Práctica 1 UD3

Alejandro Gata

2ºDAM

Este es el script de la base de datos completo:

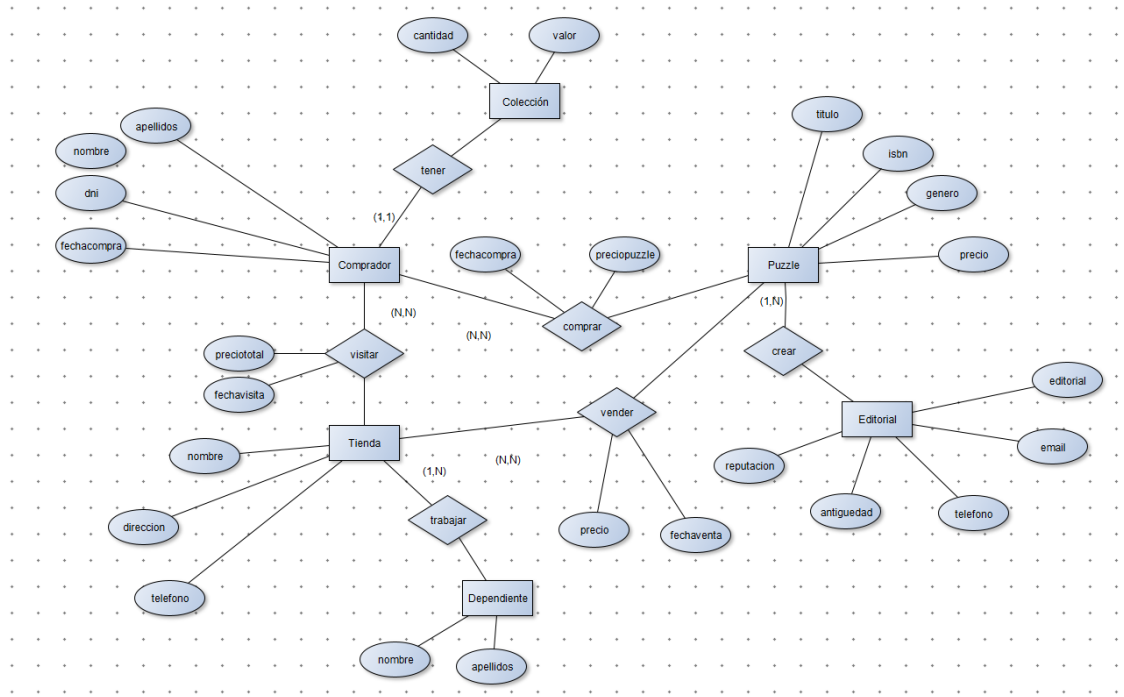
```
DROP DATABASE if exists tiendapuzzles;
CREATE DATABASE if not exists tiendapuzzles;
--
USE tiendapuzzles;
--
create table if not exists coleccion(
idcoleccion int auto_increment primary key,
cantidad int not null,
valor float not null);
--
create table if not exists tienda(
idtienda int auto_increment primary key,
nombre varchar(40) not null,
direccion varchar(150) not null,
telefono varchar(9));
--
create table if not exists editorial (
ideditorial int auto_increment primary key,
editorial varchar(50) not null,
email varchar(100) not null,
telefono varchar(9),
antiguedad int,
reputacion varchar(10),
web varchar(500));
--
create table if not exists tienda(
idtienda int auto_increment primary key,
nombre varchar(50) not null,
direccion varchar(100) not null,
telefono varchar(9));
--
create table if not exists comprador(
idcomprador int auto_increment primary key,
nombre varchar(50) not null,
apellidos varchar(150) not null,
```

```
dni varchar(10),
fechacompra date,
idcoleccion int,
idtienda int,
foreign key (idcoleccion) references coleccion(idcoleccion),
foreign key (idtienda) references tienda(idtienda));
--
create table if not exists puzzle(
idpuzzle int auto_increment primary key,
titulo varchar(50) not null,
isbn varchar(40) not null UNIQUE,
genero varchar(30),
precio float not null,
fechaedicion date,
ideditorial int,
idcomprador int,
idtienda int,
foreign key(ideditorial) references editorial(ideditorial),
foreign key(idcomprador) references comprador(idcomprador),
foreign key(idtienda) references tienda(idtienda));
--
create table if not exists comprador_puzzle(
idcomprador int,
idpuzzle int,
preciopuzzle int,
fechacompra date,
primary key(idcomprador,idpuzzle));
--
create table if not exists puzzle_tienda(
idpuzzle int,
idtienda int,
precio int,
fechaventa date,
primary key(idpuzzle, idtienda));
--
create table if not exists comprador_tienda(
idcomprador int,
idtienda int,
```

```
preciototal int,
fechavisita date,
primary key(idcomprador, idtienda));
--
delimiter ||
create function existeIsbn(f_isbn varchar(40))
returns bit
begin
        declare i int;
  set i = 0;
  while ( i < (select max(idpuzzle) from puzzles)) do
  if  ((select isbn from puzzle where idpuzzle = (i + 1)) like f_isbn) then return 1;
  end if;
  set i = i + 1;
  end while;
  return 0;
end; ||
delimiter ;
--
delimiter ||
create function existeNombreEditorial(f_name varchar(50))
returns bit
begin
        declare i int;
  set i = 0;
  while ( i < (select max(ideditorial) from editoriales)) do
  if  ((select nombre from editorial where ideditorial = (i + 1)) like f_name) then return 1;
  end if;
  set i = i + 1;
  end while;
  return 0;
end; ||
delimiter ;
--
delimiter ||
create function existeNombreComprador(f_name varchar(202))
returns bit
begin
```

```
        declare i int;
    set i = 0;
    while ( i < (select max(idcomprador) from comprador)) do
    if  ((select concat(apellidos, ', ', nombre) from comprador where idcomprador = (i + 1)) like f_name) then return 1;
    end if;
    set i = i + 1;
    end while;
    return 0;
end; ||
delimiter ;
```

