Buổi 14

ORM ?

<https://www.tutorialspoint.com/jpa/jpa_architecture.htm>

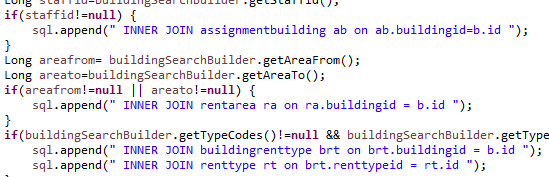
tài liệu sau sẽ hướng dẫn build JPA với java in none-Boot

https://www.baeldung.com/the-persistence-layer-with-spring-and-jpa

-ORM là một kỹ thuật/cơ chế lập trình thực hiện ánh xạ sdl sang đối tượng trong các ngôn ngũ lập trình hướng đối tượng như java,C# (các table tương ứng với các class) , ORM là của hibernate chứ không phải của java spring nhé.

Việc sử dụng ORM cho phép lập trình viên thao tác với database 1 cách hoàn toàn tự nhiên, dễ hiểu thông qua các đối tượng. Lập trình viên không cần tới loại database, kiểu dữ liệu trong database…

-Việc thao tác với ORM cho phép chúng ta giảm sự phụ thuộc vào các hệ quản trị cơ sở dữ liệu ,bởi trước đây dùng JDBC thì chúng ta sẽ cần xem đó là loại nào , Oracal ,mysql ,ví dụ hệ thống của chúng ta đang dùng MYSQL khách lại bảo ,muốn chuyển sang Oracal thì lệnh truy vấn khác => phức tạp



-JDBC:tự Build 100% dài và rườm rà dễ gặp bug,phụ thuộc vào DB => ra đời JPA(ORM)

-khi sử dụng ORM thì chúng ta chỉ thao tác với đối tượng thôi.

JDBC(100%) -> JPA(60%) -> Spring Data JPA (Spring Framework) (30%)

**Quá trình chuyển đổi trạng thái của entity trong JPA và Hibernate**

**Reference: https://shareprogramming.net/qua-trinh-chuyen-doi-trang-thai-cua-entity-trong-jpa-va-hibernate/**

|  |  |
| --- | --- |
| JDBC | JPA |
| Load Driver và Khởi tạo Connection | -Entity ManagerFactory |
| Thực thi câu querry  -Statement  +Excute Querry  +ExcuteUpdate | @PersistenceContext  -EntityManager:  .persist  .merger  .remove  .find |
| ResultSet | Query  -> getResultList  ->getSingleResult |

-Hibernate giúp các dev giảm bớt các câu lệnh SQL thay vào đó họ chỉ quan tâm đến quá trình chuyển đổi trạng thái của thực thể (tương ứng với một dòng dữ liệu dưới database-entity) trong java. Khi một entity được Hibernate quản lý, tất cả các thay đổi đó sẽ tự động cập nhật xuống BD thông qua Hibernate

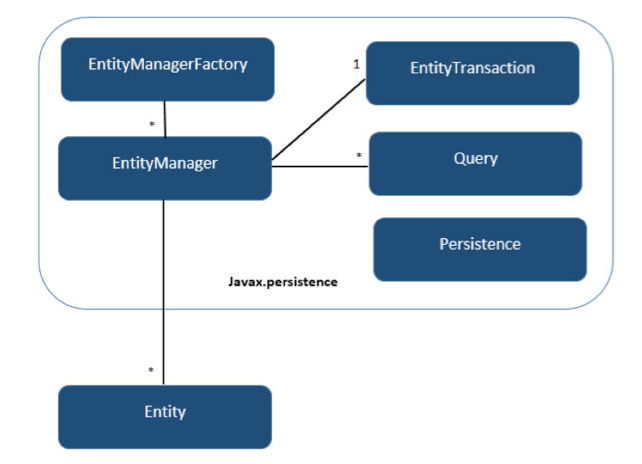
-Chúng ta thấy rằng việc thao tác trên các Entity-java object dễ hơn rất nhiều so với việc chúng ta phải viết và bảo trì các câu lệnh SQL.

