

PROGRAMACION N-CAPAS LABORATORIO #5



Anotaciones de JPA @Entity, @Table, @Id, @Column Mapeo de clases en Java

Catedrático:

Lic. Juan Lozano

Instructores:

Karla Beatriz Morales Alfaro 00022516@uca.edu.sv
Sara Noemy Romero Menjivar 00030716@uca.edu.sv
Salvador Edgardo Campos Gómez 00117716@uca.edu.sv

Java Persistence API (JPA)

JPA nos permite establecer una correlación entre una base de datos relacional y un sistema orientado a objetos.

Esta correlación es llamada **ORM (Object Relational Mapping)**, la cual genera anotaciones sobre entidades.

Una entidad es una clase **POJO (Plain Old Java Object)** que debe proporcionar un método constructor por defecto, no debe ser final, y debe implementar serializable para accesos remotos

JPA está estructurado por spring con los siguientes componentes:

- **EntityManager:** Es una interfaz encargada de manejar las operaciones de persistencia de los objetos.
- **EntityManagerFactory:** Es el encargado de administrar y crear los objetos EntityManager, gestiona la conexión a la base de datos.
- **Entidad:** Son los objetos de persistencia, es cada registro que se encuentra en una tabla de una base de datos.
- **EntityTransaction:** Es una interfaz encargada de manejar las transacciones de las operaciones de persistencia.
- **Persistence:** Es una clase que permite obtener los objetos EntityManagerFactory.
- **Query:** Es una interfaz que permite hacer consultas en base a distintos criterios.

Ejemplo de laboratorio

Se creará una base de datos en PostgreSQL con una tabla poblada, y se mostrará su contenido en una aplicación web utilizando Spring Boot por medio de JPA e Hibernate.

- En PostgreSQL crear una base de datos llamada **"BDEscuela"**.
- Crear una tabla llamada **"estudiante"** (es importante nombrar correctamente ya que para el ejemplo del laboratorio se ha trabajado exactamente con esos nombres).
- Agregar en **"estudiante"** los siguientes campos:
 - id_estudiante (primary key).
 - nombre (varchar 30).
 - apellido (varchar 30).
 - edad (Integer).
 - estado Boolean.

Query Editor

Query History

1

select * from estudiante;

2

3

Data Output

4		id_estudiante	nombre	apellido	edad	estado
5	▲	integer	character varying (30)	character varying (30)	integer	boolean
6						
7						
8						

Poblar la tabla con mínimo 5 estudiantes.

Query Editor

Query History

1

select * from estudiante;

2

3

Data Output

4

	id_estudiante integer	nombre character varying (30)	apellido character varying (30)	edad integer	estado boolean
5	1	Karla	Morales	22	true
6	2	Ayleen	Alfaro	20	false
7	3	Wilfredo	Maqueen	20	true
8	4	Monica	Herrera	22	true
9	5	Alan	Brito	29	false

5

6

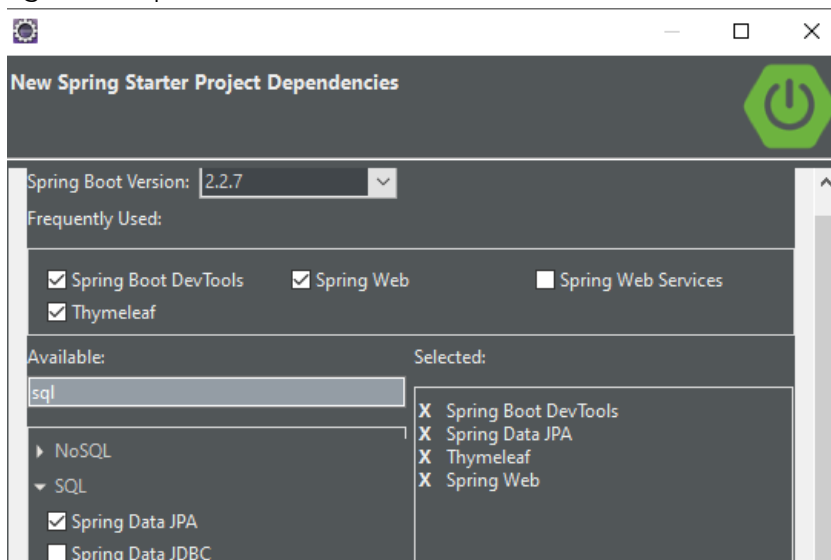
7

8

9

- Crear un proyecto Spring Boot.
- Agregar las siguientes dependencias:

figura 1. Dependencias.



