MAPEO CON ANOTACIONES EN HIBERNATE

Gracias a las anotaciones conseguiremos librarnos de los ficheros de xml de configuración de los mapeos (los famosos .hbm.xml). De esta manera tendremos que mantener menos ficheros ya que sólo es necesario el .java. Además escribiremos menos.

Las anotaciones tienen atributos los cuales tienen un valor por defecto, y en muchas ocasiones no tiene por que ser modificado.
Lo primero que deberemos hacer es en el archivo hibernate.cfg.xml poner la siguiente propiedad: <mapping class="paquete.subpaquete.Clase"/> por ejemplo: <mapping class="com.github.baaarbz.model.Employee"/>
Y será en los POJOS dónde usaremos las siguientes anotaciones para mapear los atributos con la base de datos

Existen más anotaciones/atributos de las que vamos a ver a continuación pero estas son las principales, en la página oficial de Hibernate podrás encontrar más información sobre las anotaciones y los atributos de cada una de ellas

Anotaciones a nivel de clase:

@Entity

· Sirve únicamente para indicarle a JPA que esa clase es una entidad. Cada entidad tiene asociada una tabla en la base de datos relacional y cada instancia de la entidad representa una fila en la tabla.

· El POJO deberá tener un constructor vacio.

@Table

Te permite especificar los detalles de la tabla que va a ser usada para persistir la entidad en la base de datos.
La anotación @Table tiene 4 atributos:

· name = "nombre_tabla"

· catalog

· schema = "nombre_db"

· uniqueConstraints

Anotaciones a nivel de métodos/atributos:

@Id

- Marca cual va a ser la PRIMARY KEY de nuestra tabla (puede haber una combinación de diferentes campos)
    - En caso de que sea necesario se puede determinar cuál va a ser la forma para generar la PRIMARY KEY con la anotación @GeneratedValue.

@GeneratedValue

Tienes dos parametros:

- strategy:

strategy	Funcionamiento
*GenerationType.***AUTO**	Es la que está seleccionada por defecto. Selecciona la estrategia a seguir automáticamente basándose en el dialecto de tu base de datos. Para la gran mayoría de bases de datos selecciona el tipo *GenerationType.SEQUENCE*
*GenerationType.***IDENTITY**	Es el más fácil de usar pero no el mejor a la hora de rendimiento, es autoincremental, dejando a la base de datos generar un nuevo valor por cada *INSERT*
*GenerationType.***SEQUENCE**	Usa una secuencia de una base de datos para generar valores únicos, requiere sentencias adicionales para obtener el siguiente valor de la base de datos refiriendote a la anotación *@SequenceGenerator* desde el parámetro generator. *No suele afectar al rendimiento de tu aplicación.*
*GenerationType.***TABLE**	Raramente usado a día de hoy. Simula una secuencia por guardar y actualizar su valor en el momento, este método realentiza tu aplicación. Puedes usar la anotación *@TableGenerator* de la misma forma que en el *GenerationType.SEQUENCE*

- generator

@Column

Es usada para especificar los detalles de la columna a la que está referenciada, los siguientes atributos son los más usados en esta anotación:

- name = "nombre_columna"
    - length = tamaño de la columna, valor por defecto 255

value	nullable
true	Es el valor por defecto, si está en *true* el campo puede ser *NULL*
false	Si está en *false* el campo será *NOT NULL*

value	unique
true	Crea restricción *UNIQUE*
false	Es el valor por defecto

@OneToMany / @ManyToMany / @ManyToOne / @OneToOne

Es la relación X a muchos.

```
@ManyToMany(cascade = CascadeType.ALL)
```

Algunos de sus atributos son:

value	cascade
Valor por defecto	Nada está en *cascade*
CascadeType.**ALL**	Todas las operaciones en *cascade*
CascadeType.**REFRESH**	Cuando el objeto referenciado es actualizado
CascadeType.**REMOVE**	Cuando el objeto referenciado es borrado
CascadeType.**DETACH**	Operación de desalojo en cascada. Lo mismo que arriba
CascadeType.**PERSIST**	La operación persistente de la sesión en cascada. Suponiendo que la anotaci clase Student y el campo de profesores está configurada con cascade = {Cas entonces, cuando el objeto stu1 establece una colección de profesor (los obje transitorios), cuando el objeto stu1d persiste , Todos los objetos transitorios d conectarán en cascada y se conservarán en la base de datos.
CascadeType.**MERGE**	Operación de fusión en cascada. Lo mismo que arriba

- mappedBy = "atributo" Referencia al atributo con el que está relacionado del otro POJO

`@ManyToMany`, adicional, hemos definido la anotación `@JoinTable`, la cual nos sirve para definir la estructura de la tabla intermedia que contendrá la relación entre los libros y los autores.

La anotación `@JoinTable` no es obligatoria en sí, ya que en caso de no definirse JPA asumirá el nombre de la tabla, columnas, longitud, etc. Para no quedar a merced de la implementación de JPA, siempre es recomendable definirla, así, tenemos el control total sobre ella.

Hemos definidos las siguientes propiedades de la anotación `@JoinTable`:

- **name**: Nombre de la tabla que será creada físicamente en la base de datos.
- **joinColumns**: Corresponde al nombre para el ID de la Entidad Book.
- **inverseJoinColumns**: Corresponde al nombre para el ID de la Entidad Author

En la otra entidad es más simple, pues solo marcamos la colección con `@ManyToMany`, pero en este caso ya no es necesario definir la anotación `@JoinTable`, en su lugar, definimos la

propiedad `mappedBy` para indicar la relación bidireccional y al mismo tiempo, JPA puede tomar la configuración del `@JoinTable`

```
@OneToMany(fetch = FetchType.LAZY, mappedBy = "user")
```

`FetchType.LAZY` está bajo demanda (es decir, cuando requerimos los datos).

`FetchType.EAGER` es inmediato (es decir, antes de que llegue nuestro requisito, estamos recuperando el registro innecesariamente)

@ManyToMany(fetch=FetchType.LAZY,casecade=CascadeType.MERGE)

De forma predeterminada, para todos los objetos de colección y mapa, la regla de obtención es `FetchType.LAZY`y

para otras instancias sigue la `FetchType.EAGER` política.

En resumen, `@OneToMany`y las `@ManyToMany`relaciones no obtienen los objetos relacionados (colección y mapa) implícitamente, LAZY

pero la operación de recuperación se conecta en cascada a través del campo en `@OneToOne`y `@ManyToOne`. EAGER