Ví dụ: khi xóa một dòng dữ liệu bằng SQL thì chúng ta viết thêm các câu lệnh SQL để xóa các entity đang tham chiếu đến dòng bị xóa.Hibernate thực thi cơ chế này một cách tự động khi một entity bị xóa thông qua một số cấu hình được hibernate cung cấp

-Để sử dụng được EntityManage thì bắt buộc chúng ta phải khai báo @PersistenceContext

-The default persistence context type is PersistenceContextType.TRANSACTION

-Ta có thể thấy cái tính TRANSACTION trong SQL thì đó chính là khái niệm ở trong SQL và ở đây cũng vậy,chỉ một số ít trường hợp khác mà ta không dùng đến thằng này



-Nhìn hình ảnh trên ta có thể thấy ban đầu kết nối CSDL,load Driver bằng EintityManagerFactory -> thực thi câu Querry bằng EntityManager trong bước này đảm bảo tính Transaction và đối tượng Querry quyết định trả về một list hay một single entity,nếu là list thì sử dụng .getResultLitst nếu là đơn thì sử dung .getSingleResult

Có 2 cơ chế làm việc với JPA

+JPQL (JPA Query Language) : làm việc 100% với Entity

+Sql Native: sử dụng ngôn ngữ của JPQ (gần giống sql) để làm việc

Config:

Thêm vào file porm:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

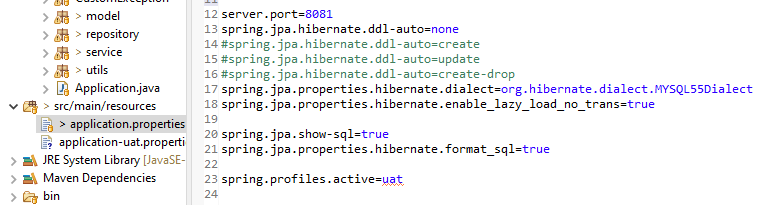
**Tìm hiểu về phân tách dữ môi trường**

Domain:hay còn gọi là tên miền là địa chỉ trang web.

Sau này khi đi làm , 1 Trang web chúng ta sẽ mua nhiều hơn 1 domain , thường là 3 cái , 1 cái là sau khi phát triển và hoàn thiện xong thì giao cho khách hàng sử dụng thường gọi là PRO, còn 1 domain khác sẽ dùng cho lập trình viên để phát triển tính năng gọi là DEV, còn cái còn lại là UAT nghĩa là sau khi phát triển xong thì sẽ đẩy lên doamain này cho tester check nếu mà gặp vấn đề thì sẽ quay lại môi trường DEV để fix rồi lại cập nhật lên UAT.

-Phân tách môi trường trong spring boot ta sẽ sử dụng @SpringProfile

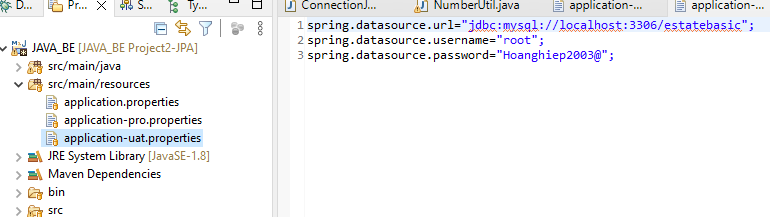
-Config:



ở dòng 23 tên môi trường chúng ta sẽ tạo là uat

Tên file mà ta phân tách sang có cú pháp **profile-[specific].properties**

**ở ví dụ trên sẽ là profile-uat.properties điều này là bắt buộc**

****

Cùng đặt ra câu hỏi tại sao mà khi ta khai báo các cái dữ liệu trong file application.properties nằm trong folder src/main/resources mà thằng mà dự án spring-boot nó đọc và lấy dữ liệu ?