- Agregar en **pom.xml** las dependencias de PostgreSQL y EntityManager:

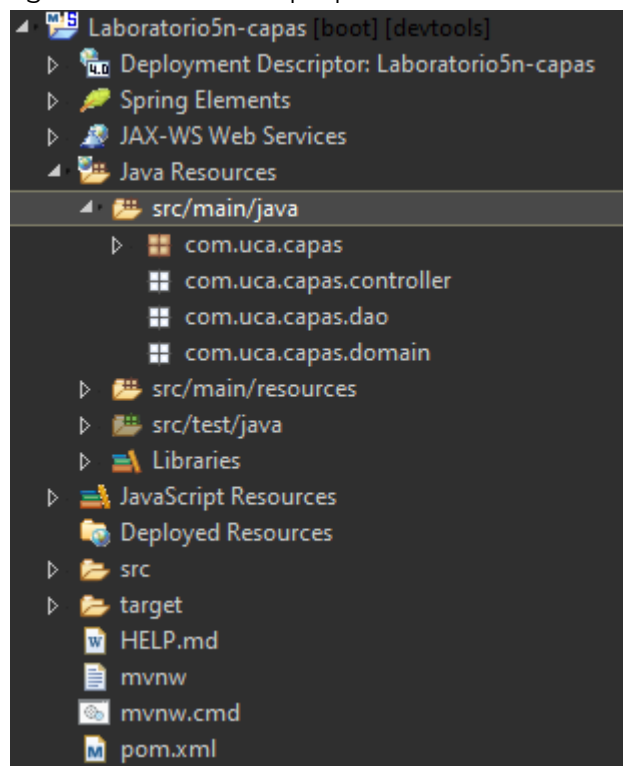
figura 1.2 Dependencia PostgreSQL y entityManager

```
<!-- Dependencia de PostgreSQL -->
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>42.2.11</version>
</dependency>

<!-- DEPENDENCIA PARA MANEJAR LOS ENTITYMANAGER -->
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>5.4.0.Final</version>
</dependency>
</dependencies>
```

- Crear los paquetes a manera de hacer el “esqueleto” del proyecto.

figura 2. Creación de paquetes.



- En el paquete **com.uca.capas** crear **JPAConfiguration.java**

figura 3. JPAConfiguration.java

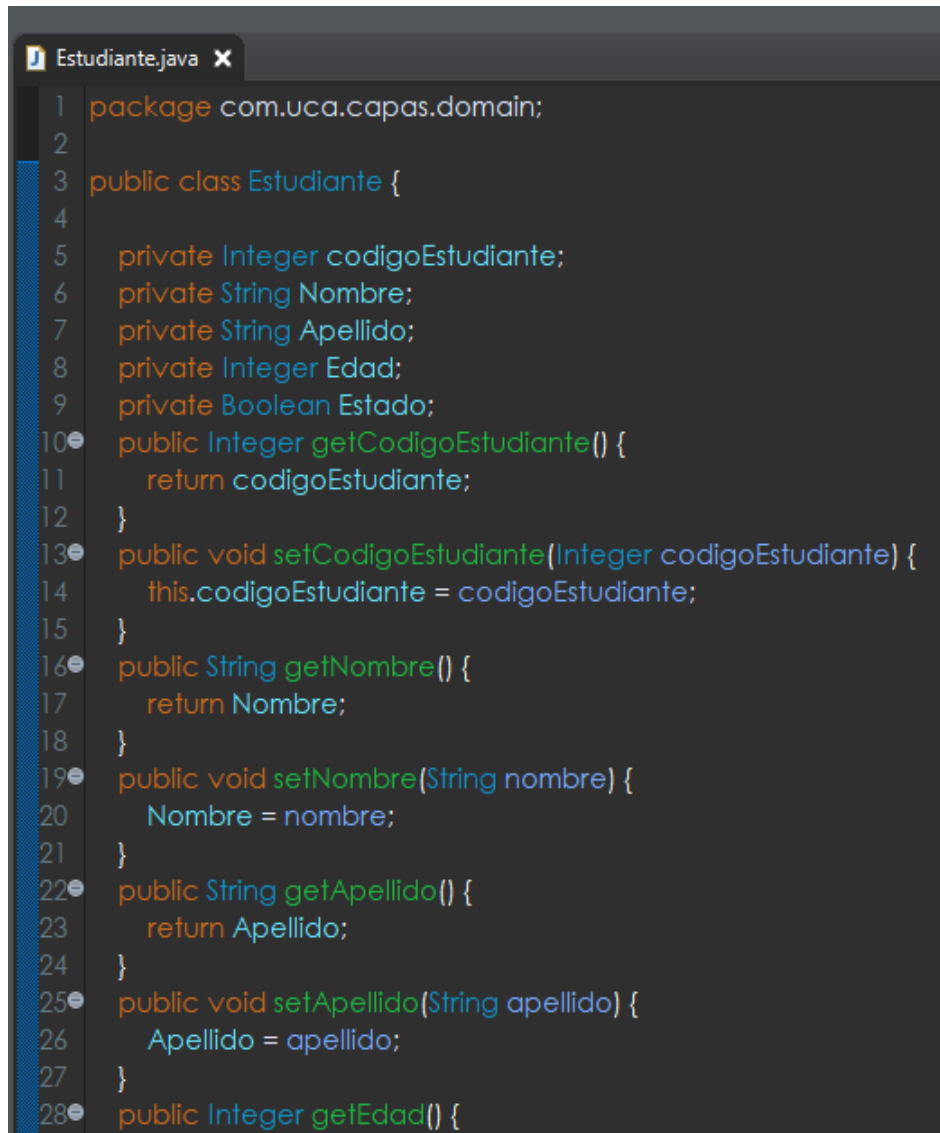
```
JPAConfiguration.java x
13
14 @Configuration
15 public class JPAConfiguration {
16
17     @Bean
18     public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
19         LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();
20         em.setDataSource(dataSource());
21         em.setPersistenceUnitName("capas");
22         em.setPackagesToScan("com.uca.capas.domain");
23
24         JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
25         em.setJpaVendorAdapter(vendorAdapter);
26         em.setJpaProperties(hibernateProperties());
27         return em;
28     }
29
30     @Bean
31     public DataSource dataSource(){
32         DriverManagerDataSource dataSource = new DriverManagerDataSource();
33         dataSource.setDriverClassName("org.postgresql.Driver");
34         dataSource.setUrl("jdbc:postgresql://127.0.0.1:5432/BDEscuela");
35         dataSource.setUsername("--- SU USUARIO DE POSTGRESQL");
36
37         dataSource.setPassword("--SU PASSWORD DE POSTGRESQL");
38         return dataSource;
39     }
```

