UNIVERSIDAD SIGLO 21

Seminario de Práctica de Informática

Licenciatura en Informática

Sistema de Gestión y Análisis de Ventas

Optimización de procesos y apoyo a la toma de decisiones

Titular Experto: Ana Carolina Ferreyra

Alumno: Agustín Tiziano Greco

Fecha de Entrega: 16/11/25

Índice

Sección TP N°1 3
Introducción del TP N°1 3
Justificación del TP N°1 3
Definición del Proyecto 3
Objetivo del Proyecto 4
Definición del Sistema 4
Objetivo del Sistema 4
Elicitación 5
Requerimientos Funcionales 5
Requerimientos No Funcionales 5
Conocimiento del Negocio 6
Propuesta de Solución 7
Propuesta Funcional 7
Propuesta Técnica 8
Casos de Uso 8
Casos de Uso Detallados 10
Sección TP N°2 14
Introducción del TP N°2 14
Justificación del TP N°2 14
Etapa de análisis 15
Etapa de diseño 22
Etapa de implementación 22
Etapa de pruebas 23
Definición de base de datos 25
Diagrama entidad-relación de la base de datos 26
Creación de las tablas MySQL 26
Inserción, consulta y borrado de registros 28
Presentación de las consultas SQL 30
Definiciones de comunicación 32
Sección del TP N°3 34
Explicación del Desarrollo en Java del TP N°3 34
Presentación del Desarrollo en Java del TP N°3 42
Sección del TP N°4 43
Explicación del Desarrollo en Java y Conexión a la Base de Datos 43

Presentación del Desarrollo en Java del TP N°4 57

Referencias..... 58

Sección TP N°1

Introducción del TP N°1

Las PyMEs desempeñan un papel clave en la economía, pero muchas enfrentan dificultades para organizar la información de productos, clientes y ventas. Esto genera ineficiencias y limita la toma de decisiones. Con este proyecto se busca desarrollar un sistema que centralice y automatice esos procesos, apoyando la competitividad de estas empresas. Según la CEPAL (2022), la transformación digital es uno de los principales retos de las PyMEs en la región, lo cual refuerza la relevancia de la propuesta. Este trabajo surge para dar respuesta a esa necesidad real de las PyMEs, ofreciendo una herramienta accesible, escalable y alineada con su crecimiento.

Justificación del TP N°1

La implementación de un sistema de gestión y análisis de ventas en una PyME resulta necesaria para afrontar dificultades como la dispersión de la información, duplicación de tareas y falta de datos confiables en tiempo real. El proyecto se justifica porque responde a la necesidad de mejorar la administración integral de productos, clientes, ventas y usuarios, permitiendo a las empresas tomar decisiones basadas en información precisa y actualizada. Además, ofrece la posibilidad de generar reportes y métricas que facilitan el control del negocio y la planificación estratégica. A nivel tecnológico, promueve la digitalización; y a nivel social/empresarial, empodera a las PyMEs como motor económico y generador de empleo.

Definición del Proyecto

El proyecto consiste en analizar, diseñar y documentar un sistema informático para la gestión y análisis de ventas en PyMEs.
Dentro de este trabajo se realizará:

- La identificación y justificación del problema.
- La definición de requerimientos funcionales y no funcionales.
- La elaboración de casos de uso, diagramas y propuesta técnica.
- La construcción de un prototipo inicial que permita validar la viabilidad de la solución.

Objetivo del Proyecto

Objetivo General

- Analizar, diseñar y documentar una solución informática para la gestión y análisis de ventas en PyMEs, aplicando buenas prácticas de desarrollo de software.

Objetivos Específicos

- Identificar los procesos actuales de gestión de ventas y clientes en PyMEs.
- Relevar y documentar requerimientos funcionales y no funcionales.
- Elaborar modelos UML (casos de uso, diagramas de dominio) que representen el sistema.
- Diseñar una propuesta técnica factible con Java y MySQL.
- Desarrollar un prototipo inicial que permita validar el diseño planteado.

Definición del Sistema

El sistema propuesto será una aplicación de gestión de ventas que permitirá:

- Registrar, consultar y modificar productos con control de stock.
- Registrar clientes y mantener actualizada su información.
- Gestionar usuarios con diferentes roles (administrador, vendedor).
- Registrar ventas vinculadas a clientes y productos, con cálculo automático de totales.
- Generar reportes e indicadores de desempeño (por cliente, producto, período).

Objetivo del Sistema

Objetivo General

- Brindar a la PyME una herramienta digital que centralice y optimice la gestión de ventas, clientes, usuarios y productos, aportando reportes e indicadores útiles para la toma de decisiones.

Objetivos Específicos

- Permitir el registro, consulta y modificación de productos con control de stock.
- Administrar la información de clientes y su historial de compras.
- Registrar ventas y asociarlas a clientes y productos.
- Generar reportes de ventas y métricas de desempeño.
- Gestionar usuarios con distintos roles y privilegios de acceso.
- Garantizar integridad de los datos y una interfaz de uso intuitivo.

Elicitación

A continuación, se presentan los requerimientos funcionales y no funcionales del sistema:

Requerimientos Funcionales

RF-01 – Actor: Administrador. Registrar un producto nuevo con nombre, categoría, precio y stock inicial.

RF-02 – Actor: Administrador. Modificar o eliminar datos de productos existentes.

RF-03 – Actor: Administrador. Gestionar usuarios con diferentes roles (administrador y vendedor).

RF-04 – Actor: Vendedor. Registrar ventas vinculadas a clientes y productos.

RF-05 – Actor: Administrador/Vendedor. Consultar ventas por rango de fechas y generar reportes de totales.

RF-06 – Actor: Administrador. Consultar reportes de desempeño por cliente, producto o vendedor.

Requerimientos No Funcionales

RNF-01: La base de datos debe estar implementada en MySQL.

RNF-02: El sistema debe desarrollarse en Java, utilizando JDBC para la persistencia.

RNF-03: La información debe mantenerse íntegra y evitar duplicados mediante el uso de claves primarias y foráneas.

RNF-04: La interfaz del sistema debe ser simple e intuitiva para el usuario.

RNF-05: El sistema debe permitir escalar en cantidad de clientes y productos sin perder rendimiento.

Conocimiento del Negocio

Contexto de la Organización

La PyME se encuentra ocupada en la comercialización de productos, realizando ventas a clientes ocasionales, así como a clientes recurrentes. Su operación principal se centra en el registro de ventas diarias, la administración básica de clientes y el manejo de los productos disponibles. Actualmente, la empresa realiza parte de estas actividades de manera manual o con herramientas muy básicas, lo que causa dificultades para centralizar y analizar la información.

Procesos de negocio principal

• **Captura de ventas**: el personal captura cada venta manualmente o en hojas de cálculo electrónicas.

• **Gestión de clientes**: se almacena cierta información de los clientes, pero no existe un historial centralizado que facilite el análisis de patrones de compra.

• **Gestión de productos**: los productos se listan sin una estructura estandarizada, lo que dificulta la consulta de precios o categorías de productos.

• **Análisis de información**: el análisis de la información no es sistemático ni se generan informes; las decisiones se toman con base en estimaciones o información parcial.

Problemas detectados

• Los datos de ventas están dispersos y no centralizados.

• La empresa no cuenta con informes históricos ni comparativos.

• Es difícil identificar los productos más vendidos o a la mayoría de los clientes habituales.

• La decisión se basa en datos parciales o de baja credibilidad.

• Sin control integral para anticipar picos en demanda o estacionalidad.

Oportunidades de mejora

La implantación de un sistema informático permitirá:

• Incorporar toda la información en una base de datos estructurada.

• Hacer más sencillo el análisis de ventas con métricas e indicadores creíbles.

• Optimización de gestión de clientes y productos.

• Errores de eliminación mediante manipulación manual de datos.

• Suministro de una herramienta tecnológica a las PyMEs que aumentará su competitividad y crecimiento a largo plazo.

Propuesta de Solución

Propuesta Funcional

Qué hará el sistema: Registrar productos clientes y ventas. Además, generará reportes con indicadores clave como productos más vendidos, clientes frecuentes y ventas por período.

El software estará organizado en los siguientes módulos funcionales:

1. **Módulo de Gestión de Productos**

- Registrar, consultar, modificar y eliminar productos.
- Control de stock y categorización de artículos.

2. **Módulo de Gestión de Clientes**

- Registrar y consultar información de clientes.
- Mantener historial de compras y clientes frecuentes.

3. **Módulo de Ventas**

- Registrar ventas vinculadas a clientes y productos.
- Cálculo automático de subtotales y totales.
- Actualización de stock.

4. **Módulo de Reportes y Métricas**

- Reportes de ventas por período, cliente, producto o vendedor.
- Indicadores de desempeño: productos más vendidos, clientes recurrentes, etc.

5. **Módulo de Usuarios y Seguridad**

- Administración de usuarios con distintos roles (administrador, vendedor).
- Control de accesos y privilegios.

Propuesta Técnica

Cómo se implementará el sistema: Se desarrollará en Java utilizando JDBC para la conexión con una base de datos MySQL. Se implementará una arquitectura en capas para separar la lógica de negocio, la persistencia y la interfaz. Los datos se almacenarán en un esquema normalizado en tercera forma normal (3NF).

- **Lenguaje de programación**: Java.
- **Base de datos**: MySQL, con un modelo relacional normalizado (3NF).
- **Persistencia**: conexión mediante JDBC.
- **Arquitectura**: en capas (presentación, lógica de negocio y persistencia).
- **Escalabilidad**: posibilidad de añadir módulos futuros (ej. inventario avanzado, predicciones de demanda).
- **Seguridad**: control de acceso por roles y validación de integridad de datos.

Casos de Uso

CU-01: Registrar producto – Actor principal: Administrador

CU-02: Modificar o eliminar producto – Actor principal: Administrador

CU-03: Consultar productos – Actor principal: Administrador, Vendedor

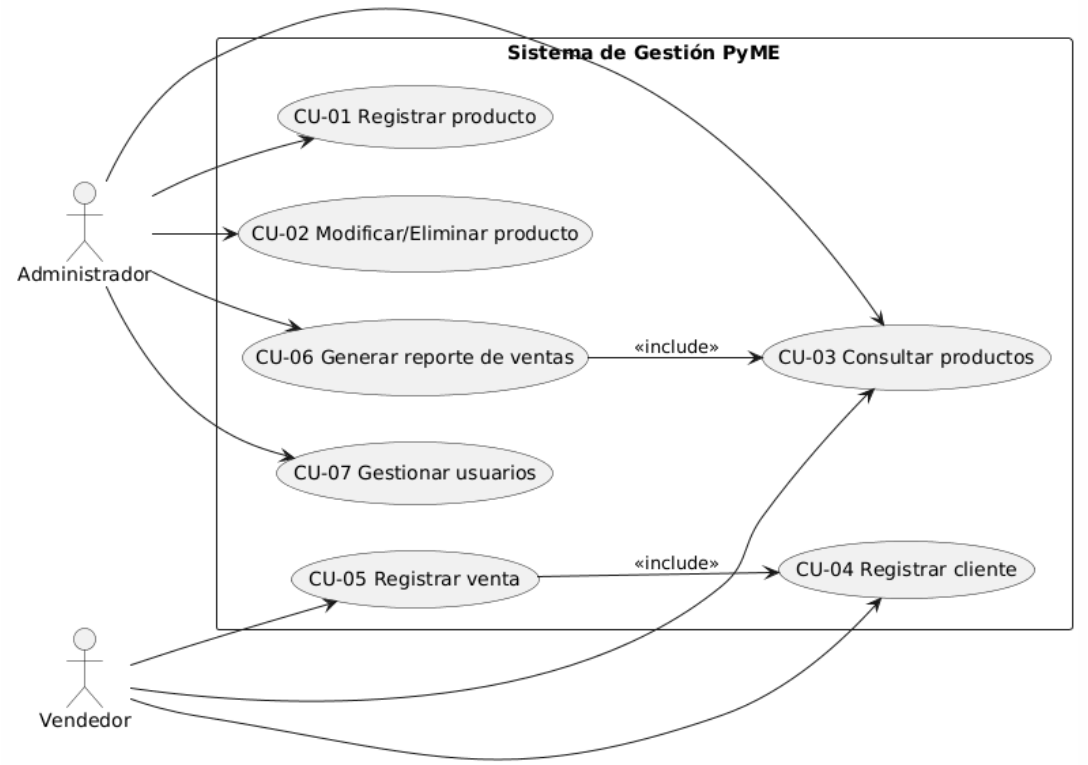
CU-04: Registrar cliente – Actor principal: Vendedor

CU-05: Registrar venta – Actor principal: Vendedor

CU-06: Generar reporte de ventas – Actor principal: Administrador

CU-07: Gestionar usuarios – Actor principal: Administrador

El diagrama de casos de uso para el sistema que se está elaborando es el siguiente:

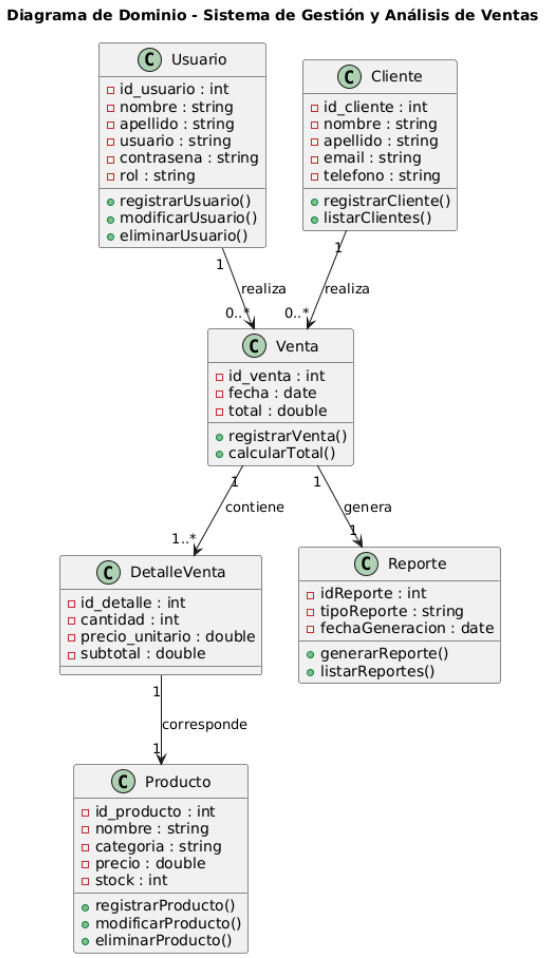


Identificación de actores

Administrador: Es la persona encargada de gestionar el sistema. Tiene privilegios para registrar, modificar o eliminar productos, generar reportes de ventas y gestionar usuarios.

