Acceso a Datos

UT4. GENERACIÓN AUTOMÁTICA DE ENTIDADES

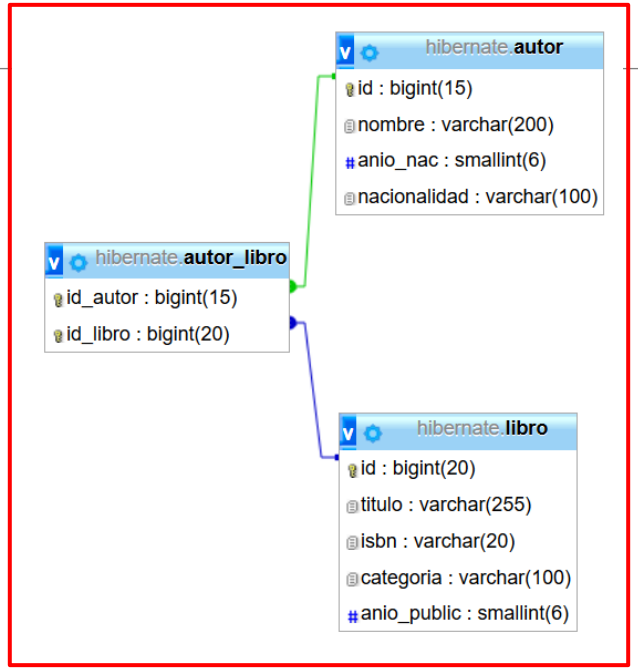
1. Introducción

- Eclipse, o Spring Tool Suite, nos ofrece otra funcionalidad que nos puede ahorrar algo de trabajo: para una base de datos dada, es capaz de escanear la misma, y generar las entidades suficientes para manejarla.

- Este esquema de trabajo puede ser útil en un contexto en el que ya tengamos una base de datos creada y queramos comenzar un proyecto JPA-Hibernate para gestionarla, o simplemente porque nuestra experiencia tecnológica con SQL sea dilatada, y por ende seamos capaces de aprovechar al máximo las funcionalidades del RDBMS concreto que vayamos a usar.

- Supongamos que tenemos una base de datos que gestiona autores y libros, y que se crearía a partir del siguiente script *bd-autor-libros.sql*:

1. Introducción



Para manejar esta base de datos, deberíamos tener dos entidades, Autor y Libro y una asociación ManyToMany bidireccional entre ellas.

