Colegio Litterator

2º Desarrollo de Aplicaciones Multiplataforma

Proyecto SQL (Supermercado)

Acceso a Datos

Autor:
Elena López Félix , Mayra Barrantes Pi

Índice de Contenido

Índice de Figuras

1 Introducción

El presente documento describe el desarrollo e implementación de un sistema de gestión para un supermercado, programado en **Java** y con base de datos **MySQL**. Este sistema permite la administración eficiente de productos, clientes, empleados, pedidos a proveedores y ventas.
El objetivo principal de este documento es proporcionar una visión detallada del diseño y funcionamiento del sistema, asegurando su mantenibilidad y escalabilidad.

2 Descripción General de la Aplicación

La aplicación está diseñada para ser ejecutada en la línea de comandos y proporciona un menú interactivo que permite a los usuarios navegar entre sus opciones de manera sencilla e intuitiva. Está enfocada en la gestión de una base de datos para un negocio, donde los usuarios pueden administrar clientes, productos, ventas, empleados, proveedores y pedidos a proveedores.

3 Configuración del sistema

3.1 Diseño de la base de datos

La base de datos utilizada en el sistema es una base de datos relacional basada en **MySQL**. Se han definido varias tablas que almacenan la información clave del sistema, manteniendo la integridad de los datos a través de claves primarias y foráneas.

3.1.1 Principales tablas y relaciones

A continuación, se describen las principales tablas junto con sus relaciones:

- **Cliente:** Almacena los datos de los clientes, quienes pueden realizar múltiples compras. Relacionada con la tabla *Venta*.
- **Venta:** Registra cada transacción de compra, con un total asociado y la fecha en que se realizó. Relacionada con *Detalle-Venta*, que almacena los productos comprados en cada venta.
- **Detalle-venta:** Representa los productos incluidos en una venta específica, registrando la cantidad y el precio unitario de cada uno.
- **Producto:** Contiene información de los productos disponibles para la venta. Está relacionada con *Categoría* (*para clasificar los productos*) y con *Proveedor* (*que suministra el producto*).

- **Proveedor:** Almacena la información de los proveedores, quienes pueden suministrar múltiples productos. Está relacionado con *Pedido_Proveedor*, que registra los pedidos realizados a los proveedores.
- **Pedido-Proveedor:** Contiene los pedidos realizados a los proveedores, con fecha y total de la compra.
- **Detalle-Pedido-Proveedor:** Similar a *Detalle_Venta*, pero para los pedidos a proveedores, registrando los productos pedidos, su cantidad y el precio unitario.

3.1.2 Diagramas

Se han generado los siguientes diagramas para representar el diseño de la base de datos:

1. **Diagrama Entidad-Relación:** Representa las entidades principales del sistema y sus relaciones.