dataSource.setUrl("jdbc:postgresql://<Dirección ip>:<puerto>/<nombre_base")

```
Properties hibernateProperties() {
    Properties properties = new Properties();
    properties.setProperty("hibernate.show_sql", "true");
    properties.setProperty("hibernate.dialect", "org.hibernate.dialect.PostgreSQLDialect");
    return properties;
}
```

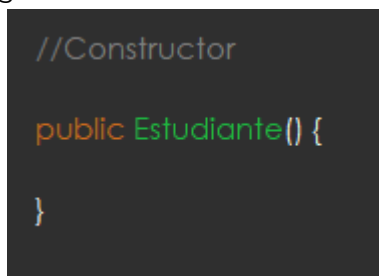
- Crear en **com.uca.capas.domain** el objeto **Estudiante.java** con los atributos y los setters y getters:

figura 3. Estudiante.java



```
1 package com.uca.capas.domain;
2
3 public class Estudiante {
4
5     private Integer codigoEstudiante;
6     private String Nombre;
7     private String Apellido;
8     private Integer Edad;
9     private Boolean Estado;
10    public Integer getCodigoEstudiante() {
11        return codigoEstudiante;
12    }
13    public void setCodigoEstudiante(Integer codigoEstudiante) {
14        this.codigoEstudiante = codigoEstudiante;
15    }
16    public String getNombre() {
17        return Nombre;
18    }
19    public void setNombre(String nombre) {
20        Nombre = nombre;
21    }
22    public String getApellido() {
23        return Apellido;
24    }
25    public void setApellido(String apellido) {
26        Apellido = apellido;
27    }
28    public Integer getEdad() {
```

figura 4. Constructor vacío



```
//Constructor
public Estudiante() {
}
}
```

Nota: Si no se agrega el constructor vacío, la aplicación dará error.

Hibernate: select * from public.estudiante
org.springframework.orm.jpa.JpaSystemException: No default constructor for entity: :
com.uca.capas.domain.Estudiante; nested exception is
org.hibernate.InstantiationException: No default constructor for entity: :
com.uca.capas.domain.Estudiante

- Crear una función delegate el cual en lugar de retornar “true” o “false” retorna si el estudiante es “Activo” o “Inactivo”
 - **Los métodos “delegate” son como cualquier método getter, pero con lógica por dentro**

figura 5. Funciones Delegate

```
//Funciones delegate
public String getEstadoDelegate() {
    if(this.Estado == null) return "";
    else {
        return Estado == true ? "Activo": "Inactivo";
    }
}
```

- Se procede a agregar las anotaciones de JPA @Entity, @Table, @Id, @Column
 - **@Entity:** Indica que se está trabajando con una entidad.
 - **@Table:** Se hace referencia al nombre de la tabla de la base de datos BDEstudiante
 - **@Id:** Hace referencia a la llave primaria de la tabla.
 - **@Column:** Se le indica la columna de la tabla que hará referencia al atributo del objeto Estudiante.

figura 6. Anotaciones de JPA

```
@Entity
@Table(schema="public",name="estudiante")
public class Estudiante {

    @Id
    @Column(name="id_estudiante")
    private Integer codigoEstudiante;

    @Column(name="nombre")
    private String Nombre;

    @Column(name="apellido")
    private String Apellido;

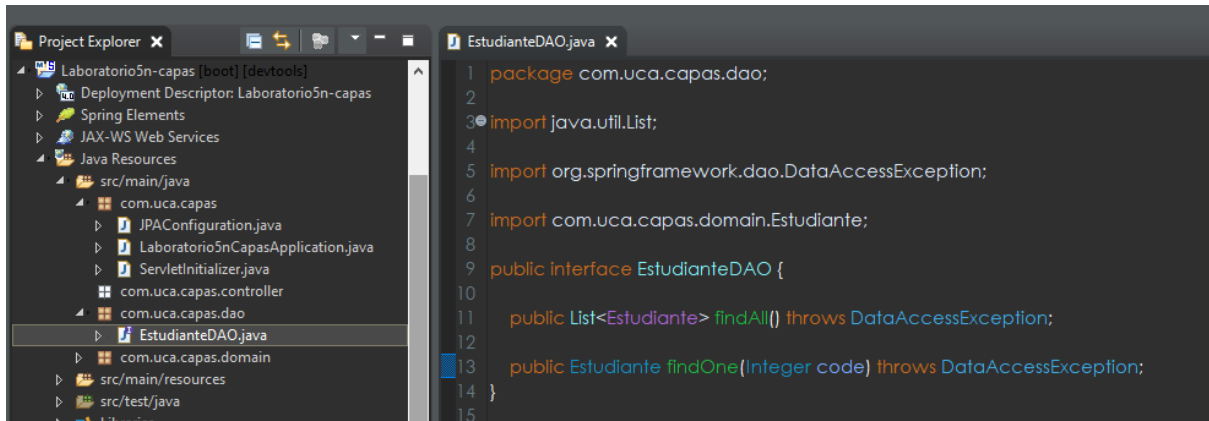
    @Column(name="edad")
    private Integer Edad;

    @Column(name="estado")
    private Boolean Estado;
```

