Enunciado
Debes completar el desarrollo del sistema TechDAM implementando todas las funcionalidades requeridas y demostrando dominio de JDBC avanzado.
Objetivos de la Actividad
Esta actividad evalúa los siguientes Criterios de Evaluación (CE):

CE	Descripción	Peso
CE2.1	Establecer conexiones con BD usando controladores	2 pts
CE2.2	Uso de sentencias DDL, DML y de control	2 pts
CE2.3	Uso de procedimientos y funciones	2 pts
CE2.4	Control de transacciones	2 pts
CE2.5	Seguridad frente a inyección SQL	2 pts
CE2.6	Documentación de pruebas	2 pts
TOTAL		12 pts

Parte 1: Base de Datos (CE2.2)
Requisitos

1. Crear script SQL techdam_completo.sql que incluya:
   Creación de base de datos
   3 tablas: empleados , proyectos , asignaciones
   Claves primarias y foráneas
   Índices apropiados
   Al menos 5 registros de prueba en cada tabla
   2 procedimientos almacenados
   1 función almacenada

Ejemplo de Procedimiento Requerido

```
DELIMITER $$
CREATE PROCEDURE actualizar_salario_departamento(
IN p_departamento VARCHAR(50),
IN p_porcentaje DECIMAL(5,2),
OUT p_empleados_actualizados INT

)
BEGIN
UPDATE empleados
SET salario = salario * (1 + p_porcentaje / 100)
WHERE departamento = p_departamento AND activo = TRUE;
SET p_empleados_actualizados = ROW_COUNT();
```

END$$
DELIMITER ;
Parte 2: Código Java (CE2.1, CE2.5)
2.1 Configuración de Conexión
Archivo: DatabaseConfig.java o DatabaseConfigPool.java
Requisitos:
Usar HikariCP (pool de conexiones)
Configuración en archivo .properties (no hardcoded)
Método getConnection() que devuelve conexión del pool
Método close() para cerrar el pool
Puntuación:
DriverManager simple: 1 punto
HikariCP configurado: 2 puntos
2.2 Modelo de Datos
Archivos: Empleado.java , Proyecto.java , Asignacion.java
Requisitos:
Getters y setters para todos los campos
Constructor vacío y constructor con parámetros
Método toString() sobrescrito
Uso de tipos apropiados (BigDecimal para dinero, LocalDate para fechas)
2.3 Capa DAO (CE2.2, CE2.5)
Archivos: EmpleadoDAO.java , ProyectoDAO.java
Requisitos obligatorios:
CRUD completo para Empleados y Proyectos:
crear(T objeto) → devuelve ID generado
obtenerTodos() → devuelve List
obtenerPorId(int id) → devuelve Optional
actualizar(T objeto) → devuelve boolean

eliminar(int id) → devuelve boolean
PreparedStatement en TODAS las consultas (nunca Statement)
Try-with-resources o finally para cerrar recursos
Manejo apropiado de excepciones SQLException
Puntuación CE2.5 (Seguridad):
Statement con concatenación: 0 puntos ❌
PreparedStatement en 50% consultas: 1 punto
PreparedStatement en 100% consultas: 2 puntos ✅
Parte 3: Procedimientos Almacenados (CE2.3)
Archivo: ProcedimientosService.java
Requisitos:
Implementar al menos 2 métodos que invoquen procedimientos almacenados

Usar CallableStatement
Manejar parámetros IN y OUT correctamente
Mostrar resultados de los parámetros OUT
Ejemplo Requerido

```
public int actualizarSalariosDepartamento(String departamento, double
porcentaje) {
try (Connection conn = DatabaseConfig.getConnection();
CallableStatement cstmt = conn.prepareCall(
"{call actualizar_salario_departamento(?, ?, ?)}")) {
cstmt.setString(1, departamento);
cstmt.setBigDecimal(2, BigDecimal.valueOf(porcentaje));
cstmt.registerOutParameter(3, Types.INTEGER);
cstmt.execute();
return cstmt.getInt(3);
} catch (SQLException e) {
e.printStackTrace();
return -1;
}
}
```

Puntuación:
0-1 procedimientos: 1 punto
2+ procedimientos funcionando: 2 puntos
Parte 4: Transacciones (CE2.4)