Thật ra là không chỉ riêng mình file application.properties thì bản chất folder src/main/resources sẽ là nơi để config, cùng lấy ví dụ khi ta sử dụng JPA để kết nối csdl bằng @PersistenceContext ---EntityManager nó sẽ đọc ở file application.properties để lấy các thông tin mà csdl ta khai báo như ảnh trên để thực hiện connect ,1 trong những cách có thể mà Spring-boot thực hiện:

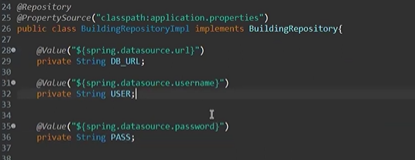
Khai báo anotation này

@PropertySource(“classpath:name-file”)

Trong class mà ta muốn xử lí sẽ có câu lênh sau để lấy value:

@Value(“$(spring.datasource.name-varlible)”)

Áp dụng :



Nhưng trên chỉ là để tham khảo để hiểu hơn về luồng của spring-boot.

Lý thuyết số 02:

1. JDBC có nhược điểm gì mà phải ra đời JPA

**Nhược điểm chính**

**Nhược điểm phụ**

2. ORM giải quyết bài toán nào?

1,

Nhược điểm chính: Phụ thuộc vào DB,khi thay đổi hệ quản trị cơ sở dữ liệu khác lại phải cấu hình lại từ đầu.

Nhược điểm phụ: câu lệnh rườm rà dễ gặp bug

2,

ORM giải quyết vấn đề :

+thay vì thao tác với DB ta sẽ thao tác với các Entity

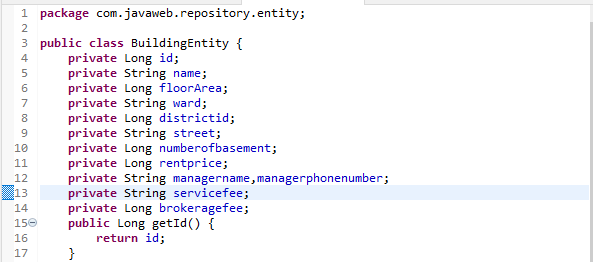
+giảm sự phụ thuộc vào DBMS, nghĩa là dù cho đó có là DBMS nào thì cũng không quan trọng vì ta thao tác với Entity

+khắc phục được các nhược điểm của JDBC

Buổi 15

Phần tách môi trường rồi, giờ cùng quay lại JPA

-Như trước đây chúng ta thường sử dụng Entity để hứng dữ liệu từ phía DB nhưng Entity đó chỉ là tên gọi mà chúng ta quy ước hay coi nó là 1 Entity



Nhưng giờ ta sử dụng thằng JPA là thao tác trên chính các Entity đó nhưng phải thêm một vài thứ như sau:

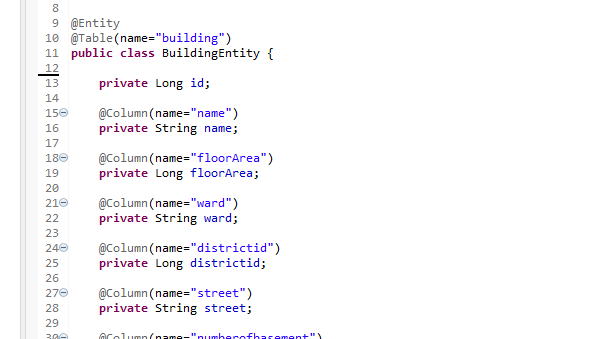
Nhưng mấy thằng anotation này là của JPA nhé

@Entity :thằng này cho biết đây là 1 Entity

@Table(name=”Tên table ánh xạ”)

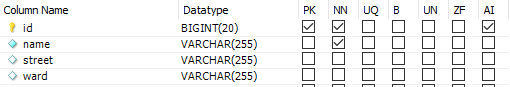
@Column(name=”tên của cột trong DB hay còn gọi là các field)

-lưu ý là name và kiểu dữ liệu của biến hứng dữ liệu phải khớp với name của column trong DB



Có phải là bạn đang thắc mắc là tại sao thằng id nó lại chưa được map đúng không ?

Bởi nó là khóa chính và not null và được tăng tự động nên nó sẽ có cách map riêng



Để map được ta sẽ sử dụng những thứ như sau :

@Id : cho biêt đây là khóa chính

@GeneratedValue(strategy=GenerationType.***IDENTITY***) : tự động tăng

Giờ chúng ta đã Map Entity với bảng trong DB.

