Relación entre entidades

En todos los ejemplos vistos hasta el momento, hemos trabajado siempre con entidades asociadas a tablas independientes, pero en las bases de datos del mundo real la información se encuentra repartida en tablas relacionadas, es decir, los datos se distribuyen en varias tablas y dichas tablas cuentan con campos comunes que nos permiten identificar que registros de una tabla se corresponden con los de otras.

En JPA las relaciones entre tablas se pueden representar también en el mundo de las entidades. En esta lección te enseñaré a establecer relaciones entre entidades y estudiaremos los beneficios que ello nos aportará a la hora de implementar la lógica de negocio de una aplicación. Y es que, al relacionar entidades, un objeto de una entidad contendrá el objeto u objetos de la entidad relacionada, lo que implica que al recuperar un objeto de una entidad recuperamos con él el objeto u objetos relacionados.

De cara a poder relacionar entidades, no es necesario que las tablas correspondientes se encuentren relacionadas explícitamente en la base de datos, basta con que cuenten con una columna común que permita identificar los registros de una tabla que se corresponden con los de la otra.

Tipos de relaciones entre entidades

Lo primero que hay que tener en cuenta es que para establecer una relación entre entidades **no es necesario que las tablas asociadas se encuentren relacionadas** físicamente en la base de datos, aunque deberán tener campos o columnas comunes que permitan identificar que registros de una tabla se corresponden con los de la otra.

Según las posibles relaciones que se pudieran establecer entre las tablas, se distinguen los siguientes tipos de relaciones entre entidades:

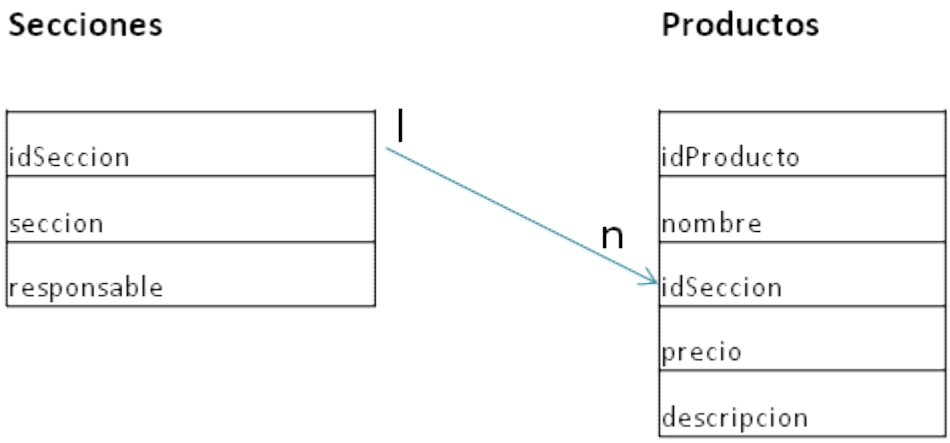
- **Relación uno a uno**. A un objeto entidad le corresponde un objeto de otra entidad. Este tipo de relación se da cuando cada registro de una tabla de la base de datos está asociado a un único registro de otra tabla. Es el tipo de relación menos común.

- **Relación uno a muchos**. A un objeto entidad le corresponden varios objetos de otra entidad. En sentido contrario, esta relación se puede interpretar como de muchos a uno. Por ejemplo, cuando tenemos una tabla de empelados y otra de departamentos, la entidad Departamento se relaciona uno a muchos con la entidad Empleado, pero en sentido contrario, Empleado se relaciona muchos a uno con Departamento. Es el tipo de relación más habitual.

- **Relación muchos a muchos**. Un objeto de una entidad se asocia con varios objetos de otra entidad y viceversa. Este tipo de relación requiere la existencia de tres tablas en la base de datos, las tablas principales y una de unión que relacione las claves primarias de las tablas principales.

Relación uno a muchos / muchos a uno

Para estudiar esta relación vamos a partir de las siguientes tablas de ejemplo:



Se trata de dos tablas correspondientes a la base de datos de un almacén. En la primera tabla tenemos registradas las secciones en las que se divide el almacén, mientras que en la segunda tenemos el registro de productos existentes. En este caso, es claro que se trata de una relación uno a muchos, pues en cada sección podemos tener varios productos almacenados.

La clave primaria de la tabla Secciones, idSeccion, aparece también en la tabla de Productos como clave ajena (fereing key). Esto nos permite identificar la sección a la que pertenece cada producto almacenado.

Atributos de relación

La relación entre entidades implica que en una entidad podemos almacenar el objeto u objetos de la entidad con la que se relaciona. Por tanto, lo primero que tenemos que hacer para poder establecer la relación será definir dentro de la clase un atributo del tipo de objeto con el que se va a relacionar.