Archivo: TransaccionesService.java
Debes implementar AL MENOS DOS de estas operaciones transaccionales:
Opción A: Transferencia de Presupuesto

```
public boolean transferirPresupuesto(int proyectoOrigenId,
int proyectoDestinoId,
BigDecimal monto) {
Connection conn = null;
try {
conn = DatabaseConfig.getConnection();
conn.setAutoCommit(false); // Iniciar transacción
// 1. Restar de origen
proyectoDAO.restarPresupuesto(conn, proyectoOrigenId, monto);
// 2. Sumar a destino
proyectoDAO.sumarPresupuesto(conn, proyectoDestinoId, monto);
conn.commit(); // Confirmar
return true;
} catch (SQLException e) {
```

```
if (conn != null) {
try {
conn.rollback(); // Revertir
} catch (SQLException ex) {}
}
return false;
} finally {
DatabaseConfig.closeConnection(conn);
}
}
```

Opción B: Asignación Múltiple con Savepoint

```
public void asignarEmpleadosConSavepoint(int proyectoId, List
empleadoIds) {
Connection conn = null;
try {
conn = DatabaseConfig.getConnection();
conn.setAutoCommit(false);
for (int empId : empleadoIds) {
Savepoint sp = conn.setSavepoint("SP_" + empId);
try {
asignacionDAO.crear(conn, empId, proyectoId);
} catch (SQLException e) {
conn.rollback(sp); // Rollback parcial
}
}
conn.commit();
} catch (SQLException e) {

// rollback...
}
}
```

Puntuación:

Solo commit, sin rollback: 0.5 puntos

Commit + rollback básico: 1 punto

Commit + rollback + savepoints: 2 puntos

Parte 5: Documentación (CE2.6)

Archivo: DOCUMENTACION.pdf o README.md

Debe incluir:

1. Portada
   Nombre del alumno

Fecha
Título: "Proyecto TechDAM - JDBC Avanzado"

2. Diagrama de Base de Datos
   Diagrama ER con las 3 tablas
   Indicar claves primarias y foráneas
3. Capturas de Pantalla
   Incluir capturas de:
   Tablas creadas en MySQL Workbench / HeidiSQL
   Ejecución exitosa de CRUD de empleados
   Ejecución exitosa de CRUD de proyectos
   Invocación de procedimiento almacenado
   Transacción con commit
   Transacción con rollback
   Pool de conexiones configurado (logs o métricas)
4. Explicación de Decisiones Técnicas
   Responde:
5. ¿Por qué usar PreparedStatement en lugar de Statement?
6. ¿Qué ventajas tiene el pool de conexiones?
7. ¿En qué casos usarías procedimientos almacenados?
8. ¿Por qué es importante el control de transacciones?
9. Problemas Encontrados y Soluciones

Describe al menos 2 problemas que encontraste y cómo los solucionaste.
Puntuación:
Sin documentación: 0 puntos
Documentación básica (capturas): 1 punto
Documentación completa (capturas + explicaciones): 2 puntos
Entregables
Debes entregar un archivo ZIP con:
apellido_nombre_techdam.zip

```
├── README.txt # Instrucciones de ejecución
├── techdam_completo.sql # Script SQL completo
├── pom.xml # Dependencias Maven
├── src/
│ └── main/java/com/techdam/
│ ├── Main.java
│ ├── config/
│ ├── model/
│ ├── dao/
│ └── service/
```

└── DOCUMENTACION.pdf # Documentación con capturas

Rúbrica de Evaluación

Resumen de Puntos

Criterio	Puntos	Descripción
CE2.1	Conexiones	2 HikariCP configurado
CE2.2	DDL/DML	2 CRUD completo
CE2.3	Procedimientos	2 2+ procedimientos funcionando
CE2.4	Transacciones	2 Commit + rollback + savepoints
CE2.5	Seguridad	2 PreparedStatement en 100%
CE2.6	Documentación	2 Completa con capturas
TOTAL	12	Normalizado a 10

Escala de Calificación

12 puntos = 10 (Sobresaliente)

10-11 = 8-9 (Notable)

8-9 = 6-7 (Bien)

6-7 = 5 (Suficiente)

<6 = <5 (Insuficiente)

Plazo de Entrega

Fecha límite: [A definir por el profesor]

Plataforma: Aula Virtual / Moodle

Formato: Archivo ZIP nombrado como apellido_nombre_techdam.zip

FAQs

¿Puedo usar JPA o Hibernate?

No. Esta actividad evalúa JDBC puro. JPA lo veremos más adelante.