Tiếp theo chúng ta sẽ biểu diễn mối quan hệ giữa các Entity với nhau.

Ta cùng xét hai Entity đó là Building và District có quan hệ N-1 với nhau:

Bên Nhiều:

@Entity

public class BuildingEntity{

@ManyToOne

@JoinColumn(name = “districtid”)

private DistrictEntity district;

…

}

Bên nhiều ta sẽ để 2 anotation:

@ManyToOne : cho biết thằng này là bên N

@JoinColumn(name = “fk”) :khóa ngoại

Và nó sẽ trả về 1 Entity District ( 1 tòa nhà nằm trên 1 quận)

Bên one:

@Entity

public class DistrictEntity {

@OneToMany(mappedBy=”district”,fetch=FetchType.LAZY)

private List<BuildingEntity> buildingentitys = new ArrayList<>();

…

}

Bên 1 ta sẽ để 2 anotation:

@OneToMany : cho biết thằng này là bên N

@JoinColumn(name = “fk”) :khóa ngoại

Và nó sẽ trả về 1 List Entity Bulding ( 1 quận có nhiều tòa nhà )

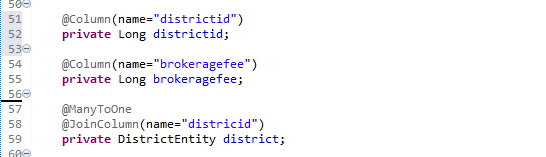
Lưu ý:



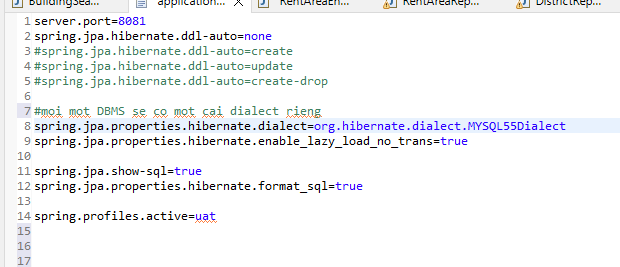
Bên one có đoạn mappedBy là district thì bên N khi hứng dữ liệu cũng phải là district



Lưu ý: dòng 57 khi mà ta đã thực hiện Join bơi buildingid thì lúc này nó sẽ tự động sinh cột id cho chúng ta nên việc lấy có dòng 51 là không cần.



Tiếp theo chúng ta cùng quay lại với file cấu hình Application.properties



Bên trên sẽ có vài cái tôi tìm hiểu như sau:

+Khi map với DB ta sẽ có thể chọn 1 trong những lựa chọn sau:

spring.jpa.hibernate.ddl-auto=none :chỉ map thôi ngoài ra không làm gì cả.

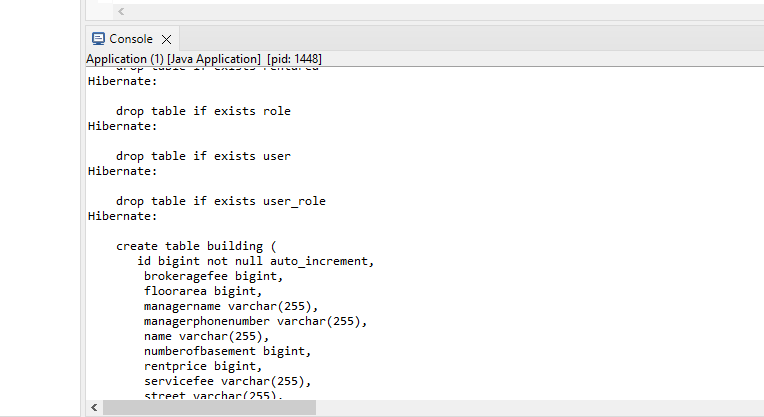
spring.jpa.hibernate.ddl-auto=create

…

Ngoài none ra thì dùng những thằng khác có thể gây thay đổi dữ liệu, ví dụ đơn giản như khi chúng thao tác với các entity và tạo ra 1 DB có tên là “Subject” với câu lệnh create mà trong máy của chúng ta đã tồn tại một DB có tên là Subject rồi thì nó sẽ cập nhật và thay đổi db trong máy của ta.

