Manual del Proyecto JDBC: Tienda MultiProductos

Beltrán González y Javier Olivas

9 de febrero de 2025

Índice

1. Introducción

En este documento se describe en detalle el desarrollo del sistema de gestión de inventario y ventas para la tienda virtual **Tienda MultiProductos**. Este proyecto utiliza **Java DataBase Connectivity (JDBC)** para conectar y manipular una base de datos relacional creada en **MySQL**.

El objetivo principal del sistema es gestionar de manera eficiente el inventario, los proveedores, los pedidos y las ventas de la tienda. Para ello, se ha implementado una base de datos relacional que permite realizar operaciones **CRUD** (Crear, Leer, Actualizar, Eliminar) y gestionar transacciones de manera segura y eficiente.

Este manual está dirigido a desarrolladores, administradores de sistemas y cualquier persona interesada en comprender el funcionamiento interno del proyecto.

2. Descripción del Proyecto

El proyecto consiste en un sistema de gestión de inventario y ventas que permite realizar las siguientes operaciones:

2.1. Registro de Productos

Permite ingresar nuevos productos al inventario, asociándolos a un proveedor, estableciendo un precio y una cantidad inicial de stock. Esta función es esencial para mantener actualizado el inventario y garantizar que los productos se registren correctamente en el sistema.

2.2. Actualización del Inventario

El sistema actualiza automáticamente el stock después de cada venta o cuando se reciben pedidos de proveedores. Esta función asegura que el inventario refleje con precisión la cantidad de productos disponibles, evitando errores en la gestión del stock.

2.3. Eliminación de Productos

Permite eliminar productos por categoría o buscar por nombre o código. Esta función facilita la eliminación de productos dentro del inventario, mejorando la eficiencia en su gestión.

2.4. Venta de Productos

Las ventas se registran ingresando productos y cantidades. El sistema calcula automáticamente el precio total, incluyendo impuestos, agilizando el proceso de venta y asegurando que las transacciones se realicen correctamente.

El sistema está diseñado para ser utilizado por personal administrativo, permitiendo una gestión eficiente del inventario y las ventas de la empresa.

3. Arquitectura del Sistema

3.1. Capa de Lógica de Negocio (Business Logic Layer)

Es el núcleo del sistema, donde se implementan las reglas y procesos que definen la gestión de los datos. Esta capa actúa como intermediaria entre la capa de presentación y la capa de acceso a datos.

Funciones principales:

- Gestión del registro, actualización, eliminación y consulta de proveedores, productos, pedidos y ventas.
- Validación de datos antes de enviarlos a la base de datos.
- Cálculo del stock disponible después de cada venta.
- Gestión de transacciones para garantizar la integridad de los datos.

3.2. Capa de Acceso a Datos (Data Access Layer)

Es responsable de interactuar directamente con la base de datos. En este proyecto, se encarga de ejecutar consultas SQL y gestionar la conexión con la base de datos.

Funciones principales:

- Establecer y gestionar la conexión con la base de datos.
- Ejecutar consultas SQL para insertar, actualizar, eliminar y recuperar datos.
- Mapear resultados de consultas SQL a objetos para su uso en la capa de lógica de negocio.
- Gestionar transacciones para asegurar que las operaciones se completen correctamente o se reviertan en caso de error.

4. Base de Datos

La base de datos relacional está compuesta por cuatro tablas principales: **Proveedores**, **Productos**, **Pedidos** y **Ventas**.

4.1. Diseño de la Base de Datos

El diseño sigue los principios de normalización para garantizar la integridad de los datos.

Tablas principales:

- **Proveedores:** Almacena información sobre los proveedores, incluyendo nombre, contacto y dirección. Cada proveedor tiene un identificador único (`Id_proveedor`).
- **Productos:** Contiene detalles de los productos, como nombre, categoría, precio, stock y fecha de caducidad. Cada producto está asociado a un proveedor mediante la clave foránea `Id_proveedor`.

- **Pedidos:** Registra los pedidos realizados, incluyendo fecha y cantidad. Cada pedido está vinculado a un producto mediante la clave foránea `Id_producto`.
- **Ventas:** Registra las ventas realizadas, incluyendo fecha, cantidad vendida y monto total. Cada venta está asociada a un producto mediante la clave foránea `Id_producto`.