Este es el diagrama de la base de datos tiendapuzzles



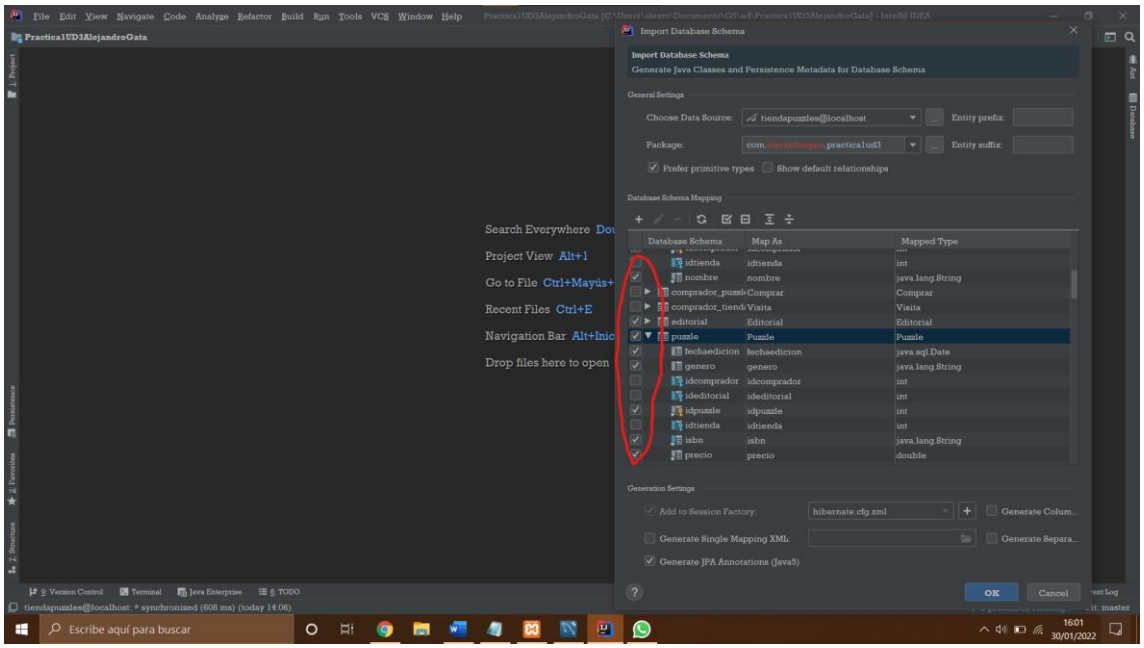
Esta es la tabla que describe el modelo entidad-relación

```
Comprador - Tienda -> N-N
Un comprador compra en muchas tiendas
En una tienda compran muchos compradores


Comprador - Colección -> 1-1
Un comprador tiene una colección
Una colección la tiene un comprador


Comprador - Puzzle -> N-N
Un comprador compra muchos puzzles
Un puzzle lo compran muchos compradores


Tienda - Dependiente -> 1-N
Un dependiente trabaja en una tienda
En una tienda trabajan muchos dependientes


Tienda - Puzzle -> N-N
Un puzzle se vende en muchas tiendas
En una tienda se venden muchos puzzles


Puzzle - Editorial -> 1-N
Un puzzle es creado en una editorial
Un editorial crea muchos puzzles
```