En el ejemplo que estamos utilizando, dentro de la entidad Seccion tendremos que definir un atributo de tipo Collection,List o Set, que contendrá la colección de objetos Producto asociados a cada objeto Seccion, mientras que en la entidad Producto, **en vez de definir el atributo clave ajena "idSeccion", se creará una atributo de tipo de entidad** (en este caso de tipo Seccion), que guardará el objeto sección asociado a cada producto.

En el siguiente listado te mostramos como quedaría la definición de atributos de cada clase:

Entidad Sección

```
@Entity

@Table(name = "Secciones")

public class Seccion implements Serializable {
```

```
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer idSeccion;
    private String seccion;
    private String responsable;
     private List<Producto> productos;
    //resto de código: constructores y métodos getter y setter
    :
}
```

Entidad Producto

```
@Entity
@Table(name = "Productos")
public class Producto implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int idProducto;
    private String nombre;
    private double precio;
    private String descripcion;
    private Seccion seccion;
    //resto de código: constructores y métodos getter y setter
    :
}
```

Tanto en una como en otra entidad, deberíamos añadir también la pareja de métodos set/get para acceder a los atributos de relación

Configuración de la relación

Además de la definición de los atributos de relación, es necesario indicar cierta información de configuración al motor de persistencia para que sepa cómo obtener los objetos de las entidades relacionadas. Esta información se proporcionará a través de una serie de **anotaciones** que se colocarán delante de los atributos de relación.

Entidad del lado uno

En primer lugar, veremos cómo configurar el atributo de relación en la entidad del lado uno. Dicho atributo será de tipo colección y tendrá que estar definido con la anotación **@OneToMany**, que indica que un objeto de la clase actual está asociado al conjunto de objetos sobre el que se aplica la anotación.

Esta anotación deberá incluir al menos el atributo *mappedBy*, en el que se indicará el nombre del atributo de relación de la entidad relacionada que contendrá el objeto de la entidad actual.

Con el ejemplo de la entidad Seccion lo veremos más claro; así tendría que quedar definido el atributo productos:

```
@OneToMany(mappedBy = "seccion")

private List<Producto> productos;
```

Como vemos, aquí *mappedBy* hace referencia al atributo "seccion" de la entidad Producto que contendrá el objeto de tipo Seccion asociado a cada producto.

Como indicamos anteriormente, estos campos de colección pueden ser Collection, List o Set.

Entidad del lado muchos

La entidad del lado muchos es la propietaria de la relación, por ello la configuración del atributo de relación de esta entidad requiere algo más de trabajo, puesto que deberá indicarse en ella la información de relación. Así, en este atributo utilizaremos las siguientes anotaciones para la configuración de la relación

- **@ManyToOne**. Anotación utilizada en el campo de relación del lado muchos. Indica básicamente que varios objetos de la clase actual (en nuestro caso Producto) están asociados al objeto al que se le aplica esta anotación (Seccion)
- **@JoinColumn**. Proporciona la información de relación, indicando al motor de persistencia como están relacionadas las tablas en la base de datos. A través de los siguientes atributos se especifican los campos de relación:
  - **name**. Nombre de la columna foreing key (lado muchos)
  - **referencedColumnName**. Nombre de la columna primary key (lado uno)

Así pues en nuestro ejemplo, el atributo *seccion* de la entidad Producto deberá estar definido como se indica a continuación:

```
@JoinColumn(name = "idSeccion", referencedColumnName = "idSeccion")
 @ManyToOne
 private Seccion seccion;
```

Al configurar las relaciones de este modo, cada vez que se obtenga un objeto de una entidad se obtendrá también el objeto u objetos de la entidad relacionada.

El siguiente bloque de código de ejemplo nos mostraría por pantalla el nombre de la sección a la que pertenece un producto, a partir del código del mismo:

```
Producto p=em.find(Producto.class,2222);
If(p!=null){
        System.out.println("Sección: "+p.getSeccion().getSeccion());
}
```

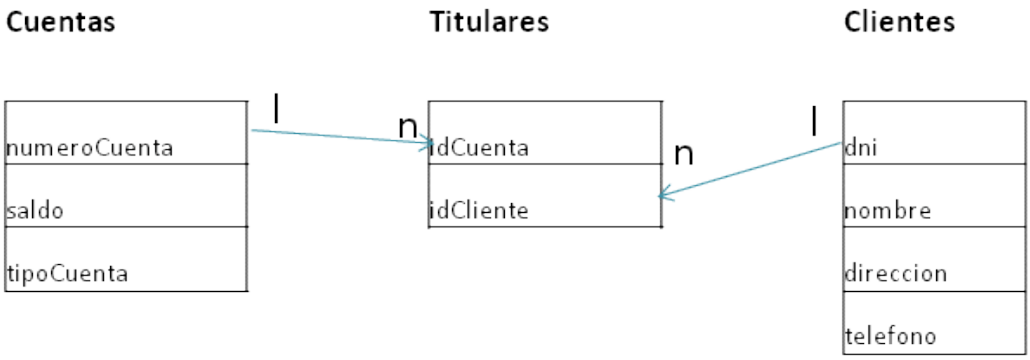
Este otro ejemplo nos muestra los nombres de todos los productos asociados a una determinada sección:

```
Query q=em.createQuery("select e from Seccion e where e.idSeccion=?1");
q.setParameter(1,31);
Seccion m=(Seccion)q.getSingleResult();
//recupera los objetos Producto relacionados
List<Producto> productos=m.getProductos();
for(Producto p:productos){
        System.out.println("Producto: "+p.getNombre());
}
```