4.2. Esquema de la Base de Datos

El esquema se define mediante sentencias SQL. A continuación, se muestra un ejemplo:

```sql
-- Creaci n de la tabla Proveedores
CREATE TABLE IF NOT EXISTS Proveedores (
    Id_proveedor INT AUTO_INCREMENT PRIMARY KEY,
    Nombre VARCHAR(255),
    Contacto VARCHAR(255),
    Direccion VARCHAR(255)
);

-- Creaci n de la tabla Productos
CREATE TABLE IF NOT EXISTS Productos (
    Id_producto INT AUTO_INCREMENT PRIMARY KEY,
    Id_proveedor INT,
    Nombre VARCHAR(255),
    Categoria VARCHAR(255),
    Precio DECIMAL(10, 2),
    Stock INT,
    FOREIGN KEY (Id_proveedor) REFERENCES Proveedores(Id_proveedor)
    ON DELETE CASCADE ON UPDATE CASCADE
);

-- Creaci n de la tabla Pedidos
CREATE TABLE IF NOT EXISTS Pedidos (
    Id_pedidos INT AUTO_INCREMENT PRIMARY KEY,
    Id_producto INT,
    Fecha_pedido DATETIME,
    Cantidad_total INT,
    FOREIGN KEY (Id_producto) REFERENCES Productos(Id_producto)
    ON DELETE SET NULL ON UPDATE CASCADE
);

-- Creaci n de la tabla Ventas
CREATE TABLE IF NOT EXISTS Ventas (
    Id_venta INT AUTO_INCREMENT PRIMARY KEY,
    Fecha_venta DATETIME,
    Id_producto INT,
    Cantidad_producto INT,
    Monto_venta DECIMAL(10, 2),
    FOREIGN KEY (Id_producto) REFERENCES Productos(Id_producto)
    ON DELETE SET NULL ON UPDATE CASCADE
);
```

4.3. Inserción de Datos Iniciales

Para facilitar pruebas y demostraciones, se han insertado datos iniciales en las tablas. A continuación, un ejemplo de inserción:

```sql
-- Inserci n de datos en la tabla Proveedores
INSERT INTO Proveedores (Nombre, Contacto, Direccion)
VALUES
('TechDistribuidora', 'tech@distribuidora.com', 'Calle 123, Madrid'),
('ElectronicaGlobal', 'info@electronica.com', 'Avenida 456, Barcelona'),
('InnovaTech', 'contact@innovatech.com', 'Calle 789, Valencia');

-- Inserci n de datos en la tabla Productos
INSERT INTO Productos (Id_proveedor, Nombre, Categoria, Precio, Stock)
VALUES
(1, 'ThinkPad X1 Lenovo', 'Electr nica', 1800.00, 40),
(2, 'MacBook Pro Apple', 'Electr nica', 2200.00, 50),
(3, 'Dell XPS 13', 'Electr nica', 1600.00, 46);
```

5. Tecnologías Utilizadas

5.1. JDBC (Java DataBase Connectivity)

JDBC es una API de Java que permite a las aplicaciones interactuar con bases de datos relacionales. En este proyecto, se utiliza para ejecutar consultas SQL y gestionar transacciones.

5.2. MySQL

MySQL es un sistema de gestión de bases de datos relacionales (RDBMS) de código abierto. Se eligió por su robustez, escalabilidad y facilidad de uso.

5.3. Java

Java es el lenguaje de programación principal utilizado en este proyecto. Su portabilidad y amplio ecosistema de librerías lo hacen ideal para el desarrollo de aplicaciones empresariales.

6. Desarrollo del Proyecto

6.1. Conexión a la Base de Datos

La conexión a la base de datos se establece mediante JDBC. A continuación, un ejemplo de código:

```java
protected Connection getConnection() throws SQLException {
        Properties properties = new Properties();

        try (InputStream input = Tabla.class.getClassLoader().
   getResourceAsStream(CONFIG_FILE)) {
            if (input == null) {
                throw new RuntimeException("No se encontr  el archivo
   de configuraci n.");
            }
            properties.load(input);
        } catch (Exception e) {
```

```
            throw new RuntimeException("Error cargando configuraci n",
    e);
        }

        String url = properties.getProperty("db.url");
        String user = properties.getProperty("db.user");
        String password = properties.getProperty("db.password");

        return DriverManager.getConnection(url, user, password);
    }
```

6.2. Operaciones CRUD

Las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) se implementan utilizando JDBC. A continuación, un ejemplo de inserción:

```
public void insertar() throws SQLException {
    System.out.print("    Id_proveedor: ");
    int id_proveedor = sc.nextInt();
    System.out.print("    Nombre: ");
    String nombre = sc.next();
    System.out.print("    Precio: ");
    float precio = sc.nextFloat();
    System.out.print("    Stock: ");
    int stock = sc.nextInt();
    String sql = "INSERT INTO PRODUCTOS (id_proveedor, nombre, precio,
   stock) VALUES (?, ?, ?, ?)";
    try (Connection c = getConnection();
    PreparedStatement ps = c.prepareStatement(sql)){
        ps.setInt(1, id_proveedor);
        ps.setString(2, nombre);
        ps.setFloat(3, precio);
        ps.setInt(4, stock);
        ps.executeUpdate();
    }
}
```

