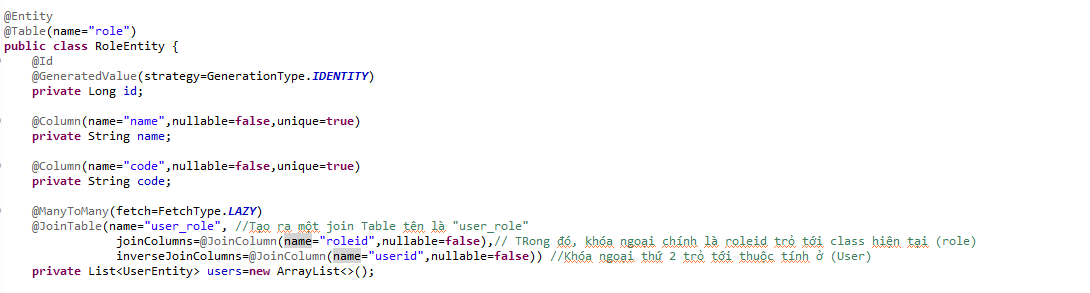
Buổi 16.

Anotation @ManyToMany

Tài liệu: <https://loda.me/articles/jpahuong-dan-manytomany>

Giờ ta có 2 bảng là role và user có quan hệ N-N

ở bảng role



ở bảng user



Thế tại sao bên user lại cài đặt ngắn hơn ?

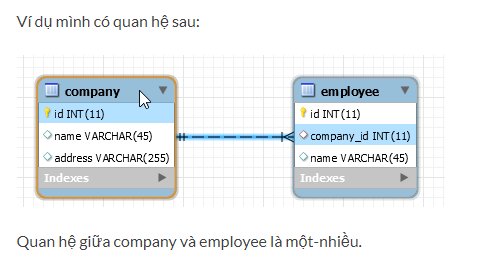
Bới vì cta đã cài đặt hết bên bảng role rồi ,và chúng ta có thể đổi ngược lại là cài đặt bên user

-----------------------------

Hibernate FetchType ?

Tài liệu : <https://stackjava.com/hibernate/hibernate-fetchtype-la-gi-phan-biet-fetchtype-lazy-voi-eager.html>

-Trong Hibernate, Fetchtype là một thuộc tính trong các anotation @OneToOne,@OneToMany,@ManyToMany ,được dùng để định nghĩa phương thức lấy các đối tượng liên quan.



Khi mapping sang class với Hibernate sẽ như sau:

* @Entity
* @Table(name = "company")
* public class Company {
* //...
* @OneToMany(fetch = FetchType.LAZY, mappedBy = "company")
* private List<Employee> listEmployee = new ArrayList<>();
* }

Annotation @OneToMany định nghĩa quan hệ giữa company và employee là 1-n. Trong đó có thuộc tính fetch:

+fetch = FetchType.LAZY tức là khi bạn select đối tượng company từ DB thì nó sẽ không lấy các đối tượng Employee liên quan trừ khi được gọi bởi hàm getListEmployee thì mới lấy nhé,còn không là nó sẽ không lưu đâu.



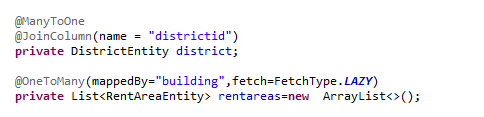
+fetch = FetchType.EAGER tức là khi bạn select đối tượng company từ DB thì nó sẽ lấy tất cả các đối tượng Employee liên quan sẽ được lưu và listEmployee, mà không chỉ mình thằng Employee đâu mà tất cả những bảng liên quan tới company cũng sẽ được lấy

Lưu ý:

* fetch = FetchType.LAZY tức là mặc định không lấy ra các đối tượng liên quan nhưng bên trong transaction, **bạn gọi method company.getListEmployee() thì nó vẫn có dữ liệu nhé, bởi vì khi bạn gọi method nó sẽ query các đối** tượng Employee liên quan và lưu vào listEmployee, và khi kết thúc transaction listEmployee sẽ chứa các employee liên quan. Tuy nhiên nếu bạn không gọi method đó thì listEmployee không có dữ liệu và khi kết thúc transaction listEmployee sẽ không có đối tượng employee nào
* fetch = FetchType.EAGER thì khi lấy đối tượng Company là nó mặc định query luôn các đối tượng Employee liên quan và lưu vào listEmployee, do đó khi kết thúc transaction, listEmployee sẽ có chứa các đối tượng Employee của Company đó.

-Với annotation @ManyToOne và @OneToOne thì fetchType mặc định là EAGER

-Với annotation @ManyToMany và @OneToMany thì fetchType mặc định là LAZY



Nhìn trên kia là 1 building chỉ nằm trên 1 district là quan hệ N-1 nên ta có thể để fetchtype nó là eager cũng không ảnh hưởng nhiều và ta không cần set cho nó là eager vì defaut của thằng ManyToOne là eager .

