



UNIVERSIDAD DE  
**CUNDINAMARCA**  
Generación Siglo 21

EXPLICACION BASE DE DATOS POSTGRESQL  
LÍNEA III

WILLIAM ALFREDO ROJAS RINCON

WILLIAM MATAALLANA  
DOCENTE

UNIVERSIDAD DE CUNDINAMARCA EXTENION CHÍA  
PROGRAMA DE INGENIERIA DE SISTEMAS  
FALCULTAD DE INGENIERIA  
2025

## Relaciones



### 1. Relación ManyToOne (Muchos a Uno)

En la clase Estudiante:

```
@ManyToOne
@JoinColumn(name="profesor_id", nullable = false)
private Profesor profesor;
```

- Significado: Muchos estudiantes pueden estar asociados a un mismo profesor.
- Implementación: Se crea una columna profesor\_id en la tabla Estudiante que es clave foránea a la tabla Profesor.
- Restricción: nullable = false indica que cada estudiante DEBE tener un profesor asignado.

En la clase Materia:

```
@ManyToOne
@JoinColumn(name = "profesor_id", nullable = false)
private Profesor profesor;
```

- Significado: Muchas materias pueden estar asociadas a un mismo profesor.
- Implementación: Similar al anterior, se crea profesor\_id en la tabla Materia.
- Restricción: Cada materia DEBE tener un profesor asignado.

### 2. Relación OneToMany (Uno a Muchos)

#### En la clase Profesor (con Estudiante):

```
@OneToMany(mappedBy = "profesor", cascade = CascadeType.ALL)
private List<Estudiante> estudiantes;
```

- Significado: Un profesor puede tener muchos estudiantes.
- Implementación: Es el lado inverso de la relación @ManyToOne en Estudiante.
- mappedBy: Indica que la relación está mapeada por el campo profesor en la clase Estudiante.
- cascade: Las operaciones (persist, remove, etc.) se propagarán a los estudiantes.

#### En la clase Profesor (con Materia):

```
@OneToMany(mappedBy = "profesor", cascade = CascadeType.ALL)
private List<Materia> materias;
```

- Significado: Un profesor puede impartir muchas materias.
- Implementación: Similar al anterior, es el lado inverso de la relación en Materia.

### 3. Relación ManyToMany (Muchos a Muchos)

#### En la clase Estudiante:

```
@ManyToMany(cascade = CascadeType.ALL)
@JoinTable(
    name = "estudiante_materia",
    joinColumns = @JoinColumn(name = "estudiante_id"),
    inverseJoinColumns = @JoinColumn(name = "materia_id")
)
private List<Materia> materias;
```

- Significado: Un estudiante puede estar inscrito en muchas materias y una materia puede tener muchos estudiantes.
- Implementación: Se crea una tabla intermedia llamada estudiante\_materia con:  
estudiante\_id: Referencia al estudiante  
materia\_id: Referencia a la materia

#### En la clase Materia:

```
@ManyToMany(mappedBy = "materias")
private List<Estudiante> estudiantes;
```

- Significado: Es el lado inverso de la relación en Estudiante.
- Implementación: mappedBy indica que la relación está controlada por el campo materias en Estudiante.

## Explicación BDPostgresApplication.java

Este es el punto de entrada principal de una aplicación Spring Boot que se conecta a una base de datos PostgreSQL. Vamos a analizarlo parte por parte:

### 1. Anotación @SpringBootApplication

```
@SpringBootApplication  
public class BdPostgresApplication {
```

Esta anotación combina tres anotaciones importantes:

@Configuration: Marca la clase como fuente de definiciones de beans

@EnableAutoConfiguration: Habilita la configuración automática de Spring Boot

@ComponentScan: Habilita el escaneo de componentes en el paquete actual y subpaquetes

### 2. Método main

```
public static void main(String[] args) {  
    loadEnv();  
    SpringApplication.run(BdPostgresApplication.class, args);  
}
```

Es el punto de entrada de la aplicación Java

Primero llama a loadEnv() para cargar variables de entorno.

Luego inicia la aplicación Spring Boot con SpringApplication.run()

### 3. Método loadEnv()

```
private static void loadEnv() {  
    Dotenv dotenv = Dotenv.load();  
    System.setProperty("BD_URL", dotenv.get("BD_URL"));  
    System.setProperty("BD_USERNAME", dotenv.get("BD_USERNAME"));  
    System.setProperty("BD_PASSWORD", dotenv.get("BD_PASSWORD"));  
}
```

Propósito: Carga variables de entorno desde un archivo .env y las establece como propiedades del sistema

Biblioteca Dotenv: Usa la librería dotenv-java para manejar variables de entorno

Proceso:

Dotenv.load() carga las variables del archivo .env

dotenv.get() obtiene los valores específicos

System.setProperty() establece estas variables como propiedades del sistema JVM



**Variables que se cargan:**

BD\_URL: URL de conexión a la base de datos PostgreSQL

BD\_USERNAME: Nombre de usuario para la conexión

BD\_PASSWORD: Contraseña para la conexión

**Link github: <https://github.com/william-1007-ctrl/EjercicioDBPostgres/tree/main>**