Implementación de DAO con hibernate

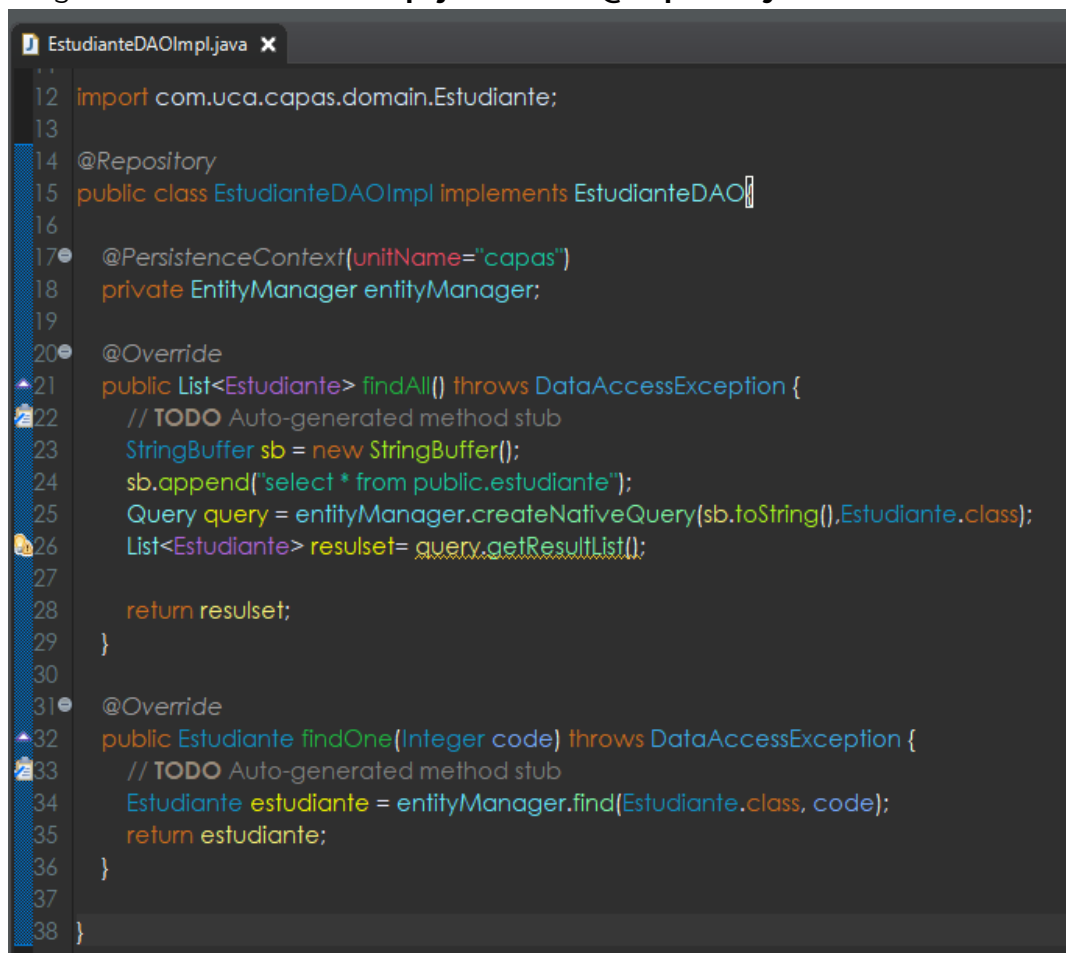
- En el paquete **com.uca.capas.dao** crear:
 - Interfaz EstudianteDAO.
 - Implementación de interfaz EstudianteDAO.

