Resumen del funcionamiento del programa

Este programa, ProvisysDB, desarrollado en Java, tiene como objetivo establecer una conexión con una base de datos PostgreSQL. Esta BD está alojada en un contenedor Docker accesible en localhost:5432, y la idea es permitir al usuario realizar un par de consultas básicas mediante un menú en consola. Es un programa simple, pero supuestamente cumple con los requisitos para una buena nota, según los requisitos de entrega del reto 1.

Antes de empezar

Asegúrate de tener Maven instalado en tu ordenador. Las instrucciones que necesitas con tal de instalar Maven están en su página web, https://maven.apache.org/install.html.

1. Conexión a la Base de Datos

Al iniciar el programa se establece la conexión con la base de datos PostgreSQL utilizando JDBC (que importamos usando Maven).
Se configura la URL de conexión, el usuario y la contraseña como parámetros de acceso.
En caso de que la conexión falle, el programa muestra un mensaje de error y finaliza.

```
C:\Users\garik\2DM3\Che\Reto1\Proyecto\ProviSys\provisysDB>mvn exec:java -Dexec.mainClass="com.provisys.Main"
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< com.provisys:provisys >-----------------------
[INFO] Building provisys 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.6.1:java (default-cli) @ provisys ---
Conexión exitosa a la base de datos PostgreSQL.
```

```java
try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD)) {
    if (conn != null) {
        //si la conexión no es nula, procede con el resto del programa
        System.out.println(x:"Conexión exitosa a la base de datos PostgreSQL.");
        //creación del scanner para lectura en el menú
        Scanner sc = new Scanner(System.in);
```

2. Menú Interactivo

Tras conectarse, el programa muestra un menú en consola con tres opciones principales:

1. Listar los primeros 5 clientes
2. Contar el número total de clientes en la base de datos
3. Salir del programa

El menú se ejecuta dentro de un bucle do-while para permitir que el usuario elija varias opciones hasta decidir salir.

```
Seleccione una opción:
1. Mostrar 5 primeros clientes junto a sus datos relevantes
2. Contar clientes en total
3. Salir del programa
```

```java
Scanner sc = new Scanner(System.in);
do {
    System.out.println(x:"Seleccione una opción:\n1. Mostrar 5 primeros clientes junto a sus datos relevantes\n2. Contar clientes en
    try{ //try catch para asegurar que el valor introducido sea un integer
        int opt = sc.nextInt();
        sc.nextLine();
        //switch para el menú
        switch(opt) {
            case 1:
                //llama al método de listado de clientes utilizando la conexión preestablecida
                listarClientes(conn);
                break;
            case 2:
                //llama al método de contado de clientes utilizando la conexión preestablecida
                contarClientes(conn);
                break;
            case 3:
                //salida del sistema con cerrado del scanner
                System.out.println(x:"Adios.");
                sc.close();
                return;
            default:
                System.out.println(x:"No.");
                break;
        }
    } catch (InputMismatchException e) {
        //vuelve a intentarlo en caso de introducirlo equivocadamente
        System.out.println(x:"Opción invalida.");
        break;
    }
```

3. Opción 1: Listado de Clientes

Esta opción realiza una consulta SQL sobre la tabla res_partner para obtener los siguientes campos de los 5 primeros registros:

