Tema 2 – Spring Boot mapeando relaciones: 1 a N

Grado en Ingeniería Informática en Tecnologías de la Información

Departamento de Ingeniería de Sistemas Informáticos y Telemáticos

Área de Lenguajes y Sistemas Informáticos

Dr. Luis V. Calderita

Recordatorio E/R

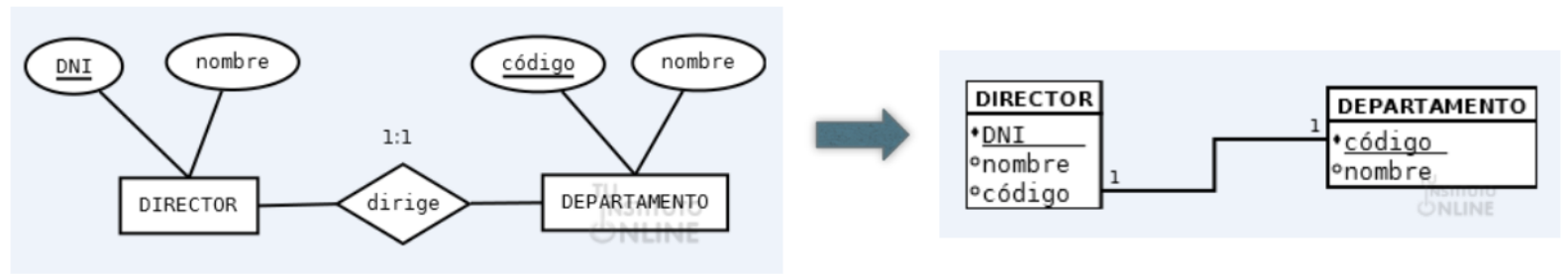
- ENTIDADES:
  - Cada entidad del modelo E-R genera una tabla.
  - Dicha tabla contiene como columnas cada uno de los atributos de la entidad.
  - Además puede contener otras columnas fruto de relaciones con otras entidades.

Recordatorio E/R

- RELACIONES:
  - Dependiendo de las **cardinalidades** de las relaciones, éstas pueden **generar una tabla** o por el contrario traducirse **en campos o columnas** dentro de las tablas asociadas a las entidades que participan en la relación.

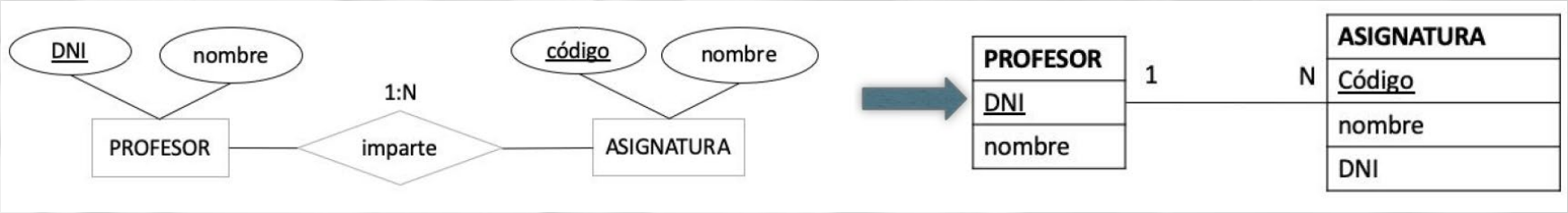
Recordatorio E/R

- RELACIONES 1:1.
  - Se añade en una de las tablas la clave primaria de la otra tabla.
  - Además se establecerá una relación entre estas dos tablas por dicho campo.



Recordatorio E/R

- RELACIONES 1:N.
  - Se añade en la tabla de la entidad que tiene la parte del N la clave principal de la otra tabla.
  - Además se establecería una relación entre estas dos tablas por dicho campo.



Recordatorio E/R

- RELACIONES N:N.
  - Se resuelve mediante la creación de una tabla intermedia cuyas campos serán las claves primarias de ambas entidades. Si la relación tiene atributos se añaden a la tabla de la relación.
  - Esta tabla se relaciona con las dos tablas mediante dos relaciones de 1:N, teniendo siempre la parte del N en la nueva tabla creada.
  - La clave primaria puede ser el conjunto de las claves primarias de las otras dos tablas. O un campo auto numeración.