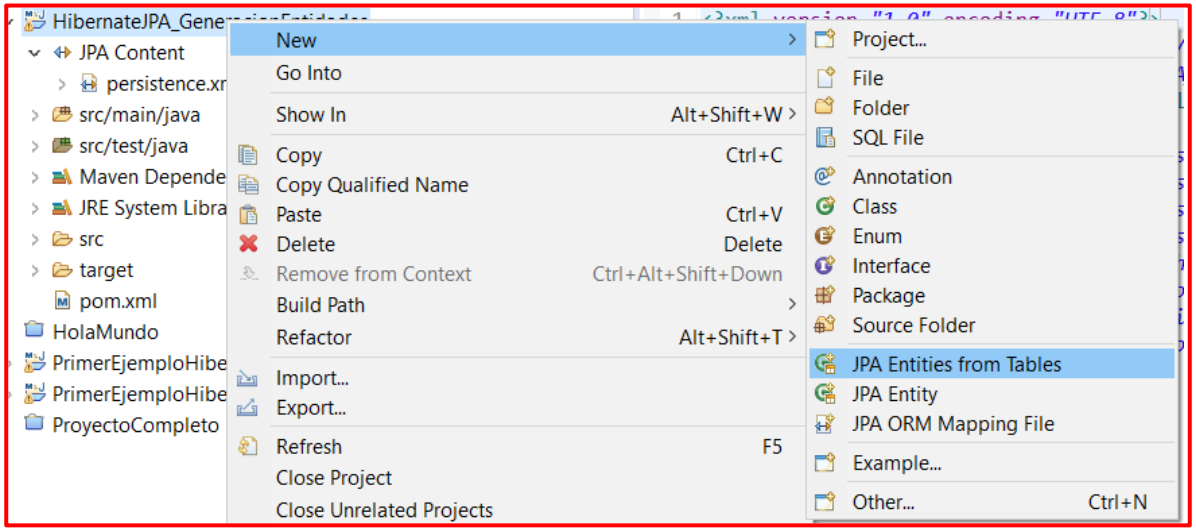
1. Introducción

Nos vamos a crear un proyecto Maven y le vamos añadir los Project Facets de JPA tal y como hemos visto en puntos anteriores a excepción de la propiedad hibernate.hbm2ddl que no la configuraremos y añadiremos al pom.xml las mismas dependencias de los proyectos anteriores: *hibernate-entitymanager y mysql-connector-java*

Antes de comenzar el proceso, tenemos que tener habilitada la conexión con la base de datos.

1. Introducción

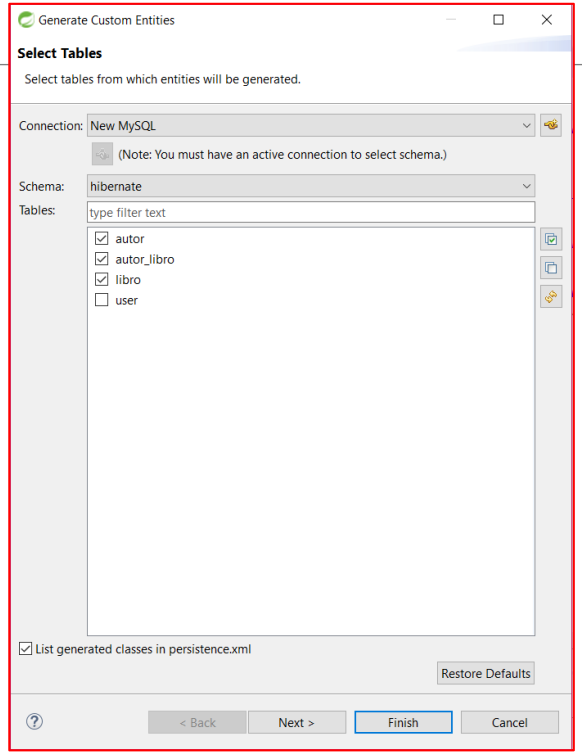
Para generar las entidades, pulsamos sobre el proyecto con el botón derecho, **New → JPA Entities from Tables**.

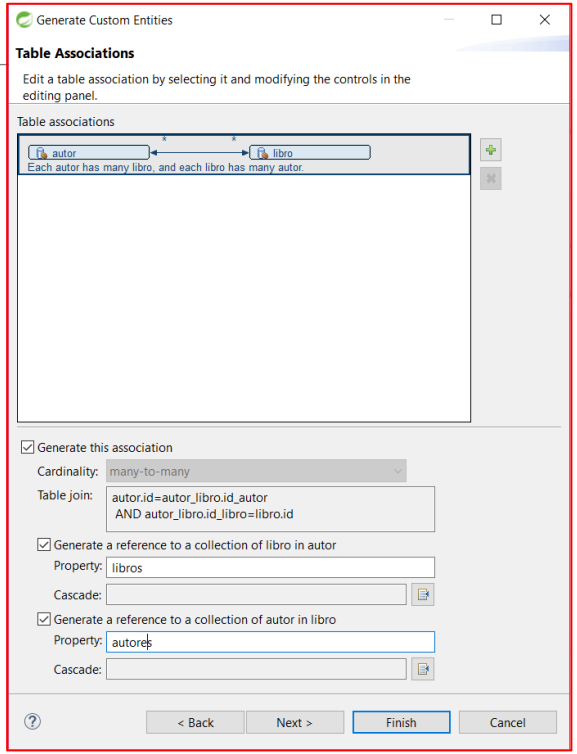


1. Introducción

Seleccionamos las tablas cuyas entidades vamos a generar (autor, autor_libro y libro) y pulsamos sobre el botón Next.