Vendedor: Es el usuario autorizado para registrar clientes y ventas, así como consultar productos disponibles.

El diagrama de dominio para el sistema que se está elaborando es el siguiente:



Casos de Uso Detallados

CU-01 Registrar producto

- Actor principal: Administrador
- Precondición: Usuario autenticado como administrador.
- Flujo principal:
    1. Selecciona “Registrar producto”.
    2. Ingresa nombre, categoría, precio y stock.
    3. El sistema valida datos y guarda el producto.
- Postcondición: Producto almacenado en Productos.
- RF asociados: RF-01

CU-02 Modificar o eliminar producto

- Actor principal: Administrador
- Precondición: Usuario autenticado como administrador.
- Flujo principal:
    1. Selecciona producto a modificar o eliminar.
    2. Realiza los cambios o confirma la eliminación.
    3. El sistema valida y actualiza la información.
- Postcondición: Producto actualizado o eliminado.
- RF asociados: RF-02

CU-03 Consultar productos

- Actores principales: Administrador, Vendedor
- Precondición: Usuario autenticado.
- Flujo principal:
    1. Selecciona “Consultar productos”.
    2. El sistema muestra listado de productos.
    3. Permite filtrar por nombre, categoría o stock.
- Postcondición: Se obtiene un listado actualizado de productos.
- RF asociados: RF-01, RF-02 (para reflejar los cambios)

CU-04 Registrar cliente

- Actor principal: Vendedor
- Precondición: Vendedor autenticado.
- Flujo principal:
    1. Selecciona “Registrar cliente”.

2. Ingresa datos personales y de contacto.
3. El sistema guarda la información.

- Postcondición: Cliente almacenado en Clientes.
- RF asociados: RF-04

CU-05 Registrar venta

- Actor principal: Vendedor
- Precondición: Vendedor autenticado; existe producto en stock.
- Flujo principal:
  1. Selecciona “Registrar venta”.
  2. Selecciona cliente existente o lo registra.
  3. Agrega productos y cantidades.
  4. El sistema calcula subtotal y total.
  5. Confirma y se guarda la venta.
- Postcondición: Venta y detalles almacenados en Ventas y Detalle_Venta; stock actualizado.
- RF asociados: RF-04, RF-05

CU-06 Generar reporte de ventas

- Actores principales: Administrador, Vendedor
- Precondición: Usuario autenticado.
- Flujo principal:
  1. Selecciona rango de fechas o criterios.
  2. El sistema consulta ventas.
  3. Muestra reporte con totales.
- Postcondición: Reporte generado en pantalla, exportable.
- RF asociados: RF-05

CU-07 Gestionar usuarios

- Actor principal: Administrador
- Precondición: Usuario autenticado como administrador.
- Flujo principal:
    1. Selecciona “Gestionar usuarios”.
    2. Registra, modifica o elimina usuarios.
    3. Asigna roles y contraseñas.
- Postcondición: Usuarios actualizados en Usuarios.
- RF asociados: RF-03

Sección TP N°2

Introducción del TP N°2

En la primera entrega se presentó la problemática que enfrentan muchas PyMEs a la hora de gestionar sus ventas, clientes y reportes. Se estableció que la falta de herramientas efectivas dificulta y hace menos eficiente la toma de decisiones en los procesos internos.

Este segundo trabajo práctico lo retoma desde ese punto y busca dar un paso más: convertir el problema en una solución real. Para ello, se desarrolló un proceso iterativo e incremental que abarca el análisis, diseño, implementación y pruebas de un sistema de gestión de ventas. El objetivo no es solamente mostrar cómo funcionaría el sistema en teoría, sino también evidenciar que puede materializarse en un entorno real mediante el uso de Java, MySQL y buenas prácticas de desarrollo.

Justificación del TP N°2

En el TP1 se basó la necesidad de llevar a cabo un sistema informático que optimice la gestión de ventas en las PyMEs. En este TP2, esa base se potencia mostrando la viabilidad técnica de la propuesta.

La solución propuesta es factible porque:

- **Respuesta a una necesidad específica**: deja a las PyMEs poder organizar mejor su información de clientes, productos y ventas.
- **Apoya la toma de decisiones**: los informes generados permiten reconocer tendencias y evaluar el desempeño empresarial.
- **Iguala la factibilidad tecnológica**: el prototipo implementado demuestra que el sistema es capaz de realizarse con tecnologías disponibles y de amplio uso (Java y MySQL).

En resumen, este estudio no solo contesta al "por qué" del problema, sino también al "para qué": ofrecer una herramienta confiable que ayude a las PyMEs a mejorar su competitividad y eficacia.

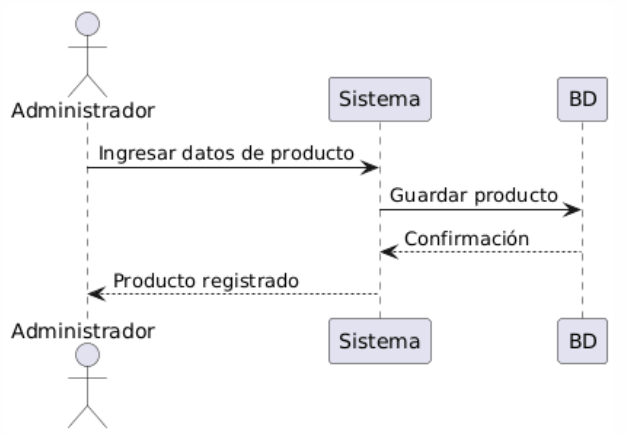
Etapa de análisis

El análisis constituye la base del sistema, ya que permite comprender cómo interactúan los actores con la aplicación y qué funcionalidades se esperan de ella.

En esta etapa se retomaron los casos de uso definidos en el TP1, ahora expresados de manera más detallada mediante diagramas de secuencia. Estos diagramas muestran, paso a paso, cómo un administrador o un vendedor utilizan el sistema para registrar productos, clientes y ventas, así como para generar reportes.

El análisis también incluye el diseño conceptual de la base de datos, asegurando que cada entidad (clientes, productos, usuarios, ventas, reportes) esté correctamente definida y vinculada. De este modo, se garantiza que los procesos de inserción, consulta y borrado de información respondan adecuadamente a las necesidades reales del negocio.

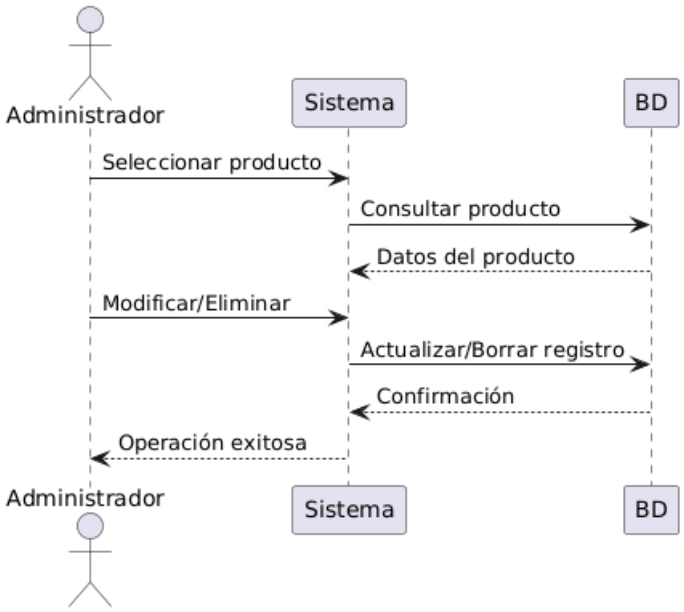
A continuación, se presenta el diagrama de secuencia para CU001, Registrar un producto:



Resultado esperado

El sistema guarda un nuevo producto en la base de datos y lo confirma el administrador.

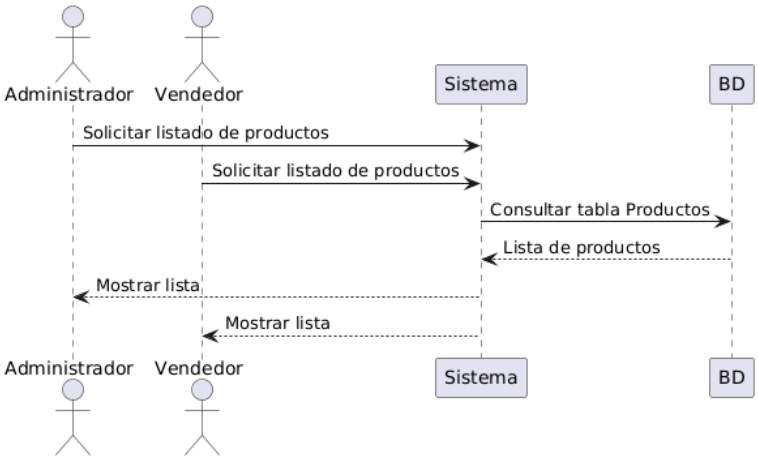
A continuación, se presenta el diagrama de secuencia para CU002, Modificar o eliminar producto:



Resultado esperado

El sistema actualiza o elimina correctamente el producto y confirma la operación al administrador.

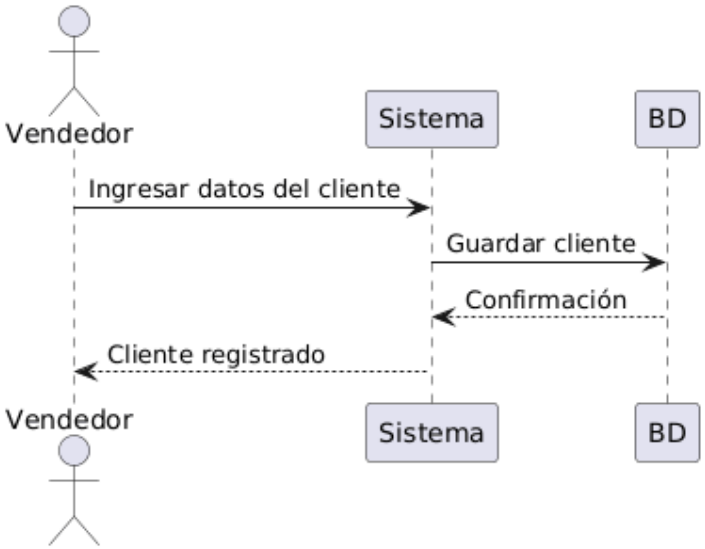
A continuación, se presenta el diagrama de secuencia para CU003, Consultar productos:



Resultado esperado

El sistema muestra la lista de productos actualizada para el administrador o vendedor.

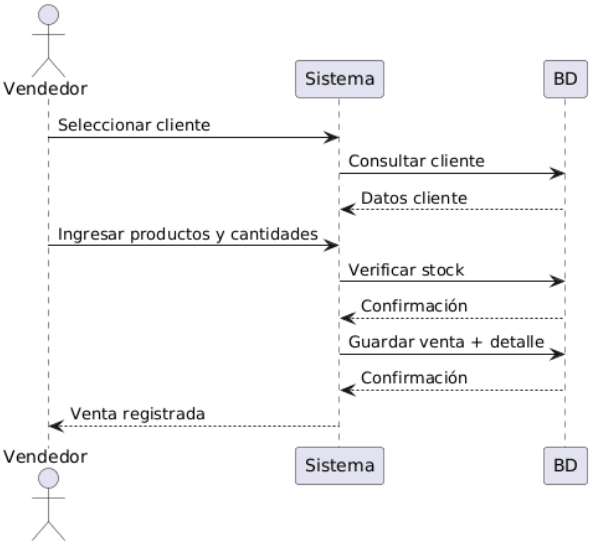
A continuación, se presenta el diagrama de secuencia para CU004, Registrar cliente:



Resultado esperado

El cliente queda registrado en la base de datos y el sistema devuelve confirmación al vendedor.

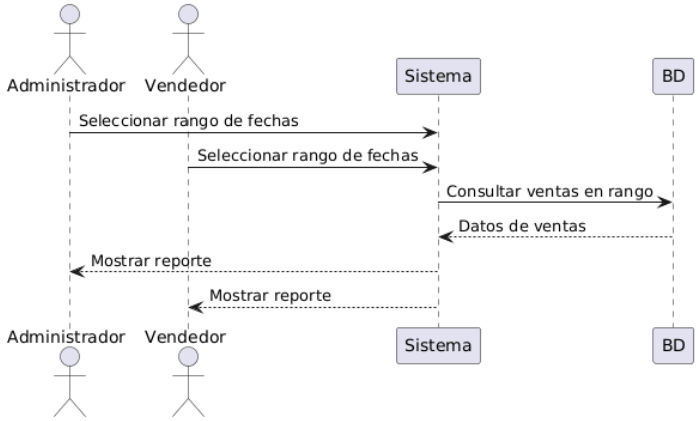
A continuación, se presenta el diagrama de secuencia para CU005, Registrar venta:



Resultado esperado

La venta se registra junto a su detalle, y se descuenta el stock de los productos vendidos.

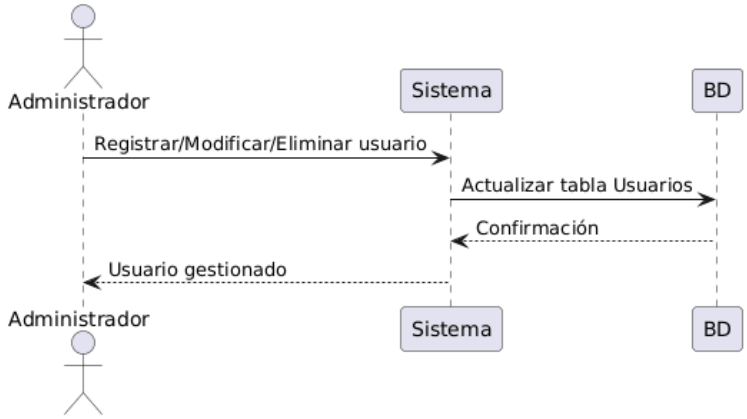
A continuación, se presenta el diagrama de secuencia para CU006, Generar reporte de ventas:



Resultado esperado

El sistema presenta un reporte con las ventas del rango de fechas solicitado.