figura 7. New Interface -> EstudianteDAO



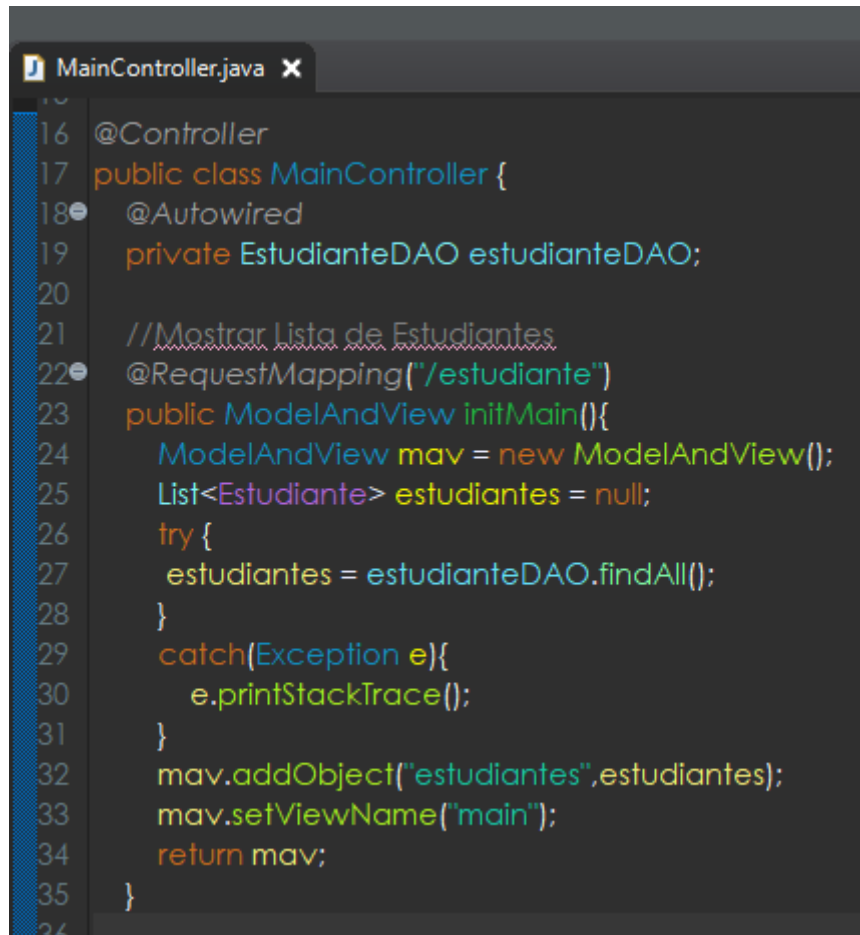
- **findAll():** Devolverá una lista de estudiantes que contendrá a todos los estudiantes que se encuentren en la tabla **estudiante**, como objetos de tipo Estudiante.
- **findOne():** Devolverá un objeto de tipo Estudiante, el cual recibirá de parámetro el valor de la llave primaria (un integer).

figura 8. **EstudianteDAOImpl.java** Incluir **@Repository**



- Crear el controlador **MainController.java** en el cual se va a tomar desde EstudianteDAO para acceder a los datos y enviarlos a la capa de **"Vista"** en donde se mostrarán los datos de la base de datos.

figura 9. **MainController.java** Mostrar todos los estudiantes

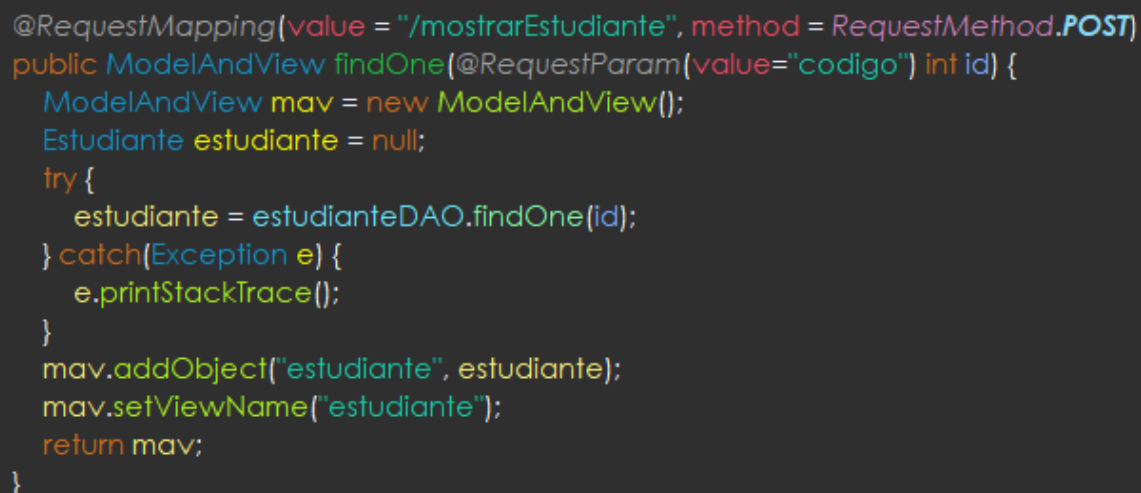


```

16 @Controller
17 public class MainController {
18     @Autowired
19     private EstudianteDAO estudianteDAO;
20
21     //Mostrar Lista de Estudiantes
22     @RequestMapping("/estudiante")
23     public ModelAndView initMain(){
24         ModelAndView mav = new ModelAndView();
25         List<Estudiante> estudiantes = null;
26         try {
27             estudiantes = estudianteDAO.findAll();
28         }
29         catch(Exception e){
30             e.printStackTrace();
31         }
32         mav.addObject("estudiantes",estudiantes);
33         mav.setViewName("main");
34         return mav;
35     }
36

