ xóa tất cả các employee đang tham chiếu tới nó trước rồi sau đấy mới xóa company đó chứ không thực hiện xóa company trước rồi sử dụng cascade để xóa các employee đang tham chiếu tới.

Ta sẽ sử dụng cascade trong các trường hợp dữ liệu tham chiếu ít, các dữ liệu tham chiếu chỉ có ý nghĩa khi gắn liền với đối tượng tham chiếu.

Ví dụ một người có nhiều tên (các tên đó chỉ gắn với người đó), thì khi xóa người đó đi thì ta sẽ xóa luôn các tên đó (vì nếu giữ lại nó sẽ không có tác dụng gì cả) -> trường hợp này dùng cascade.

### Sử dụng cascade trong Hibernate, JPA

Các loại cascade trong JPA (khi sử dụng với EntityManagerFactory/EntityManager)

| **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. |