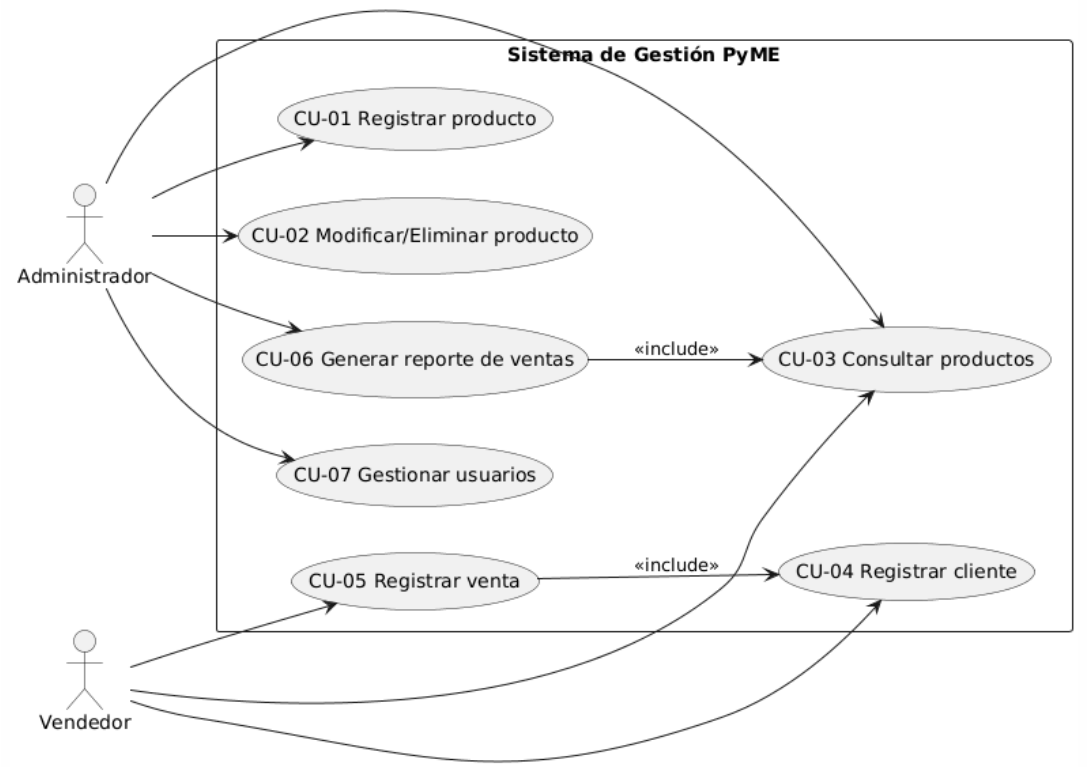
A continuación, se presenta el diagrama de secuencia para CU007, Gestionar usuarios:



Resultado esperado

El sistema actualiza la tabla de usuarios con los cambios solicitados (alta, baja o modificación).

Diagrama de casos de uso



CU-01 Registrar producto

- Actor principal: Administrador
- Precondición: Usuario autenticado como administrador.
- Flujo principal:
    1. Selecciona “Registrar producto”.
    2. Ingresa nombre, categoría, precio y stock.
    3. El sistema valida datos y guarda el producto.
- Postcondición: Producto almacenado en Productos.
- RF asociados: RF-01

CU-02 Modificar o eliminar producto

- Actor principal: Administrador
- Precondición: Usuario autenticado como administrador.
- Flujo principal:

1. Selecciona producto a modificar o eliminar.
2. Realiza los cambios o confirma la eliminación.
3. El sistema valida y actualiza la información.

- Postcondición: Producto actualizado o eliminado.
- RF asociados: RF-02

CU-03 Consultar productos

- Actores principales: Administrador, Vendedor
- Precondición: Usuario autenticado.
- Flujo principal:
  1. Selecciona “Consultar productos”.
  2. El sistema muestra listado de productos.
  3. Permite filtrar por nombre, categoría o stock.
- Postcondición: Se obtiene un listado actualizado de productos.
- RF asociados: RF-01, RF-02 (para reflejar los cambios)

CU-04 Registrar cliente

- Actor principal: Vendedor
- Precondición: Vendedor autenticado.
- Flujo principal:
  1. Selecciona “Registrar cliente”.
  2. Ingresa datos personales y de contacto.
  3. El sistema guarda la información.
- Postcondición: Cliente almacenado en Clientes.
- RF asociados: RF-04

CU-05 Registrar venta

- Actor principal: Vendedor

- Precondición: Vendedor autenticado; existe producto en stock.
- Flujo principal:
    1. Selecciona “Registrar venta”.
    2. Selecciona cliente existente o lo registra.
    3. Agrega productos y cantidades.
    4. El sistema calcula subtotal y total.
    5. Confirma y se guarda la venta.
- Postcondición: Venta y detalles almacenados en Ventas y Detalle_Venta; stock actualizado.
- RF asociados: RF-04, RF-05

CU-06 Generar reporte de ventas

- Actores principales: Administrador, Vendedor
- Precondición: Usuario autenticado.
- Flujo principal:
    1. Selecciona rango de fechas o criterios.
    2. El sistema consulta ventas.
    3. Muestra reporte con totales.
- Postcondición: Reporte generado en pantalla, exportable.
- RF asociados: RF-05

CU-07 Gestionar usuarios

- Actor principal: Administrador
- Precondición: Usuario autenticado como administrador.
- Flujo principal:
    1. Selecciona “Gestionar usuarios”.
    2. Registra, modifica o elimina usuarios.
    3. Asigna roles y contraseñas.

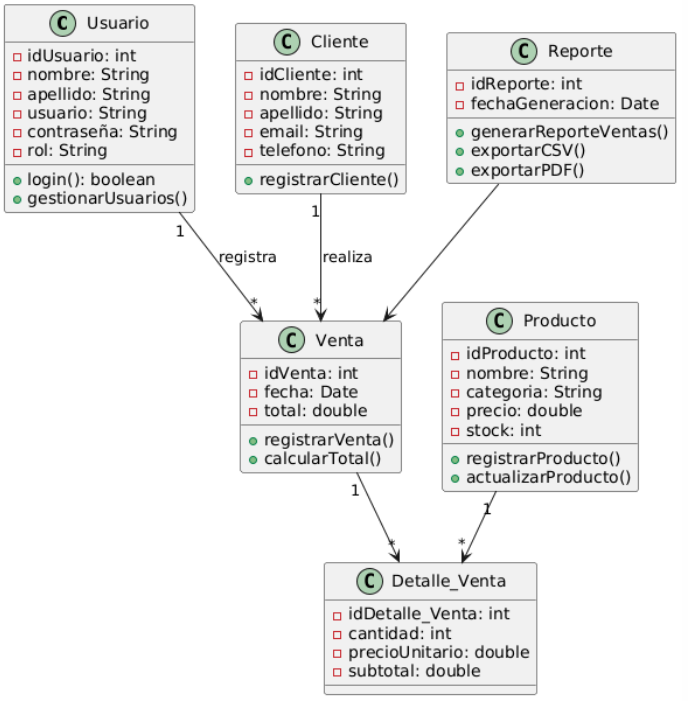
- Postcondición: Usuarios actualizados en Usuarios.
- RF asociados: RF-03

Etapa de diseño

El diseño del sistema se apoya en diagrama de clases, donde cada clase corresponde a una entidad de la base de datos. Cada clase contiene atributos y operaciones que reflejan las responsabilidades del sistema.

Además, se elaboró un diagrama de despliegue que muestra la arquitectura de comunicación: la aplicación Java (cliente) se conecta mediante JDBC al servidor MySQL (base de datos). Este esquema facilita la escalabilidad y garantiza que múltiples clientes puedan acceder de manera concurrente a la información centralizada.

Diagrama de clases de diseño



Etapa de implementación

La etapa de implementación consiste en transformar los diseños realizados en código ejecutable, definiendo la estructura de la base de datos y el esqueleto de las funcionalidades del sistema.

Requerimientos del sistema

RNF-01: La base de datos debe estar implementada en MySQL.

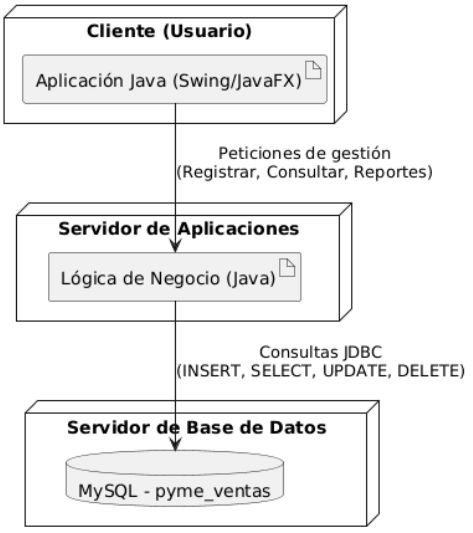
RNF-02: El sistema debe desarrollarse en Java, utilizando JDBC para la persistencia.

RNF-03: La información debe mantenerse íntegra y evitar duplicados mediante el uso de claves primarias y foráneas.

RNF-04: La interfaz del sistema debe ser simple e intuitiva para el usuario.

RNF-05: El sistema debe permitir escalar en cantidad de clientes y productos sin perder rendimiento.

Diagrama de despliegue



Etapa de pruebas

Plan de pruebas

Id Prueba	Caso de uso	Acción	Datos de entrada	Resultado esperado	Resultado obtenido
P-01	CU-01 Registrar producto	Insertar producto en la base	Nombre: "Notebook Lenovo", Categoría: "Tecnología",	El producto se guarda en la tabla Producto con un	✅ Producto insertado correctamente

			Precio: 450000, Stock: 10	idProducto único	
P-02	CU-02 Modificar o eliminar producto	Cambiar precio del mouse	idProducto=2, nuevo precio=16000	El precio se actualiza en la tabla Producto	✅ Precio actualizado
P-03	CU-03 Consultar productos	Ejecutar consulta SELECT * FROM Producto	-	Devuelve lista de productos disponibles	✅ Se listan los productos cargados
P-04	CU-04 Registrar cliente	Insertar cliente en la base	Nombre: "María", Apellido: "Pérez", Email: "mperez@mail.com", Tel: "1122334455"	El cliente se almacena en la tabla Cliente con un idCliente único	✅ Cliente registrado correctamente
P-05	CU-05 Registrar venta	Registrar venta de cliente con productos	Cliente=María (id=1), Productos: (Notebook, Mouse)	Venta registrada en Venta y detalles en Detalle_Venta; stock actualizado	✅ Venta registrada y stock descontado
P-06	CU-06 Generar reporte de ventas	Consultar ventas por rango de fechas	Fecha = '2025-09-25'	Devuelve ventas del día y total acumulado	✅ Reporte generado correctamente
P-07	CU-07 Gestionar usuarios	Eliminar usuario Carlos López	idUsuario=2	El usuario se elimina; sus ventas y detalles asociados también	✅ Usuario eliminado y datos relacionados borrados

Validación de pruebas

- Todas las pruebas se ejecutaron correctamente.
- El sistema permite insertar, consultar, modificar y eliminar registros de acuerdo con los casos de uso.
- Se mantiene la integridad referencial gracias al control de claves foráneas y restricciones en la base de datos.

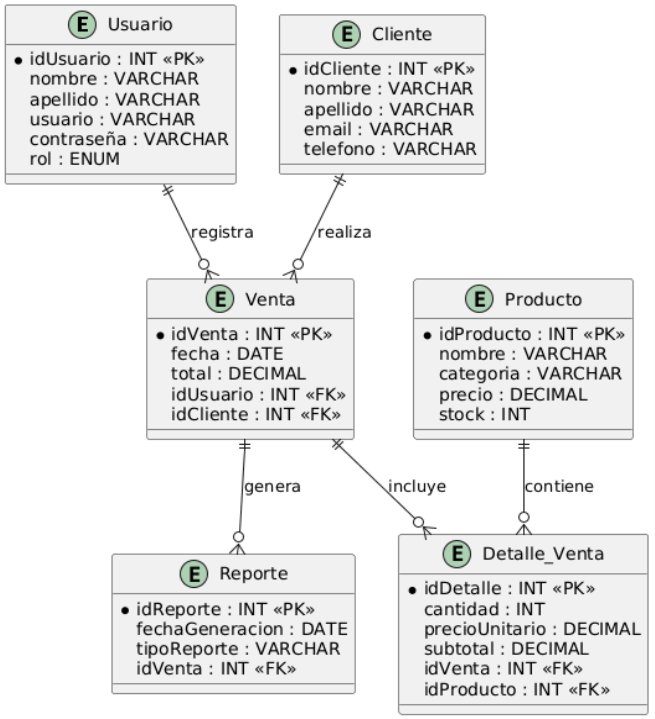
Definición de base de datos

Se diseñó una base de datos MySQL con las siguientes tablas: Usuario, Cliente, Producto, Venta, Detalle_Venta y Reporte. Cada tabla posee un identificador único y claves foráneas que garantizan la integridad referencial.

Las principales entidades son:

- **Usuario**: gestiona el acceso al sistema y define roles (Administrador, Vendedor).
- **Cliente**: almacena los datos de los clientes.
- **Producto**: contiene la información de productos disponibles.
- **Venta**: registra las operaciones de venta, asociando cliente, usuario y fecha.
- **Detalle_Venta**: descompone cada venta en productos, cantidades y subtotales.
- **Reporte**: almacena reportes generados a partir de ventas, con fecha y criterios de generación.

Diagrama entidad-relación de la base de datos



Relaciones:

- Un Usuario puede registrar varias Ventas.
- Un Cliente puede estar asociado a varias Ventas.
- Una Venta puede incluir múltiples Productos a través de Detalle_Venta.
- Reporte está vinculado a una Venta (cada reporte corresponde a una venta o grupo de ventas).

Creación de las tablas MySQL

En esta fase se tradujo el diseño lógico a estructuras físicas dentro de la base de datos MySQL. Se crearon las tablas necesarias para gestionar usuarios, clientes, productos y ventas, respetando las claves primarias y foráneas que aseguran la integridad referencial. Posteriormente, se realizaron inserciones de datos de prueba, consultas para obtener información relevante y operaciones de borrado controlado para validar el correcto funcionamiento del modelo. A continuación, se muestran las definiciones de cada tabla junto con ejemplos de uso.

