MÓDULO X - SPRING DATA JPA II

ESTABLECIENDO RELACIONES ENTRE LAS TABLAS

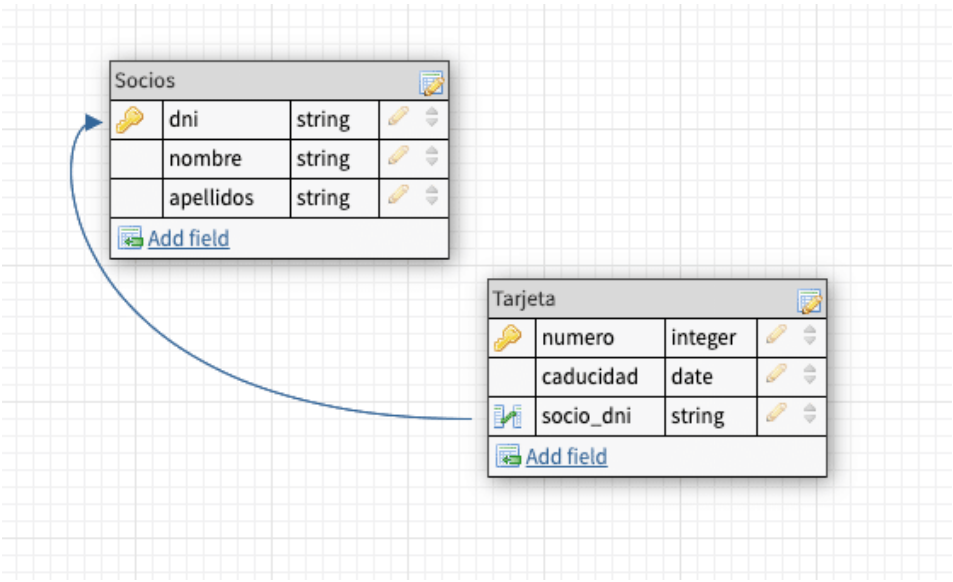
Al crear una base de datos, el sentido común dicta que usamos tablas separadas para diferentes tipos de entidades. Algunos ejemplos son: clientes, pedidos, artículos, mensajes, etc... Pero también necesitamos tener relaciones entre estas tablas. Por ejemplo, los clientes realizan pedidos y los pedidos contienen elementos. Estas relaciones deben ser representadas en la base de datos.

Hay varios tipos de relaciones de base de datos:

- Relaciones de uno a uno.
- Relaciones de uno a muchos y muchos a uno.
- Relaciones muchos a muchos.

RELACIONES DE UNO A UNO - @ONETOONE

En esta relación un registro de una entidad A se relaciona con solo un registro en una entidad B. Ejemplo dos entidades socio y tarjeta, ya que cada socio tiene asignada su tarjeta y cada tarjeta pertenece a un Socio. Este tipo de relaciones está desaconsejado su uso por motivos de rendimiento, ya que los datos en una relación de uno a uno, suelen meterse todos los datos en una misma tabla. En el mundo empresarial este tipo de relación no se usa, optándose siempre por la relación @ManyTone.

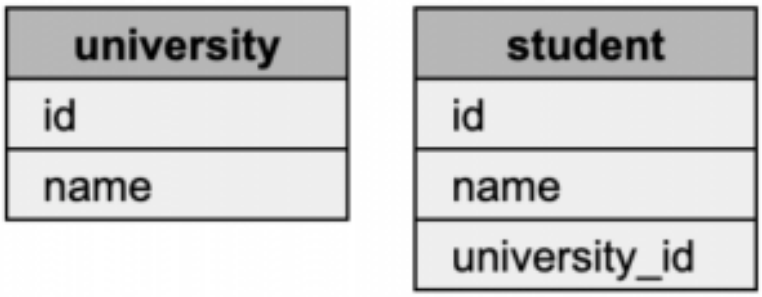


Aquí podemos observar como a través de JoinColumn se genera la relación.

RELACIONES DE MUCHOS A UNO - @MANYTOONE

Una relación many to one, es aquella en la que la tabla origen tiene un campo que hace referencia a la tabla destino. En JPA para establecer una relación many to one entre dos entidades debemos mapear el atributo de la clase propietaria (clase padre) de la relación con la anotación **@ManyToOne** y el atributo que hace referencia a la otra entidad (la clase hija) debe anotarse con **@ManyToOne(mappedBy=”propiedad”)**.

En el fondo es muy similar a la relación OneToOne, la principal diferencia es que la relación ManyToOne siempre mantiene una clave foránea de la tabla origen a la tabla destino, mientras que en la relación OneToOne la clave foránea puede estar en cualquiera de las dos tablas, origen o destino. También podemos usar (y es una buena práctica) la anotación **@JoinColumn** o **@JoinColumns** para especificar cual es el campo de la tabla que actúa de clave foránea.

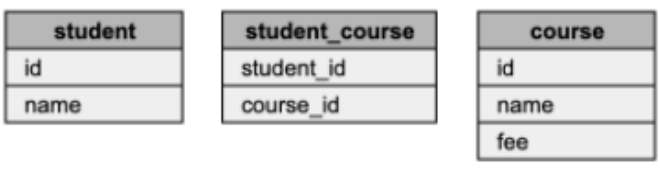


```
@Entity
@Table(name = "university")
public class University {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @OneToMany(mappedBy = "university", cascade =
CascadeType.ALL, orphanRemoval = true)
    private List<Student> students;

    /* Getters and setters */
}
```

```
@Entity
@Table(name = "students")
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @ManyToOne()
    @JoinColumn(name = "university_id")
    private University university;

    /* Getters and setters */
}
```

RELACIONES DE MUCHOS A MUCHOS - @MANYTOMANY

En una relación ManyToMany una entidad A se puede relacionar con 1 o con muchas entidades en B y viceversa (ejemplo asociaciones-ciudadanos, donde muchos ciudadanos pueden pertenecer a una misma asociación, y cada ciudadano puede pertenecer a muchas asociaciones distintas).



```
@Entity
@Table(name="course")
public class Course {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private Double fee;

    @ManyToMany(mappedBy = "courses")
    private Set<Student> students;

    /* Getters and setters */
}

@Entity
@Table(name="student")
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @ManyToMany(cascade = {
            CascadeType.PERSIST,
            CascadeType.MERGE
    })
    @JoinTable(
            name = "student_course",
            joinColumns = {@JoinColumn(name = "student_id")},
            inverseJoinColumns = {@JoinColumn(name =
"course_id")}
    )
    private Set<Course> courses;

    /* Getters and setters */
}
```

En este ejemplo el owning side es student y es donde se usa la anotación @JoinTable. Con ella, especificamos el nombre de la tabla que realiza el mapeo (student_course). JoinColumns apunta a la tabla del owning side (student) e InverseJoinColumns apunta a la tabla inversa del owning side (course). Usamos el cascade Merge y Persist, pero no cascade.Remove ya que si eliminamos un curso, no queremos eliminar los estudiantes asociados a él.

Como podemos ver en el ejemplo, estamos usando un Set en vez de List en la asociación. Esto es porque usando List, Hibernate elimina las filas del objeto que queremos eliminar y vuelve a insertar las demás. Esto es completamente innecesario e ineficiente.