+Dòng số 8 là dialect thì mỗi một DBMS sẽ có một cái dialect :hiểu đơn giản sẽ là do cái thằng jpa nó không phụ thuộc vào DBMS nên khi nó config rồi sau đó sinh code ấy thì nó sẽ có cái dialect để nó xem của DBMS nào để nó sinh code tương ứng với cái DBMS đó

+spring.jpa.show-sql=true thì nó sẽ show cái code mà Hibernate nó sinh ra cho chúng ta tại cửa sổ console (bằng false là k show)



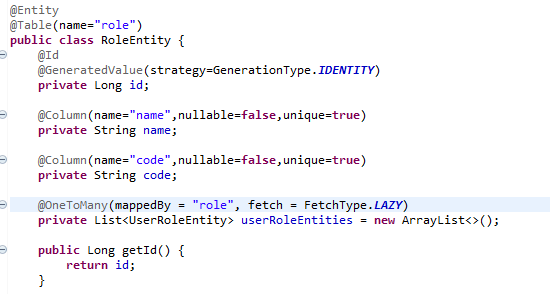
+spring.profiles.active=môi trường mà ta muốn thao tác

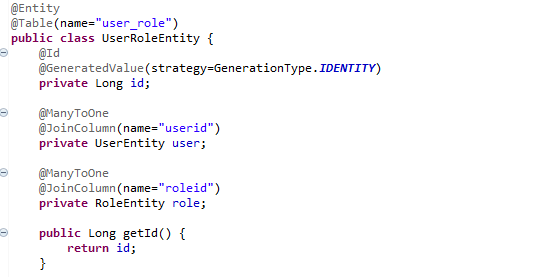
Bên trên ta đã tìm hiểu về cách biểu diễn quan hệ 1\_N và N\_1

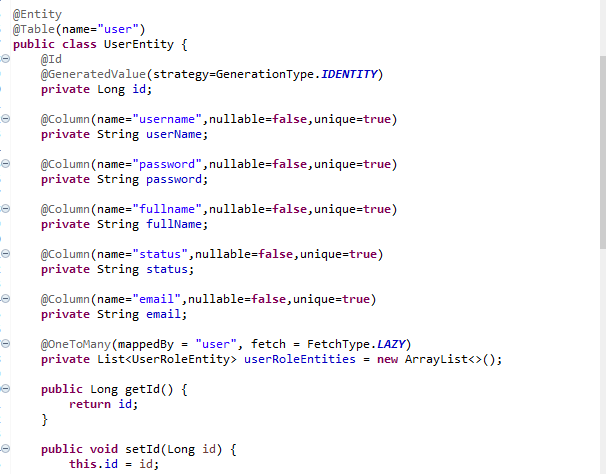
Giờ ta sẽ tìm hiểu về quan hệ N-N và build chay nhé.

- giờ ta sẽ có 2 Bảng là User và Role có mối quan hệ N-N, để thể hiện nó theo Entity thì cũng giống như trong SQL chúng ta sẽ tạo ra một bảng chung gian là UserRole và tách nó thành 2 cái 1-N