Tabla «usuario»

Contiene un identificador único (idUsuario) para cada usuario del sistema. Se registran datos personales, credenciales de acceso y el rol asignado, que define los permisos dentro del sistema.

```sql
-- Tabla Usuarios
CREATE TABLE Usuario (
    idUsuario INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    usuario VARCHAR(50) UNIQUE NOT NULL,
    contraseña VARCHAR(100) NOT NULL,
    rol ENUM('Administrador','Vendedor') NOT NULL
);
```

Tabla «cliente»

Contiene un identificador único (idCliente) para cada cliente del sistema. Se registran datos personales, que define los permisos dentro del sistema.

```sql
-- Tabla Clientes
CREATE TABLE Cliente (
    idCliente INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    email VARCHAR(100),
    telefono VARCHAR(50)
);
```

Tabla «producto»

Contiene un identificador único (idProducto) para cada producto del sistema. Se registran datos sobre el producto en cuestión.

```sql
-- Tabla Productos
CREATE TABLE Producto (
    idProducto INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    categoria VARCHAR(100),
    precio DECIMAL(10,2) NOT NULL,
    stock INT NOT NULL
);
```

Tabla «venta»

Contiene un identificador único (idVenta) para cada venta del sistema. Se registra la fecha y el total de la venta, junto con la referencia al usuario y cliente que la registró.

```sql
-- Tabla Ventas
CREATE TABLE Venta (
    idVenta INT AUTO_INCREMENT PRIMARY KEY,
    idUsuario INT NOT NULL,
    idCliente INT NOT NULL,
    fecha DATE NOT NULL,
    total DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (idUsuario) REFERENCES Usuario(idUsuario),
    FOREIGN KEY (idCliente) REFERENCES Cliente(idCliente)
);
```

Tabla «Detalle_Venta»

Relaciona los productos con cada venta, indicando la cantidad, precio unitario y subtotal.

```sql
-- Tabla Detalle_Venta
CREATE TABLE Detalle_Venta (
    idDetalle_Venta INT AUTO_INCREMENT PRIMARY KEY,
    idVenta INT NOT NULL,
    idProducto INT NOT NULL,
    cantidad INT NOT NULL,
    precioUnitario DECIMAL(10,2) NOT NULL,
    subtotal DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (idVenta) REFERENCES Venta(idVenta),
    FOREIGN KEY (idProducto) REFERENCES Producto(idProducto)
);
```

Tabla «Reporte»

Almacena los reportes generados en el sistema, indicando el tipo (por ejemplo, ventas o desempeño) y la fecha correspondiente.

```sql
CREATE TABLE Reporte (
    idReporte INT AUTO_INCREMENT PRIMARY KEY,
    idVenta INT,
    fechaGeneracion DATE NOT NULL,
    tipoReporte VARCHAR(50),
    FOREIGN KEY (idVenta) REFERENCES Venta(idVenta) ON DELETE CASCADE
);
```

Inserción, consulta y borrado de registros

Se insertan datos de prueba en las tablas para verificar que el modelo de datos funcione correctamente. Luego, se mostrará con una consulta cómo obtener resultados y finalmente, se procederá a limpiar las tablas.

Inserción de datos de prueba

Insertar datos de prueba de **«usuario»**

```sql
-- Insertamos un usuario administrador y un vendedor
INSERT INTO Usuario (nombre, apellido, usuario, contraseña, rol)
VALUES ('Ana', 'Gómez', 'agomez', '1234', 'Administrador'),
       ('Luis', 'Martínez', 'lmartinez', 'abcd', 'Vendedor');
```

Insertar datos de prueba **«cliente»**

```sql
-- Insertamos clientes
INSERT INTO Cliente (nombre, apellido, email, telefono)
VALUES ('Carlos', 'López', 'carlos.lopez@mail.com', '351111111'),
       ('María', 'Pérez', 'maria.perez@mail.com', '351222222');
```

Insertar datos de prueba **«producto»**

```sql
-- Insertamos productos
INSERT INTO Producto (nombre, categoria, precio, stock)
VALUES ('Laptop HP', 'Tecnología', 450000, 10),
       ('Mouse Logitech', 'Accesorios', 15000, 50);
```

Insertar datos de prueba **«venta»** a Carlos

```sql
-- Insertamos una venta (Luis vende a Carlos)
INSERT INTO Venta (idUsuario, idCliente, fecha, total)
VALUES (2, 1, '2025-09-25', 465000);
```

Insertar datos de prueba **«venta»** a María

```sql
 -- Otra venta: Luis vende a María Pérez
INSERT INTO Venta (idUsuario, idCliente, fecha, total)
VALUES (2, 2, '2025-09-26', 15000);
```

Insertar datos de prueba **«detalle_venta»**

```sql
-- Insertamos detalle de la venta
INSERT INTO Detalle_Venta (idVenta, idProducto, cantidad, precioUnitario, subtotal)
VALUES (1, 1, 1, 450000, 450000),
       (1, 2, 1, 15000, 15000);
```

Insertar datos de prueba **«reporte»**

```sql
-- Insertamos un reporte asociado a la venta con idVenta = 1
INSERT INTO Reporte (idVenta, fechaGeneracion, tipoReporte)
VALUES (1, '2025-09-25', 'Reporte Detallado');
```

Presentación de las consultas SQL

Consultar datos de prueba

Consulta – Listado de usuarios registrados

```
88  SELECT * FROM Usuario
```

Salida de la consulta

Output:

idUsuario	nombre	apellido	usuario	contraseña	rol
1	Ana	Gómez	agomez	1234	Administrador
2	Luis	Martínez	lmartinez	abcd	Vendedor

Consulta – Listado de clientes registrados

```
88  SELECT * FROM Cliente
```

Salida de la consulta

Output:

idCliente	nombre	apellido	email	telefono
1	Carlos	López	carlos.lopez@mail.com	351111111
2	María	Pérez	maria.perez@mail.com	351222222

Consulta – Total de ventas por cliente

```
95  SELECT c.nombre, c.apellido, COALESCE(SUM(v.total),0) AS Total_Ventas
96  FROM Cliente c
97  LEFT JOIN Venta v ON c.idCliente = v.idCliente
98  GROUP BY c.idCliente;
```

Salida de la consulta

Output:

nombre	apellido	Total_Ventas
Carlos	López	465000.00
María	Pérez	15000.00

Consulta – Listado de productos registrados

```
95  SELECT * FROM Producto
```

Salida de la consulta

Output:

idProducto	nombre	categoria	precio	stock
1	Laptop HP	Tecnología	450000.00	10
2	Mouse Logitech	Accesorios	15000.00	50

Consulta – Reporte generado

```
95  SELECT * FROM Reporte
```

Salida de la consulta

Output:

idReporte	idVenta	fechaGeneracion	tipoReporte
1	1	2025-09-25	Reporte Detallado

Datos de prueba borrados y verificación

```
76  -- Primero borro detalles de esas ventas
77  DELETE FROM Detalle_Venta
78  WHERE idVenta IN (
79      SELECT idVenta FROM Venta WHERE idUsuario = 2
80  );
81
82  -- Luego borro las ventas
83  DELETE FROM Venta
84  WHERE idUsuario = 2;
85
86  -- Finalmente borro el usuario
87  DELETE FROM Usuario
88  WHERE idUsuario = 2;
89
90  SELECT * FROM Usuario;
```

```
Output:

+-----------+--------+----------+---------+-------------+---------------+
| idUsuario | nombre | apellido | usuario | contraseña  | rol           |
+-----------+--------+----------+---------+-------------+---------------+
|         1 | Ana    | Gómez    | agomez  | 1234        | Administrador |
+-----------+--------+----------+---------+-------------+---------------+
```

Definiciones de comunicación

El sistema sigue un modelo cliente-servidor:

- **Cliente:** Aplicación Java.
- **Servidor:** MySQL para persistencia de datos.
- **Canal de comunicación:** protocolo TCP/IP.
- **Repositorio externo:** GitHub para control de versiones.

Este esquema permite escalabilidad y garantiza que múltiples usuarios accedan a información consistente, reduciendo redundancias y mejorando la eficiencia de la gestión.

Como parte de la implementación, se utilizó JDBC para la conexión a MySQL:

```
95  Connection conn = DriverManager.getConnection(
96  "jdbc:mysql://localhost:3306/pyme_ventas", "user", "password");
97  System.out.println("Conexión exitosa a MySQL");
```

Esto asegura la comunicación cliente-servidor y la persistencia de datos.

Conclusión:

La redacción de este trabajo permitió ofrecer continuidad a lo trabajado en el TP1, incluyendo las correcciones sugeridas y ampliando el ámbito hacia los procesos de diseño, implementación, pruebas y definición de la base de datos. A lo largo del texto se fue desarrollando una propuesta segura y consistente que refleja la evolución del sistema de gestión para PyMEs.

En esta segunda entrega se pudo dar forma a la idea original en una solución un poco más concreta, con el diseño de casos de uso extendido, diagramas que representan el comportamiento de actores y entidades, y definición de una base de datos relacional que respalda procesos de negocio principales. También se incluyeron consultas,

inserciones y eliminaciones en SQL, para poder verificar el comportamiento práctico del modelo propuesto.

El enfoque incremental e iterativo no solo mejoró la calidad de las tareas, sino que también hizo que mantener una continuidad de línea en las entregas fuera más sencillo. Esto garantiza que el proyecto se desplace en orden y de manera consistente, reforzando una visión clara de cómo un sistema de información podría mejorar la gestión interna de las PyMEs.

En resumen, el TP2 representa un gran paso hacia la implementación de un sistema operativo, demostrando la importancia de integrar el análisis teórico con la aplicación práctica. El proceso documental y de desarrollo resultó enriquecedor, ya que no solo permitió aplicar los conceptos aprendidos en materias anteriores, sino también comprender cómo se construye un proyecto de software de forma progresiva, con mejoras y ajustes continuos.

Sección del TP N°3

Explicación del Desarrollo en Java del TP N°3

El presente desarrollo constituye la implementación práctica del sistema de gestión para PYMES de ventas, continuando con el análisis y diseño elaborados en las etapas anteriores (TP1 y TP2). En esta instancia, el enfoque se centra en trasladar los requerimientos funcionales definidos a una aplicación desarrollada íntegramente en el lenguaje **Java**, aplicando los principios de la **Programación Orientada a Objetos (POO)**.

El proyecto fue implementado en el entorno de desarrollo **Eclipse IDE**, respetando los fundamentos de modularidad, legibilidad y mantenibilidad del código. Se adoptó una estructura basada en clases y paquetes que separan las responsabilidades lógicas del sistema:

- **Modelos** que representan las entidades del dominio (Usuario, Cliente, Producto, Venta, Reporte).
- **Gestores** encargados de la lógica operativa y la manipulación de los datos de cada entidad.
- **Clase Main** que actúa como interfaz de usuario mediante un menú de selección interactivo.

Aplicación de los principios de POO

En el desarrollo se aplicaron los cuatro pilares fundamentales de la Programación Orientada a Objetos:

1. **Encapsulamiento:**
   Cada clase define sus atributos como privados, garantizando la protección de los datos y el acceso controlado mediante métodos *getters* y *setters*. De esta manera, se evita la manipulación directa de los atributos desde fuera de la clase, preservando la integridad de los objetos.
2. **Herencia:**
   Las clases comparten características comunes a través de jerarquías que permiten reutilizar código. Este principio facilita la extensión del sistema ante futuras funcionalidades sin comprometer la estructura existente.
3. **Polimorfismo:**
   Se aplicó para permitir la reutilización de métodos con comportamientos distintos según el tipo de objeto. Gracias a este principio, las operaciones sobre

diferentes entidades pueden gestionarse de forma genérica, aumentando la flexibilidad del código.

4. **Abstracción:**
   Cada clase fue diseñada representando únicamente los atributos y comportamientos esenciales de cada entidad del negocio, evitando complejidades innecesarias y mejorando la comprensión general del sistema.

Estructuras de control y manejo de excepciones

El menú principal se implementó utilizando **estructuras condicionales y repetitivas** (como switch, if-else y bucles do-while), que permiten navegar entre las distintas opciones del sistema: registrar, modificar o eliminar usuarios y productos, registrar clientes, registrar ventas y generar reportes.
Asimismo, se incorporó el **manejo de excepciones** mediante bloques try-catch para prevenir fallos durante la ejecución (por ejemplo, entradas de datos incorrectas o errores de tipo), mejorando la estabilidad y robustez del programa.

Constructores y creación de objetos

Cada entidad del sistema cuenta con **constructores parametrizados y por defecto**, permitiendo la creación dinámica de objetos con sus valores iniciales. Esta práctica asegura la correcta inicialización de los datos y facilita la reutilización de las clases en futuras expansiones del proyecto.

Algoritmos de búsqueda y ordenamiento

Si bien el enfoque principal estuvo orientado al manejo de datos básicos, se implementaron **métodos de búsqueda** para localizar registros específicos (por ejemplo, buscar productos por nombre o usuarios por ID). Esto permite validar la existencia de un registro antes de realizar operaciones como modificaciones o eliminaciones.

Menú de interacción

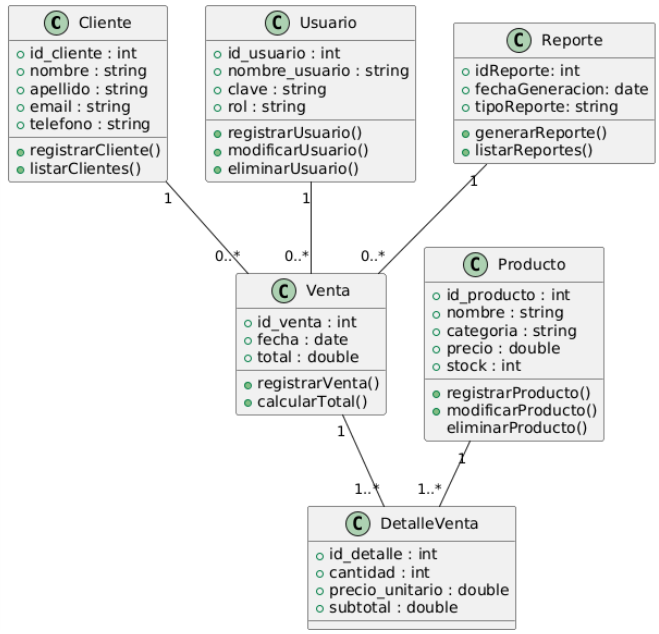
El sistema cuenta con un **menú de selección textual**, donde el usuario puede elegir distintas operaciones a realizar. Este menú se ejecuta en un bucle hasta que el usuario decida salir, permitiendo la ejecución continua del programa sin necesidad de reiniciarlo.

Características de las clases más importantes

A continuación, se presentan las clases principales del sistema y sus características más relevantes.