El primer paso es abrir MySQL, PHPMyAdmin o similares y eliminar las claves foráneas (foreign key) de las entidades que tienen varias foreign key, sustituyéndolas por un único id.

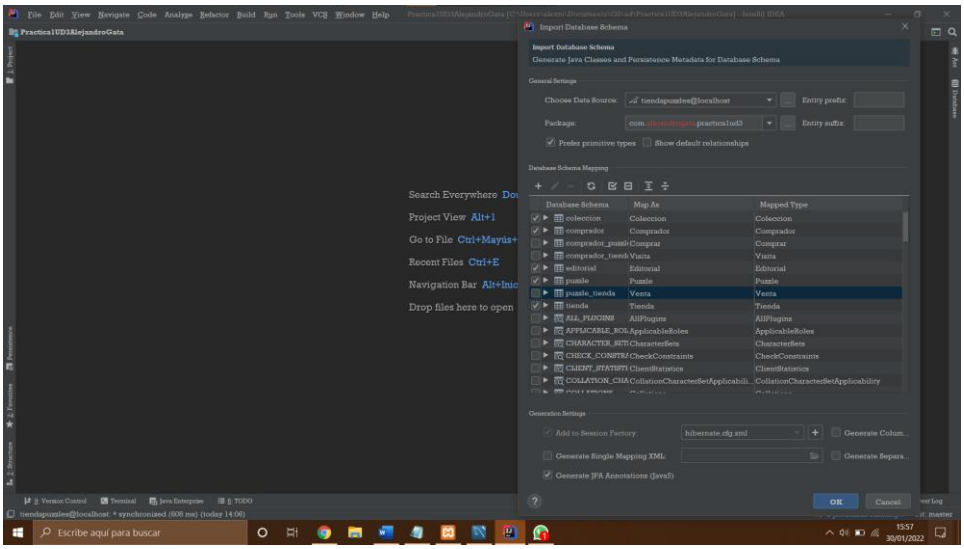
Una vez que tenemos esto, abrimos Intellij y seleccionamos las pestañas de Hibernate y SQL Support. En la siguiente ventana, seleccionamos las dos casillas que aparecen (*Create default hibernate configuration and main class* and *import database schema*).

Se abrirá la siguiente ventana:



Una vez ahí, deseleccionamos las claves foráneas como se muestra en la imagen.

