

GUÍA COMPLETA PARA EL EXAMEN: PATRONES DAO, SINGLETON Y FACTORY EN JAVA CON POSTGRESQL

Este documento es una **guía definitiva** para resolver cualquier caso de uso que puedan pedirte en el examen. Incluye:

- ✓ Explicación general de la estructura
 - ✓ Ejemplos de implementación
 - ✓ Lista de casos de uso comunes
 - ✓ Consultas SQL necesarias
 - ✓ Errores comunes y soluciones
-

1. Estructura del Proyecto

Este proyecto sigue una **arquitectura basada en DAO (Data Access Object)** y usa **Singleton** para manejar la conexión a la base de datos.

- ✓ **Pelicula y Genero** → Son las clases modelo con atributos y métodos **getter** y **setter**.
 - ✓ **PeliculaDAO y GeneroDAO** → Son los objetos de acceso a datos (DAO), encargados de ejecutar consultas SQL.
 - ✓ **Singleton** → Se encarga de la conexión con PostgreSQL.
 - ✓ **Factory** → Permite crear objetos sin necesidad de hacerlo en **Main.java**.
 - ✓ **Main.java** → Donde ejecutamos los **casos de uso**.
-

2. Creación de la Base de Datos

Si necesitas **crear todo desde cero**, aquí tienes los comandos SQL para PostgreSQL:

♦ 1. Crear la Base de Datos

```
CREATE DATABASE peliculasDB;
```

♦ 2. Crear la Tabla generos

```
CREATE TABLE generos (  
    id SERIAL PRIMARY KEY,  
    nombre VARCHAR(100) NOT NULL UNIQUE  
);
```

- ♦ 3. Crear la Tabla peliculas con Clave Foránea a generos

```
CREATE TABLE peliculas (  
    id SERIAL PRIMARY KEY,  
    titulo VARCHAR(200) NOT NULL,  
    director VARCHAR(100) NOT NULL,  
    anio INT NOT NULL,  
    genero_id INT,  
    FOREIGN KEY (genero_id) REFERENCES generos(id)  
);
```

- ♦ 4. Insertar Datos de Prueba

```
INSERT INTO generos (nombre) VALUES  
('Acción'), ('Comedia'), ('Drama'), ('Terror'), ('Ciencia Ficción');
```

```
INSERT INTO peliculas (titulo, director, anio, genero_id) VALUES  
('Gladiator', 'Ridley Scott', 2000, 1),  
('Titanic', 'James Cameron', 1997, 3),  
('Matrix', 'Lana Wachowski', 1999, 5);
```

3. Clases JAVA Principales

Clase **Pelicula**

```
package org.example;
```

```
public class Pelicula {  
    private int id;  
    private String titulo;  
    private String director;  
    private int anio;  
    private Genero genero;  
  
    public Pelicula(int id, String titulo, String director, int anio, Genero genero) {  
        this.id = id;  
        this.titulo = titulo;  
        this.director = director;  
        this.anio = anio;  
        this.genero = genero;  
    }  
  
    // Getters y Setters  
    public int getId() { return id; }
```

```

public String getTitulo() { return titulo; }
public String getDirector() { return director; }
public int getAnio() { return anio; }
public Genero getGenero() { return genero; }

public void setId(int id) { this.id = id; }
public void setTitulo(String titulo) { this.titulo = titulo; }
public void setDirector(String director) { this.director = director; }
public void setAnio(int anio) { this.anio = anio; }
public void setGenero(Genero genero) { this.genero = genero; }
}

```

Clase **PeliculaDAO** (Ejecuta las consultas SQL)

```

package org.example;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class PeliculaDAO implements DAO {
    @Override
    public void add(Pelicula pelicula) {
        String query = "INSERT INTO peliculas (titulo, director, anio, genero_id) VALUES (?, ?, ?, ?)";
        Connection conn = Singleton.getInstance().getConnection();
        try (PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setString(1, pelicula.getTitulo());
            ps.setString(2, pelicula.getDirector());
            ps.setInt(3, pelicula.getAnio());
            ps.setInt(4, pelicula.getGenero().getId());
            ps.executeUpdate();
            System.out.println("✅ Película agregada: " + pelicula.getTitulo());
        } catch (SQLException e) {
            System.out.println("❌ Error al agregar la película: " + e.getMessage());
        }
    }

    @Override
    public void findByGenero(String generoNombre) {
        String query = "SELECT p.id, p.titulo, p.director, p.anio FROM peliculas p " +
            "JOIN generos g ON p.genero_id = g.id WHERE g.nombre = ?";
        Connection conn = Singleton.getInstance().getConnection();
        try (PreparedStatement ps = conn.prepareStatement(query)) {

```

```

ps.setString(1, generoNombre);
try (ResultSet rs = ps.executeQuery()) {
    System.out.println("\n🎬 Películas del género: " + generoNombre);
    while (rs.next()) {
        System.out.println("  • " + rs.getInt("id") + " | "
            + rs.getString("titulo") + " | "
            + rs.getString("director") + " | "
            + rs.getInt("anio"));
    }
}
} catch (SQLException e) {
    System.out.println("❌ Error al filtrar películas por género: " + e.getMessage());
}
}
}

```



4. Casos de USO COMUNES


Estos son los **casos de uso más típicos** que te pueden pedir en el examen:

#	Caso de Uso	Método en DAO	Consulta SQL Interna
1	Insertar película	<code>add(Pelicula p)</code>	<code>INSERT INTO peliculas ...</code>
2	Buscar película por ID	<code>find(int id)</code>	<code>SELECT * FROM peliculas WHERE id = ?</code>
3	Actualizar película	<code>update(Pelicula p)</code>	<code>UPDATE peliculas SET ... WHERE id = ?</code>
4	Eliminar película	<code>delete(int id)</code>	<code>DELETE FROM peliculas WHERE id = ?</code>
5	Listar todas las películas	<code>findAll()</code>	<code>SELECT * FROM peliculas</code>
6	Filtrar por género	<code>findByGenero(String g)</code>	<code>SELECT * FROM peliculas WHERE genero = ?</code>

5. Errores Comunes y Soluciones

- ✓ **ERROR:** `This connection has been closed`
 - ♦ **SOLUCIÓN:** La conexión **se está cerrando antes de ejecutar las consultas**. Verifica que **Singleton no cierre la conexión antes de tiempo**.
- ✓ **ERROR:** `Table "peliculas" does not exist`
 - ♦ **SOLUCIÓN:** Asegúrate de haber creado la base de datos con `CREATE TABLE peliculas (...)`.
- ✓ **ERROR:** `Error near "s" en SQL`
 - ♦ **SOLUCIÓN:** Si un nombre tiene **apóstrofes (")**, usa `' '` en lugar de `\'` en SQL.

6. Resumen Final

- ✓ Usa **Singleton** para manejar la conexión con PostgreSQL.
 - ✓ Crea los objetos **Pelicula** con un **Factory** para mantener el código limpio.
 - ✓ Los **DAO** ejecutan las consultas SQL y devuelven los resultados.
 - ✓ Estructura clara y fácil de modificar en `Main.java`.
-  Con esta guía, estarás listo para cualquier cosa en el examen. Si te piden algo nuevo, **seguro estará basado en estos mismos patrones**. ¡Éxito! 🔥💪

Si en el examen te piden un **caso de uso que no has hecho**, lo que harás será:

- 1 Copiar la estructura de otro método en **PeliculaDAO**.
- 2 Modificar la consulta **SQL** para que haga lo que te pidan.
- 3 Adaptar los **parámetros de entrada y salida** según sea necesario.

Ejemplo de Caso de Uso Extraño en **PeliculaDAO**

? Imagina que en el examen te piden:

"Crea un método para obtener las **películas más antiguas** en la base de datos, es decir, aquellas que se estrenaron antes del año ingresado por el usuario."

💡 **Solución:** Creamos un nuevo método en **PeliculaDAO** llamado **findOldMovies(int anioLimite)**, que buscará todas las películas con un año menor al proporcionado.

 Código del Nuevo Caso de Uso en **PeliculaDAO.java**

```
// 🚀 Método para encontrar películas más antiguas que un año dado

public void findOldMovies(int anioLimite) {

    String query = "SELECT id, titulo, director, anio FROM peliculas
    WHERE anio < ?";

    Connection conn = Singleton.getInstance().getConnection();

    try (PreparedStatement ps = conn.prepareStatement(query)) {

        ps.setInt(1, anioLimite);

        try (ResultSet rs = ps.executeQuery()) {

            System.out.println("\n🎬 Películas más antiguas que el
            año " + anioLimite + ":");

            boolean encontrado = false;

            while (rs.next()) {

                encontrado = true;

                System.out.println("    • " + rs.getInt("id") + " | "
```

```

        + rs.getString("titulo") + " | "
        + rs.getString("director") + " | "
        + rs.getInt("anio"));
    }

    if (!encontrado) {
        System.out.println("⚠ No hay películas anteriores a
ese año.");
    }
}

} catch (SQLException e) {
    System.out.println("❌ Error al buscar películas antiguas: "
+ e.getMessage());
}
}

```

Ahora lo Usamos en **Main.java**

```

// Caso de uso: Buscar películas más antiguas que un año
ingresado por el usuario

Scanner scanner = new Scanner(System.in);

System.out.print("\n📅 Introduce un año para ver películas más
antiguas: ");

int anioBuscado = scanner.nextInt();

peliculaDAO.findOldMovies(anioBuscado);

```

Otros Casos de Uso Extraños que podrían pedirte

- ♦ 1. Contar cuántas películas hay por género

findMovieCountByGenre()

```
public void findMovieCountByGenre() {  
  
    String query = "SELECT g.nombre, COUNT(p.id) as total FROM  
    peliculas p " +  
  
        "JOIN generos g ON p.genero_id = g.id GROUP  
    BY g.nombre";  
  
    Connection conn = Singleton.getInstance().getConnection();  
  
    try (PreparedStatement ps = conn.prepareStatement(query);  
  
        ResultSet rs = ps.executeQuery()) {  
  
        System.out.println("\n🎬 Cantidad de películas por  
género:");  
  
        while (rs.next()) {  
  
            System.out.println("    • " +  
rs.getString("nombre") + ": " + rs.getInt("total") + "  
películas");  
  
        }  
  
    } catch (SQLException e) {  
  
        System.out.println("❌ Error al contar películas por  
género: " + e.getMessage());  
  
    }  
  
}
```


 **Uso en `Main.java`:**

```
peliculaDAO.findMovieCountByGenre();
```