Cada clase fue diseñada aplicando encapsulamiento, herencia y abstracción cumpliendo los principios de la **Programación Orientada a Objetos (POO)** en Java.

Figura 2. Diagrama de clase definido en la etapa de diseño



- **Clase Usuario**

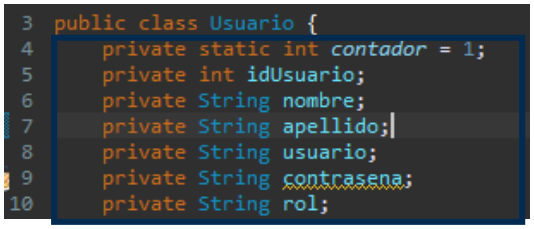
Descripción: Representa a los usuarios del sistema (Administrador/Vendedor).

Atributos: idUsuario, nombre, apellido, usuario, contraseña, rol.

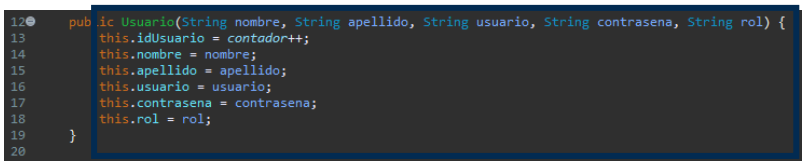
Métodos principales:

- registrarUsuario(): Crea un nuevo usuario.
- modificarUsuario(): Actualiza datos existentes.
- eliminarUsuario(): Elimina usuario y sus registros.

Figura 3. Atributos y Constructor «Usuario»



Atributos



Constructor

- **Clase Cliente**

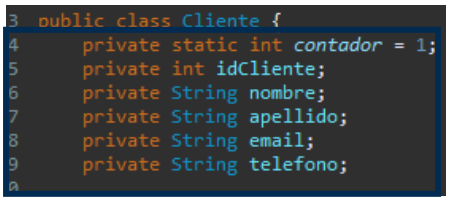
Descripción: Modela los clientes con sus datos personales y de contacto.

Atributos: idCliente, nombre, apellido, email, telefono.

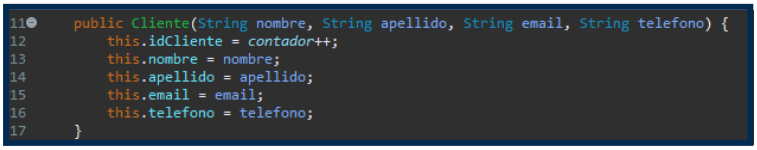
Métodos principales:

- registrarCliente(): Guarda un nuevo cliente.
- listarClientes(): Devuelve la lista de clientes registrados.

Figura 4. Atributos y Constructor «Cliente»



Atributos



Constructor

- **Clase Producto**

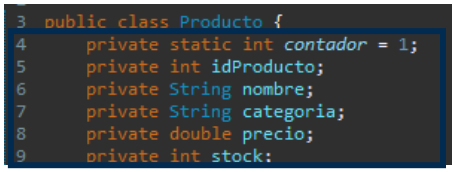
Descripción: Define los productos comercializados.

Atributos: idProducto, nombre, categoria, precio, stock.

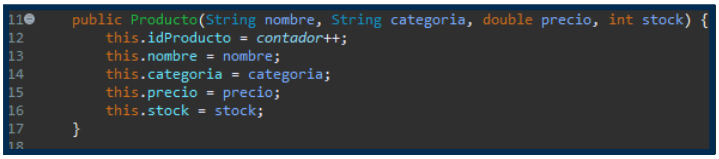
Métodos principales:

- registrarProducto(): Permite incorporar un nuevo producto.
- modificarProducto(): Actualiza información como precio o stock.
- eliminarProducto(): Elimina productos inactivos.

Figura 5. Atributos y Constructor «Producto»



Atributos



Constructor

- **Clase Venta**

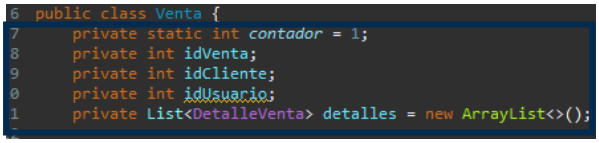
Descripción: Registra las ventas efectuadas por los usuarios a los clientes.

Atributos: idVenta, cliente, usuario, total, fecha.

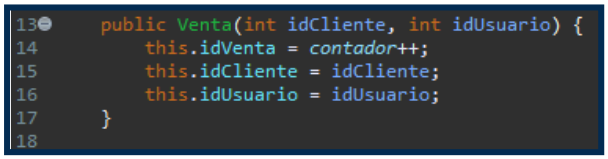
Métodos principales:

- registrarVenta(): Guarda una nueva venta y actualiza el stock.
- calcularTotal(): Calcula el monto final.

Figura 6. Atributos y Constructor «Venta»



Atributos



Constructor

- **Clase Reporte**

Descripción: Genera y lista reportes sobre las ventas.

Atributos: idReporte, fechaGeneracion, tipoReporte, venta.

Métodos principales:

- generarReporte(): Crea un nuevo reporte con base en las ventas.
- listarReportes(): Muestra todos los reportes generados.

Figura 7. Atributos y Constructor «Reporte»

```
3  public class Reporte {
4      private static int contador = 1;
5      private int idReporte;
6      private String tipo;
7      private double total;
```

Atributos

```
9      public Reporte(String tipo, double total) {
0          this.idReporte = contador++;
1          this.tipo = tipo;
2          this.total = total;
3      }
```

Constructor

- **Clase Main**

Descripción: Controla la ejecución del sistema mediante un menú interactivo.

Características destacadas:

- Implementa bucles do-while y switch.
- Maneja excepciones try-catch.
- Conecta los gestores y modelos.

Figura 8. Clase «Main»

```
1  package app;
2
3  import java.util.Scanner;
4
5  import modelo.*;
6  import servicio.*;
7
8  public class Main {
9      public static void main(String[] args) {
0          Scanner sc = new Scanner(System.in);
1
2          // Inicialización de gestores
3          GestorUsuarios gestorUsuarios = new GestorUsuarios();
4          GestorClientes gestorClientes = new GestorClientes();
5          GestorProductos gestorProductos = new GestorProductos();
6          GestorVentas gestorVentas = new GestorVentas();
7          GestorReportes gestorReportes = new GestorReportes();
8
9          // Datos iniciales
0          gestorUsuarios.registrarUsuario(new Usuario("Ana", "Gómez", "agomez", "1234", "Administrador"));
1          gestorUsuarios.registrarUsuario(new Usuario("Carlos", "López", "clopez", "5678", "Vendedor"));
2          gestorProductos.registrarProducto(new Producto("Monitor", "Electrónica", 80000, 10));
3          gestorProductos.registrarProducto(new Producto("Teclado", "Periféricos", 15000, 25));
4          gestorClientes.registrarCliente(new Cliente("Pedro", "Martínez", "pedro@mail.com", "11335577"));
5          gestorClientes.registrarCliente(new Cliente("Lucía", "Fernández", "lucia@mail.com", "11774455"));
6
7          int opcion;
```

```
        int opcion;
        do {
            System.out.println("\n========= MENÚ PRINCIPAL =========");
            System.out.println("1. Gestionar Usuarios");
            System.out.println("2. Gestionar Clientes");
            System.out.println("3. Gestionar Productos");
            System.out.println("4. Registrar Venta");
            System.out.println("5. Listar Ventas");
            System.out.println("6. Generar Reporte de Ventas");
            System.out.println("0. Salir");
            System.out.print("Seleccione una opción: ");
            opcion = sc.nextInt();
            sc.nextLine(); // limpiar buffer

            switch (opcion) {
                case 1 -> {
                    System.out.println("\n--- GESTIÓN DE USUARIOS ---");
                    System.out.println("1. Registrar usuario");
                    System.out.println("2. Modificar usuario");
                    System.out.println("3. Eliminar usuario");
                    System.out.print("Seleccione una opción: ");
                    int sub = sc.nextInt(); sc.nextLine();

                    switch (sub) {
                        case 1 -> {
                            System.out.print("Nombre: ");
                            String nom = sc.nextLine();
```

```
                            System.out.print("Apellido: ");
                            String ape = sc.nextLine();
                            System.out.print("Usuario: ");
                            String usr = sc.nextLine();
                            System.out.print("Contraseña: ");
                            String pass = sc.nextLine();
                            System.out.print("Rol: ");
                            String rol = sc.nextLine();
                            gestorUsuarios.registrarUsuario(new Usuario(nom, ape, usr, pass, rol));
                        }
                        case 2 -> {
                            System.out.print("ID de usuario a modificar: ");
                            int id = sc.nextInt(); sc.nextLine();
                            System.out.print("Nuevo nombre: ");
                            String nuevoNombre = sc.nextLine();
                            System.out.print("Nuevo apellido: ");
                            String nuevoApellido = sc.nextLine();
                            gestorUsuarios.modificarUsuario(id, nuevoNombre, nuevoApellido);
                        }
                        case 3 -> {
                            System.out.print("ID de usuario a eliminar: ");
                            int id = sc.nextInt();
                            gestorUsuarios.eliminarUsuario(id);
                        }
                        default -> System.out.println("⚠ Opción inválida.");
                    }
```

```
                case 4 -> {
                    System.out.println("\n--- REGISTRAR VENTA ---");
                    System.out.print("ID Cliente: ");
                    int idCliente = sc.nextInt();
                    System.out.print("ID Usuario (Vendedor): ");
                    int idUsuario = sc.nextInt();
                    Venta venta = new Venta(idCliente, idUsuario);

                    String continuar;
                    do {
                        System.out.print("ID Producto: ");
                        int idProducto = sc.nextInt();
                        System.out.print("Cantidad: ");
                        int cant = sc.nextInt();
                        gestorVentas.agregarDetalleVenta(venta, idProducto, cant, gestorProductos);
                        System.out.print("¿Agregar otro producto? (s/n): ");
                        continuar = sc.next();
                    } while (continuar.equalsIgnoreCase("s"));

                    gestorVentas.registrarVenta(venta);
                }

                case 5 -> {
                    System.out.println("\n--- LISTADO DE VENTAS ---");
                    for (Venta v : gestorVentas.getVentas()) {
                        System.out.println(v);
                        v.getDetalles().forEach(d -> System.out.println("   " + d));
```

```
                    }
                }

                case 6 -> gestorReportes.generarReporte(gestorVentas);

                case 0 -> System.out.println("👋 Saliendo del sistema. ¡Gracias!");
                default -> System.out.println("⚠ Opción inválida.");
            }
        } while (opcion != 0);

        sc.close();
    }
}
```

Ejemplos de Ejecución del Desarrollo en Java

Figura 9. Registrar «Cliente»

Agrego un nuevo cliente al sistema (Juan Pérez) con sus respectivos datos personales.

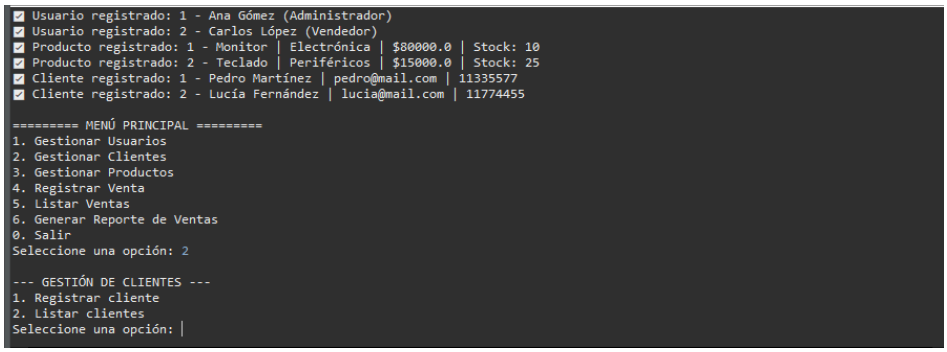


Figura 10. Registrar «Producto»

Registramos un nuevo producto (Laptop Lenovo) en el sistema.

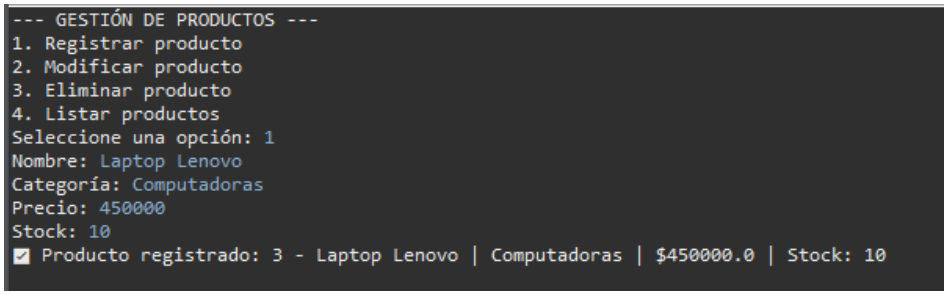


Figura 11. Registrar «Venta»

El Usuario (Vendedor) Carlos López vende el nuevo producto registrado (x5 Laptop Lenovo) a Juan Pérez (Cliente).

```
========= MENÚ PRINCIPAL =========
1. Gestionar Usuarios
2. Gestionar Clientes
3. Gestionar Productos
4. Registrar Venta
5. Listar Ventas
6. Generar Reporte de Ventas
0. Salir
Seleccione una opción: 4

--- REGISTRAR VENTA ---
ID Cliente: 3
ID Usuario (Vendedor): 2
ID Producto: 3
Cantidad: 5
📝 Agregado: Laptop Lenovo x5 = $2250000.0
¿Agregar otro producto? (s/n): n
✅ Venta registrada: Venta #1 - Cliente ID: 3 | Total: $2250000.0
```

Figura 12. Generar «Reporte de Ventas»

Se genera el reporte de ventas de la última venta registrada.

```
========= MENÚ PRINCIPAL =========
1. Gestionar Usuarios
2. Gestionar Clientes
3. Gestionar Productos
4. Registrar Venta
5. Listar Ventas
6. Generar Reporte de Ventas
0. Salir
Seleccione una opción: 6
📊 Reporte #1 - Tipo: Ventas totales | Total: $2250000.0
```

Presentación del Desarrollo en Java del TP N°3

A continuación, se adjuntan las clases Java utilizadas para crear el proyecto en una IDE y comprobar la salida con la ejecución del programa desarrollado.