Anotaciones JPA

- Las anotaciones disponibles para el mapeo de entidades son:
  - @OneToOne, relación 1 a 1
  - @OneToMany, relación 1 a Muchos
  - @ManyToOne, relación Muchos a 1
  - @ManyToMany, relación Muchos a Muchos
- Estas anotaciones tienen atributos para matizar la relación
- Vamos a realizar un ejemplo 1 a N

Relación 1 a N: Intuitiva

- Un usuario puede tener más de una dirección

Relación 1 a N: Relacional

- Representación en Tablas:
  - La tabla *Usuario* se relaciona con la *Direccion*
  - ¿Cómo?
    - En la tabla Dirección se incluye la clave primaria del Usuario

Relación 1 a N: Objetos

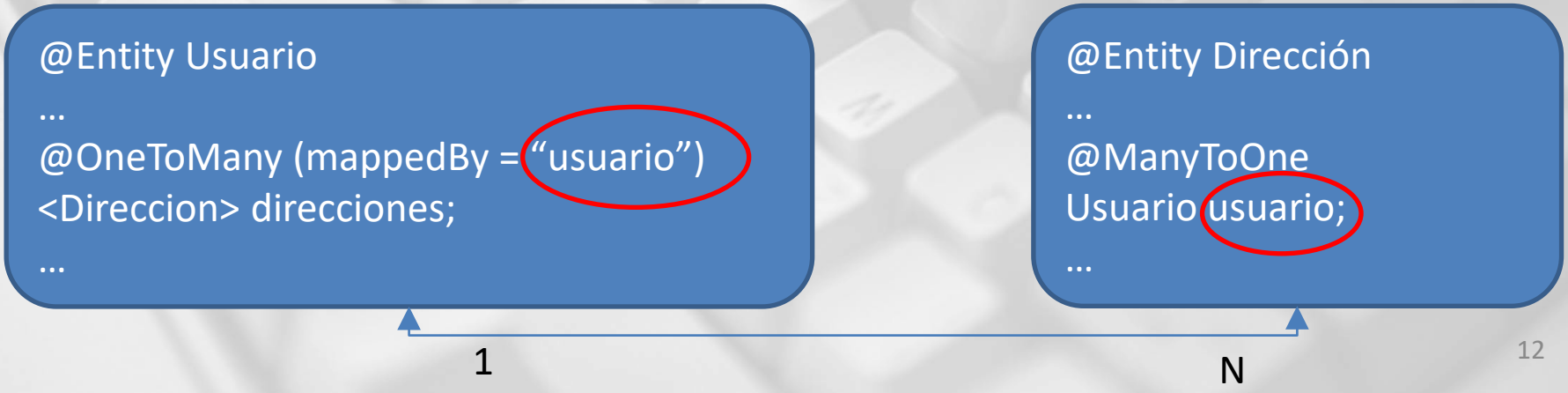
- Representación en Clases:
  - La clase Usuario contiene un *vector* que almacena las Direcciones

Relación 1 a N: Asociando Entidades

- Un usuario puede tener varias direcciones. (Muchas direcciones son del mismo Usuario)
  - Asocio cada Dirección con un Usuario.
  - Añado @ManyToOne en la entidad Dirección sobre la propiedad que los relaciona
  - De momento, es *unidireccional*: Desde Dirección puedo acceder a Usuario. No al revés.

Relación 1 a N: Asociando Entidades

- Añadimos el otro lado de la relación:
  - Añado @OneToMany en la entidad Usuario sobre la propiedad que los relaciona
  - Y le indico que propiedad de Dirección se encarga del mapeo
- La asociación es *Bidireccional.*
- El Usuario contiene una colección de *Direccion.*



Relación 1 a N: Implementación

- Necesitamos dos Entidades
  - Usuario y Direccion
- Condiciones:
  - Un Usuario puede tener varias direcciones.
  - Queremos que desde el usuario se pueda acceder a sus direcciones
- Solución (posible):
  - Implementar una asociación 1 a N y bidireccional

Entity Dirección

```
@Entity
public class Direccion {

    @Id
    @GeneratedValue (strategy = GenerationType.AUTO)
    private long id;

    private String dir;
    private String ciudad;

    //Esta es la entidad propietaria de la relación. Contiene la clave ajena
    @ManyToOne
    private Usuario usuario;

    . . .
```

Entity Usuario

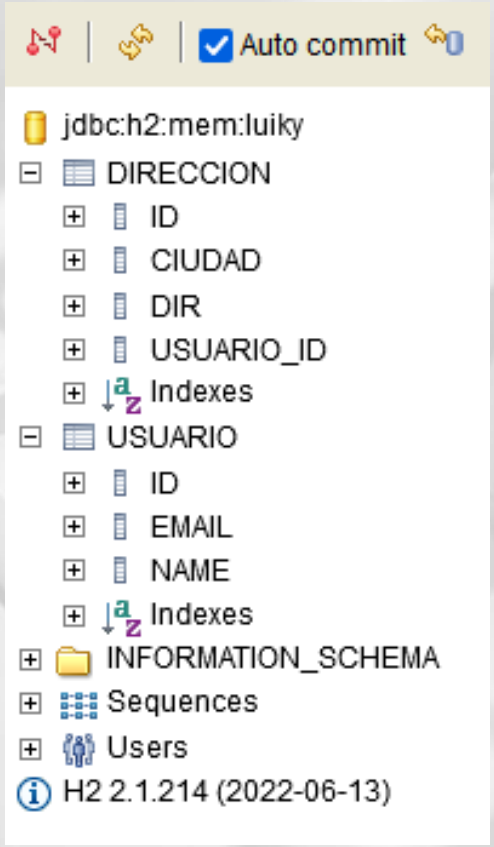
```
@Entity
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    private String name;
    private String email;

    //Esta es la entidad NO propietaria.
    @OneToMany (mappedBy ="usuario") //La magia se empieza a acabar…
    private Set<Direccion> direcciones = new HashSet<>();

    . . .
```