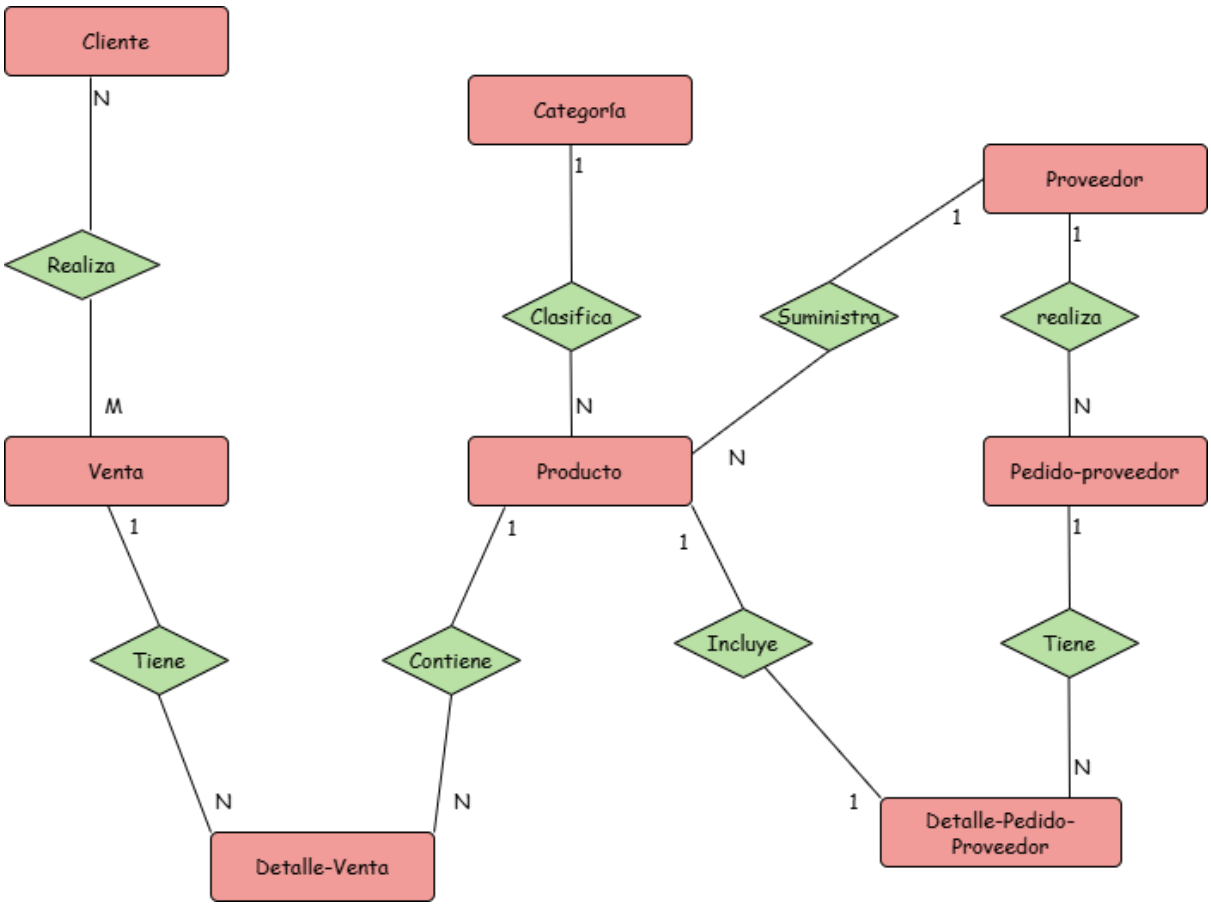

Figure 1: Diagrama Entidad Relación

2. **Diagrama Relacional:** Muestra la estructura final de las tablas con sus atributos, claves primarias y claves foráneas.

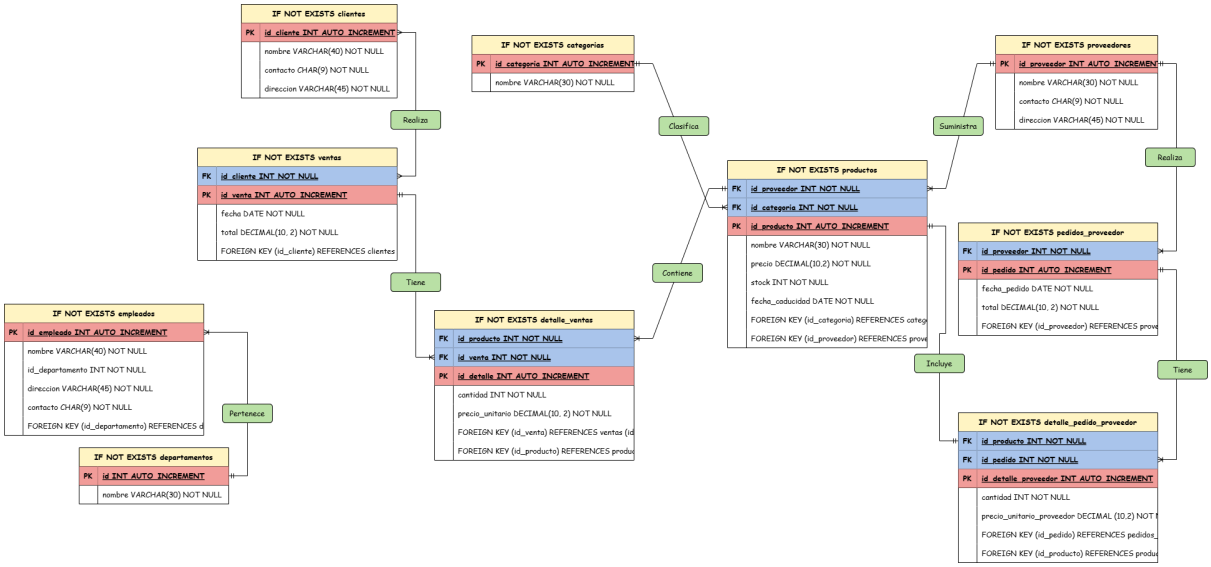

Figure 2: Diagrama Relacional

Estos modelos garantizan que la base de datos esté bien estructurada, optimizando la consulta y manipulación de los datos.