GitHub: https://github.com/agustingreco2020/TP3_PymesVentas

Sección del TP N°4

Explicación del Desarrollo en Java y Conexión a la Base de Datos

Introducción

El presente trabajo práctico forma parte del proyecto integrador *PYME Ventas*, un sistema desarrollado en Java que permite la gestión completa de una pequeña empresa comercial.
Entre sus principales funcionalidades se destacan:

- Registro y administración de Usuarios, Clientes y Productos.
- Registro de Ventas con detalle de productos.
- Generación de Reportes de ventas totales.
- Conexión con una base de datos MySQL remota, permitiendo la persistencia de datos.

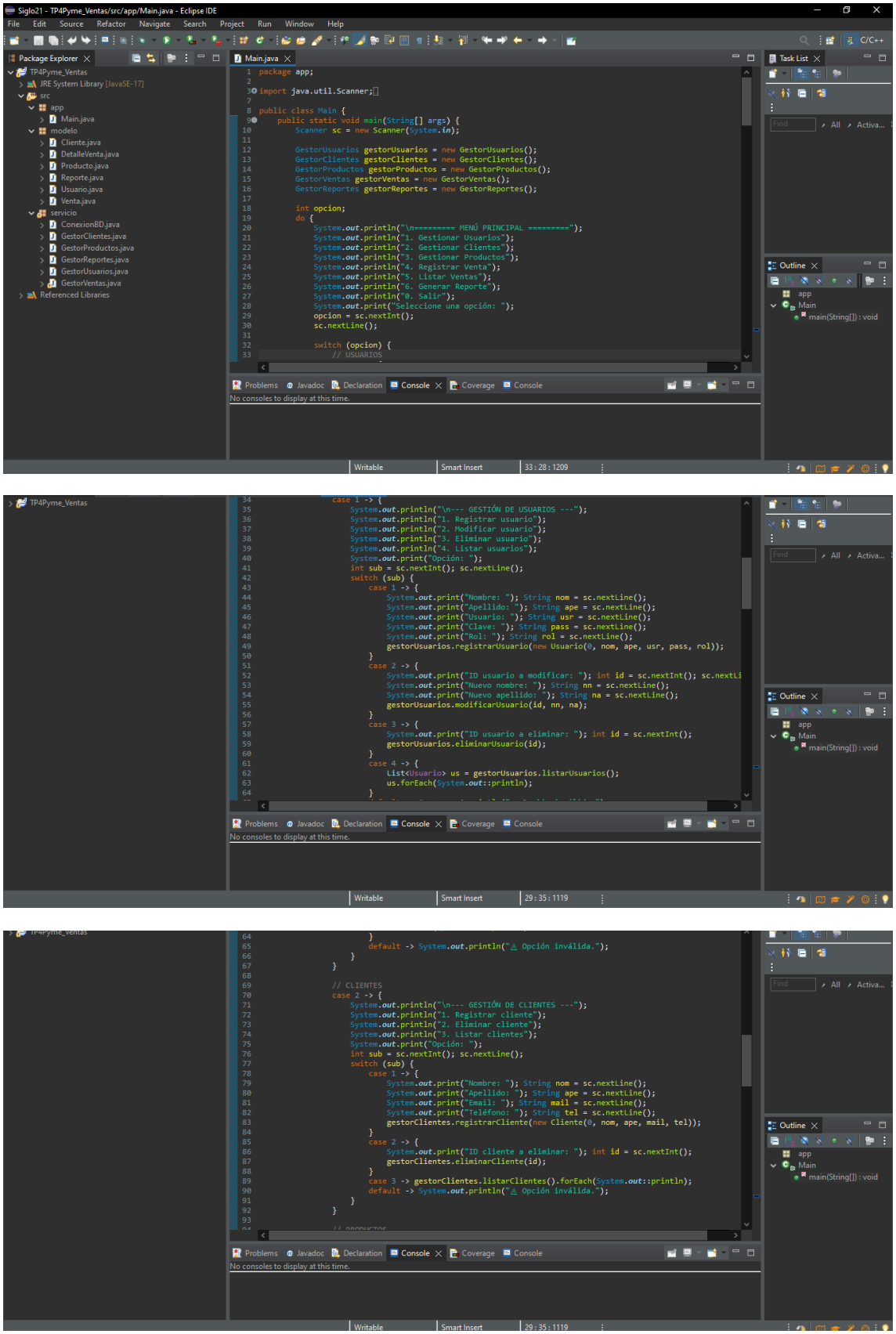
Este proyecto incorpora los principios de la Programación Orientada a Objetos (POO), junto con la implementación de conexión a base de datos, manejo de excepciones y colecciones como ArrayList.

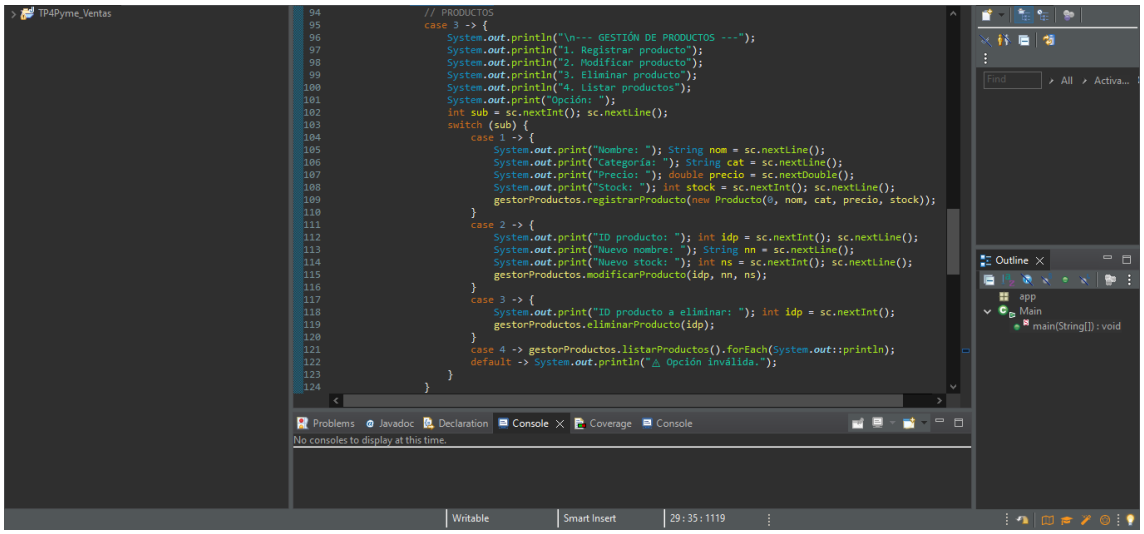
2. Estructura del proyecto

El sistema se encuentra organizado en paquetes dentro del entorno de desarrollo **Eclipse IDE**, siguiendo una estructura modular:

- **modelo:** contiene las clases principales (Usuario, Cliente, Producto, Venta, DetalleVenta, Reporte).
- **servicio:** incluye los gestores responsables de cada entidad (GestorUsuarios, GestorClientes, etc.). Además, se encuentra la clase ConexionBD, que gestiona la conexión con MySQL.
- **app:** contiene la clase principal Main.

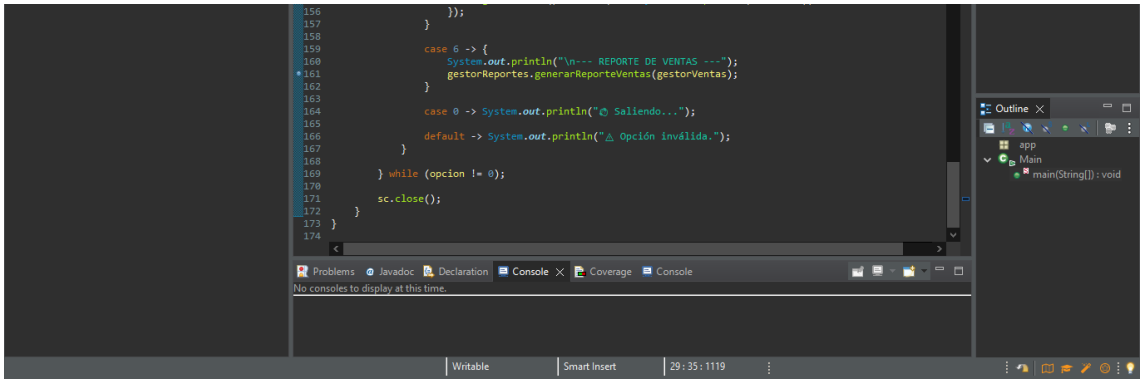
Figura 2. Proyecto realizado en Eclipse





```
// PRODUCTOS
case 3 -> {
    System.out.println("\n--- GESTIÓN DE PRODUCTOS ---");
    System.out.println("1. Registrar producto");
    System.out.println("2. Modificar producto");
    System.out.println("3. Eliminar producto");
    System.out.println("4. Listar productos");
    System.out.print("Opción: ");
    int sub = sc.nextInt(); sc.nextLine();
    switch (sub) {
        case 1 -> {
            System.out.print("Nombre: "); String nom = sc.nextLine();
            System.out.print("Categoría: "); String cat = sc.nextLine();
            System.out.print("Precio: "); double precio = sc.nextDouble();
            System.out.print("Stock: "); int stock = sc.nextInt(); sc.nextLine();
            gestorProductos.registrarProducto(new Producto(0, nom, cat, precio, stock));
        }
        case 2 -> {
            System.out.print("ID producto: "); int idp = sc.nextInt(); sc.nextLine();
            System.out.print("Nuevo nombre: "); String nn = sc.nextLine();
            System.out.print("Nuevo stock: "); int ns = sc.nextInt(); sc.nextLine();
            gestorProductos.modificarProducto(idp, nn, ns);
        }
        case 3 -> {
            System.out.print("ID producto a eliminar: "); int idp = sc.nextInt();
            gestorProductos.eliminarProducto(idp);
        }
        case 4 -> gestorProductos.listarProductos().forEach(System.out::println);
        default -> System.out.println("⚠ Opción inválida.");
    }
}
```

```
// VENTAS
case 4 -> {
    System.out.println("\n--- REGISTRAR VENTA ---");
    System.out.print("ID Cliente: "); int idC = sc.nextInt();
    System.out.print("ID Usuario (vendedor): "); int idU = sc.nextInt();
    Venta venta = new Venta(idC, idU);

    String mas;
    do {
        System.out.print("ID Producto: "); int idP = sc.nextInt();
        System.out.print("Cantidad: "); int cant = sc.nextInt();
        Producto p = gestorProductos.buscarProducto(idP);
        if (p == null) System.out.println("⚠ Producto no encontrado.");
        else {
            DetalleVenta d = new DetalleVenta(p, cant);
            venta.agregarDetalle(d);
            System.out.println("✔ Agregado: " + d);
        }
        System.out.print("Agregar otro producto? (s/n): "); mas = sc.next();
    } while (mas.equalsIgnoreCase("s"));

    gestorVentas.registrarVenta(venta);
}

case 5 -> {
    System.out.println("\n--- LISTADO DE VENTAS ---");
    List<Venta> ventas = gestorVentas.listarVentas();
    ventas.forEach(v -> {
        System.out.println(v);
        v.getDetalles().forEach(d -> System.out.println("   " + d));
    });
}

case 6 -> {
    System.out.println("\n--- REPORTE DE VENTAS ---");
    gestorReportes.generarReporteVentas(gestorVentas);
}

case 0 -> System.out.println("👋 Saliendo...");

default -> System.out.println("⚠ Opción inválida.");
}

} while (opcion != 0);

sc.close();
}
}
```



Conexión a la base de datos

Para la persistencia de datos, se implementó una clase específica llamada **ConexionBD**, la cual utiliza el **Driver JDBC de MySQL** para establecer la comunicación entre el sistema y la base de datos remota hospedada en *FreeSQLDatabase.com*.

Figura 3. Base de Datos FreeSQLDatabase

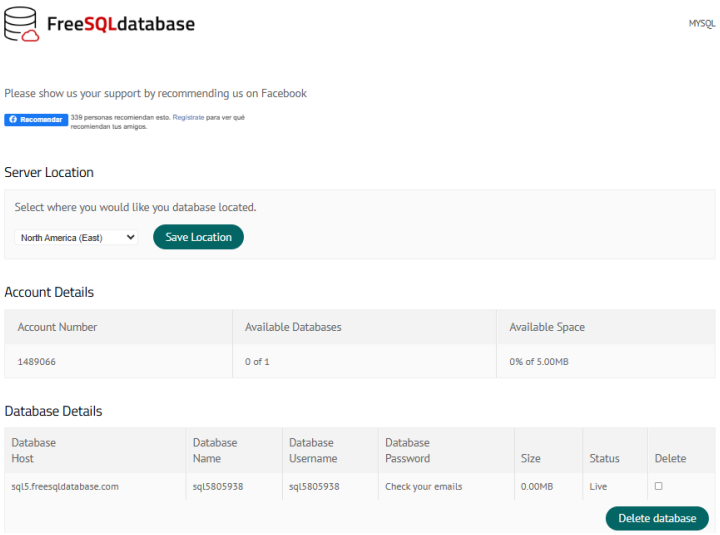
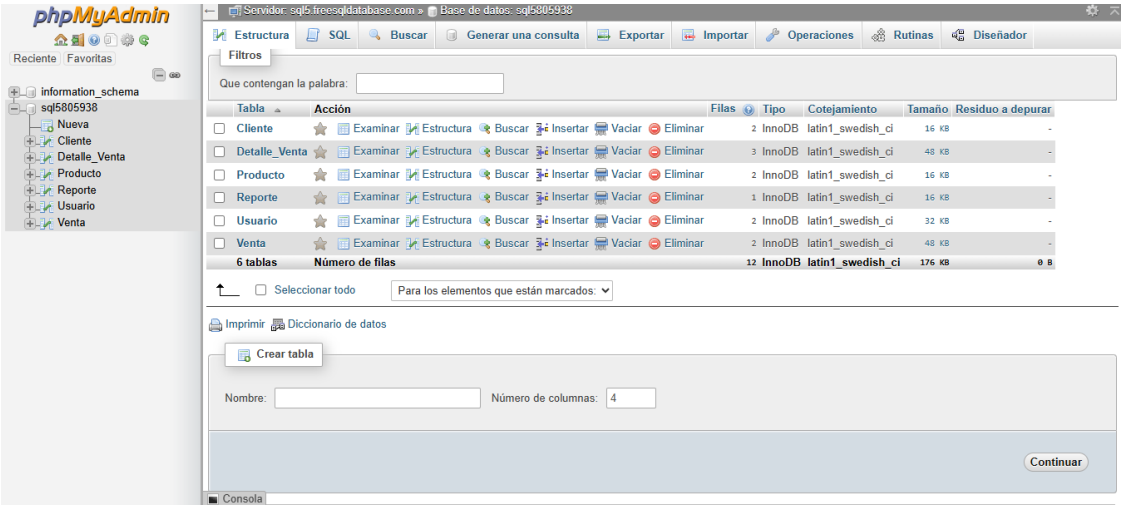