Propiedad colección

- Las colecciones de elementos que pueden usarse como persistentes son:
  - java.util.Set, java.util.List
  - java.util.SortedMap, java.util.SortedSet,
- La **propiedad persistente** que contenga una colección ha de ser un interface del tipo **Map, Set o List**; nunca HashMap, TreeSet o ArrayList.
  - Esta restricción es debida a que Hibernate reemplaza las instancias de Map, Set y List con instancias de sus propias implementaciones de Map, Set o List.

Relación 1 a N mapeada en la BD

- Las dos entidades anteriores establecen la relación 1 a N a nivel de tabla en la BD:



Matizando la relación.

- *La magia se empieza a acabar…*
  - Necesitamos matizar, mediante parámetros de configuración, la anotación correspondiente
- En este caso, sólo son necesarios del lado de la relación del Usuario:
  - Decidir como propagar ciertos eventos a la entidad relacionada (Tipo de Cascade)
  - Decidir como carga los datos de la entidad relacionada (Tipo de Fetch)
- Para logarlo hay que:
  - Modificar/añadir parámetros a la anotación @OneToMany de la entidad Usuario

Tipos de operación en Cascada

- Permite establecer la forma en que deben propagarse ciertos eventos (como persistir, eliminar, actualizar, etc) entre entidades que forman parte de una asociación.
  - CascadeType.PERSIST: Cuando una entidad es persistida, su entidad relacionada debe ser persistida también.
  - CascadeType.REMOVE: Cuando una entidad es borrada, su entidad relacionada debe ser borrada también.
  - CascadeType.REFRESH: Cuando una entidad es refrescada, su entidad relacionada debe ser refrescada también.
  - CascadeType.MERGE: Cuando una entidad es actualizada, su entidad relacionada debe ser actualizada también.
  - CascadeType.ALL: Cuando una entidad es persistida, borrada, refrescada o actualizada, su entidad relacionada debe ser persistida, borrada, refrescada o actualizada también

Matizando la relación: Cascade

- Propagar Eventos.
  - Queremos que todos los eventos que afectan a la entidad Usuario se propaguen a la entidad Direccion.

```
@OneToMany (mappedBy ="usuario",
                    cascade = CascadeType.ALL)
```

Tipo de carga

- Permite configurar cuándo se debe cargar la entidad relacionada
  - FetchType.**EAGER**. Al cargar una entidad cargamos su entidad relacionada junto con el resto de campos
  - FetchType.**LAZY**. La carga será bajo demanda al llamar al método get correspondiente
    - StackOverFlow Memo:
      - LAZY = fetch when needed
      - EAGER = fetch immediately

Matizando la relación: FecthType

- Tipo de carga:
  - Queremos que al Usuario se le carguen inmediatamente sus direcciones asociadas (EAGER)

```
@OneToMany (mappedBy ="usuario",
            cascade = CascadeType.ALL,
            fetch = FetchType.EAGER)
```

- **Nota**: En este caso, la elección de EAGER, simplifica la ejecución en el test Junit. En otro caso la opción LAZY puede ser perfectamente válida. Sin embargo, para este ejemplo lanzaría una excepción (probarlo)

Entity Usuario: @OneToMany Matizado

```
@Entity
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    private String name;
    private String email;

    //Esta es la entidad NO propietaria. La magia…
    @OneToMany (mappedBy ="usuario", cascade = CascadeType.ALL , fetch =
    FetchType.EAGER)
    private Set<Direccion> direcciones = new HashSet<>();

    . . .
```

Practicando

- Crea un proyecto Spring Boot, con las dependencias habituales
- Crea las entidades anteriores. Usuario y Dirección. Mapea la relación 1 a N entre ellas.
- Crea un repository para Usuario y otro para Dirección.
- Crea un test, con dos usuarios. Uno con dos direcciones y otro con una. Guárdalos.
  - Muestra todos los usuarios de la BD
  - Elimina un usuario
  - Muestra todos los usuarios de la BD
- Practica con algún test más de tu invención, mapea otras relaciones ...

Recursos

- Jakarta Persistence API:
  - https://jakarta.ee/specifications/persistence/3.0/jakarta-persistence-spec-3.0.html
- Hibernate Guide:
  - https://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html