3.2 Estructura del código

El sistema es una aplicación de consola, estructurada en diferentes clases que representan las entidades del negocio, siguiendo el paradigma de **Programación Orientada a Objetos (POO)**. Se utiliza **JDBC** para la conexión con la base de datos, permitiendo realizar operaciones **CRUD** (*Crear, Leer, Actualizar y Eliminar*).

3.2.1 Diagrama de Clases

El siguiente diagrama muestra la organización de las clases en el sistema:

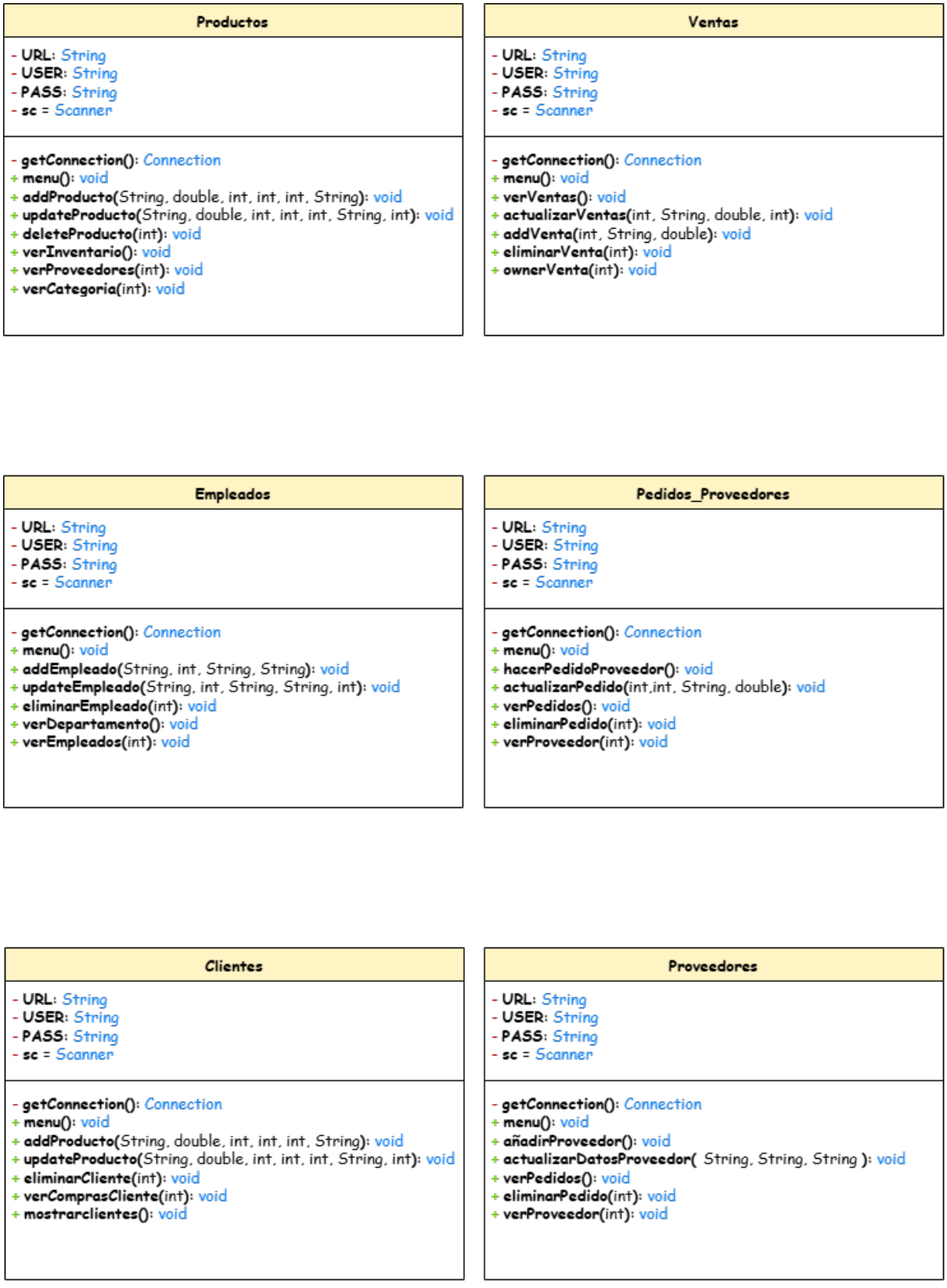

Figure 3: Diagrama de clases

3.2.2 Descripción de las clases

Cada una de las clases gestiona una tabla específica en la base de datos y contiene métodos para realizar operaciones **CRUD** (*Create, Read, Update, Delete*).