Figura 4. Clase ConexionBD

```java
package servicio;

import java.sql.Connection;

public class ConexionBD {
    private static final String URL = "jdbc:mysql://sql5.freesqldatabase.com:3306/sql5805938";
    private static final String USER = "sql5805938";
    private static final String PASSWORD = "ekuFlu9qAX";

    // devuelve conexión o lanza SQLException
    public static Connection obtenerConexion() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

Requisito	**Implementación**
Conexión a MySQL	Clase ConexionBD con JDBC
Excepciones	Bloques try-catch en consultas SQL
Patrón de diseño	Modelo-Servicio-Controlador (MSC)
Clases abstractas / interfaces	Implementadas en gestores y modelo
Estructuras dinámicas	Uso de ArrayList en gestores

Figura 5. Tablas MySQL importadas a phpMyAdmin



Funcionalidades principales

El sistema permite realizar las siguientes operaciones sobre la base de datos MySQL:

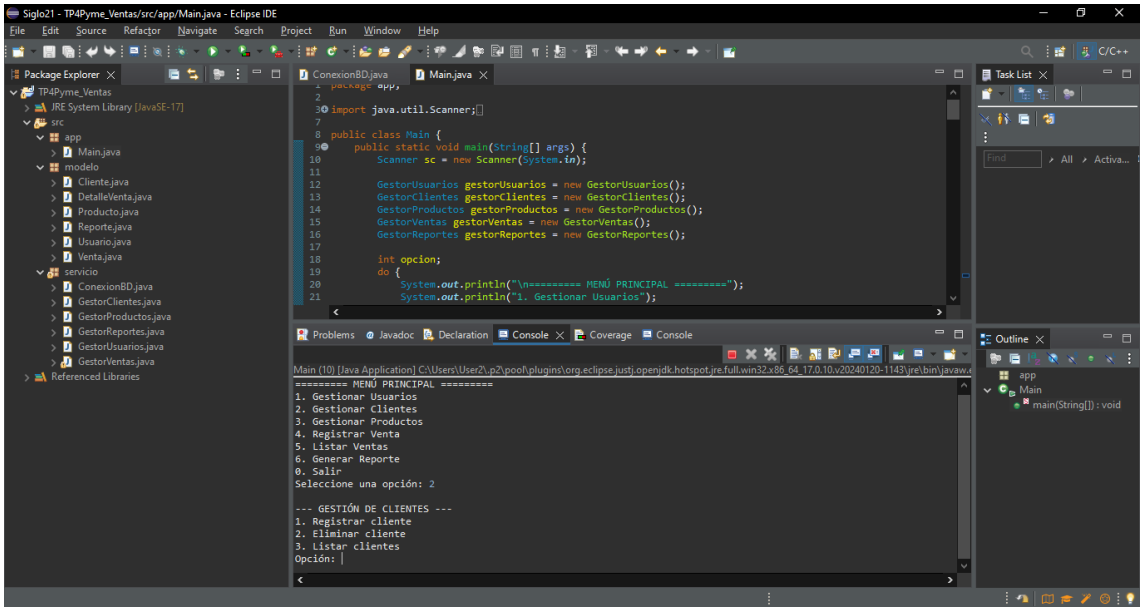
- **INSERT:** agregar nuevos registros (clientes, productos, usuarios, ventas).
- **UPDATE:** modificar registros existentes (por ejemplo, actualizar el stock o el nombre de un producto).
- **DELETE:** eliminar registros.
- **SELECT:** listar o consultar información almacenada.

Las pruebas fueron realizadas desde la clase Main utilizando el menú de consola, verificando la persistencia de los cambios en phpMyAdmin.

Ejemplos de Consulta de datos en mi base MySQL (Conectándose al sistema por medio de JDBC)

Figura 6. Registrar «Cliente»

Registramos un nuevo cliente al sistema y comprobamos su conexión en la base de datos.



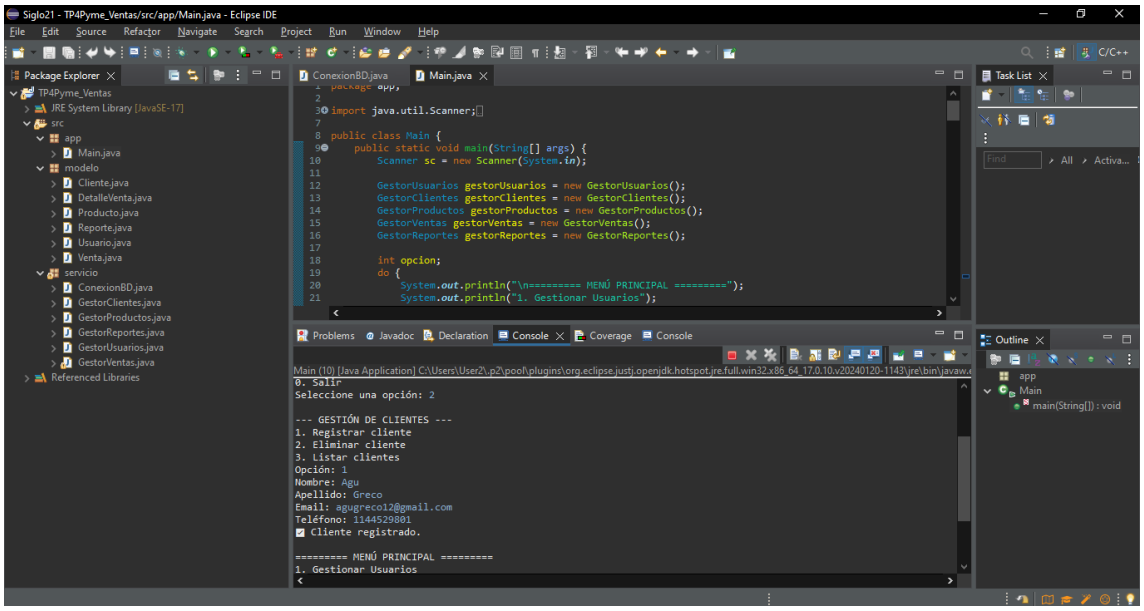


Figura 7. Conexión a la Base de Datos MySQL (Nuevo Cliente)

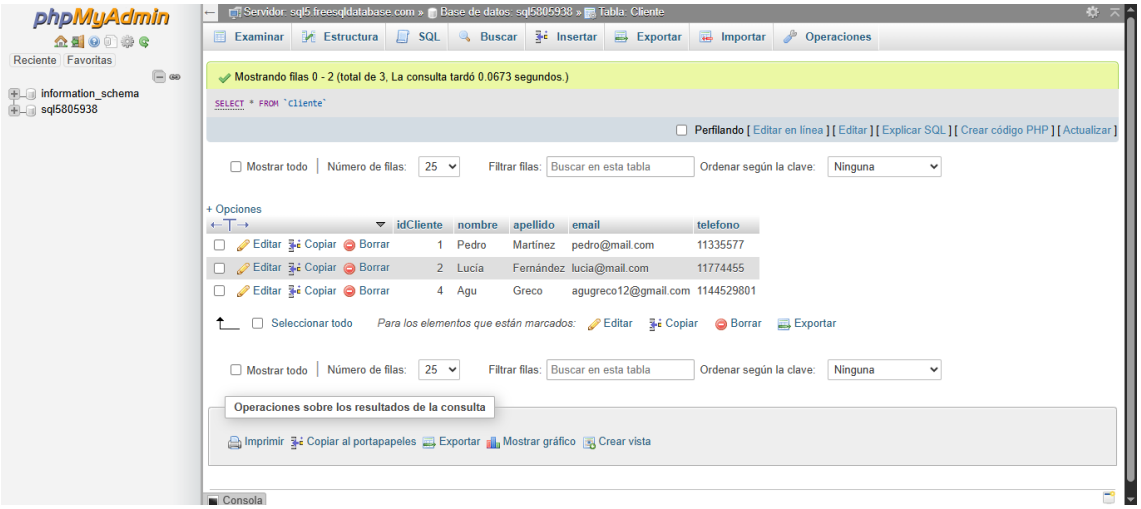


Figura 8. Eliminar «Cliente»

Realizamos una consulta para borrar el Nuevo Cliente.

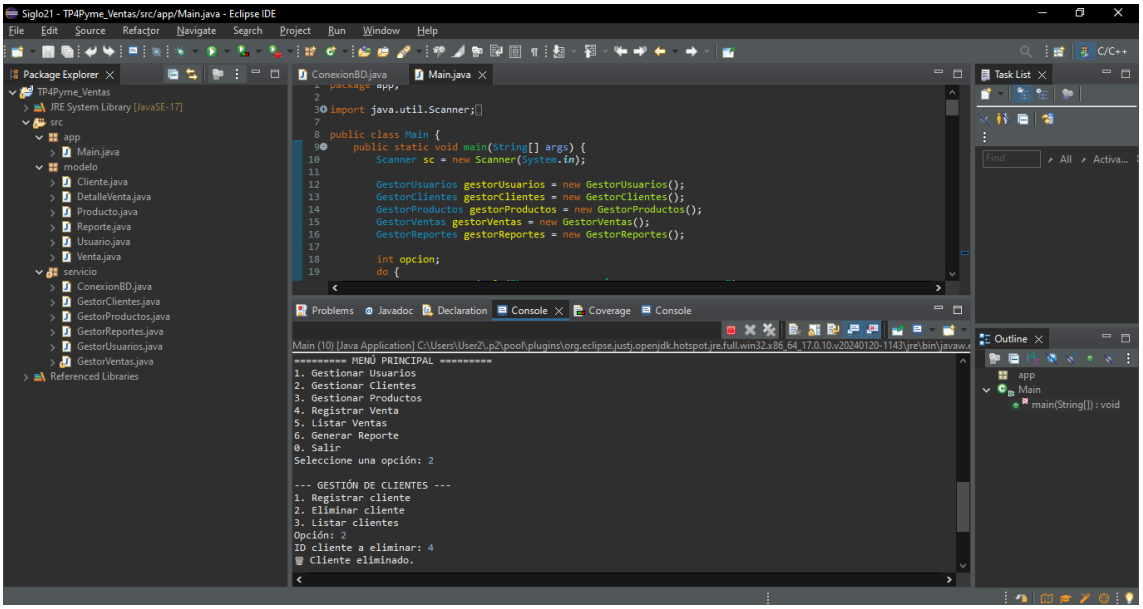


Figura 9. Conexión a la Base de Datos MySQL (Eliminar Cliente)

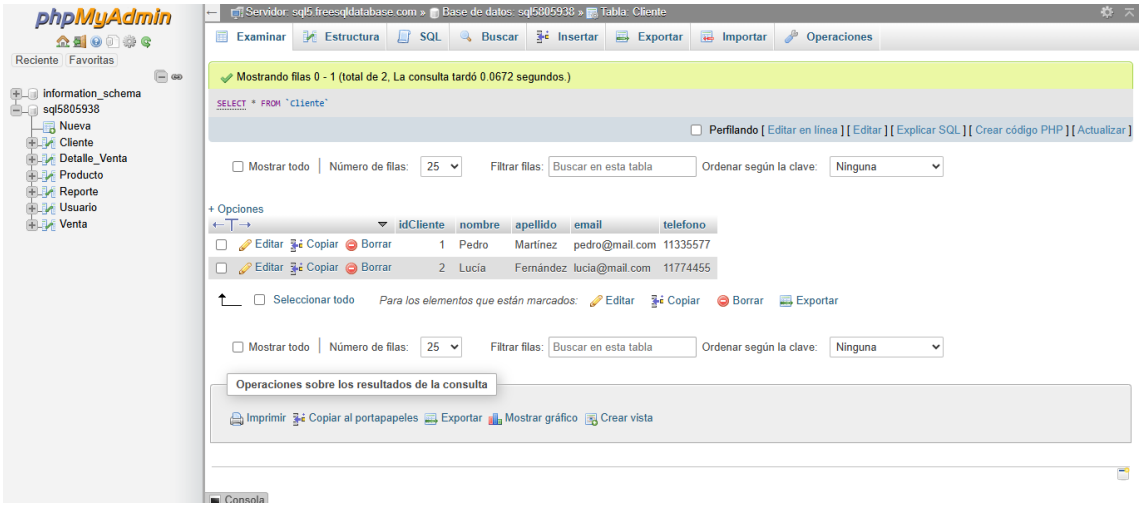


Figura 10. Manejo de Errores