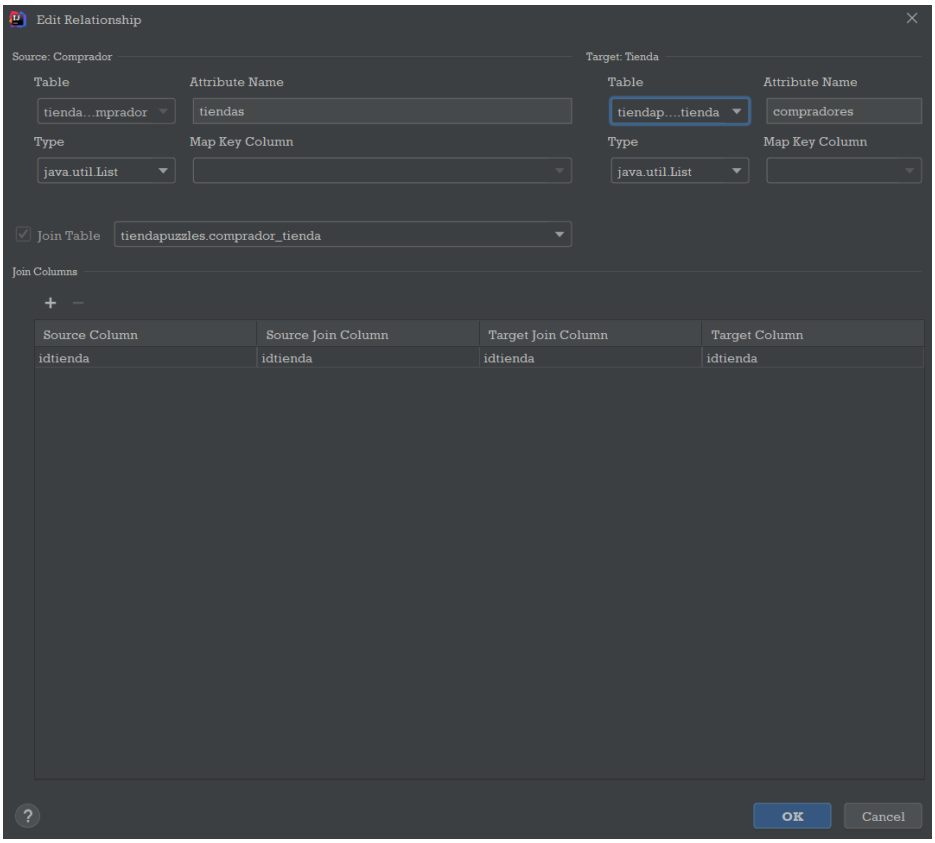
En la segunda columna de "*Map as*" escribimos el nombre que queremos que tengan las clases de Java.



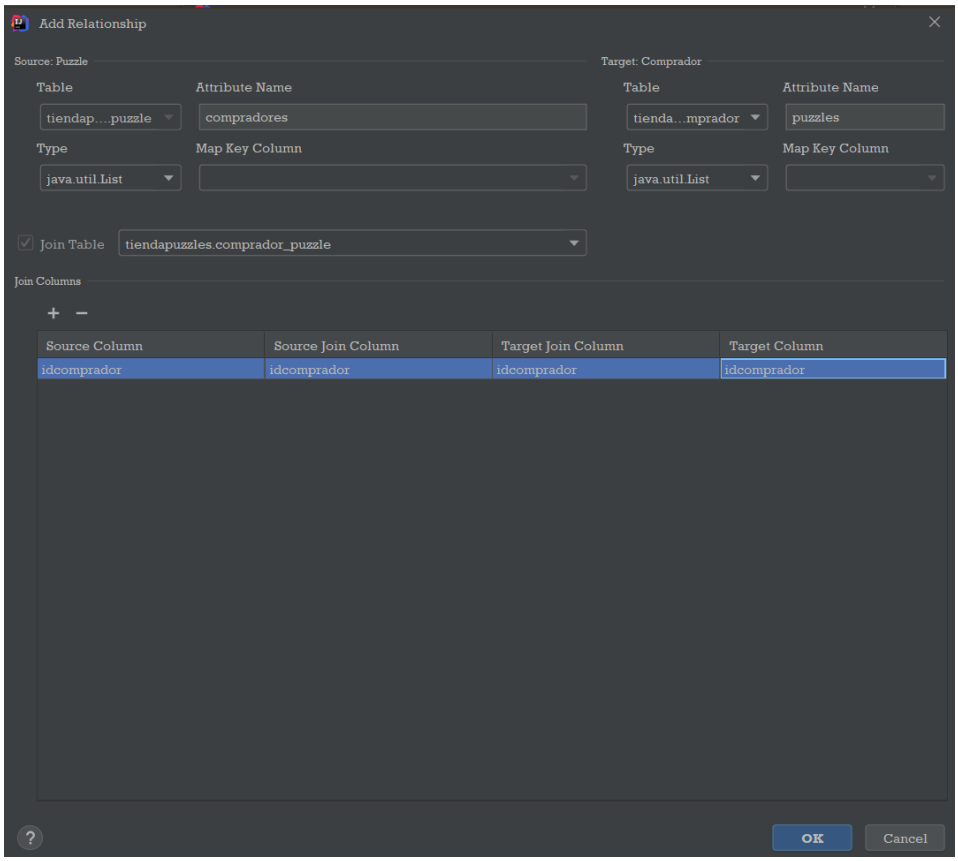
Ahora tenemos que relacionar las tablas entre ellas. Debemos crear una relación por cada una de las seis que aparecen en el MER.