- **Clientes:** Maneja todas las operaciones relacionadas con la tabla *Clientes*. Permite añadir, actualizar y eliminar clientes, así como visualizar sus compras.
- **Productos:** Contiene los métodos necesarios para gestionar la tabla *Productos*, incluyendo la inclusión de nuevos productos, actualización de información y eliminación.
- **Proveedores:** Maneja todas las operaciones relacionadas con la tabla *Proveedores*. Permite gestionar los proveedores del sistema, permitiendo su registro, modificación y eliminación.
- **Ventas:** Contiene los métodos necesarios para gestionar la tabla *Ventas*. Gestiona las ventas realizadas en el sistema, permitiendo registrar nuevas ventas y consultar ventas previas.
- **Empleados:** Maneja todas las operaciones relacionadas con la tabla *Empleados*. Administra la información de los empleados del sistema, con métodos para registrar nuevos empleados, actualizar datos y eliminarlos.
- **Pedidos_Proveedores:** Contiene los métodos necesarios con la tabla *Pedidos_Proveedores*. Maneja los pedidos realizados a los proveedores, incluyendo la inserción de nuevos pedidos y la consulta de pedidos anteriores.

3.2.3 Organización del código en clases

Cada una de estas clases sigue una estructura similar, basada en los siguientes elementos clave:

3.2.4 Clase Clientes

Esta clase contiene 4 atributos: `URL, USUARIO, PASWORD, sc` los tres primeros son indisplensables para crear la conexión con la base de datos. El atributo `sc` es un objeto de la clase `Scanner` para que el usuario pueda interactuar con la consola de comandos.
Esta clase contiene varios métodos:

- `private static Connection getConnection()`: Este método crea la conexión con la base de datos, nos devuelve un objeto de tipo `Connection`.
- `private void clean()`: Este método simula que limpia la consola, para que el texto sea más legible.

- `public void menu()`: Este método muestra todas las opciones disponibles que puede hacer el usuario con la tabla **Clientes**. Las funcionalidades actuales son:
  - **Añadir cliente**.
  - **Actualizar cliente**.
  - **Eliminar cliente**.
  - **Ver las compras de un cliente**.
  - **Ver todos los clientes**.
- `public void añadirCliente (String nombre, String contacto, String direccion)`: Este método añade un cliente a la tabla **Clientes**, el usuario elige de manera dinámica los datos del cliente.
- `public void actualizarDatosCliente(int idCliente, String nombre, String contacto, String direccion)`: Este método actualiza los datos de un cliente en específico. De igual manera, el usuario escribe y elige de manera dinámica los nuevos datos del cliente.
- `public void eliminarCliente(int idCliente)`: Este método elimina un cliente de la tabla **Clientes**. El usuario elige que cliente quiere eliminar en función de su `idCliente`.
- `public void verComprasCliente(int idCliente)`: Este método te muestra todas las compras de un cliente en específico.
- `public void mostrarCLientes()`: Este método te muestra todos los clientes de la tabla **Clientes**.

3.2.5 Clase Empleados

Esta clase contiene 4 atributos: `URL, USER, PASS, sc` los tres primeros son indisplensables para crear la conexión con la base de datos. El atributo `sc` es un objeto de la clase `Scanner` para que el usuario pueda interactuar con la consola de comandos.
Esta clase contiene varios métodos:

- `private static Connection getConnection()`: Este método crea la conexión con la base de datos, nos devuelve un objeto de tipo `Connection`.
- `private void clean()`: Este método simula que limpia la consola, para que el texto sea más legible.
- `public void menu()`: Este método muestra todas las opciones disponibles que puede hacer el usuario con la tabla **Empleados**. Las funcionalidades actuales son:

  - **Añadir empleado**.
  - **Actualizar empleado**.
  - **Eliminar empleado**.
  - **Ver el departamento del empleado**.
  - **Ver todos los empleados**.