¿Puedo trabajar en parejas?

Consulta con el profesor. Por defecto es individual.

¿Qué pasa si no funciona algo?

En la documentación explica qué intentaste, qué error obtuviste y cómo lo solucionaste (o por qué no pudiste).

¿Debo crear tests unitarios?

No es obligatorio, pero suma puntos extra si los incluyes.

¿Puedo añadir funcionalidades extra?

Sí, puedes añadir un menú CLI, más procedimientos, más tablas, etc. Se valorará positivamente.

Consejos Finales

1. Empieza por la base de datos - Si la BD no funciona, nada funciona
2. Prueba cada parte por separado - No intentes hacer todo de golpe

3. Haz commits frecuentes - Usa Git para versionar tu código
4. Documenta mientras desarrollas - No dejes la documentación para el final
5. Haz capturas de TODO - Es mejor tener muchas que pocas
6. Revisa la rúbrica - Asegúrate de cumplir cada punto

Recursos de Ayuda
Notebooks del curso (bloques 1-8)
Código de ejemplo en GitHub
Documentación oficial de JDBC
Consultas al profesor en horario de tutoría

Checklist Final
Antes de entregar, verifica:
El script SQL se ejecuta sin errores
El código Java compila sin errores
Todas las operaciones CRUD funcionan
Los procedimientos se ejecutan correctamente
Las transacciones hacen commit/rollback
No hay ni un solo Statement, solo PreparedStatement
Tienes capturas de pantalla de todo
La documentación está completa
El nombre del archivo es correcto
El README explica cómo ejecutar el proyecto
¡Mucha suerte con tu proyecto TechDAM!

RESULTADO

estructura

```
TechDAM/
│
├── src/main/java/
│   ├── config/
│   │   └── DatabaseConfig.java <-- conexión HikariCP
│   │
│   ├── modelo/
│   │   ├── Empleado.java
│   │   ├── Proyecto.java
│   │   └── Asignacion.java
│   │
│   ├── dao/
│   │   ├── EmpleadoDAO.java
```

```
│   │   └── ProyectoDAO.java
│   │
│   ├── service/
│   │   ├── ProcedimientosService.java
│   │   └── TransaccionesService.java
│   │
│   └── main/
│       └── Main.java
│
└── src/main/resources/
    └── db.properties
```