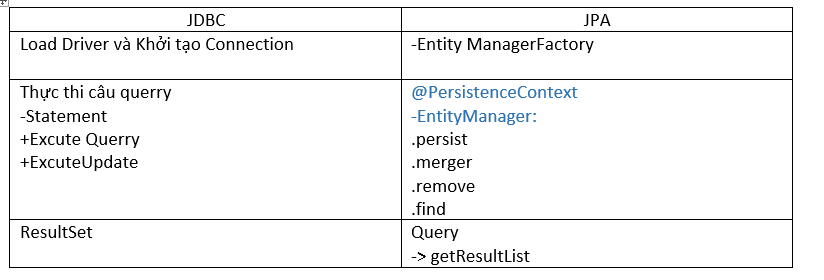




Nullable default là true nghĩa là có thể null là false thì không thể null

Unique thì là độc nhất không trùng

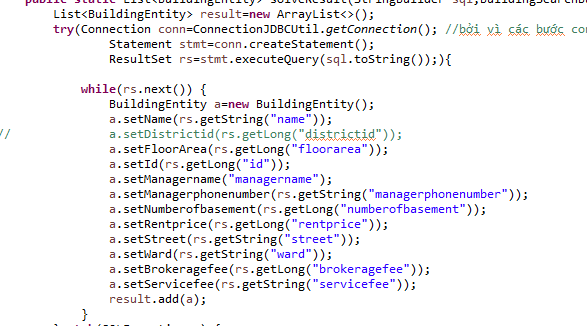
Như tôi đã nói qua phía trên thì có 2 cơ chế làm việc với JPA và giờ cùng tìm hiểu nhé:



+JPQL (JPA Query Language) : làm việc 100% với Entity

Lớp EntityManager là thuộc cái anotation PersistenceContext nhé.

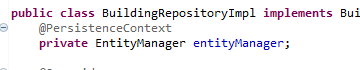
Xem lại ảnh trên thì khi ta dùng JDBC thực thi câu query rồi gán dữ liệu cho enity bằng statement :