- `public void addEmpleado(String nombre, int id_departamento, String direccion, String contacto)`: Este método añade un empleado a la tabla **Empleados**, el usuario elige de manera dinámica los datos del empleado.
- `public void updateEmpleado(String nombre, int id_departamento, String direccion, String contacto, int id_empleado)`: Este método actualiza los datos de un empleado en específico. De igual manera, el usuario escribe y elige de manera dinámica los nuevos datos del empleado.
- `public void eliminarEmpleado(int id_empleado)`: Este método elimina un empleado de la tabla **Empleados**. El usuario elige que empleado quiere eliminar en función de su `id_empleado`.
- `public void verDepartamento(int id_empleado)`: Este método te muestra el departamento en el que trabaja un empleado en específico.
- `public void verEmpleados()`: Este método te muestra todos los empleados de la tabla **Empleados**.

3.2.6 Clase Engine

Esta clase actúa como el menú principal del sistema, permitiendo al usuario navegar entre las diferentes funcionalidades. Esta clase instancia los objetos de las demás clases y llama a sus respectivos métodos `menu()`, asegurando una estructura modular y organizada.
Este método solo contiene un atributo: `sc`, que hemos dicho anteriormente es una instancia de la clase `Scanner` para que el usuario pueda interactuar con la consola de comandos.
Esta clase solo contiene un sólo método:

- `public void start()`: Este método llama a los métodos menu() de las diferentes clases, el usuario es quien elige que clase desea explorar.

3.2.7 Clase Main

Esta clase solo contiene un método:

- `public static void main (String []args)`: Este método inicializa el proyecto, creando un objeto de la clase `Engine` para llamar así a su método `start()`.

3.2.8 Clase Pedidos_Proveedores

Esta clase contiene 4 atributos: `URL, USUARIO, PASWORD, sc` los tres primeros son indisplensables para crear la conexión con la base de datos. El atributo `sc` es un objeto de la clase `Scanner` para que el usuario pueda interactuar con la consola de comandos.
Esta clase contiene varios métodos:

- `private static Connection getConnection()`: Este método crea la conexión con la base de datos, nos devuelve un objeto de tipo `Connection`.
- `private void clean()`: Este método simula que limpia la consola, para que el texto sea más legible.
- `public void menu()`: Este método muestra todas las opciones disponibles que puede hacer el usuario con la tabla **Pedidos_Proveedores**. Las funcionalidades actuales son:
  - **Hacer pedido a proveedor**.
  - **Modificar datos de pedido**.
  - **Eliminar pedido**.
  - **Ver el proveedor que ha hecho un pedido**.
  - **Ver todos los pedidos**.
- `public void hacerPedidoProveedor()`: Este método simula el hacer un pedido a un proveedor determinado.
- `public void actualizarPedido(int idPedido, int idProveedor, String fechapedido, double total)`: Este método actualiza los datos de un pedido en específico. El usuario escribe y elige de manera dinámica los nuevos datos del pedido.
- `public void eliminarPedido(int idPedido)`: Este método elimina un pedido de la tabla **Pedidos_Proveedores**. El usuario elige que pedido quiere eliminar en función de su `idPedido`.
- `public void verPedidos()`: Este método te muestra todos los pedidos de la tabla **Pedidos_Proveedores**.

3.2.9 Clase Productos

Esta clase contiene 4 atributos: `URL, USER, PASS, sc` los tres primeros son indisplensables para crear la conexión con la base de datos. El atributo `sc` es un objeto de la clase `Scanner` para que el usuario pueda interactuar con la consola de comandos.
Esta clase contiene varios métodos:

