Otra forma de establecer una secuencia de id en las tablas

@SequenceGenerator(  
 name = "client\_sequence",  
 sequenceName = "client\_sequence",  
 allocationSize = 1,  
 initialValue = 999  
)  
@GeneratedValue(strategy = GenerationType.*SEQUENCE*,generator = "client\_sequence")

Relacion uno a uno

@OneToOne  
User user;

Relación uno a muchos bidireccional

Del lado del one

@OneToMany(fetch = FetchType.*EAGER*,  
cascade = CascadeType.*ALL*)  
Set<Address> addressList=new HashSet<>();

Del lado del many

@ManyToOne(fetch = FetchType.*EAGER*,optional = false)  
@JoinColumn(name = "client\_id")  
Client client;

Relacion muchos a muchos unidireccional

Class Entity 1

@ManyToMany(fetch = FetchType.*EAGER*)  
@JoinTable(name = "client\_product",  
 joinColumns = {@JoinColumn(name = "fk\_client")},  
 inverseJoinColumns = {@JoinColumn(name = "fk\_product")}  
)  
Set<Product> products=new HashSet<>();

Relacion bidireccional OneToMany

En este caso seria de usuario a roles , un usuario puede tener un solo rol pero un rol puede tener muchos usuarios o sea que desde rol seria una relación entre entidades 1(rol) a N(usuarios)

Del lado de Rol

@Entity

public class Rol {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int id;

private String nombreRol;

private String descripcion;

@OneToMany(mappedBy = "unRol")

private List<Usuario> listaUsuarios;

del lado de Usuario

@Entity

public class Usuario implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int id;

private String nombreUsuario;

private String contrasena;

@ManyToOne

@JoinColumn(name = "fk\_rol")

private Rol unRol;

Relacion bidireccional OneToMany y del otro lado ManyTonOne

Relacion: “Una Biblioteca tiene muchos libros”

public class Libro {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 int id;  
 @NotNull  
 String nombre;  
 @ManyToOne(fetch = FetchType.*LAZY*,optional = false)  
 @JoinColumn(name = "id\_biblioteca")  
 @JsonProperty(access = JsonProperty.Access.*WRITE\_ONLY*)  
 Biblioteca biblioteca;  
  
}

para evitar lazyInitializationException se agrega la línea : @JsonProperty(access = JsonProperty.Access.*WRITE\_ONLY*)

public class Biblioteca {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 int id;  
 String nombre;  
 @OneToMany(mappedBy = "biblioteca",cascade = CascadeType.*ALL*)  
 Set<Libro> listaDeLibros=new HashSet<>();  
}

cascade = CascadeType.*ALL*)  
se coloca por que si se borra la biblioteca también sus libros.

Nuevo intento de relaciones entre tablas

Relación OneToOne bidireccional

1-llave foránea FK

De un lado:

public class InformacionDeContacto {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "email",nullable = false)  
 private String email;  
 @Column(name = "telefono",nullable = false)  
 private String telefono;  
 @Column(name = "direccion",nullable = false)  
 private String direccion;  
 @Column(name = "ciudad",nullable = false)  
 private String ciudad;  
 @Column(name = "provincia",nullable = false)  
 private String provincia;  
 @Column(name = "codigo\_postal",nullable = false)  
 private String codigoPostal;  
   
 @OneToOne(cascade = CascadeType.*ALL*,optional = false)  
 @JoinColumn(name = "estudiante\_id")  
 private Estudiante estudiante;

del otro:

con mappedBy decimos que la llave de la relación va a estar almacenada en la otra entidad.

public class Estudiante {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "nombre\_estudiante",nullable = false)  
 private String nombre;  
 @Column(name = "apellido\_estudiante",nullable = false)  
 private String apellido;  
 @Column(name = "fecha\_de\_nacimiento",nullable = false)  
 private LocalDate fechaDeNacimiento;  
  
 @OneToOne(mappedBy = "estudiante",cascade = CascadeType.*ALL*)  
 private InformacionDeContacto informacionDeContacto;

2-llave primaria compartida por ambas entidades

Por un lado:

public class InformacionDeContacto {  
 @Id  
 @Column(name = "estudiante\_id")  
 private Long id;  
 @Column(name = "email",nullable = false)  
 private String email;  
 @Column(name = "telefono",nullable = false)  
 private String telefono;  
 @Column(name = "direccion",nullable = false)  
 private String direccion;  
 @Column(name = "ciudad",nullable = false)  
 private String ciudad;  
 @Column(name = "provincia",nullable = false)  
 private String provincia;  
 @Column(name = "codigo\_postal",nullable = false)  
 private String codigoPostal;  
  
 @OneToOne(cascade = CascadeType.*ALL*,optional = false)  
 @MapsId  
 @JoinColumn(name = "estudiante\_id")  
 private Estudiante estudiante;

Por el otro:

public class Estudiante {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "nombre\_estudiante",nullable = false)  
 private String nombre;  
 @Column(name = "apellido\_estudiante",nullable = false)  
 private String apellido;  
 @Column(name = "fecha\_de\_nacimiento",nullable = false)  
 private LocalDate fechaDeNacimiento;  
  