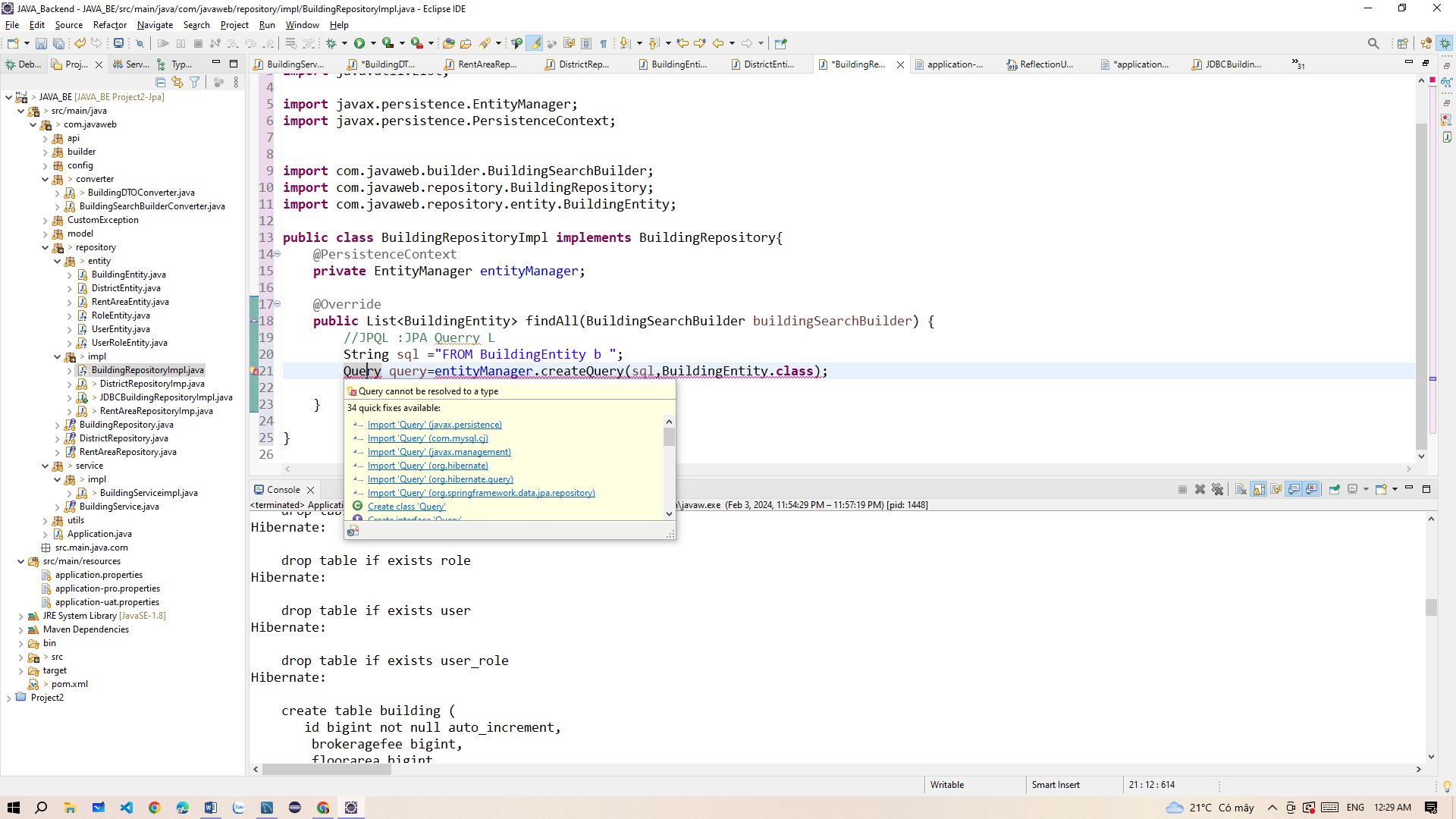


Rất là nhọc đúng không ?

Giờ sang JPA chúng ta sẽ thực hiện theo cơ chế JPQL((Java Persistence Query Language) để thao tác thực thi query với Entity Manager :

Đầu tiên hãy thêm những thứ như sau:



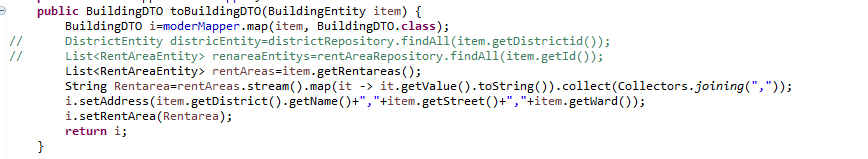


Khai báo lớp đối tượng Query của persistence nhé. Rồi sau đó sử dụng đối tượng entityManager thuộc lớp EntityManager gọi phương thức creatquery Phương thức này được sử dụng để tạo một đối tượng truy vấn JPA (Query) dựa trên chuỗi JPQL đã cung cấp (sql). JPQL tương tự như SQL nhưng hoạt động trên các thực thể (entities) thay vì các bảng.

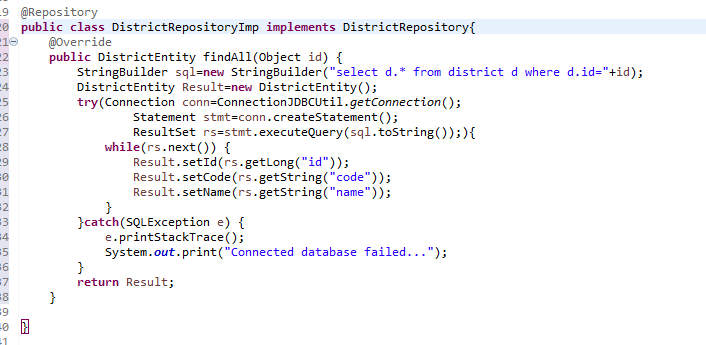
-nhìn bức ảnh trước mà ta dùng JDBC thì phải dùng tay để set cho các entity ,giờ thằng này tự động set cho chúng ta và việc tiếp theo là trả về một list hay single.



Thì sau khi chúng ta đã lấy được data từ phía DB rồi giờ việc của ta là chuẩn hóa rồi ném ra view đúng không, hay nhìn 2 dòng coment sử dụng JDBC để só sánh với những dòng dưới :

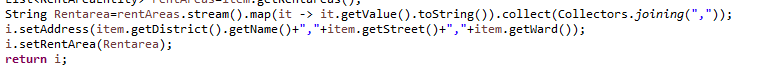


Như trước dùng JDBC ta muốn lấy name của thằng District thì ta phải thông qua từ 1 thằng building chứa id của quận và lấy id ta lại phải tạo thêm 1 thằng districtrepository để lấy dữ liệu từ DB thông qua thằng id đó và thằng renareas cũng vậy:

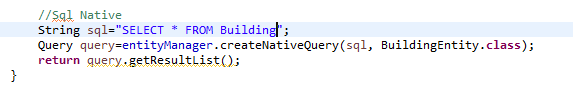


-Giờ khi sử dụng đến JPA ta thao tác với các Entity, bạn có nhớ khi ta thiết lập quan hệ giữa 2 entity building và district thì khi đó từ thằng entity building ta trả về đc 1 thằng entity district và từ 1 thằng entity district ta trả về được 1 list building

Như bức ảnh trên thì khi muốn lấy name của district thông qua id nằm trong bằng building thì chỉ cần sử dụng hàm get mà ta đã thiết lập trong entitybuilding thôi và việc dùng các repo khác để lấy data lúc này là không cần thiết.