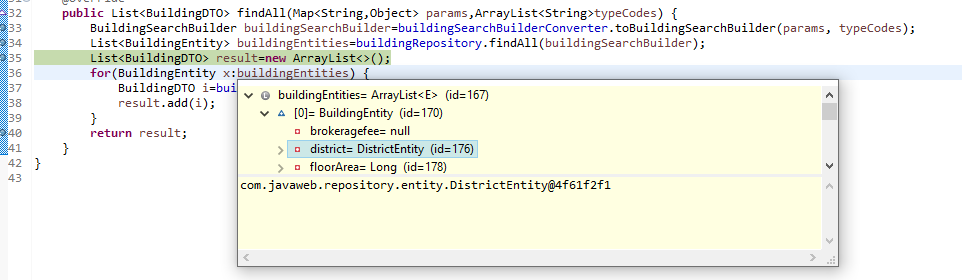
Lấy ví dụ thực tế với project:

Đây là tôi đang đánh debug ở tầng sevice

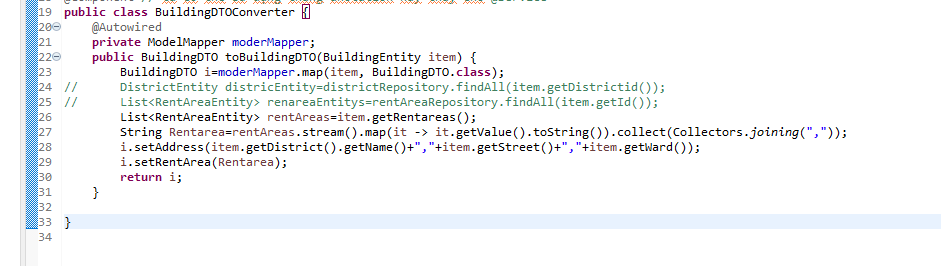
Bảng building: ở đây eager



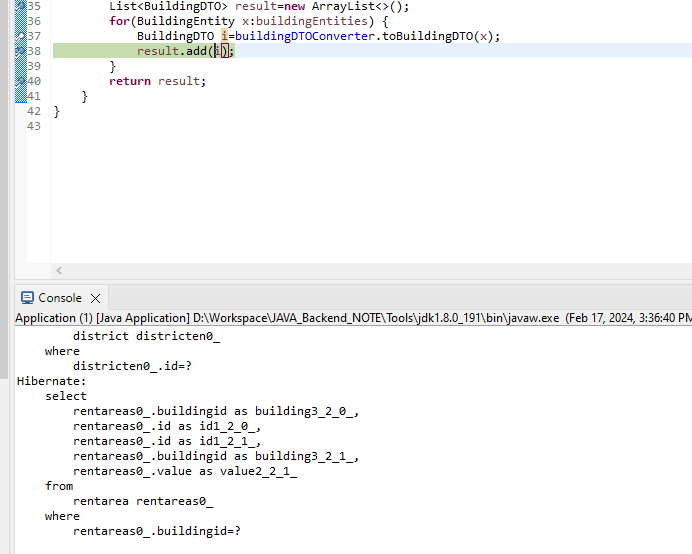
Nhìn ở đây khi mà ta chưa gọi hàm getlist thì nó đã lấy sẵn 1 district entity rồi



Đây là hàm buildingDTOConverter



ở dòng 27 nó mới bắt đầu gọi List rentarea thì đồng thời code của hibernate lúc này sẽ sinh ra và lấy dữ liệu



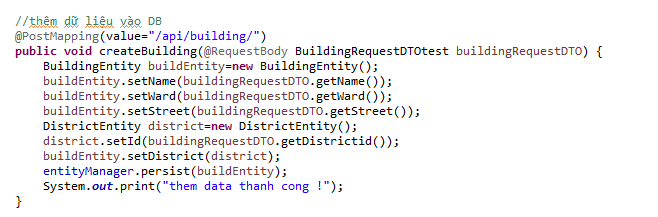
Tóm lại cứ làm đúng theo 2 điều sau:  
-Với annotation @ManyToOne và @OneToOne thì fetchType mặc định là EAGER

-Với annotation @ManyToMany và @OneToMany thì fetchType mặc định là LAZY

-----------------------------------------------

Các hàm persist,remove,merger,find trong JPA

Persist : dùng để thêm Data (insert)



Trên là đang ở tầng controller để test nhưng nếu lúc làm thì cứ làm theo mô hình 3 MVC

Đầu tiền kết nối cơ sở DL dùng EntityManager vì cập nhật thì chúng ta sẽ dùng Post cho đúng với rule và cập nhập và sửa thì client thường gửi về dạng body nên tôi sẽ tạo 1 DTO hứng data cùng nhìn từ dòng 59->61 tại sao phải làm vậy , bởi trong Thằng BuildingEntity thằng district nó là cả 1 cái DistrictEntiy vậy nên phải làm như vậy nhưng lưu ý là khi thêm data cho thằng này phải tồn tại sẵn trong bảng district nhé.