- `private static Connection getConnection()`: Este método crea la conexión con la base de datos, nos devuelve un objeto de tipo `Connection`.
- `private void clean()`: Este método simula que limpia la consola, para que el texto sea más legible.
- `public void menu()`: Este método muestra todas las opciones disponibles que puede hacer el usuario con la tabla **Productos**. Las funcionalidades actuales son:
  - **Añadir producto**.
  - **Actualizar producto**.
  - **Eliminar producto**.
  - **Ver la información del proveedor**.
  - **Ver la categoría del producto**
  - **Ver todos los productos**.
- `public void addProducto(String nombre, double precio, int stock, int id_categoria int id_proveedor, String fecha_caducidad)`: Este método añade un producto a la tabla **Productos**, el usuario elige de manera dinámica los datos del producto.
- `public void updateProducto(String nombre, double precio, int stock, int id_categoria, int id_proveedor, String fecha_caducidad, int id_producto)`: Este método actualiza los datos de un producto en específico. De igual manera, el usuario escribe y elige de manera dinámica los nuevos datos del producto.
- `public void deleteProducto(int id_producto)`: Este método elimina un producto de la tabla **Productos**. El usuario elige que producto quiere eliminar en función de su `id_producto`.
- `public void verInventario()`: Este método te muestra todos los productos de la tabla **Productos**.
- `public void verProveedores(int id_producto)`: Este método te muestra información sobre el proveedor que ha traído el producto.
- `public void verCategoria(int id_producto)`: Este método te muestra la categoría a la que pertenece el producto.

3.2.10 Clase Proveedores

Esta clase contiene 4 atributos: `URL, USUARIO, PASWORD, sc` los tres primeros son indisplensables para crear la conexión con la base de datos. El atributo `sc` es un objeto de

la clase `Scanner` para que el usuario pueda interactuar con la consola de comandos.
Esta clase contiene varios métodos:

- `private static Connection getConnection()`: Este método crea la conexión con la base de datos, nos devuelve un objeto de tipo `Connection`.
- `private void clean()`: Este método simula que limpia la consola, para que el texto sea más legible.
- `public void menu()`: Este método muestra todas las opciones disponibles que puede hacer el usuario con la tabla **Proveedores**. Las funcionalidades actuales son:
  - **Añadir proveedor**.
  - **Actualizar proveedor**.
  - **Eliminar proveedor**.
  - **Ver la información de los pedidos**.
  - **Ver todos los proveedores**.
- `public void añadirProveedor(String nombre, String contacto, String direccion)`: Este métoto añade un proveedor a la tabla **Proveedores**, el usuario elige de manera dinámica los datos del proveedor.
- `public void actualizarDatosProveedor(int id, String nombre, String contacto, String direccion)`: Este método actualiza los datos de un proveedor en específico. De igual manera, el usuario escribe y elige de manera dinámica los nuevos datos del proveedor.
- `public void eliminaProveedor(int id)`: Este método elimina un proveedor de la tabla **Proveedores**. El usuario elige que proveedor quiere eliminar en función de su `id`.
- `public void mostrarProveedor()`: Este método te muestra todos los proveedores de la tabla **Proveedores**.
- `public static void mostrarDetallePedidosProveedor(int idProveedor)`: Este método te muestra los pedidos que ha hecho un proveedor en específico.

3.2.11 Clase Ventas

Esta clase contiene 4 atributos: `URL, USER, PASS, sc` los tres primeros son indisplensables para crear la conexión con la base de datos. El atributo `sc` es un objeto de la clase `Scanner` para que el usuario pueda interactuar con la consola de comandos.
Esta clase contiene varios métodos:

- `private static Connection getConnection()`: Este método crea la conexión con la base de datos, nos devuelve un objeto de tipo `Connection`.
- `private void clean()`: Este método simula que limpia la consola, para que el texto sea más legible.
- `public void menu()`: Este método muestra todas las opciones disponibles que puede hacer el usuario con la tabla **Ventas**. Las funcionalidades actuales son:
  - **Actualizar venta**.
  - **Eliminar venta**.
  - **Ver cliente que ha hecho la venta**.
  - **Ver todas las ventas**.
- `public void verVentas()`: Este método te muestra todas las ventas de la tabla **Ventas**.
- `public void actualizarVenta(int id_cliente, String fecha, double total, int id_venta)`: Este método actualiza los datos de una venta en específico. El usuario escribe y elige de manera dinámica los nuevos datos de la venta.
- `public void hacerVenta()`: Este método simula una compra de productos. Añade una venta a la tabla **Ventas** y modifica el *stock* de la tabla **Productos**. Para así aumentar el realismo de la compra, el usuario elige que productos y comprar y la cantidad deseada.
- `public void eliminarVenta(int id_venta)`: Este método elimina una venta de la tabla **Ventas**. El usuario elige que venta quiere eliminar en función de su `id_venta`
- public void ownerVenta(int id_venta): Este método te muestra información sobre el cliente que ha realizado la venta.