DatabaseConfig

```java
package config;


import com.zaxxer.hikari.HikariConfig;

import com.zaxxer.hikari.HikariDataSource;


import java.io.InputStream;

import java.sql.Connection;

import java.sql.SQLException;

import java.util.Properties;


public class DatabaseConfig {

    private static HikariDataSource dataSource;


    static {

        try {

            Properties props = new Properties();
```

```
            try (InputStream input =
DatabaseConfig.class.getClassLoader().getResourceAsStream("db.properties")) {

                props.load(input);

            }



            HikariConfig config = new HikariConfig();

            config.setJdbcUrl(props.getProperty("db.url"));

            config.setUsername(props.getProperty("db.user"));

            config.setPassword(props.getProperty("db.password"));

            config.setMaximumPoolSize(10);

            config.setMinimumIdle(2);

            config.setIdleTimeout(10000);

            config.setConnectionTimeout(10000);

            config.setPoolName("TechDAMPool");



            dataSource = new HikariDataSource(config);

            System.out.println("Pool de conexiones HikariCP inicializado
correctamente.");

        } catch (Exception e) {

            throw new RuntimeException("Error al inicializar el pool de
conexiones", e);

        }

    }
```

```
    public static Connection getConnection() throws SQLException {
        return dataSource.getConnection();
    }

    public static void close() {
        if (dataSource != null && !dataSource.isClosed()) {
            dataSource.close();
            System.out.println("Pool de conexiones cerrado.");
        }
    }
}
**
**
EmpleadoDAO
**
**
package dao;

import config.DatabaseConfig;
import modelo.Empleado;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;
```

```
import java.util.Optional;



public class EmpleadoDAO {



   public int crear(Empleado emp) {

       String sql = "INSERT INTO empleados(nombre, departamento, salario,
fecha_ingreso, activo) VALUES (?, ?, ?, ?, ?)";

       try (Connection conn = DatabaseConfig.getConnection();

            PreparedStatement ps = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {



           ps.setString(1, emp.getNombre());

           ps.setString(2, emp.getDepartamento());

           ps.setBigDecimal(3, emp.getSalario());

           ps.setDate(4, Date.valueOf(emp.getFechaIngreso()));

           ps.setBoolean(5, emp.isActivo());



           ps.executeUpdate();

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

               if (rs.next()) return rs.getInt(1);

           }

       } catch (SQLException e) { e.printStackTrace(); }

       return -1;

   }
```

```java
public List<Empleado> obtenerTodos() {
    List<Empleado> lista = new ArrayList<>();
    String sql = "SELECT * FROM empleados";
    try (Connection conn = DatabaseConfig.getConnection();
         PreparedStatement ps = conn.prepareStatement(sql);
         ResultSet rs = ps.executeQuery()) {

        while (rs.next()) {
            lista.add(new Empleado(
                    rs.getInt("id"),
                    rs.getString("nombre"),
                    rs.getString("departamento"),
                    rs.getBigDecimal("salario"),
                    rs.getDate("fecha_ingreso").toLocalDate(),
                    rs.getBoolean("activo")
            ));
        }
    } catch (SQLException e) { e.printStackTrace(); }
    return lista;
}

public Optional<Empleado> obtenerPorId(int id) {
```

```
    String sql = "SELECT * FROM empleados WHERE id = ?";
    try (Connection conn = DatabaseConfig.getConnection();
         PreparedStatement ps = conn.prepareStatement(sql)) {

        ps.setInt(1, id);
        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                return Optional.of(new Empleado(
                        rs.getInt("id"),
                        rs.getString("nombre"),
                        rs.getString("departamento"),
                        rs.getBigDecimal("salario"),
                        rs.getDate("fecha_ingreso").toLocalDate(),
                        rs.getBoolean("activo")
                ));
            }
        }
    } catch (SQLException e) { e.printStackTrace(); }
    return Optional.empty();
}

public boolean actualizar(Empleado emp) {
    String sql = "UPDATE empleados SET nombre=?, departamento=?, salario=?,
```

```
fecha_ingreso=?, activo=? WHERE id=?";

        try (Connection conn = DatabaseConfig.getConnection();

             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setString(1, emp.getNombre());

            ps.setString(2, emp.getDepartamento());

            ps.setBigDecimal(3, emp.getSalario());

            ps.setDate(4, Date.valueOf(emp.getFechaIngreso()));

            ps.setBoolean(5, emp.isActivo());

            ps.setInt(6, emp.getId());

            return ps.executeUpdate() > 0;

        } catch (SQLException e) { e.printStackTrace(); }

        return false;

    }



    public boolean eliminar(int id) {

        String sql = "DELETE FROM empleados WHERE id=?";

        try (Connection conn = DatabaseConfig.getConnection();

             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setInt(1, id);

            return ps.executeUpdate() > 0;

        } catch (SQLException e) { e.printStackTrace(); }

        return false;

    }
```

```
}
```

