l proceso de mapear las tablas de la base de datos a clases en nuestra capa de modelo se conoce como ORM (Object Relational Mapping), vamos a convertir las columnas de las tablas en atributos de las clases y las relaciones entre tablas pasan a ser atributos de tipo objeto o lista de objetos.

Antes de comenzar debemos configurar las dependencias para la base de datos que vamos a utilizar y el manejo/soporte del administrador de capas de persistencia (JPA) en el archivo pom.xml.

<!-- Dependencias de JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<version>42.1.4</version>

</dependency>

/\*\*

\*

\*/

package me.camilocastillo.reservation.model;

import java.util.Set;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;

import lombok.Data;

/\*\*

\* Clase que representa la tabla Cliente

\*

\*/

*@Data*

*@Entity* // da soporte para que represente una tabla de la base de datos

*@Table*(name = "cliente") // nombre de la tabla que debe mapear

public class Cliente {

*@Id*

*@GeneratedValue*(generator = "system-uuid")

*@GenericGenerator*(name="system-uuid", strategy="uuid2")

    private *String* idCli; // llave primaria de la tabla cliente

    private *String* nombreCli;

    private *String* apellidoCli;

    private *String* identificacionCli;

    private *String* direccionCli;

    private *String* telefonoCli;

    private *String* emailCli;

    /\* Set indica que es una coleccion de objetos por lo que sera tomada

     \* como una relacion de uno a muchos , es decir un cliente puede tener muchas

     \* reservas asociadas

     \*/

*@OneToMany*(mappedBy="cliente")

    private *Set*<Reserva> reservas;

    public Cliente() {

    }

}

Luego se crea la clase de reservas donde tiene el campo al que también hace referencia a la tabla clientes

/\*\*

\*

\*/

**package** me.camilocastillo.reservation.model;

**import** java.util.Date;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.Id;

**import** javax.persistence.JoinColumn;

**import** javax.persistence.ManyToOne;

**import** javax.persistence.Table;

**import** org.hibernate.annotations.GenericGenerator;

**import** lombok.Data;

/\*\*

\* Clase que representa la tabla reserva

\* **@author** jcgat

\*

\*/

@Data

@Entity

@Table(name="reserva")

**public** **class** Reserva {

@Id

@GeneratedValue(generator = "system-uuid")

@GenericGenerator(name="system-uuid", strategy="uuid2")

**private** String idRes;

**private** Date fechaIngresoRes;

**private** Date fechaSalidaRes;

**private** **int** cantidadPersonasRes;

**private** String descripcionRes;

@ManyToOne // notacion de la dependencia de muchos a uno

@JoinColumn(name="idCli")// llave primaria con la que hace referencia

**private** Cliente cliente;

}

Una vez mapeados los campos , pasamos a la configuración de a que datasource se va a referenciar y a que base de datos se va a conectar. Para ello seguimos los siguientes pasos

1. Crear archivo **application.yaml**  en la carpeta **src/main/resources**

Dentro de este con la siguiente configuración

spring:

datasource:

url: jdbc:postgresql://localhost:5433/postgres

username: postgres

password: 123456789

diver-class-name: org.postgresql.Driver

jpa:

hibernate:

ddl-auto: create-drop

generate-ddl: true <-- debe ir en falso en produccion

show-sql: true <-- debe ir en falso en produccion

finalmente se corre la aplicación para que la empaquete

Nota: el contenedor de postgres debe esta corriendo