```

figura 10. **MainController.java** Mostrar un estudiante por llave primaria.



```

@RequestMapping(value = "/mostrarEstudiante", method = RequestMethod.POST)
public ModelAndView findOne(@RequestParam(value="codigo") int id) {
    ModelAndView mav = new ModelAndView();
    Estudiante estudiante = null;
    try {
        estudiante = estudianteDAO.findOne(id);
    } catch(Exception e) {
        e.printStackTrace();
    }
    mav.addObject("estudiante", estudiante);
    mav.setViewName("estudiante");
    return mav;
}

```

- Por último, crear en **template** las vistas donde se mostrará la lista de estudiantes almacenados en la base de datos, y donde se mostrará un estudiante que se buscará por llave primaria.

figura 11. **main.html**

```

main.html x
<!-- HTML5 doctype -->
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Lista de estudiantes</title>
6 </head>
7 <body>
8 <table>
9 <thead>
10 <th>Nombre</th>
11 <th>Apellido</th>
12 <th>Edad</th>
13 <th>Estado</th>
14 </thead>
15 <th:block th:each="estudiante, row: ${estudiantes}">
16 <tr>
17 <th th:text="${estudiante.codigoEstudiante}"/>
18 <th th:text="${estudiante.nombre}"/>
19 <th th:text="${estudiante.apellido}"/>
20 <th th:text="${estudiante.edad}"/>
21 <th th:text="${estudiante.estadoDelegade}"/>
22 </tr>
23 </th:block>
24 </table>
25 <br>
26 <form th:action="@{/mostrarEstudiante}" method="post">
27 <label>Buscar por codigo de estudiante: </label><input type="number" name="codigo"><br>
28 <input type="submit" value="Buscar">
29 </form>
30 </body>
31 </html>

```

figura 12. **estudiante.html**

```

estudiante.html x
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Mostrar estudiante por llave primaria</title>
6 </head>
7 <body>
8 <label>Estudiante con codigo</label> <label th:text="${estudiante.codigoEstudiante}"/></label>
9 <table>
10 <thead>
11 <th>Nombre</th>
12 <th>Apellido</th>
13 <th>Edad</th>
14 <th>Estado</th>
15 </thead>
16
17 <tr>
18 <th th:text="${estudiante.nombre}"/>
19 <th th:text="${estudiante.apellido}"/>
20 <th th:text="${estudiante.edad}"/>
21 <th th:text="${estudiante.estadoDelegade}"/>
22 </tr>
23 </table>
24 </body>
25 </html>

```

RESULTADO:

← → ↻ ⓘ localhost:8080/estudiante

	Nombre	Apellido	Edad	Estado
1	Karla	Morales	22	Activo
2	Ayleen	Alfaro	20	Inactivo
3	Wilfredo	Maqueen	20	Activo
4	Monica	Herrera	22	Activo
5	Alan	Brito	29	Inactivo

Buscar por codigo de estudiante:

← → ↻ ⓘ localhost:8080/mostrarEstudiante

Estudiante con codigo 5

Nombre	Apellido	Edad	Estado
Alan	Brito	29	Inactivo

Referencias:

- [1] <https://spring.io/guides/gs/accessing-data-jpa/>
- [2] <https://www.oscarblancarteblog.com/tutoriales/java-persistence-api-jpa/>
- [3] <https://www.uv.es/grimo/teaching/SpringMVCv4PasoAPaso/part5.html>
- [4] Clase 12, Clase 13 Programación N-Capas

Tarea 100%:

Implementando JPA, Hibernate y la base de datos proporcionada (consultar guía para importar base de datos):

```
CREATE TABLE estudiante(  
    c_usuario serial NOT NULL PRIMARY KEY,  
    nombre character varying (50),  
    apellido character varying (50),  
    carne character varying(10),  
    carrera character varying(100)  
);
```

Deberán realizar un video de no más de 10 minutos en donde se visualice todo el proceso que siguieron para realizar una aplicación que permite guardar y visualizar Estudiantes.

La aplicación contiene lo siguiente:

- JpaConfiguration.java
- Dominio Estudiante.java con las anotaciones de JPA e Hibernate.
- EstudianteDAO.java y EstudianteDAOImpl.java que tendrán:
 - **Método para insertar un estudiante.**
 - **Método para obtener todos los estudiantes.**
- **index.html (Insertar datos de un estudiante)** -> Contendrá el formulario con validación Hibernate haciendo uso de "th:errors". El formulario debe contener:
 - Nombre.
 - Apellido.
 - Carne.
 - Carrera.
- **listado.html** -> Contendrá el listado de estudiantes obtenidos de la base de datos.

El proyecto debe iniciar en **"/inicio"** que lo redirigirá al formulario en donde se va a insertar un nuevo estudiante, cada vez que guarde debe volver al formulario, en **index.html** debe haber un botón que lleve a **"/listado"** para mostrar los estudiantes guardados.

En el vídeo deben mostrar la base de datos en pgAdmin4 y mostrar los resultados obtenidos luego de haber guardado varios estudiantes haciendo un "Select * from ESTUDIANTE".