ProyectoDAO

```
package dao;


import config.DatabaseConfig;

import modelo.Proyecto;


import java.math.BigDecimal;

import java.sql.*;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;


public class ProyectoDAO {


   public int crear(Proyecto p) {

       String sql = "INSERT INTO proyectos(nombre, presupuesto, fecha_inicio,
fecha_fin) VALUES (?, ?, ?, ?)";

       try (Connection conn = DatabaseConfig.getConnection();

            PreparedStatement ps = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {


           ps.setString(1, p.getNombre());
```

```
            ps.setBigDecimal(2, p.getPresupuesto());
            ps.setDate(3, Date.valueOf(p.getFechaInicio()));
            ps.setDate(4, Date.valueOf(p.getFechaFin()));

            ps.executeUpdate();
            try (ResultSet rs = ps.getGeneratedKeys()) {
                if (rs.next()) return rs.getInt(1);
            }

        } catch (SQLException e) { e.printStackTrace(); }
        return -1;
    }

    public List<Proyecto> obtenerTodos() {
        List<Proyecto> lista = new ArrayList<>();
        String sql = "SELECT * FROM proyectos";
        try (Connection conn = DatabaseConfig.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql);
             ResultSet rs = ps.executeQuery()) {

            while (rs.next()) {
                lista.add(new Proyecto(
                        rs.getInt("id"),
```

```
                    rs.getString("nombre"),
                    rs.getBigDecimal("presupuesto"),
                    rs.getDate("fecha_inicio").toLocalDate(),
                    rs.getDate("fecha_fin").toLocalDate()
            ));
        }
    } catch (SQLException e) { e.printStackTrace(); }
    return lista;
}


public Optional<Proyecto> obtenerPorId(int id) {
    String sql = "SELECT * FROM proyectos WHERE id = ?";
    try (Connection conn = DatabaseConfig.getConnection();
         PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setInt(1, id);
        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                return Optional.of(new Proyecto(
                        rs.getInt("id"),
                        rs.getString("nombre"),
                        rs.getBigDecimal("presupuesto"),
                        rs.getDate("fecha_inicio").toLocalDate(),
                        rs.getDate("fecha_fin").toLocalDate()
```

```
                    ));
                }
            }
        } catch (SQLException e) { e.printStackTrace(); }
        return Optional.empty();
    }

    public boolean actualizar(Proyecto p) {
        String sql = "UPDATE proyectos SET nombre=?, presupuesto=?,
fecha_inicio=?, fecha_fin=? WHERE id=?";
        try (Connection conn = DatabaseConfig.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setString(1, p.getNombre());
            ps.setBigDecimal(2, p.getPresupuesto());
            ps.setDate(3, Date.valueOf(p.getFechaInicio()));
            ps.setDate(4, Date.valueOf(p.getFechaFin()));
            ps.setInt(5, p.getId());
            return ps.executeUpdate() > 0;
        } catch (SQLException e) { e.printStackTrace(); }
        return false;
    }

    public boolean eliminar(int id) {
```

```
        String sql = "DELETE FROM proyectos WHERE id=?";

        try (Connection conn = DatabaseConfig.getConnection();

             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setInt(1, id);

            return ps.executeUpdate() > 0;

        } catch (SQLException e) { e.printStackTrace(); }

        return false;

    }



    // Métodos para transacciones

    public void restarPresupuesto(Connection conn, int proyectoId, BigDecimal
monto) throws SQLException {

        String sql = "UPDATE proyectos SET presupuesto = presupuesto - ? WHERE
id=?";

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

            ps.setBigDecimal(1, monto);

            ps.setInt(2, proyectoId);

            ps.executeUpdate();

        }

    }



    public void sumarPresupuesto(Connection conn, int proyectoId, BigDecimal
monto) throws SQLException {

        String sql = "UPDATE proyectos SET presupuesto = presupuesto + ? WHERE
id=?";
```

```
        try (PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setBigDecimal(1, monto);

            ps.setInt(2, proyectoId);

            ps.executeUpdate();

        }

    }



    public void asignarEmpleado(Connection conn, int empleadoId, int
proyectoId) throws SQLException {

        String sql = "INSERT INTO asignaciones(empleado_id, proyecto_id,
fecha_asignacion) VALUES (?, ?, CURRENT_DATE())";

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

            ps.setInt(1, empleadoId);

            ps.setInt(2, proyectoId);

            ps.executeUpdate();

        }

    }

}
```

Asignacion

```
package modelo;



import java.time.LocalDate;
```

```java
public class Asignacion {

    private int id;

    private int empleadoId;

    private int proyectoId;

    private LocalDate fechaAsignacion;



    public Asignacion() {}



    public Asignacion(int id, int empleadoId, int proyectoId, LocalDate
fechaAsignacion) {

        this.id = id;

        this.empleadoId = empleadoId;

        this.proyectoId = proyectoId;

        this.fechaAsignacion = fechaAsignacion;

    }



    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public int getEmpleadoId() { return empleadoId; }

    public void setEmpleadoId(int empleadoId) { this.empleadoId = empleadoId; }

    public int getProyectoId() { return proyectoId; }

    public void setProyectoId(int proyectoId) { this.proyectoId = proyectoId; }

    public LocalDate getFechaAsignacion() { return fechaAsignacion; }

    public void setFechaAsignacion(LocalDate fechaAsignacion) {
```

```
this.fechaAsignacion = fechaAsignacion; }


    @Override
    public String toString() {
        return "EmpleadoID: " + empleadoId + " - ProyectoID: " + proyectoId;
    }
}
```

