UNIVERSIDAD NACIONAL DE LA MATANZA

Taller Web 1

Evaluación de aprendizaje N°2

Teruel Pablo DNI 32605368

1- Pegar código del producto que se está desarrollando en equipo donde se vea el mapeo de hibernate de una relación entre clases explicando las partes principales del mismo. Se debería mostrar un caso de un mapeo uno a muchos. Si se mapea del lado “muchos” explicar por qué es recomendable mapear de este lado de la relación; si en cambio se tomó la decisión de ir en contra de la recomendación y mapear del lado “uno” explicar el por qué de esa decisión

Extracto de código:

```java
@Entity

public class Carrera {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;
    private Integer codigo;
    private String nombre;
    private String descripcion;

    @ManyToOne
    private Departamento departamento;
```

Explicación:

El mapeo de relaciones se produce dentro de los modelos en hibernate. Para generar una relación hace falta agregar diferentes tipos de notations según el tipo de relación a mapear y luego definir la otra entidad con la cual se va a relacionar.

@ManyToOne
Se agrega cuando se quiere relacionar muchas entidades en la cual estamos parados con una sola entidad definida luego de la notation.

@OneToMany
Se agrega cuando se quiere relacionar una sola entidad en la cual estamos parados con muchas entidaded definida luego de la notation.

@OneToOne
Se agrega cuando se quiere relacionar una sola entidad en la cual estamos parados con una sola entidad definida luego de la notation.

Es recomendable mapear del lado de las N (muchos) ya que es mucho mas simple referenciar y usar un valor en vez de una lista de valores. Además, el mapeo que hace internamente hibernate es mucho mas eficiente en relaciones *@ManyToOne.*

Al generar una relación unidireccional *@OneToMany*, Hibernate tiene que ejecutar sentencias adicionales para relacionar ambas tablas a nivel base de datos. Una buena práctica es usar *@ManyToOne* si queremos que sea unidireccional o si no crear directamente una relación bidireccional, de este modo nos ahorraremos la ejecución de queries innecesarias

Cabe alcara que las relaciones bidireccionales pueden generar inconsistencia en la base de datos si son empleadas de mala manera. Ya que podriamos referenciar a la entidad_1 con la entidad_2 como *@OneToMany* del lado de la entidad_1 y luego referenciar *@OneToMany* en vez de *@ManyToOne* del lado de la entidad_2.

Se podría haber mapeado del lado del 1 con el *@OneToMany* haciendo uso de listas, la solución que se elija depende principalmente de cada situación en particular.

2- Mostrar una porción de código de un repositorio donde se haga una consulta usando Criteria (que tenga al menos 1 restricción) explicando las partes importantes del mismo

`Extracto de código:`

```
@Override
public List<Carrera> listarCarrerasPorDepartamento(Departamento
departamento) {
    Session session =  sessionFactory.getCurrentSession();
    Criteria criteria = session.createCriteria(Carrera.class);
    criteria.add(Restrictions.eq("departamento", departamento));
    List<Carrera> carreras = criteria.list();

    return carreras;

}
```

Explicación:

Criteria es una interfaz que nos permite especificar consultas programáticamente (en base a clases y métodos de estas clases) sobre nuestras entidades definiendo un conjunto de restricciones. Es una forma mas amigable de ejecutar sentencias de SQL con código no tan complejo y más genérico.

Para usar Criteria debemos crear un objeto del tipo Criteria y luego pasarle como parámetro la entidad que queremos instanciar. Una vez que tengamos la clase instanciada, tenemos que aplicar los metodos de criteria que necesitemos. Uno de los más comunes es el add que agrega una restriccion evaluando una condición. En el ejemplo del extracto, la linea:

`criteria.add(Restrictions.eq("departamento", departamento));`

Solamente selecciona de todas las carreras las que corresponden con el departamento que le pasamos por parámetro. Seria similar a un where en sentencia de SQL.

Luego devuelve la consulta en formato de lista por el método .list().

Bibliografía:

https://www.adictosaltrabajo.com

https://www.baeldung.com

https://stackoverflow.com

https://docs.jboss.org

https://docs.jboss.org

https://www.adictosaltrabajo.com

http://javaencastellano-hibernate.blogspot.com