id, create_date, name, email, street

Los datos obtenidos se muestran uno por uno en consola, en formato de tabla.

```
1

Mostrando clientes.
ID: 1 | Nombre: My Company (San Francisco) | Correo Electrónico: admin@admin.com | Calle: 250 Executive Park Blvd, Suite 3400 | Fecha de registro: 2025-09-29 08:59:13.729666
----------------------------------------------------
ID: 2 | Nombre: OdooBot | Correo Electrónico: odoobot@example.com | Calle: null | Fecha de registro: 2025-09-29 08:59:14.583452
----------------------------------------------------
ID: 3 | Nombre: Mitchell Admin | Correo Electrónico: admin@admin.com | Calle: 215 Vine St | Fecha de registro: 2025-09-29 08:59:14.583452
----------------------------------------------------
ID: 4 | Nombre: Public user | Correo Electrónico: null | Calle: null | Fecha de registro: 2025-09-29 08:59:14.583452
----------------------------------------------------
ID: 5 | Nombre: Default User Template | Correo Electrónico: null | Calle: null | Fecha de registro: 2025-09-29 08:59:14.583452
----------------------------------------------------
```

```java
private static void listarClientes(Connection conn) {
String sql = "SELECT id, create_date, name, email, street FROM public.res_partner ORDER BY id LIMIT 5";
//prepara el statement de la BBDD de antemano, y luego prueba a ejecutarlo utilizando la conexión que se le pasa en la llamada de método
try (Statement stmt = conn.createStatement();
     ResultSet rs = stmt.executeQuery(sql)) {
        //crea el resultset que guarda los resultados de la consulta, y luego asigna los valores del resultset a diferentes campos para su i
    System.out.println(x:"\nMostrando clientes.");
    while (rs.next()) {
        int id = rs.getInt(columnLabel:"id");
        Timestamp fecha = rs.getTimestamp(columnLabel:"create_date");
        String nombre = rs.getString(columnLabel:"name");
        String email = rs.getString(columnLabel:"em
        String calle = rs.getString(columnLabel:"st
        System.out.println("ID: "+id+" | Nombre: "+nombre+" | Correo Electrónico: "+email+" | Calle: "+calle+" | Fecha de registro: "+fecha.
    }

} catch (SQLException e) {
    //en caso de que ocurra cualquier error, lo reporta aquí abajo y vuelve al menú anterior
    System.out.println(x:"Error listando clientes: ");
    e.printStackTrace();
}
}
```

String nombre - Main.listarClientes(Connection)

4. Opción 2: Contar Clientes

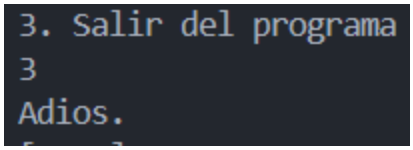
Esta opción realiza una consulta SQL para contar el número total de registros existentes en la tabla res_partner. El resultado se muestra en consola como el total de clientes almacenados en la base de datos.

```
Seleccione una opción:
1. Mostrar 5 primeros clientes junto a sus datos relevantes
2. Contar clientes en total
3. Salir del programa
2

? Total de clientes: 50
```

```java
private static void contarClientes(Connection conn) {
    String sql = "SELECT COUNT(*) AS total FROM public.res_partner";
    //prepara el statement de la BBDD de antemano, y luego prueba a ejecutarlo utilizando la conexión que se le pasa en la llamada de método
    try (Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {
            //en este caso, el resultset solo es un valor, así que no es necesario asignarlo a una variable; lo imprimimos directamente.
        if (rs.next()) {
            System.out.println("\nTotal de clientes: " + rs.getInt(columnLabel:"total"));
        }

    } catch (SQLException e) {
        //en caso de que ocurra cualquier error, lo reporta aquí abajo y vuelve al menú anterior
        System.out.println(x:"Error contando clientes: ");
        e.printStackTrace();
    }
}
}
```

5. Finalización del Programa

El usuario puede finalizar la ejecución seleccionando la opción 3 del menú. Al salir, la conexión a la base de datos se cierra automáticamente.




```java
case 3:
    //salida del sistema con cerrado del scanner
    System.out.println(x:"Adios.");
    sc.close();
    return;
```

Esto en total describiría el funcionamiento del programa. En el caso de manejo de errores, podemos comprobar a lo largo de las capturas proporcionadas que se incluye manejo de errores tanto para fallos de introducción de datos y errores de SQL.

Reflexión

Un programa sencillo, a decir verdad, pero uno que nos ha ayudado a profundizar nuestros conocimientos de cómo los programas de Java pueden pasarle statements a las bases de datos para una implementación a mayor escala. Es nuestra primera vez trabajando con un servidor remoto de BBDD, pero por suerte no hemos tenido muchos problemas en ese respecto. En resúmen, un programa sencillo pero útil, esperamos que podamos aplicar los conocimientos obtenidos más adelante.