4 Funcionalidades principales

La aplicación permite gestionar diferentes aspectos de un negocio, incluyendo productos, proveedores, ventas y clientes. Cada una de estas áreas cuenta con operaciones específicas que permiten agregar, listar, actualizar y eliminar registros en la base de datos

4.1 Gestión de producto

La gestión de productos permite administrar el inventario de la empresa mediante las siguientes operaciones:

1. **Agregar producto:** Permite insertar un nuevo producto en la base de datos, proporcionando información como:

   - Nombre del producto
   - Descripción
   - Precio
   - Cantidad en stock
   - Categoría

   Ejemplo de inserción SQL:
   ```
   INSERT INTO productos (nombre, descripcion, precio, stock, categoria)
   VALUES (?, ?, ?, ?, ?);
   ```

2. **Listar producto:** Recupera y muestra todos los productos registrados en la base de datos.
   Ejemplo de inserción SQL:
   ```
   SELECT * FROM productos;
   ```

3. **Atualizar producto:** Permite modificar los datos de un producto ya existente, como el precio, el stock o la descripción.
   Ejemplo de inserción SQL:
   ```
   UPDATE productos SET nombre = ?, descripcion = ?, precio = ?, stock =
   ?, categoria = ?  WHERE id_producto = ?;
   ```

4. **Eliminar producto:** Borra un producto identificado por su ID, asegurando que no existan dependencias antes de la eliminación.
   Ejemplo de inserción SQL:
   ```
   DELETE FROM productos WHERE id_producto = ?;
   ```

4.2 Gestión de proveedores

Los proveedores son fundamentales en el abastecimiento de productos. Las operaciones disponibles en esta área incluyen:

1. **Agregar proveedor:** Inserta un nuevo proveedor en la base de datos, registrando información como:

   - ID

- Nombre del proveedor
- Dirección
- Contacto

Ejemplo de inserción SQL:
`INSERT INTO proveedores (id, nombre, contacto, dirección) VALUES (?, ?, ?, ?);`

2. **Listar proveedores:** Permite visualizar todos los proveedores registrados en la base de datos.
   Ejemplo de consulta SQL:
   `SELECT * FROM proveedores;`

3. **Actualizar proveedor:** Permite modificar los datos de un proveedor registrado
   Ejemplo de consulta SQL:
   `UPDATE proveedores SET nombre = ?, dirección= ?, contacto = ?, WHERE id_proveedor = ?;`

4. **Eliminar proveedor:** Elimina un proveedor de la base de datos, verificando que no tenga pedidos pendientes antes de su eliminación.
   Ejemplo de eliminación SQL:
   `DELETE FROM proveedores WHERE id_proveedor = ?;`

4.3 Gestión de ventas

La gestión de ventas permite registrar las transacciones realizadas con los clientes.

1. **Registrar venta:** Inserta una nueva venta en la base de datos, registrando información como:

   - Cliente que realizó la compra.
   - Productos comprados.
   - Cantidad de productos.
   - Fecha de la venta.
   - Total de la compra.

   Ejemplo de inserción SQL:
   `INSERT INTO ventas (id_cliente, fecha, total) VALUES (?, ?, ?);`

2. **Listar ventas:** Permite visualizar todas las ventas registradas en la base de datos.
   Ejemplo de consulta SQL:
   `SELECT * FROM ventas;`

3. **Consultar ventas por cliente:** Muestra todas las compras realizadas por un cliente específico.
   Ejemplo de consulta SQL:
   ```
   SELECT v.id_venta, v.fecha, v.total FROM ventas v JOIN clientes c
   ON v.id_cliente = c.id_cliente WHERE c.id_cliente = ?;
   ```

4. **Eliminar venta:** Elimina una venta de la base de datos, asegurándose de borrar también los productos asociados en detalle_ventas.
   Ejemplo de consulta SQL:
   ```
   DELETE FROM ventas WHERE id_venta = ?;
   ```

4.4 Gestión de clientes

Permite administrar la base de clientes de la empresa.

1. **Agregar cliente:** Registra un nuevo cliente en la base de datos, incluyendo datos como:

   - Nombre.
   - Dirección.
   - Contacto.

