```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/animaldb
spring.datasource.username=2dawa
spring.datasource.password=2daw2324
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

Los datos de conexión con la BD se definen en el archivo “application.properties” del directorio “resources”. Ejemplo:

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:<port>/<DBName>
spring.datasource.username=<DBUserName>
spring.datasource.password=<password>
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#spring.jpa.show-sql: true
```

Los valores entre <> son particulares de cada implementación y se han de especificar

Arquitectura de la Persistencia

Modelos

Los modelos tendrán la anotación @Entity para que Spring pueda relacionarlos con las tablas de la BD.

Cuando la clase que representa un modelo es un *bean* de tipo @Entity, es necesario que al menos uno de sus atributos funcione como clave primaria.

El atributo que se usará como clave primaria tendrá dos anotaciones.

- @Id → para especificar que el atributo es la clave primaria
- @GeneratedValue → para indicar que la gestión de este campo la realizará el framework y no los desarrolladores.

Arquitectura de la Persistencia

Modelos

Por defecto, Spring Data JPA utilizará los nombres de las clases y atributos de las entidades para crear/mapear las tablas y columnas de la BD.

Sin embargo, **es posible especificar distintos nombres de tablas y columnas** para mapearlos con las entidades y sus atributos.

Para ello se utilizarán las siguientes anotaciones encima de la definición de la clase y de cada atributo:

- **@Table(name="<table_name>")** → mapea la clase con la tabla especificada.
- **@Column(name="<column_name>")** → mapea el atributo con la columna especificada.

Arquitectura de la Persistencia

Servicios

Definen las operaciones CRUD de las entidades.

Será buena práctica definir una interfaz de los servicios donde se definan las cabeceras de los métodos que se han de implementar.

Se podrá tener 1 o más implementaciones de una interfaz, según necesidades.

Se podrá definir una implementación como principal con la anotación @Primary.

Los servicios se han de inyectar en los controladores con @Autowired

Arquitectura de la Persistencia

Repositorios

En ellos reside toda la “magia” de Spring Data JPA para facilitar las operaciones de recuperación y persistencia de los datos de los modelos.

Existirá un repositorio por cada entidad del proyecto.

Son interfaces que extienden de la clase “JpaRepository”, indicando:

- Nombre de la entidad con la que trabajan
- Tipo de dato de la clave primaria de la entidad

Estas interfaces no necesitan definir explícitamente las operaciones básicas, pero **pueden tener definiciones de operaciones complejas** con *query keywords* (Guía).

Los repositorios se han de inyectar en el servicio de la entidad correspondiente.

Arquitectura de la Persistencia

Repositorios: Operaciones Básicas

Algunos de los métodos que ofrecen por defecto estos elementos son:

- findAll() : recupera todos los elementos de una entidad
- save() : almacena, o actualiza, un elemento de una entidad
- findById().orElse() : Recupera un elemento por su Id, o sino devuelve el elemento especificado.
- deleteById() : Elimina entidades por su Id.

Estos métodos, y muchos más, están definidos en detalle en la documentación oficial.

Arquitectura de la Persistencia

Repositorios: Operaciones Complejas

Como ya se ha comentado, los repositorios pueden tener definiciones de operaciones complejas con *query keywords* (Guía).

Gracias a esto se podrán realizar consultas concretas sobre los datos de una entidad en función de las necesidades de nuestro proyecto.

Un ejemplo de consulta *keyword*, para mostrar como se ha de incluir en el repositorio, sería la siguiente:

```
public interface ClaseRepository extends JpaRepository<Clase, Integer> {
    Clase findByName(String claseName);
}
```

Relaciones Entre Entidades

¡OJO con esto, que cascade y orphanremoval podrían ser redundantes, investigarlo!

Ejemplo de relación One to Many

Este tipo de relación se implementa de la siguiente forma en los modelos.

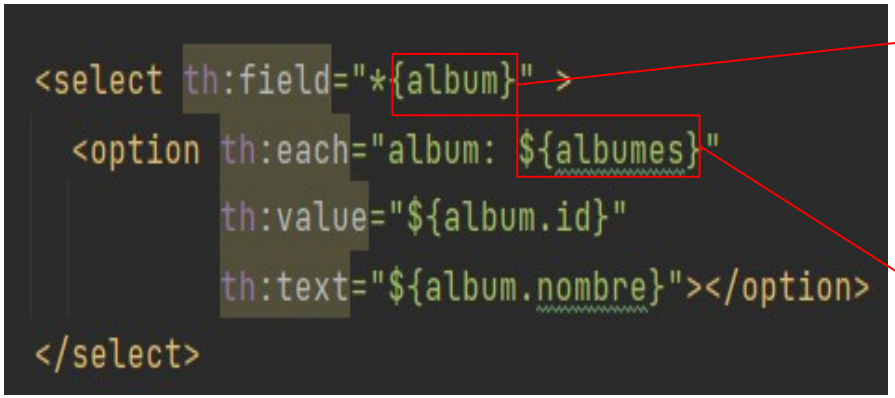
```java
@Entity
public class Album {

    //Id y otros atributos

    @OneToMany(mappedBy = "album",
            cascade = CascadeType.ALL,
            orphanRemoval = true)
    private List<Track> tracks;

    //Constructores
    //Getter/seeter
}
```

```java
@Entity
public class Track {

    //Id y otros atributos

    @ManyToOne
    private Album album;

    //Constructores
    //Getter/seeter
    //etc
}
```

Relaciones Entre Entidades

Ejemplo de relación One to Many

El desplegable del formulario para la creación del elemento del lado *many* de la relación (Track en este caso), tendrá un <select> similar al siguiente:



Nombre del atributo anotado con @ManyToOne, del lado *many* de la relación

Listado de elementos recuperado en el *controller* e incluido en el *Model*