El asistente nos muestra en este paso un diagrama con las tablas y las asociaciones y su cardinalidad. En nuestro caso, aparecen **autor, libro** y una asociación **muchos-a-muchos** entre ellos de tipo **bidireccional**.





1. Introducción

Si necesitáramos añadir alguna asociación auxiliar, lo podemos hacer en este punto del asistente. El único cambio que haremos será cambiar el nombre de la colección al plural autores.

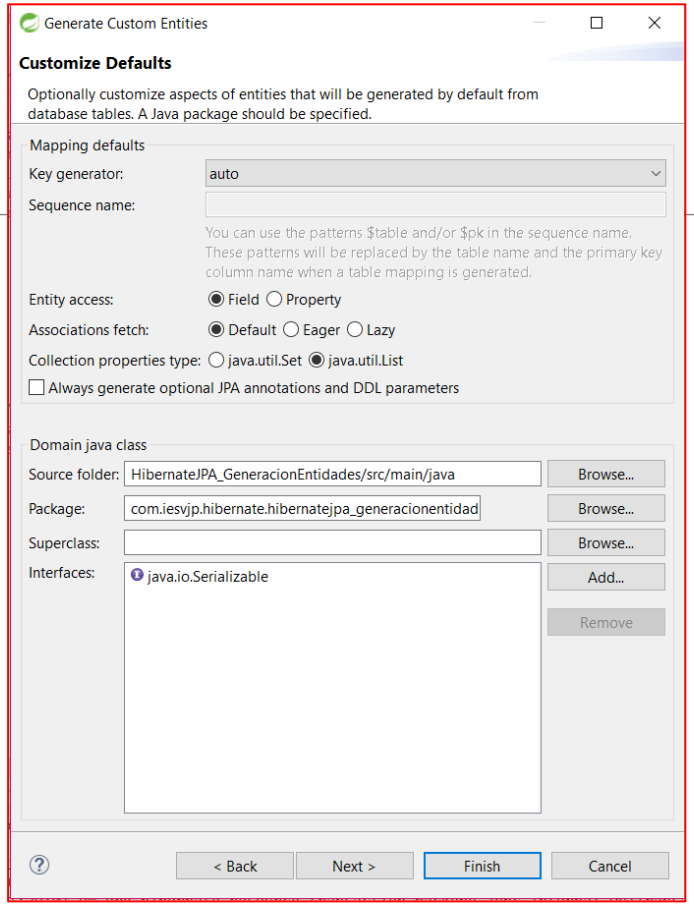
En el siguiente paso, seleccionamos algunos elementos importantes:

- El generador de identificadores. Lo pondremos a **identity**
- El tipo de acceso (a través de los atributos (field) o de los getters (property)).
- El tipo de fetching de las asociaciones (por defecto, ágil (eager) o perezoso (lazy)).