+Sql Native: sử dụng ngôn ngữ của JPQ (gần giống sql) để làm việc

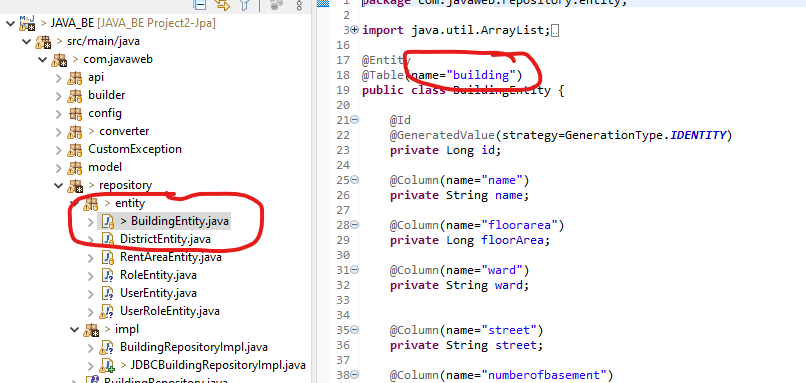


Câu cú pháp hay khai báo thì giống với thằng JPQL khác ở chỗ là thằng này câu lệnh truy vấn cảu nó gần giống với câu lệnh SQL nên hay đc sử dụng nhiểu hơn:

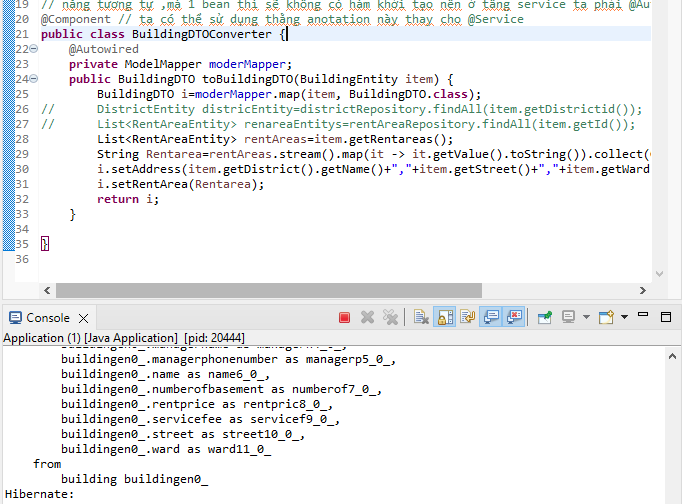
Vd: chọn toàn bộ tòa nhà

+JPQL : “FROM BuildingEntity b” vì thằng này làm việc 100% với Entity nên bạn nhìn cách nó lấy dữ liệu từ bảng cũng là tên của Entity (BuildingEntity) chứ không phải building

+Sql native : “SELECT \*FROM building b”

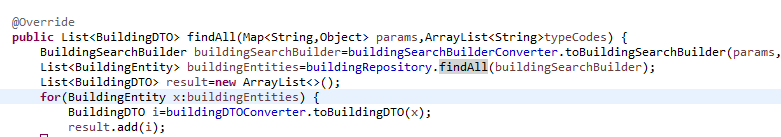


Code của hibernate sinh ra khi chạy:

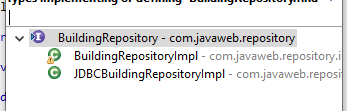


---------------------------------------------------------

Có 1 tình huống như sau:



Hàm findAll này thuộc interface buildingRepsitory và có 2 thằng triển khai từ nó:



Thế thì nó biết gọi thằng nào ? đơn giản thôi chỉ cần thêm anotation @Primary vào thằng mà ta muốn sử dụng.