Empleado

```
package modelo;


import java.math.BigDecimal;
import java.time.LocalDate;


public class Empleado {
    private int id;
    private String nombre;
    private String departamento;
    private BigDecimal salario;
    private LocalDate fechaIngreso;
    private boolean activo;


    public Empleado() {}
```

```
    public Empleado(int id, String nombre, String departamento, BigDecimal
salario, LocalDate fechaIngreso, boolean activo) {
        this.id = id;
        this.nombre = nombre;
        this.departamento = departamento;
        this.salario = salario;
        this.fechaIngreso = fechaIngreso;
        this.activo = activo;
    }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }
    public String getDepartamento() { return departamento; }
    public void setDepartamento(String departamento) { this.departamento =
departamento; }
    public BigDecimal getSalario() { return salario; }
    public void setSalario(BigDecimal salario) { this.salario = salario; }
    public LocalDate getFechaIngreso() { return fechaIngreso; }
    public void setFechaIngreso(LocalDate fechaIngreso) { this.fechaIngreso =
fechaIngreso; }
    public boolean isActivo() { return activo; }
    public void setActivo(boolean activo) { this.activo = activo; }
```

```
    @Override
    public String toString() {
        return nombre + " - " + departamento + " - " + salario;
    }
}
```

Proyecto

```
package modelo;

import java.math.BigDecimal;
import java.time.LocalDate;

public class Proyecto {
    private int id;
    private String nombre;
    private BigDecimal presupuesto;
    private LocalDate fechaInicio;
    private LocalDate fechaFin;

    public Proyecto() {}
```

```java
    public Proyecto(int id, String nombre, BigDecimal presupuesto, LocalDate fechaInicio, LocalDate fechaFin) {

        this.id = id;

        this.nombre = nombre;

        this.presupuesto = presupuesto;

        this.fechaInicio = fechaInicio;

        this.fechaFin = fechaFin;

    }



    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getNombre() { return nombre; }

    public void setNombre(String nombre) { this.nombre = nombre; }

    public BigDecimal getPresupuesto() { return presupuesto; }

    public void setPresupuesto(BigDecimal presupuesto) { this.presupuesto = presupuesto; }

    public LocalDate getFechaInicio() { return fechaInicio; }

    public void setFechaInicio(LocalDate fechaInicio) { this.fechaInicio = fechaInicio; }

    public LocalDate getFechaFin() { return fechaFin; }

    public void setFechaFin(LocalDate fechaFin) { this.fechaFin = fechaFin; }



    @Override

    public String toString() {

        return nombre + " - Presupuesto: " + presupuesto;
```

```
    }
}
```

Main

```
package org.example;

import dao.EmpleadoDAO;
import dao.ProyectoDAO;
import modelo.Empleado;
import modelo.Proyecto;
import service.ProcedimientosService;
import service.TransaccionesService;

import java.math.BigDecimal;
import java.time.LocalDate;
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {

        EmpleadoDAO empleadoDAO = new EmpleadoDAO();
        ProyectoDAO proyectoDAO = new ProyectoDAO();
        ProcedimientosService procService = new ProcedimientosService();
```

```
        TransaccionesService transService = new TransaccionesService();



        // --- Crear empleado ---

        Empleado e = new Empleado(0, "Carlos Ruiz", "IT", new
BigDecimal("2800"), LocalDate.now(), true);

        int idEmp = empleadoDAO.crear(e);

        System.out.println("Empleado creado con ID: " + idEmp);



        // --- Listar empleados ---

        System.out.println("Lista de empleados:");

        empleadoDAO.obtenerTodos().forEach(System.out::println);



        // --- Crear proyecto ---

        Proyecto p = new Proyecto(0, "TechDAM Project", new
BigDecimal("10000"), LocalDate.now(), LocalDate.now().plusMonths(6));

        int idProj = proyectoDAO.crear(p);

        System.out.println("Proyecto creado con ID: " + idProj);



        // --- Procedimiento: aumentar salarios ---

        int actualizados = procService.actualizarSalariosDepartamento("IT", 5);

        System.out.println("Empleados actualizados: " + actualizados);



        // --- Transacción: transferencia de presupuesto ---

        boolean exito = transService.transferirPresupuesto(idProj, idProj, new
```

```
BigDecimal("1000"));

        System.out.println("Transferencia exitosa: " + exito);



        // --- Transacción: asignación múltiple ---

        transService.asignarEmpleadosConSavepoint(idProj,
Arrays.asList(idEmp));

        System.out.println("Asignación de empleados completada.");

    }

}
```