Empezaremos uniendo la tabla comprador con la tabla tienda. Como hemos explicado, un comprador compra en varias tiendas y una tienda tiene varios compradores.

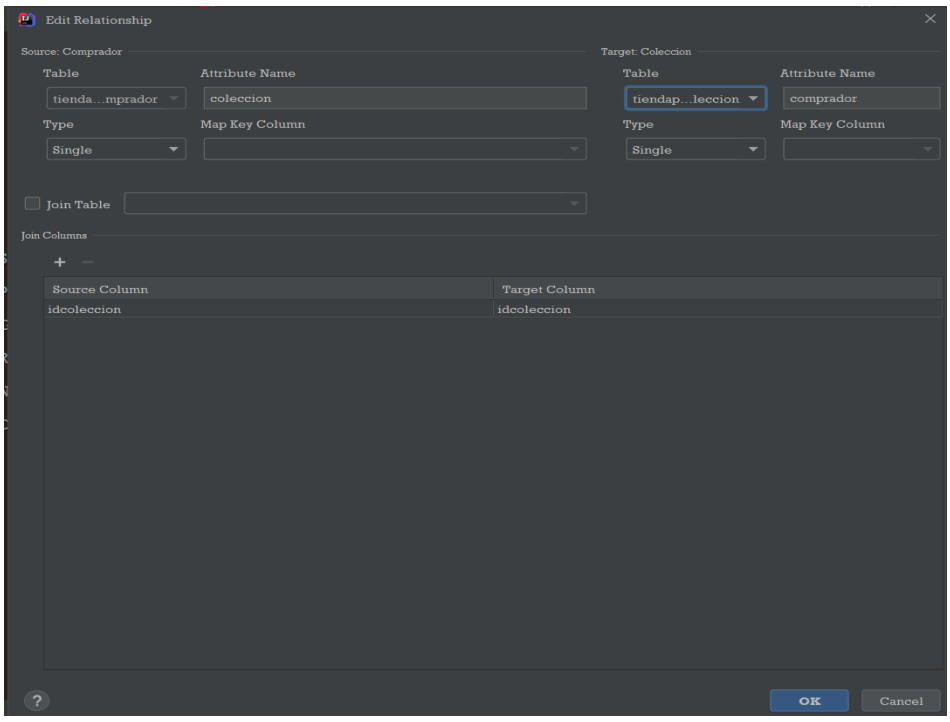
Comprador-tienda



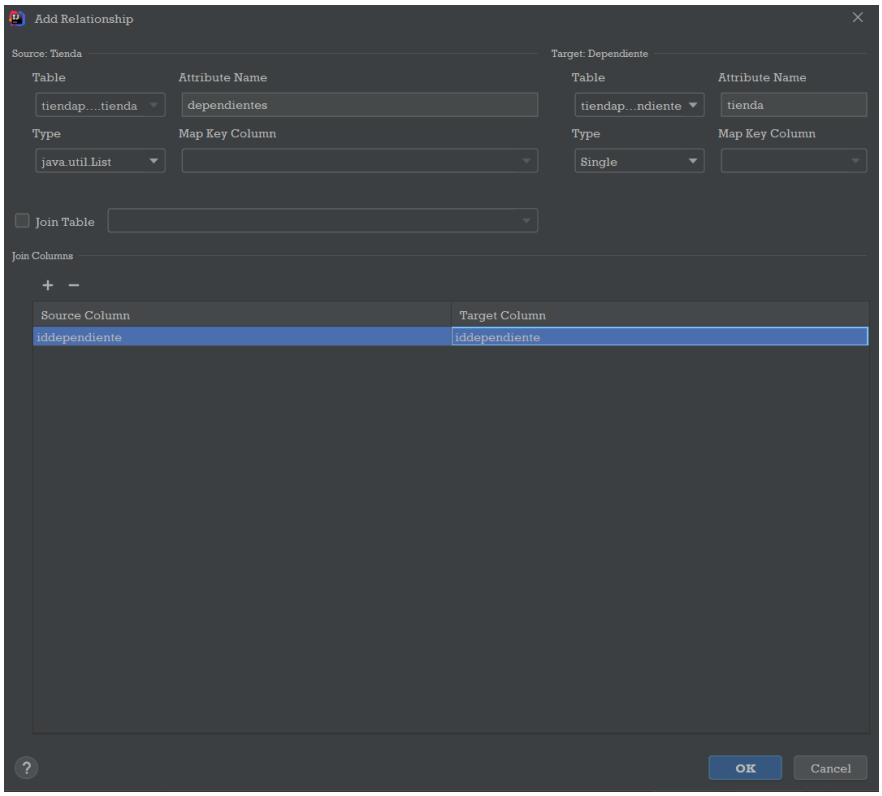
Puzzle-comprador



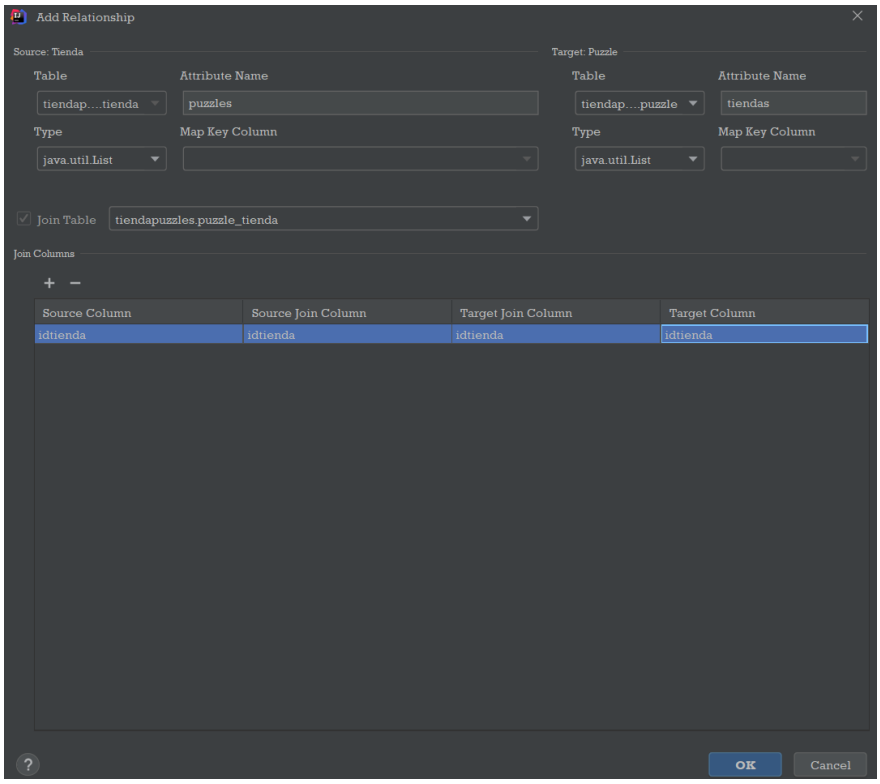
Colección-comprador



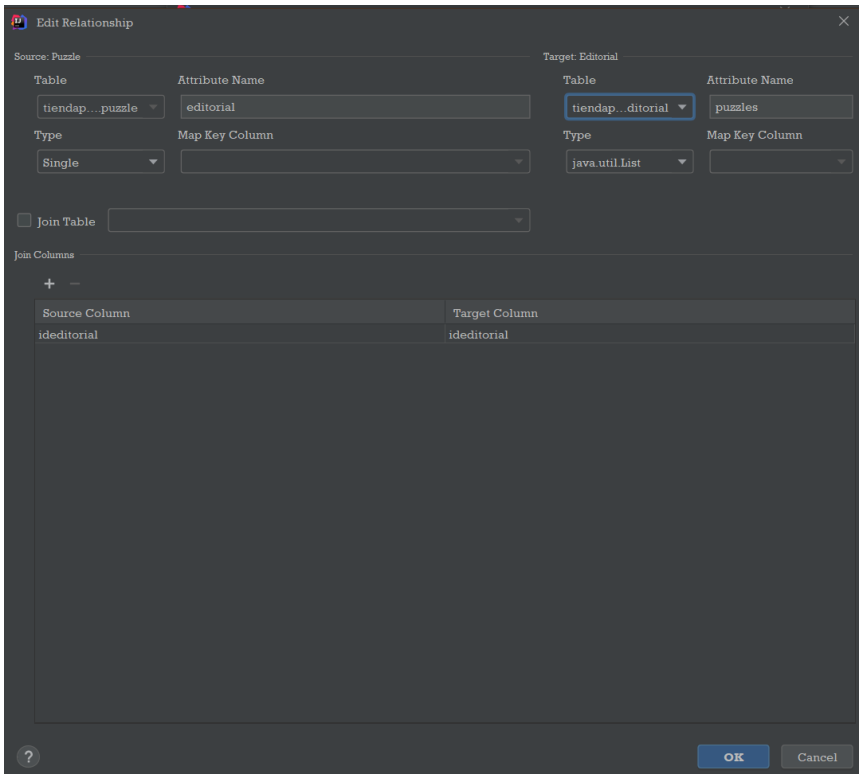
Tienda-dependiente



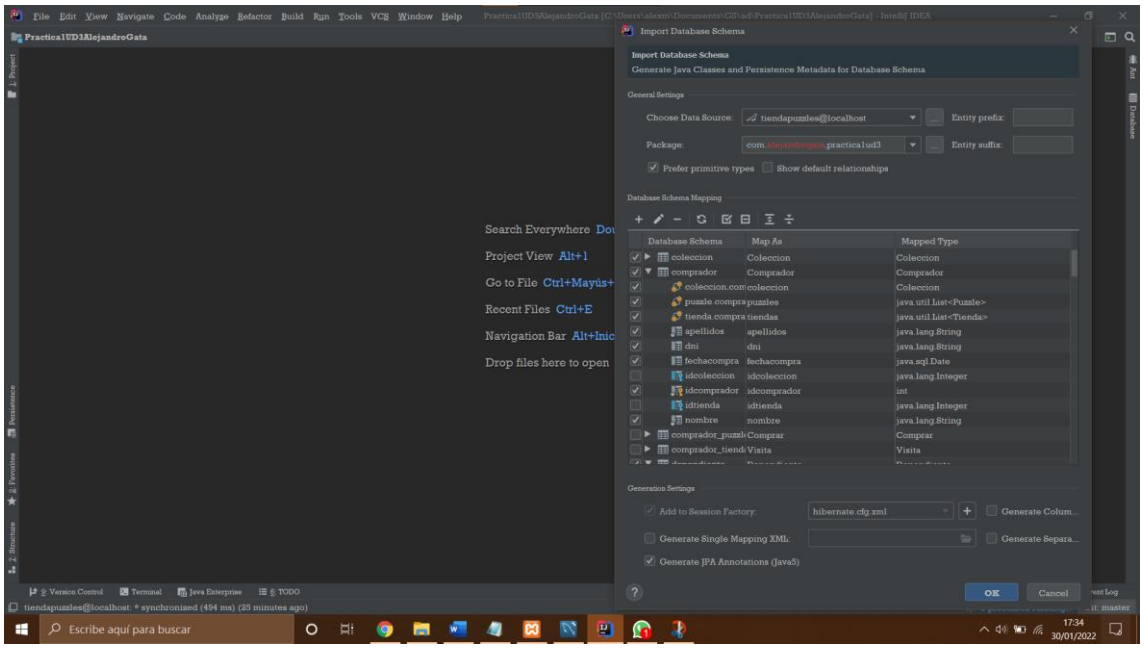
Tienda-puzzle



Puzzle-editorial



Con todo listo, le damos al botón de OK.



Ya nos ha generado el código JAVA.

Clase comprador:

```java
@ManyToMany
@JoinTable(name = "comprador_tienda", catalog = "", schema = "tiendapuzzles", joinColumns = @JoinColumn(name = "
public List<Tienda> getTiendas() { return tiendas; }

public void setTiendas(List<Tienda> tiendas) { this.tiendas = tiendas; }

