
spring

JAVA PERSISTENCE API

JPA tiene anotaciones para mapear atributos de objetos a relaciones de cardinalidad *que se necesitan en un esquema de base de datos relacional.*

De esta manera se puede abstraer del esquema de base de datos, y centrarse en el modelo de objetos, pudiendo relacionar tanto atributos de un solo objeto, como atributos de colecciones tan solo con una anotación. En JPA tiene varias anotaciones para mapear las relaciones de cardinalidad.

- ✓ *Relación Uno a Uno (**@OneToOne**)*

*Partimos de nuestro modelo de objetos, donde tenemos una clase Equipo, con un atributo Entrenador. Como un Entrenador solo puede pertenecer a un Club, y éste va a tener solo un Entrenador, utilizaremos la relación **@OneToOne**.*

```
@Entity
@Table(name = "clubes")
public class Club {
 @Id
 private Long id;

@OneToOne
 private Entrenador entrenador;
}
```

```
@Entity
@Table(name = "entrenadores")
public class Entrenador {
 @Id
 private Long id;
 private String nombre;
 private String apellido;
 private int edad;
 private String nacionalidad;
}
```

*En el modelo de objetos, el Entrenador no conoce al Club al cual pertenece y para no tener una relación bidirecional, decidimos que quien sea el propietario de la relación sea el Club, ya **que un Entrenador no tiene razón de existir por sí solo en nuestro dominio, sino que depende de la existencia de un Club**. Pero a su vez, tiene la suficiente relevancia para ser una entidad en nuestro modelo.*

Una vez realizado el mapeo, en el modelo de datos tenemos una relación de uno a uno entre un Club de futbol y su Entrenador, donde un club va a tener solo un Entrenador, y un Entrenador va a pertenecer solo a un Club. ***En esta relación, el propietario de la misma es el Club ya que tiene la FK al Entrenador.***

- *Relación Uno a Muchos (@**OneToMany**)*

Ahora a nuestro club vamos a agregarle la plantilla de jugadores. En nuestro modelo de objetos vamos a tener una lista de tipo Jugador perteneciente a la clase Club. Esto nos permite tener una colección de jugadores que van a pertenecer solo a un club, y por tal motivo podemos utilizar ***@OneToMany****.*

```
@Entity
@Table(name = "clubes")
public class Club {
 @Id
 private Long id;

@OneToOne
 private Entrenador entrenador;

@OneToMany
 private List<Jugador> jugadores;
}
```

```
@Entity
@Table(name = "jugadores")
public class Jugador {
 @Id
 private Long id;
 private String nombre;
 private String apellido;
 private int numero;
 private String posicion;
}
```

Si probamos el modelo de datos, podemos observar que se creó una nueva tabla "jugadores" y una tabla intermedia "clubes_jugadores". ***Al ser una relación de uno a muchos, pensando solamente nuestro diseño de datos, hubiésemos esperado tener una FK de la entidad perteneciente a la relación del "muchos", referenciando al id de la entidad del "uno".*** *Es decir, tener un campo id_club en la tabla "jugador" referenciando a la tabla "club". Pero al no especificar nada, JPA entiende esta relación con "muchos a muchos" y nos crea esta tabla intermedia.*

Para evitar la creación de la tabla intermedia, y que la tabla "jugador" tenga la FK a "club", debemos agregar la annotation @JoinColumn a nuestra lista de jugadores, con una propiedad "name" en la cual vamos a indicar como se llamara el campo que hace referencia a la tabla Club. *Naturalmente podríamos pensar en que es necesario hacer explicito el campo al cual hace referencia, y se puede hacer con la propiedad referencedColumnName, pero por lo general no se suele utilizar ya que JPA infiere que la columna es el ID de la entidad referenciada.*

```
@Entity
@Table(name = "clubes")
public class Club {
 @Id
 private Long id;

@OneToOne
 private Entrenador entrenador;

@OneToMany
 @JoinColumn(name = "id_club")
 private List<Jugador> jugadores;
}
```

Ahora sí, nuestro modelo de datos solo tendrá las tablas maestras relacionadas por la FK, sin necesidad de una tabla intermedia.

Relación Muchos a Muchos ***(@ManyToMany)***

Por último, queremos modelar en nuestro dominio de objetos, las competiciones en las que participan todos los clubes. De este modo podemos definir que un Club puede participar en muchas Competiciones y una competencia obviamente va a tener muchos Clubes participantes. Así es como utilizaremos la relación ***@ManyToMany****.*

```
@Entity
@Table(name = "clubes")
public class Club {
 @Id
 private Long id;

@OneToOne
 private Entrenador entrenador;

@OneToMany
 @JoinColumn(name = "id_club")
 private List<Jugador> jugadores
```

```
@ManyToMany
 private List<Competicion> competiciones;
}
```

```
@Entity
@Table(name = "competiciones")
public class Competicion {
 @Id
 private Long id;
 private String nombre;
 private int montoPremio;
 private LocalDate fechaInicio;
 private LocalDate fechaFin
}
```

Decidimos agregar en nuestra clase Club una lista de Competiciones en la que participa, aunque de igual modo podríamos haber decidido agregar una lista de Clubes en la Competicion, pero trataremos de evitar relaciones bidireccionales.

En nuestro modelo de datos, podemos ver que ***se crea una nueva tabla "clubes_competiciones"*** *con referencia a los IDs de las dos tablas relacionadas. Esta tabla no tiene entidad por sí misma, ya que solo nos sirve para romper con la relación de muchos a muchos, a diferencia de una tabla intermedia la cual cada registro tiene un id y posiblemente otros datos que no dependen de las tablas relacionadas.*

De la misma manera, podríamos cambiar la relación en nuestro modelo de objetos, agregando una lista de Clubes participantes en cada instancia de una Competicion, y nuestro esquema de base de datos será igual independientemente de que como modelemos en objetos.

```
@Entity
@Table(name = "clubes")
public class Club {
 @Id
 private Long id;

@OneToOne
 private Entrenador entrenador;

@OneToMany
 @JoinColumn(name = "id_club")
 private List<Jugador> jugadores
```

```
@Entity
@Table(name = "competiciones")
public class Competicion {
 @Id
 private Long id;
 private String nombre;
 private int montoPremio;
 private LocalDate fechaInicio;
 private LocalDate fechaFin
 @ManyToMany
 private List<Club> clubes;
}
```

El modelo de objetos no necesita relaciones bidireccionales para poder realizar relaciones entre muchos registros, ya que solo nos alcanza con agregar el mappeo en los atributos y así nos evitamos relaciones bidireccionales que nos pueden traer complicaciones a nivel