 @OneToOne(mappedBy = "estudiante",cascade = CascadeType.*ALL*)  
 @PrimaryKeyJoinColumn  
 private InformacionDeContacto informacionDeContacto;

Relacion OneToMany

1-Unidireccional

De un solo lado!!

public class Profesor {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "nombre\_profesor",nullable = false)  
 private String nombre;  
 @Column(name = "apellido\_profesor",nullable = false)  
 private String apellido;  
   
 @OneToMany  
 private Set<Curso> cursos=new HashSet<>();

crea una tercera tabla -> poco practico

2-bidireccional

De un lado:

public class Profesor {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "nombre\_profesor",nullable = false)  
 private String nombre;  
 @Column(name = "apellido\_profesor",nullable = false)  
 private String apellido;  
  
 @OneToMany(mappedBy = "profesor",cascade = CascadeType.*ALL*)  
 private Set<Curso> cursos=new HashSet<>();

del otro:

public class Curso {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "nombre\_curso",nullable = false)  
 private String nombre;  
 @ManyToOne(fetch = FetchType.*EAGER*)  
 private Profesor profesor;

Relación ManyToMany

Del lado de la propietaria de la relacion

public class Estudiante {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "nombre\_estudiante",nullable = false)  
 private String nombre;  
 @Column(name = "apellido\_estudiante",nullable = false)  
 private String apellido;  
 @Column(name = "fecha\_de\_nacimiento",nullable = false)  
 private LocalDate fechaDeNacimiento;  
  
 @OneToOne(mappedBy = "estudiante",cascade = CascadeType.*ALL*)  
 @PrimaryKeyJoinColumn  
 private InformacionDeContacto informacionDeContacto;  
  
 @ManyToMany  
 @JoinTable(name = "estudiantes\_curso",joinColumns =  
 @JoinColumn(name = "estudiante\_id"),inverseJoinColumns =@JoinColumn(name = "curso\_id") )  
 private Set<Curso> cursos=new HashSet<>();

del otro lado:

public class Curso {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "nombre\_curso",nullable = false)  
 private String nombre;  
 @ManyToOne(fetch = FetchType.*EAGER*)  
 private Profesor profesor;  
  
 @ManyToMany(mappedBy = "cursos")  
 private Set<Estudiante> estudiantes=new HashSet<>();

Relaciones entre tablas OneToOne unidireccional

Lo coloco donde quiero que se establezca la clave foránea, en este caso en club

public class Club {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String name;  
 @OneToOne(targetEntity = Coach.class,cascade = CascadeType.*PERSIST*)

@JoinColumn(name = "id\_coach")  
 private Coach coach;  
}

Relaciones entre tablas OneToMany bidireccional

Del lado de One

public class Club {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String name;  
 @OneToOne(targetEntity = Coach.class,cascade = CascadeType.*PERSIST*)  
 @JoinColumn(name = "id\_coach")  
 private Coach coach;  
  
 @OneToMany(targetEntity = Player.class,fetch = FetchType.*LAZY*,mappedBy = "club")  
 private List<Player> players;  
}

del lado del many

public class Player {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String name;  
 @Column(name = "last\_name")  
 private String lastName;  
 private String nationality;  
 private Integer age;  
 private String position;  
 @ManyToOne(targetEntity = Club.class)  
 private Club club;  
}

Relacion ManyToMany no puede ser bidireccional

public class Club {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String name;  
 @OneToOne(targetEntity = Coach.class,cascade = CascadeType.*PERSIST*)  
 @JoinColumn(name = "id\_coach")  
 private Coach coach;  
  
 @OneToMany(targetEntity = Player.class,fetch = FetchType.*LAZY*,mappedBy = "club")  
 private List<Player> players;  
 @ManyToOne(targetEntity = FootballAssociation.class)  
 private FootballAssociation footballAssociation;  
 @ManyToMany(targetEntity = FootballCompetition.class,fetch = FetchType.*LAZY*)  
 private List<FootballCompetition> competitions;  
}

ejemplo para usar el Column

@GeneratedValue(strategy = GenerationType.*IDENTITY*)  
private Long id;  
**@Column(name = "name",columnDefinition = "VARCHAR(300)")  
private String name;**@Column(name = "cuantity\_price")  
private Integer cuantityPrice;  
@Column(name = "start\_date",columnDefinition = "DATE")  
private LocalDate startDate;  
@Column(name = "end\_date", columnDefinition = "DATE")  
private LocalDate endDate;

Ejemplo de como personalizar la tabla intermedia de una relación ManyToMany

@ManyToMany(targetEntity = FootballCompetition.class,fetch = FetchType.*LAZY*)  
@JoinTable(name = "club\_competition"  
 ,joinColumns = @JoinColumn(name = "club"),  
 inverseJoinColumns = @JoinColumn(name = "competition"))  
private List<FootballCompetition> competitions;