@ManyToMany(mappedBy = "compradores")
public List<Puzzle> getPuzzles() { return puzzles; }

public void setPuzzles(List<Puzzle> puzzles) { this.puzzles = puzzles; }

@OneToOne
@JoinColumn(name = "idcoleccion", referencedColumnName = "idcoleccion")
public Coleccion getColeccion() { return coleccion; }

public void setColeccion(Coleccion coleccion) { this.coleccion = coleccion; }
}
```

@ManyToMany simboliza que un comprador puede comprar en muchas tiendas, y una tienda puede tener muchos compradores. Igual que en nuestro MER.

@OneToOne significa que un comprador tiene una colección, y una colección solo la tiene un comprador. Igual que en nuestro MER.

Clase dependiente:

```java
@ManyToOne
@JoinColumn(name = "iddependiente", referencedColumnName = "iddependiente", nullable = false)
public Tienda getTienda() { return tienda; }

public void setTienda(Tienda tienda) { this.tienda = tienda; }
}
```

@ManyToOne simboliza que una tienda puede tener muchos dependientes pero cada dependiente trabaja solo en una tienda (es cierto que en la vida real puede haber trabajadores pluriempleados, pero en la base de datos lo hemos reflejado como que cada dependiente tiene un único trabajo). Igual que en nuestro MER.

Clase puzzle:

```java
@ManyToMany
@JoinTable(name = "comprador_puzzle", catalog = "", schema = "tiendapuzzles", joinColumns = @JoinColumn(name = "idcomprad
public List<Comprador> getCompradores() { return compradores; }

public void setCompradores(List<Comprador> compradores) { this.compradores = compradores; }

@ManyToMany(mappedBy = "puzzles")
public List<Tienda> getTiendas() { return tiendas; }

public void setTiendas(List<Tienda> tiendas) { this.tiendas = tiendas; }

@ManyToOne
@JoinColumn(name = "ideditorial", referencedColumnName = "ideditorial")
public Editorial getEditorial() { return editorial; }
```

@ManyToMany significa que un comprador puede comprar muchos puzzles y un puzzle lo pueden comprar muchos compradores. Igual que en nuestro MER.

@ManyToMany, justo debajo del anterior, refleja que un puzzle se encuentra disponible a la venta en muchas tiendas a la vez, y en una tienda hay muchos puzzles diferentes. Igual que en nuestro MER.

@ManyToOne explica que una editorial tiene muchos puzzles, pero un puzzle pertenece a una única editorial. Igual que en nuestro MER.

Las 6 líneas del código JAVA reflejan las 6 relaciones de nuestro modelo entidad-relación. Todo está en orden.