   Ejemplo de inserción SQL:
   ```
   INSERT INTO clientes (nombre, direccion, contacto) VALUES (?, ?, ?);
   ```

2. **Listar clientes:** Muestra todos los clientes registrados en la base de datos.
   Ejemplo de inserción SQL:
   ```
   SELECT * FROM clientes;
   ```

3. **Actualizar cliente:** Permite modificar los datos de un cliente existente, como su dirección o contacto.
   Ejemplo de consulta SQL:
   ```
   UPDATE clientes SET nombre = ?, direccion = ?, contacto = ?  WHERE id_cliente
   = ?;
   ```

4. **Eliminar cliente:** Borra un cliente de la base de datos, asegurando que no tenga ventas pendientes.
   Ejemplo de eliminación SQL:
   ```
   DELETE FROM clientes WHERE id_cliente = ?;
   ```

5 Pruebas realizadas

Para garantizar el correcto funcionamiento de la aplicación, se realizaron pruebas manuales en cada una de las funcionalidades principales. Estas pruebas se enfocaron en la interacción con la base de datos, asegurando que las operaciones de inserción, recuperación, actualización y eliminación de datos se ejecutaran sin errores.

5.1 Inserción correcta de datos en la base de datos

Se probaron los métodos de inserción en las tablas productos, clientes, proveedores y ventas, verificando que los datos ingresados aparecieran correctamente en la base de datos.

1. **Caso de prueba:** Insertar un nuevo cliente.
   - **Entrada:**
     Nombre: Juan Pérez
     Dirección: Calle 123
     Contacto: 654537875
   - **Resultado esperado:** Cliente agregado exitosamente a la base de datos.
   - **Resultado obtenido:** Cliente registrado y visible en la tabla clientes.
2. **Caso de prueba:** Insertar un nuevo producto con datos válidos.
   - **Entrada:**
     Nombre: Coca Cola
     Precio: 200.00€
     Stock: 100 unidades
   - **Resultado esperado:** Producto agregado correctamente.
   - **Resultado Obtenido:** Producto registrado en la base de datos y visible en la lista de productos.

5.2 Recuperación de listas completas sin errores

Se realizaron consultas para obtener todos los registros de cada tabla (clientes, productos, proveedores, ventas), verificando que los datos almacenados coincidieran con los esperados.

1. **Caso de prueba:** Mostrar lista de clientes.
   - **Consulta ejecutada:** `SELECT * FROM clientes;`
   - **Resultado esperado:** Listado completo de clientes sin errores.

- **Resultado obtenido:** Todos los clientes registrados aparecieron correctamente en la consola.

2. **Caso de prueba:** Mostrar lista de ventas asociadas a un cliente específico.
   - **Consulta ejecutada:** `SELECT * FROM ventas WHERE id_cliente = 1;`
   - **Resultado esperado:** Ventas del cliente con ID 1 listadas correctamente.
   - **Resultado obtenido:** Los datos recuperados fueron los esperados.

5.3 Actualización precisa de registros existentes

Se probaron los métodos de actualización para modificar registros existentes en la base de datos.
Caso de prueba: Actualizar los datos de un cliente.

- **Entrada:**
  ID Cliente: 2
  Nuevo Nombre: María López
  Nueva Dirección: Calle 456
  Nuevo Contacto: 654537875
- **Consulta ejecutada:** `UPDATE clientes SET nombre='María López', direccion='Calle 456', contacto='654537875' WHERE id_cliente=2;`
- **Resultado esperado:** Datos del cliente actualizados correctamente.
- **Resultado obtenido:** Los cambios se reflejaron en la base de datos y al listar los clientes.

5.4 Eliminación de registros sin afectar otros datos

Se verificó que al eliminar registros, los datos relacionados en otras tablas no se vieran afectados (salvo cuando existían restricciones de clave foránea).
Caso de prueba: Eliminar un proveedor.

- **Entrada:** ID Proveedor: 3
- **Consulta ejecutada:** `DELETE FROM proveedores WHERE id_proveedor=3;`
- **Resultado esperado:** Proveedor eliminado sin afectar otros registros.
- **Resultado obtenido:** El proveedor fue eliminado correctamente, y los demás datos permanecieron intactos.

6 Tecnologías utilizadas

En el desarrollo de este sistema de gestión de supermercado, se han empleado diversas tecnologías que garantizan un funcionamiento eficiente, seguro y escalable. A continuación, se detallan los principales componentes tecnológicos utilizados:

- **Lenguaje de programación:** Java(versión JDK 26).
- **Bases de datos:** MySQL .
- **Frameworks:** JDBC para la conexión y manipulación de datos.
- **Entorno de desarrollo:** Intellij IDEA.