---Para el método de insertar implementado en EstudianteDAOImpl.java se pueden auxiliar de:

```
@Transactional  
public void insert(Estudiante estudiante) throws DataAccessException {  
    entityManager.persist(estudiante);  
}
```

@Transactional: Es una anotación que se utiliza cuando hay una operación de Insert, Delete, Update. Para garantizar la atomicidad de la consulta.

entityManager.persist(<Object>): Recibe un objeto que persiste en la base de datos.

Además, deben anotar el campo que contiene el Id en el dominio con **@GeneratedValue**:

```
@Id
@Column(name="c_usuario")
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer cUsuario;
```

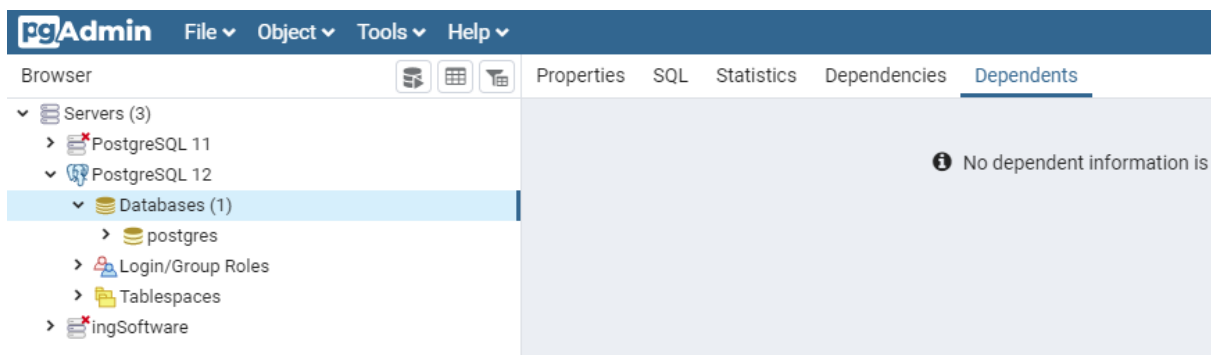
Esta anotación hace que el id sea incremental, hay formas más eficientes, pero para el ejercicio es más que suficiente.

Herramienta sugerida para creación de video:
OBS Studio.

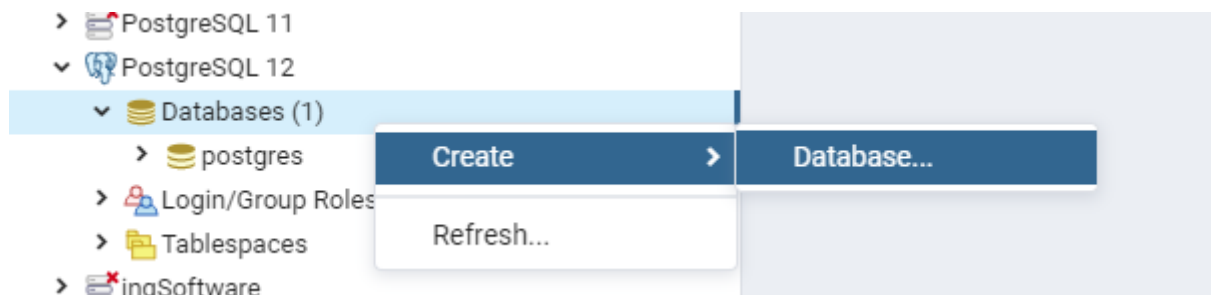
Install: <https://obsproject.com/es>

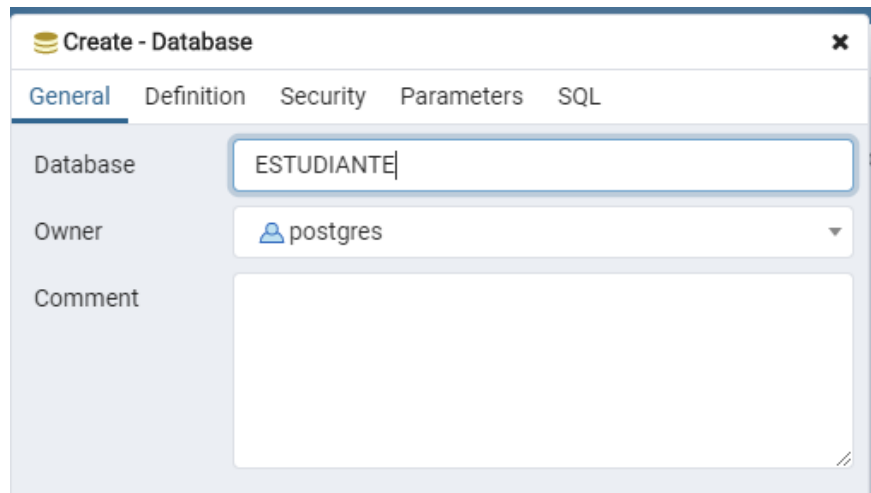
Pasos para importar la base de datos.