Relación muchos a muchos

En este tipo de relación, cada entidad tiene asociado un conjunto de objetos de la otra entidad. A nivel de tablas, se requiere de una tabla de unión.

Para analizar este tipo de relación, vamos a pensar en el caso de las cuentas bancarias y los clientes de un banco. Cada cliente puede tener varias cuentas en dicho banco, a su vez, una cuenta puede estar asociada a varios clientes. A nivel de base de datos, sería una situación como la que se muestra en el siguiente esquema, en el que hemos simplificado el número de campos de las tablas Cuentas y Clientes:



Las tablas Cuentas y Clientes se encuentran relacionadas muchos a muchos, pero lo hacen a través de una tabla intermedia Titulares, que contiene las claves ajenas de ambas tablas, lo que permite identificar las cuentas asociadas a cada cliente y los clientes propietarios de cada cuenta.

Observa cómo, de forma deliberada, he definido en la tabla Titulares los nombres de las claves ajenas de manera distinta a como están definidos en las tablas principales.

Atributos de relación

A la hora de definir las entidades, Cuenta dispondrá de un atributo colección con los objetos Cliente asociados, mientras que Cliente contará con la colección de objetos Cuenta asociados:

```
@Entity

@Table(name="Cuentas")

public class Cuenta implements Serializable{

        @Id

        private int numeroCuenta;

        private double saldo;

        private String tipoCuenta;

        private List<Cliente> clientes;

        //constructores y métodos setter getter

        :

}

@Entity
```

```
@Table(name="Clientes")
public class Cliente implements Serializable{
    @Id
    private int dni;
    private String nombre;
    private String direccion;
    private int telefono;
    private List<Cuenta> cuentas;
    //constructores y métodos setter getter
    :
}
```

Configuración de la relación

De cara a configurar la relación, tenemos que tener en cuenta que una de las entidades será la propietaria de la relación mientras que la otra será la entidad del lado inverso de la relación. Al tratarse del mismo tipo de relación en ambos sentidos, da igual cual elijamos como propietaria y cual como inversa. En nuestro ejemplo, elegiremos Cliente como entidad propietaria de la relación y Cuenta como entidad inversa.

En ambas entidades, el atributo que contiene la colección de objetos relacionados deberá ser anotado con **@ManyToMany**. En el caso de la entidad del lado inverso (en nuestro ejemplo Cuenta), su atributo *mappedBy* indicará el nombre del atributo de la otra entidad que contiene los objetos relacionados de ésta:

```
@ManyToMany(mappedBy="cuentas")
private List<Cliente> clientes;
```

Será la entidad propietaria de la relación, en nuestro caso Cliente, la que deberá incluir la información sobre la relación a través de la anotación **@JoinTable**, en la que se deberá especificar los siguientes atributos:

- **name**. Nombre de la tabla de unión. En nuestro ejemplo, sería "Titulares".
- **joinColumns** e **inverseJoinColumns**. Con estos atributos indicamos los campos de relación entre la tabla de join las tablas propietarias e inversa, respectivamente. El valor de cada uno de estos atributos es una anotación @JoinColumn con dos atributos:
  - **name**. Nombre de la columna en la tabla de join
  - **referencedColumnName**. Nombre de la columna en la tabla principal.

En nuestro ejemplo, el atributo *cuentas* de la entidad Cliente debería estar definido de la siguiente manera:

```
@ManyToMany
@JoinTable(name="Titulares",
        joinColumns=@JoinColumn(name="idCliente",referencedColumnName="dni"),
        inverseJoinColumns=@JoinColumn(name="idCuenta",
                referencedColumnName="numeroCuenta"))
private List<Cuenta> cuentas;
```

Teniendo las entidades relacionadas de esta manera, si, por ejemplo, quisiéramos mostrar los nombres de todos los titulares de una determinada cuenta procederíamos de la siguiente manera:

```
String jpql="Select c From Cuenta c Where c.numeroCuenta=?1";

TypedQuery<Cuenta> q=em.createQuery(jpql,Cuenta.class):

q.setParameter(1,6666);

Cuenta cuenta=q.getSingleResult();

List<Cliente> titulares=cuenta.getClientes();

for(Cliente cl:titulares){

        System.out.println(cl.getNombre());

}
```