Ejemplo de actualización:

```
public void actualizar() throws SQLException {
    System.out.print("    Id_producto a modificar: ");
    int id = sc.nextInt();
    System.out.print("    Id_proveedor: ");
    int id_proveedor = sc.nextInt();
    System.out.print("    Nombre: ");
    String nombre = sc.next();
    System.out.print("    Precio: ");
    Float precio = sc.nextFloat();
    System.out.print("    Stock: ");
    int stock = sc.nextInt();
    String sql = "UPDATE PRODUCTOS SET id_proveedor = ?, nombre = ?,
   precio = ?, stock = ? WHERE id_producto = ?";
    try (Connection c = getConnection();
    PreparedStatement ps = c.prepareStatement(sql)){
        ps.setInt(1, id_proveedor);
        ps.setString(2, nombre);
        ps.setFloat(3, precio);
        ps.setInt(4, stock);
```

```
        ps.setInt(5, id);
        ps.executeUpdate();
    }
}
```

6.3. Gestión de Transacciones

Las transacciones se gestionan para garantizar la integridad de los datos. A continuación, un ejemplo:

```
public void insertar() throws SQLException,
    CantidadInsuficienteException {
    List<String> productos = new ArrayList<>();
    String sql = "SELECT * FROM PRODUCTOS";
    try (Connection c = getConnection();
         PreparedStatement ps = c.prepareStatement(sql)) {
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String producto = "    ID[" + rs.getInt("id_producto") + "]
    Nombre[" + rs.getString("nombre") + "] Precio[" + rs.getFloat("
    precio") + "   ] Stock[" + rs.getInt("stock") + "]";
            productos.add(producto);
        }
    }
    for (String producto : productos) {
        System.out.println(producto);
    }

    System.out.print("    ID Producto: ");
    int Id_producto = sc.nextInt();
    System.out.print("    Fecha Venta (YYYY-MM-DD): ");
    String fecha_venta = sc.next();
    System.out.print("    Cantidad Producto: ");
    int Cantidad_producto = sc.nextInt();
    float Monto_venta = 0;

    try (Connection c = getConnection()) {
        String sql1 = "SELECT Precio, Stock FROM Productos WHERE
    Id_producto = ?";
        try (PreparedStatement ps1 = c.prepareStatement(sql1)) {
            ps1.setInt(1, Id_producto);
            try (ResultSet rs1 = ps1.executeQuery()) {
                if (rs1.next()) {
                    float precio = rs1.getFloat("Precio");
                    int stockActual = rs1.getInt("Stock");

                    if (stockActual >= Cantidad_producto) {
                        Monto_venta = precio * Cantidad_producto;
                    } else {
                        throw new CantidadInsuficienteException("
    Cantidad Insuficiente");
                    }
                } else {
                    throw new SQLException("Producto no encontrado");
                }
            }
        }
```

```
43
44          String sql2 = "INSERT INTO VENTAS (Id_producto, Fecha_venta,
    Cantidad_producto, Monto_venta) VALUES (?, ?, ?, ?)";
45          try (PreparedStatement ps2 = c.prepareStatement(sql2)) {
46              ps2.setInt(1, Id_producto);
47              ps2.setString(2, fecha_venta);
48              ps2.setInt(3, Cantidad_producto);
49              ps2.setFloat(4, Monto_venta);
50              ps2.executeUpdate();
51          }
52      }
53  }
```

7. Esquemas y Diagramas

7.1. Diagrama de Entidad-Relación (ER)

El diagrama ER muestra las relaciones entre las entidades principales: Proveedores, Productos, Pedidos y Ventas.

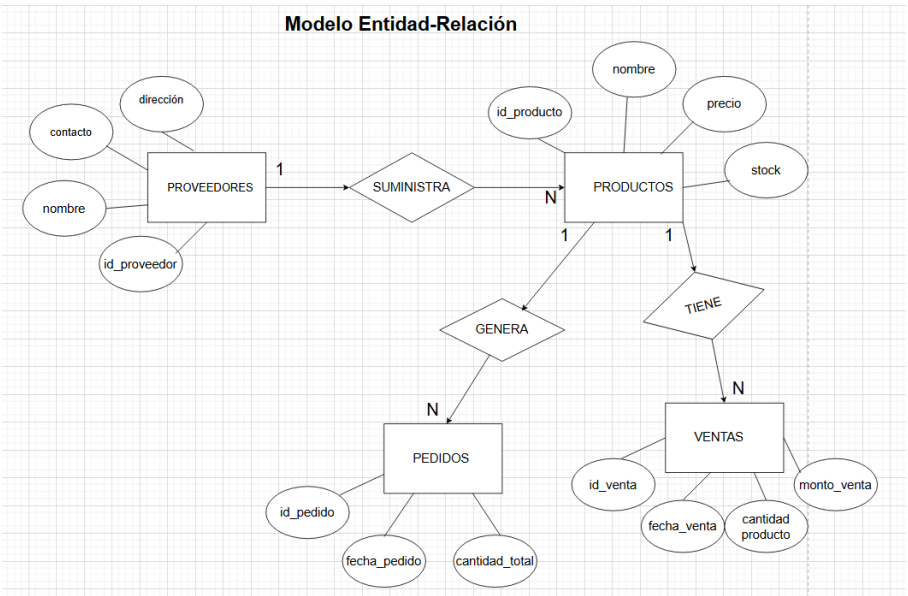

Figura 1: Modelo Entidad-Relación

7.2. Diagrama de Clases

El diagrama de clases representa la estructura de las clases en el código Java, incluyendo sus atributos y métodos.