Ejecutamos nuestro pgAdmin y nos colocamos en las bases de datos de nuestra instalación:

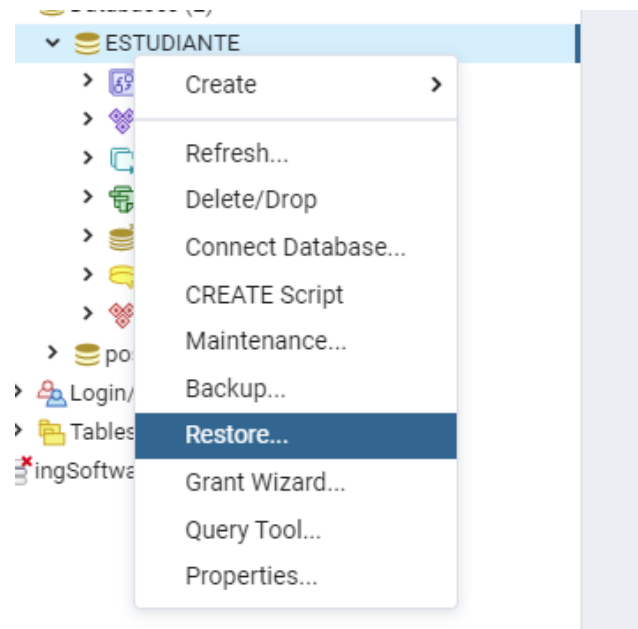


Luego creamos una base de datos vacía con el nombre de la base de datos que se va a importar en este caso es "ESTUDIANTE".

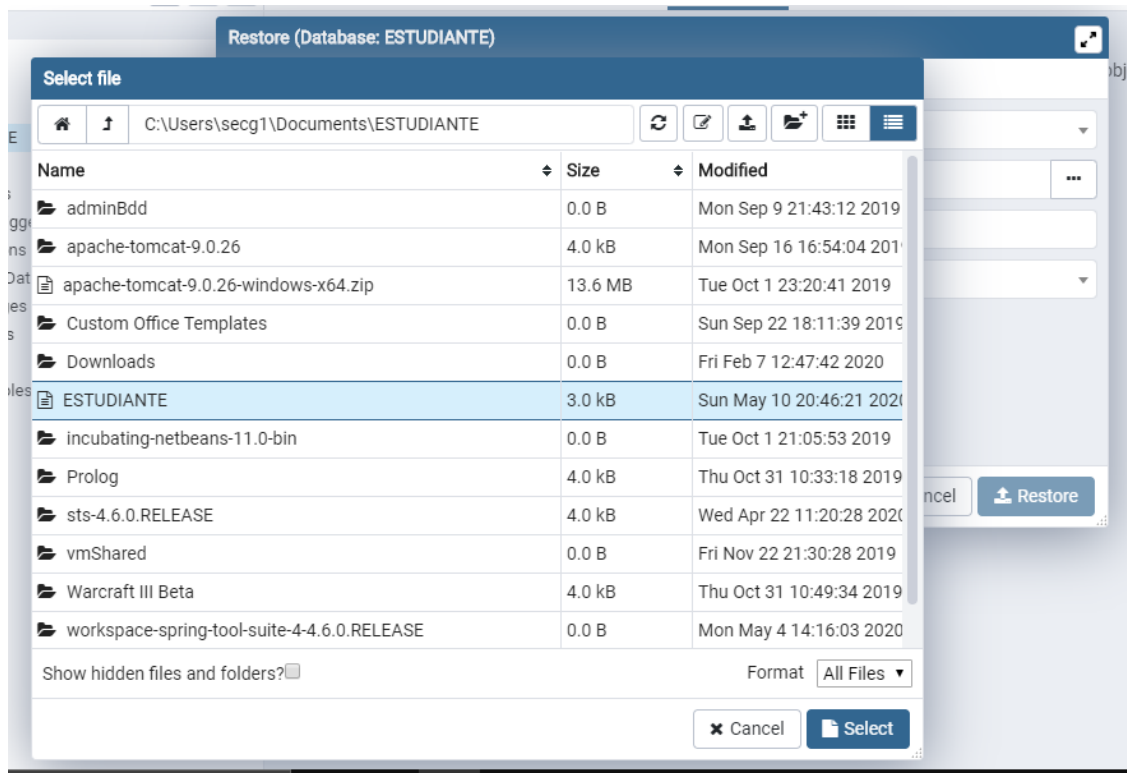





Damos en guardar. Una vez teniendo la base de datos vacía vamos a importar el archivo de respaldo de la base de datos.



En la siguiente ventana usar los 3 puntos para buscar nuestro archivo de respaldo, asegurarse de marcar "All Files".



Luego de seleccionar el archivo dar clic en “Restore” y probar si se importó correctamente.



ESTUDIANTE/postgres@PostgreSQL 12

Query Editor

Query History

1

SELECT * FROM ESTUDIANTE;

Data Output

Explain

Messages

Notifications

	<div>c_usuario</div> <div>[PK] integer</div>	<div>nombre</div> <div>character varying (50)</div>	<div>apellido</div> <div>character varying (50)</div>	<div>came</div> <div>character varying (10)</div>	<div>carrera</div> <div>character varying (100)</div>	
1	1	Salvador	Campos	00117716	Ing. Informatica	

De no funcionar la restauración (Ya que a veces hay problemas por compatibilidad entre versiones), ya que el ejemplo es sencillo pueden crear la tabla con la estructura dada.