JPA: Eager vs Lazy

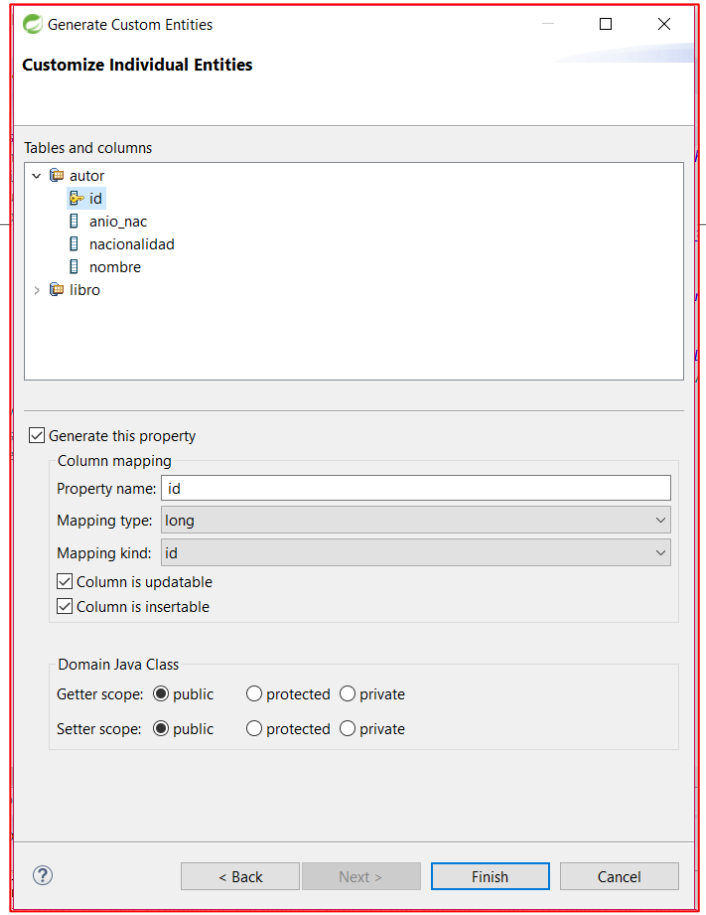
1. Introducción

Debajo, tenemos las opciones del paquete en el que vamos a ubicar las clases, si tienen algún tipo de clase base o deben implementar alguna interfaz (por defecto, *Serializable*).



1. Introducción

En el siguiente paso, podemos customizar cada una de las entidades que se van a generar. Por ejemplo los **id** en vez de String serán de tipo *long* y el **año de nacimiento** y de publicación lo cambiaremos a *java.util.Date*.



1. Introducción

La generación automática de entidades puede crear campos repetidos o meter los campos de todas las tablas que se llamen igual existentes dentro de nuestro esquema de MySQL:

https://stackoverflow.com/questions/51933308/eclipse-jpa-tools-generating-duplicate-fields-in-generate-entities-from-tables

Intenta borrar del esquema aquellas bases de datos que no necesites y tengan tablas con el mismo nombre que la BD de la cual vas a generar las entidades.

1. Introducción

Al finalizar el asistente, tendremos nuestras nuevas entidades disponibles:

```java
/**
 * The persistent class for the autor database table.
 *
 */
@Entity
@NamedQuery(name="Autor.findAll", query="SELECT a FROM Autor a")
public class Autor implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name="anio_nac")
    private Date anioNac;

    private String nacionalidad;

    private String nombre;

    //bi-directional many-to-many association to Libro
    @ManyToMany
    @JoinTable(
        name="autor_libro"
        , joinColumns={
            @JoinColumn(name="id_autor")
            }
        , inverseJoinColumns={
            @JoinColumn(name="id_libro")
            }
        )
    private List<Libro> libros;
```

```java
/**
 * The persistent class for the libro database table.
 *
 */
@Entity
@NamedQuery(name="Libro.findAll", query="SELECT l FROM Libro l")
public class Libro implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name="anio_public")
    private Date anioPublic;

    private String categoria;

    private String isbn;

    private String titulo;

    //bi-directional many-to-many association to Autor
    @ManyToMany(mappedBy="libros")
    private List<Autor> autores;
```

1. Introducción

Además, éstas han sido añadidas a nuestra unidad de persistencia. Y el archivo de persistencia debería quedar finalmente así:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
    xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
        http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="CreacionEntidades" transaction-type="RESOURCE_LOCAL">
        <class>org.hibernate.creacionentidades.Autor</class>
        <class>org.hibernate.creacionentidades.Libro</class>
        <properties>
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true" />
            <property name="hibernate.dialect"
                value="org.hibernate.dialect.MySQL5InnoDBDialect" />
            <property name="hibernate.connection.driver"
                value="com.mysql.jdbc.Driver" />
            <property name="javax.persistence.jdbc.url"
                value="jdbc:mysql://localhost:3306/hibernate" />
            <property name="javax.persistence.jdbc.user" value="root" />
            <property name="javax.persistence.jdbc.password" value="" />
            <property name="javax.persistence.jdbc.driver"
                value="com.mysql.jdbc.Driver" />
        </properties>
    </persistence-unit>
</persistence>
```

Dudas y preguntas

Práctica

A partir del script *company-sales.sql* créate la base de datos en MySQL y una vez importada, créate un nuevo un proyecto Maven y genera de forma automática las entidades de JPA.