ProcedimientosService

```
package service;


import config.DatabaseConfig;


import java.math.BigDecimal;

import java.sql.*;


public class ProcedimientosService {


    public int actualizarSalariosDepartamento(String departamento, double
porcentaje) {

        try (Connection conn = DatabaseConfig.getConnection();
```

```
            CallableStatement cstmt = conn.prepareCall("{call
actualizar_salario_departamento(?, ?, ?)}")) {

            cstmt.setString(1, departamento);
            cstmt.setBigDecimal(2, BigDecimal.valueOf(porcentaje));
            cstmt.registerOutParameter(3, Types.INTEGER);
            cstmt.execute();
            return cstmt.getInt(3);

        } catch (SQLException e) {
            e.printStackTrace();
            return -1;
        }
    }

    // Puedes agregar otro procedimiento aquí de manera similar
}
```

TransaccionesService

```
package service;

import config.DatabaseConfig;
```

```java
import dao.ProyectoDAO;


import java.math.BigDecimal;

import java.sql.Connection;

import java.sql.SQLException;

import java.sql.Savepoint;

import java.util.List;


public class TransaccionesService {


   private ProyectoDAO proyectoDAO = new ProyectoDAO();


   public boolean transferirPresupuesto(int origenId, int destinoId,
BigDecimal monto) {

       Connection conn = null;

       try {

           conn = DatabaseConfig.getConnection();

           conn.setAutoCommit(false);


           proyectoDAO.restarPresupuesto(conn, origenId, monto);

           proyectoDAO.sumarPresupuesto(conn, destinoId, monto);


           conn.commit();
```

```
            return true;
        } catch (SQLException e) {
            if (conn != null) try { conn.rollback(); } catch (SQLException ex)
{}
            e.printStackTrace();
            return false;
        } finally {
            if (conn != null) try { conn.close(); } catch (SQLException e) {}
        }
    }


    public void asignarEmpleadosConSavepoint(int proyectoId, List<Integer>
empleadoIds) {
        Connection conn = null;
        try {
            conn = DatabaseConfig.getConnection();
            conn.setAutoCommit(false);

            for (int empId : empleadoIds) {
                Savepoint sp = conn.setSavepoint("SP_" + empId);
                try {
                    proyectoDAO.asignarEmpleado(conn, empId, proyectoId);
                } catch (SQLException e) {
                    conn.rollback(sp); // rollback parcial
```

```
                }

            }



            conn.commit();

        } catch (SQLException e) {

            if (conn != null) try { conn.rollback(); } catch (SQLException ex)
{}

            e.printStackTrace();

        } finally {

            if (conn != null) try { conn.close(); } catch (SQLException e) {}

        }

    }

}
```

db.properties

```
db.url=jdbc:mysql://localhost:3306/techdam

db.user=root

db.password=root123

db.maximumPoolSize=10
```

pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

        http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>




    <groupId>org.example</groupId>

    <artifactId>3</artifactId>

    <version>1.0-SNAPSHOT</version>



    <properties>

        <maven.compiler.source>23</maven.compiler.source>

        <maven.compiler.target>23</maven.compiler.target>

        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

        <mysql.version>8.0.33</mysql.version>

        <hikaricp.version>5.0.1</hikaricp.version>

        <slf4j.version>2.0.9</slf4j.version>

    </properties>
```

```
    <dependencies>

        <!-- Driver JDBC MySQL -->

        <!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->

        <dependency>

            <groupId>com.mysql</groupId>

            <artifactId>mysql-connector-j</artifactId>

            <version>${mysql.version}</version>

        </dependency>









        <!-- HikariCP -->

        <dependency>

            <groupId>com.zaxxer</groupId>

            <artifactId>HikariCP</artifactId>

            <version>${hikaricp.version}</version>

        </dependency>




        <!-- SLF4J - Logging -->

        <dependency>

            <groupId>org.slf4j</groupId>
```

```
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
        </dependency>


        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>${slf4j.version}</version>
        </dependency>


    </dependencies>



</project>
  ,,



**
-- ------------------------------------------------
```

```sql
-- ELIMINAR BASE DE DATOS ANTIGUA
-- ------------------------------------------------------
DROP DATABASE IF EXISTS techdam;

-- ------------------------------------------------------
-- CREAR BASE DE DATOS
-- ------------------------------------------------------
CREATE DATABASE techdam;
USE techdam;