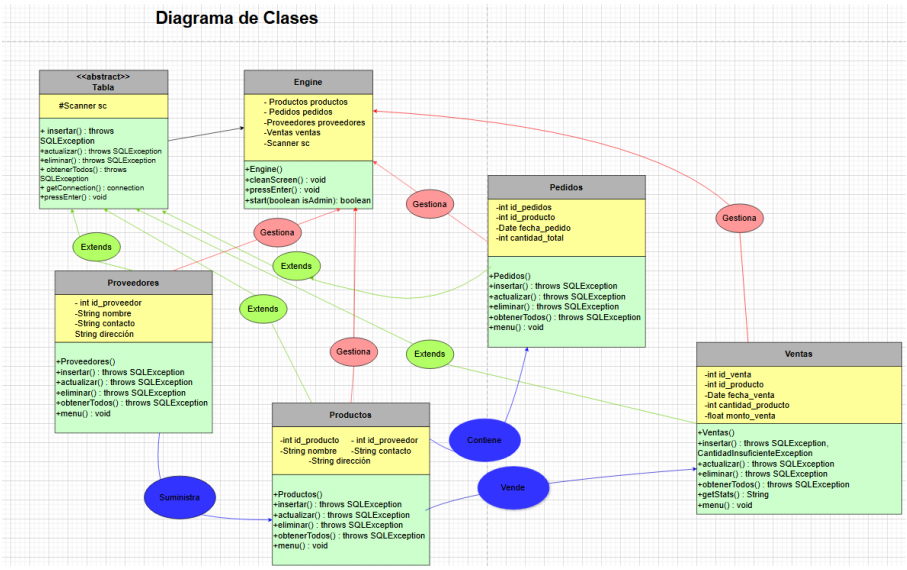

Figura 2: Diagrama de clases

7.3. Diagrama Relacional

El diagrama relacional muestra las tablas de la base de datos y sus relaciones.

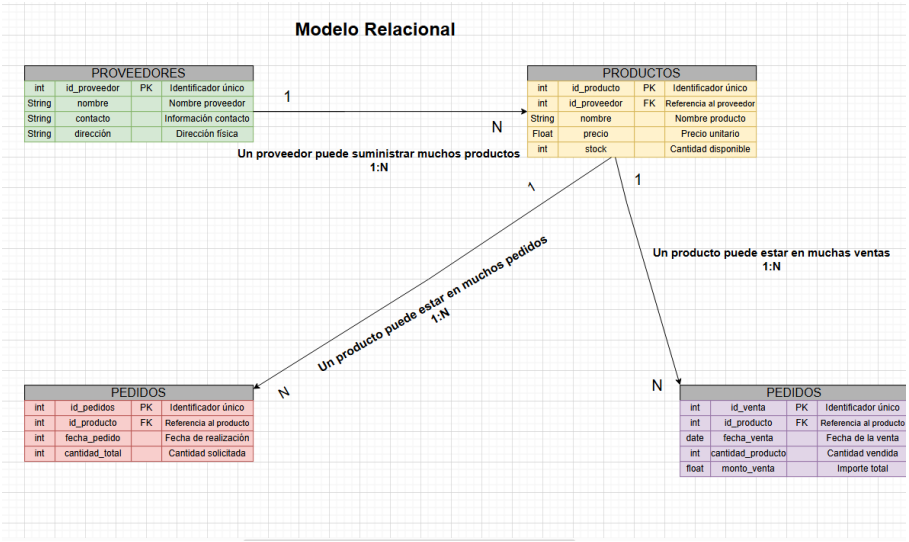

Figura 3: Modelo relacional

8. Conclusión

El sistema de gestión de inventario y ventas desarrollado en este proyecto demuestra cómo las tecnologías **JDBC**, **MySQL** y **Java** pueden combinarse para crear una solución robusta y escalable. La arquitectura de tres capas garantiza una separación clara de responsabilidades, facilitando el mantenimiento y la expansión del sistema.

Este proyecto no solo cumple con los requisitos funcionales, sino que también sirve como base para futuras mejoras, como la integración de nuevas funcionalidades o la migración a tecnologías más avanzadas.