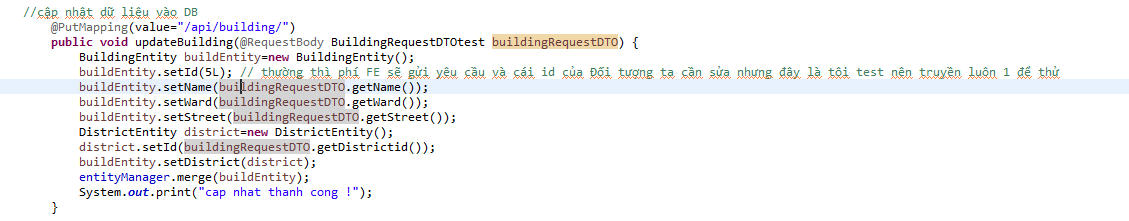
Nhớ thêm một anotation @Transactional

Tiếp theo chúng ta sẽ sang hàm Merger (update) và để sử dụng thằng này thì ta sẽ phải chỉ rõ ra id của thằng mà ta muốn update,tại sao phải vậy nếu không chỉ rõ thì đó là thêm mới rồi, update là nó phải tồn tại sẵn trong DB của chúng ta rồi

Giờ chúng ta sẽ cùng phần tích sự giống nhau giữa việc creat và update DATA :

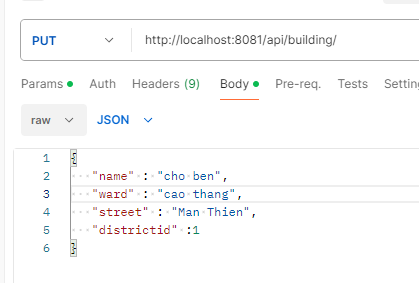
Ví dụ ta thêm data cho 1 Building thì có phải ta sẽ tạo ra một đối tượng thuộc BuildingEntity rồi sau đó set dữ liệu từ phía Client gửi về cho cái đối tượng ta vừa tạo rồi dùng hàm persit thuộc entity manager để thực hiện create.

Vậy việc update 1 building sẽ diễn ra như sau đầu tiên là các yêu cầu từ phía client và kèm theo 1 id cụ thể của building mà ta muốn update. Chúng ta cũng sẽ tạo một entityBuilding và lấy data từ dưới DB của thằng id được gửi về (dùng model maper đối với các field đơn giản) tại sao lại phải làm bước này ư ? bởi nhỡ đâu phía client chỉ yêu cầu đổi nguyên tên chẳng hạn ,rồi sau đó ta sẽ set các data cho đối tượng ta vừa tạo rồi dùng hàm merger của EntityManager

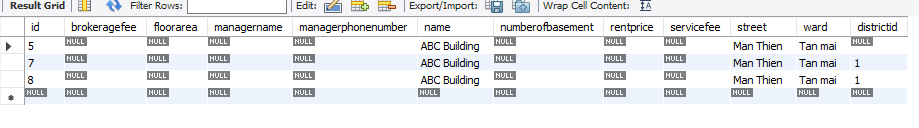


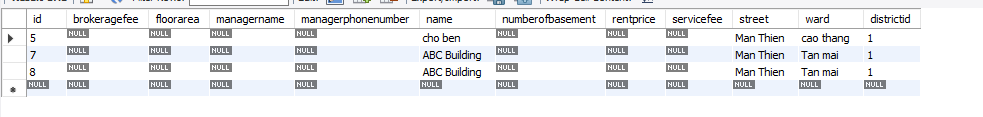
-Bạn nhìn thấy chả khác mấy khi ta thêm data đko ? ,

-Data gửi về :



Trước khi cập nhật

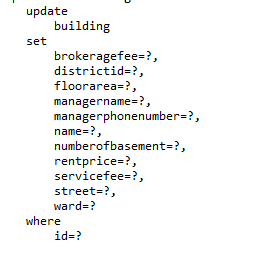


Sau: 

Thằng building có id là 5 đã đc update

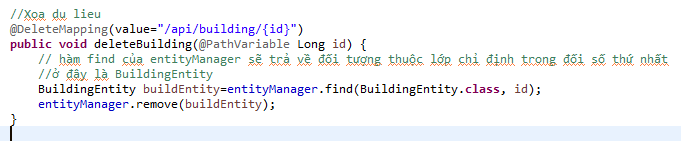
Ví dụ trên tôi test thôi nên chưa dùng model mapper =))

Code hibernate sinh ra:



Xóa Data với hàm Delete:

Xóa thì chúng ta sẽ gửi nguyên id của thằng mà ta muốn xóa , và đường nhiên các gửi nguyên id sẽ qua params và dung @PathVariable để hứng.



Lưu ý khi xóa ở đây cũng giống như trong DB cũng vậy , khi trong 1 quan hệ N-1 bên N sẽ nắm giữ khóa ngoại là khóa chính của bên 1 vậy nên khi xóa thì phải xóa bên nhiều N trước rồi xóa bên 1 nếu không nó sẽ vi phạm tính toàn vẹn và không xóa , trừ trường hợp bên One không có Data thì xóa bên nào trước cũng được , ví dụ building và district có quan hệ N-1 và building có quan hệ 1 Nhiều với rentarea và giờ tôi muốn xóa district thì tôi xóa thằng building trước ư ? đúng vậy nhưng trước cả khi xóa building thì ta phải xóa cả rentarea trước nữa.