-- ------------------------------------------------------
-- TABLA: empleados
-- ------------------------------------------------------
CREATE TABLE empleados (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    departamento VARCHAR(50) NOT NULL,
    salario DECIMAL(10,2) NOT NULL,
    fecha_ingreso DATE NOT NULL,
    activo BOOLEAN DEFAULT TRUE
);

CREATE INDEX idx_departamento ON empleados(departamento);

-- Datos de prueba
INSERT INTO empleados(nombre, departamento, salario, fecha_ingreso, activo)
VALUES
('Ana López','IT',2500,CURDATE(),TRUE),
('Luis Pérez','RRHH',2200,CURDATE(),TRUE),
('María Gómez','IT',2800,CURDATE(),TRUE),
('Jorge Díaz','Finanzas',3000,CURDATE(),TRUE),
('Laura Sánchez','Marketing',2000,CURDATE(),TRUE);

-- ------------------------------------------------------
-- TABLA: proyectos
-- ------------------------------------------------------
CREATE TABLE proyectos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    presupuesto DECIMAL(12,2) NOT NULL,
    fecha_inicio DATE NOT NULL,
    fecha_fin DATE NOT NULL
);

CREATE INDEX idx_nombre_proyecto ON proyectos(nombre);
```

```sql
-- Datos de prueba
INSERT INTO proyectos(nombre, presupuesto, fecha_inicio, fecha_fin) VALUES
('Proyecto Alpha',10000,CURDATE(),DATE_ADD(CURDATE(), INTERVAL 6 MONTH)),
('Proyecto Beta',15000,CURDATE(),DATE_ADD(CURDATE(), INTERVAL 12 MONTH)),
('Proyecto Gamma',12000,CURDATE(),DATE_ADD(CURDATE(), INTERVAL 8 MONTH)),
('Proyecto Delta',18000,CURDATE(),DATE_ADD(CURDATE(), INTERVAL 10 MONTH)),
('Proyecto Epsilon',20000,CURDATE(),DATE_ADD(CURDATE(), INTERVAL 15 MONTH));

-- ------------------------------------------------------
-- TABLA: asignaciones
-- ------------------------------------------------------
CREATE TABLE asignaciones (
    id INT AUTO_INCREMENT PRIMARY KEY,
    empleado_id INT NOT NULL,
    proyecto_id INT NOT NULL,
    fecha_asignacion DATE NOT NULL,
    FOREIGN KEY (empleado_id) REFERENCES empleados(id) ON DELETE CASCADE,
    FOREIGN KEY (proyecto_id) REFERENCES proyectos(id) ON DELETE CASCADE
);

-- Datos de prueba
INSERT INTO asignaciones(empleado_id, proyecto_id, fecha_asignacion) VALUES
(1,1,CURDATE()),
(2,2,CURDATE()),
(3,1,CURDATE()),
(4,3,CURDATE()),
(5,4,CURDATE());

-- ------------------------------------------------------
-- PROCEDIMIENTO 1: actualizar_salario_departamento
-- ------------------------------------------------------
DELIMITER $$
CREATE PROCEDURE actualizar_salario_departamento(
    IN p_departamento VARCHAR(50),
    IN p_porcentaje DECIMAL(5,2),
    OUT p_empleados_actualizados INT
)
BEGIN
    UPDATE empleados
    SET salario = salario * (1 + p_porcentaje / 100)
    WHERE departamento = p_departamento AND activo = TRUE;
    SET p_empleados_actualizados = ROW_COUNT();
END$$
DELIMITER ;

-- ------------------------------------------------------
```

```sql
-- PROCEDIMIENTO 2: asignar_empleado_proyecto
-- ----------------------------------------------------
DELIMITER $$
CREATE PROCEDURE asignar_empleado_proyecto(
    IN p_empleado_id INT,
    IN p_proyecto_id INT
)
BEGIN
    INSERT INTO asignaciones(empleado_id, proyecto_id, fecha_asignacion)
    VALUES (p_empleado_id, p_proyecto_id, CURDATE());
END$$
DELIMITER ;

-- ----------------------------------------------------
-- FUNCIÓN: salario_promedio_departamento
-- ----------------------------------------------------
DELIMITER $$
CREATE FUNCTION salario_promedio_departamento(p_departamento VARCHAR(50))
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE promedio DECIMAL(10,2);
    SELECT AVG(salario) INTO promedio
    FROM empleados
    WHERE departamento = p_departamento;
    RETURN promedio;
END$$
DELIMITER ;
```