Using type query to cast result of query to specific tpye:

Let's see a *TypedQuery* alternative, compared to our first example:

**public** UserEntity **getUserByIdWithTypedQuery**(Long id) {

TypedQuery<UserEntity> typedQuery

= getEntityManager().createQuery("SELECT u FROM UserEntity u WHERE u.id=:id", UserEntity.class);

typedQuery.setParameter("id", id);

**return** typedQuery.getSingleResult();

}

This way, **we get stronger typing for free,** avoiding possible casting exceptions down the road.

Study: JpaSpecificationExecutor

Không dùng được: Update / delete from table [as T / T ], chỉ dùng được [as T / T ] tại select

**\_ Hibernate Criteria:**

-JPQL: ngắn gọn, giúp chúng ta query chỉ với 1 câu lệnh, nhưng cũng vì điều này mà JPQL tiềm ẩn những lỗi mà cho tới khi chạy/ sd ta mới biết, hoặc chạy sau 1 tg mới biêt

-**Criteria** thì ngược lại, chúng dài dòng hơn, nhưng dễ hiểu hơn, linh động hơnn, không bị hard code trong 1 string như **JPQL**, có thể tái sd, đặc biệt vì là Java object nên 1 câu lệnh bị lỗi về cú pháp thì ta có thể biết ngay khi biên dịch trương trình rồi

- nếu có 1 yêu cầu đơn giản, ta nên sử dụng **JPQL** còn phức tạp, ta nên sử dụng **criteria**

- 3 thành phần chính để tạo nên 1 câu lệnh sd criteria là :**criteriaBuilder, criteriaQuery, Root.**

Trong đó,  **criteriaBuilder** được tạo ra từ **session.getCriteriaBuilder():**  đối tượng cung cấp các toán tử, phép tính, logic trong câu sql

CriteriaQuery được tạo ra từ CQ<X.class> cq = **criteriaBuilder.createQuery(X.class):**  X làkết quả đầu ra của query, có thể là **object,** **primatives**, hoặc **multi scala(multi fields)**: **tuple**

**Root** được tạo ra từ: **criteriaQuery.from(X.class): X là đối** tượng đại diện cho tên bảng, đứng sau mệnh đề from trong câu sql.

* Trước khi thao tác cần getTransaction().begin() và kết thúc với: getTransaction(). Commit() Nếu đang sd chỉ hibernate criteria, còn spring data jpa thì không cần thao tác với session từ phía dev

-@Lazy load and eagerLoad:

Với lazyload: tại bảng **University**

@OneToMany**(fetch = FetchType.LAZY)**

List<Student> students

… get/set

**Biến Students** chỉ được load data từ DB khi getStudents() được gọi, trước đó, khi select all/one từ bảng university thì mỗi row sẽ không chứa 1 list student tương ứng dù có student đang tồn tại ,

@OneToMany**(fetch = FetchType.EAGER)**

List<Student> students

Với eagerload: tại bảng **University, khi se**lect all/one, mỗi row đều có 1 list student tương ứng và có đẩy đủ data dùng k cần gọi: row.getStudents();

* OneToMany, ManyToMany Mặc định sẽ là lazy load,, manyToOne, OneToOne mặc định sẽ là eagerLoad do 2 annotation này nhiều nhất cũng chỉ có 1 row đi kèm

Mặc định khóa ngoại trong bảng con là id tham chiếu của bảng cha sẽ có tên là [tên bảng cha+ \_id], Nếu bạn muốn đặt một cái tên khác cho khóa ngoại thì có thể sử dụng @JoinColumn annotation chỉ định trong thuộc tính name. @JoinColumn(name = “any\_name”)

Khi Trường **ID** có kiểu **string** thay vì **Long,**  cần xác định cách để tự động gen như sau:

@Id  
@GeneratedValue(generator = "UUID")  
@GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")  
private String id;

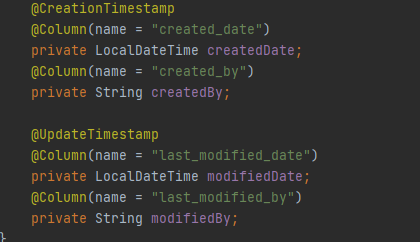
**@Transient:**  dùng cho thuộc tính không muốn xuất hiện trong Db nhưng vẫn đc Objêct sử dụng trong các context khác nhau.

**@JsonIgnore:**  Khi không muốn thuộc tính xuất hiện trong form Json khi thêm, sửa nhưng vẫn muốn có chúng trong DB.

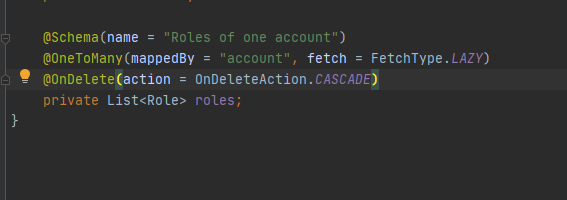
**@Enumerated(EnumType.String):** sử dụng kiểu enum **X bất kỳ** trong entity, mục đích để chắc chắn giá trị cùa filed được kiể soát 1 cách chính xác đó là các phần tử của Enum X đó.

Using : @**CreationTimestamp** To auto gen create time for raw.

And **@UpdateTimestamp** To auto save last\_modified\_time for raw



=> Trong quan hệ OneToMany và ManyToOne của School – student, khi muốn thiết lập: **on Delete: cascade** khi **Delete,** cần thêm thuộc tính: **@OnDelete(action =OnDeleteAction.CASCADE)** để ánh xạ vào DB tính chất tương ứng



Using native query:

@Query(nativeQuery = true, value = "select up.name from v2\_upload\_report up where up.status = 4 and up.period\_index = ?2 and up.period\_year = ?1" +  
 " and up.organization\_info\_oio\_id = ?3 and up.report\_form\_id in ?5 and up.period\_type = ?4")  
List<String> findReportForBctc(Integer periodYear, Integer periodIndex, Long orgId, String periodType, List<Long> ids);

Using StringBuilder to build sql vs NamedQuery  