Comprobamos el Manejo de Excepciones llamando al sistema con un ID inexistente de Usuario.

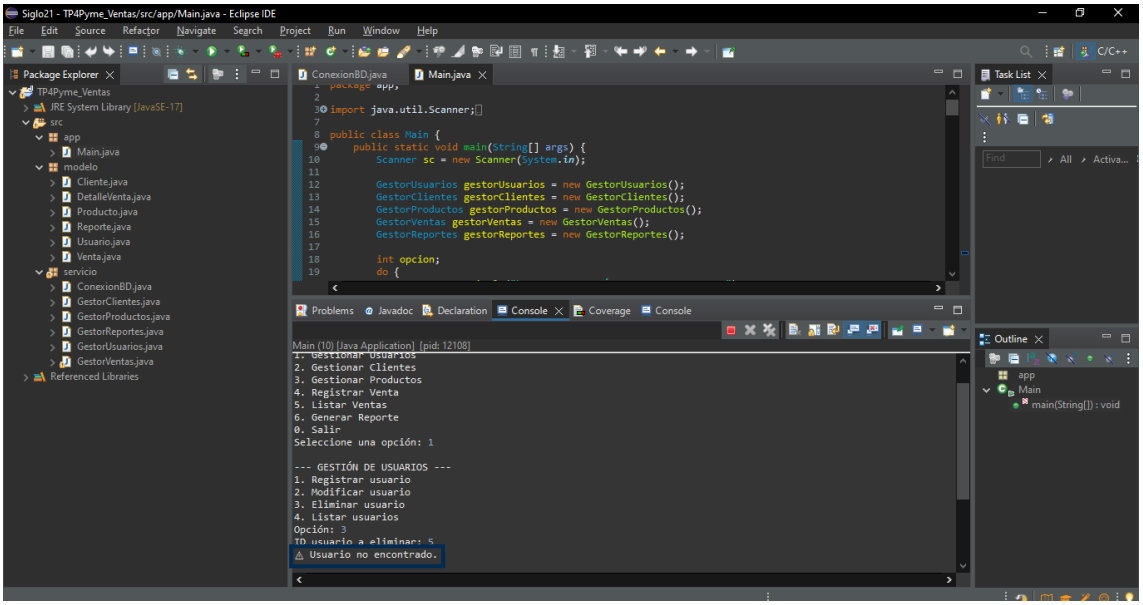


Figura 11. Registrar «Venta»

Registramos una nueva venta al sistema y comprobamos su conexión en la base de datos.

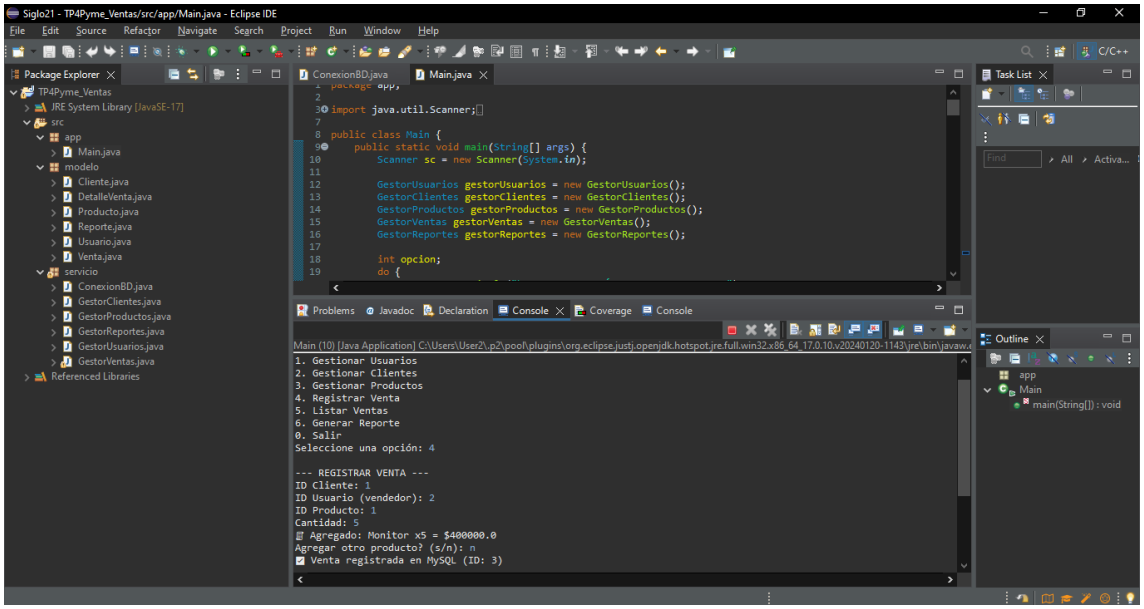
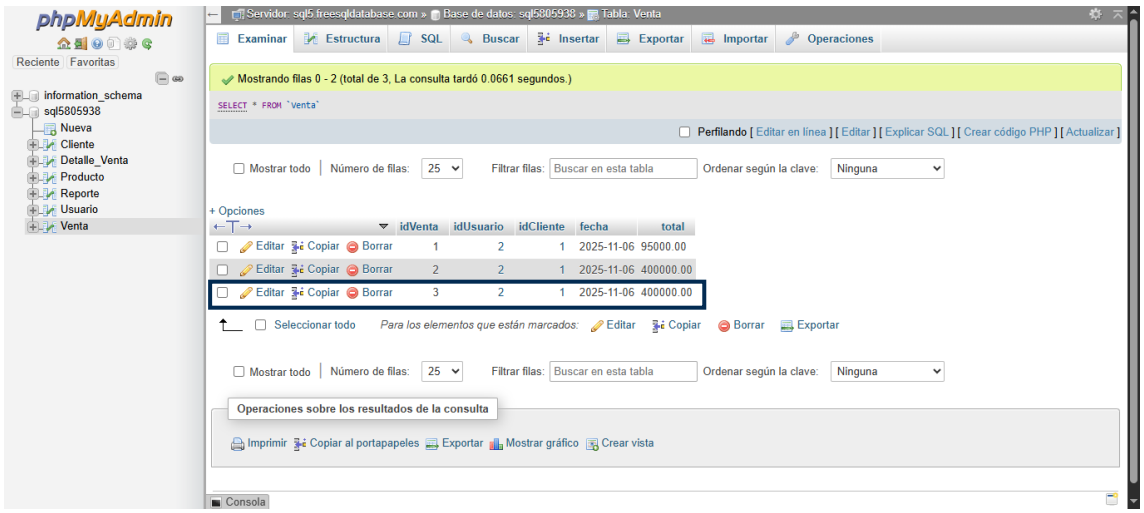
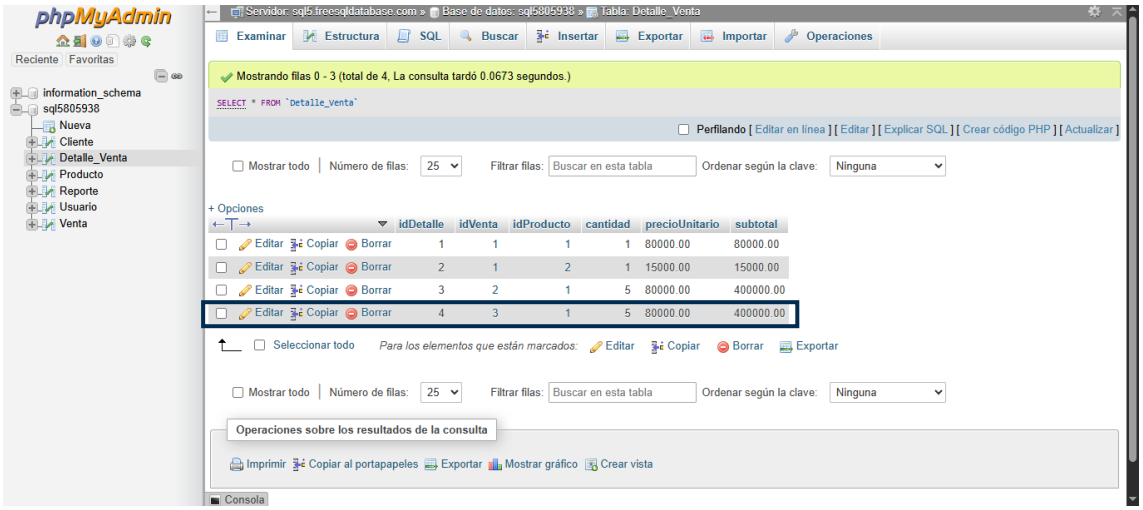


Figura 12. Conexión a la Base de Datos MySQL (Registrar Venta)




phpMyAdmin
Reciente Favoritas
information_schema
sql5805938
Nueva
Cliente
Detalle_Venta
Producto
Reporte
Usuario
Venta
Servidor: sql5.freesqldatabase.com » Base de datos: sql5805938 » Tabla: Detalle_Venta
Examinar Estructura SQL Buscar Insertar Exportar Importar Operaciones
Mostrando filas 0 - 3 (total de 4, La consulta tardó 0.0673 segundos.)
SELECT * FROM `Detalle_Venta`
Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]
Mostrar todo | Número de filas: 25 Filtrar filas: Buscar en esta tabla Ordenar según la clave: Ninguna
+ Opciones
idDetalle idVenta idProducto cantidad precioUnitario subtotal
Editar Copiar Borrar 1 1 1 1 80000.00 80000.00
Editar Copiar Borrar 2 1 2 1 15000.00 15000.00
Editar Copiar Borrar 3 2 1 5 80000.00 400000.00
Editar Copiar Borrar 4 3 1 5 80000.00 400000.00
Seleccionar todo Para los elementos que están marcados: Editar Copiar Borrar Exportar
Mostrar todo | Número de filas: 25 Filtrar filas: Buscar en esta tabla Ordenar según la clave: Ninguna
Operaciones sobre los resultados de la consulta
Imprimir Copiar al portapapeles Exportar Mostrar gráfico Crear vista
Consola

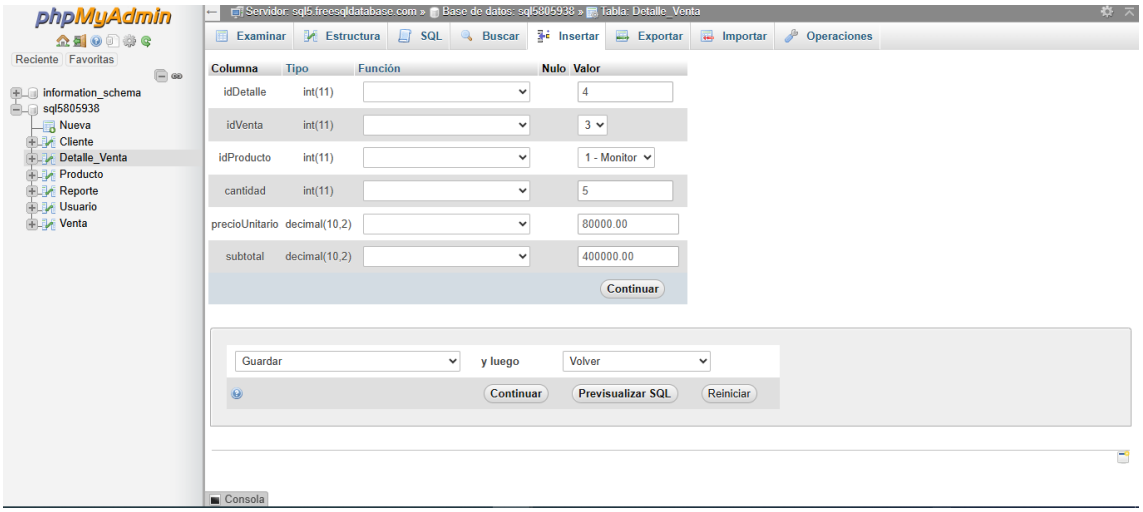
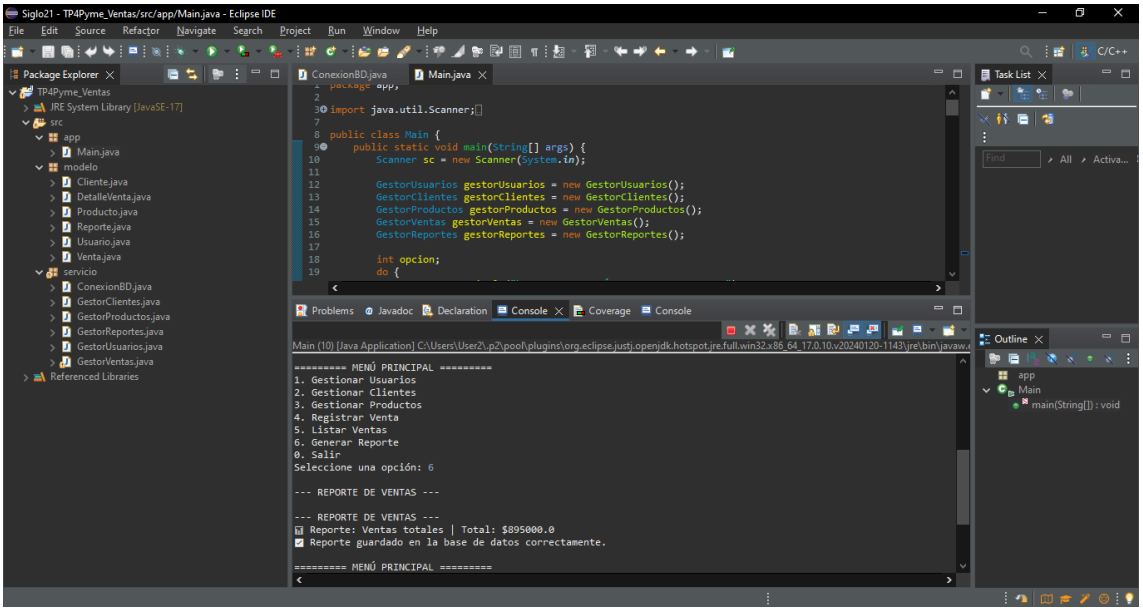
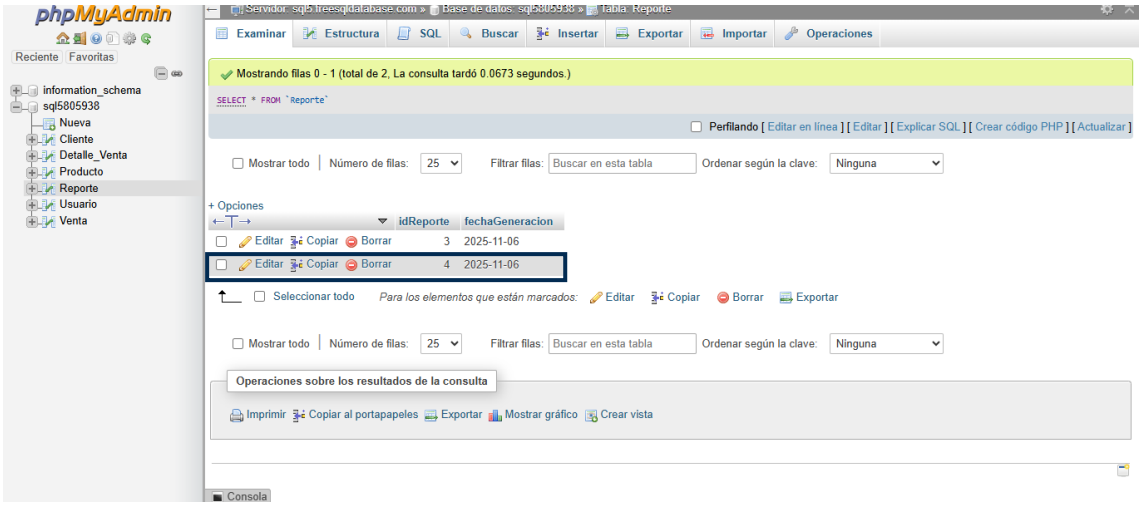

phpMyAdmin
Reciente Favoritas
information_schema
sql5805938
Nueva
Cliente
Detalle_Venta
Producto
Reporte
Usuario
Venta
Servidor: sql5.freesqldatabase.com » Base de datos: sql5805938 » Tabla: Detalle_Venta
Examinar Estructura SQL Buscar Insertar Exportar Importar Operaciones
Columna Tipo Función Nulo Valor
idDetalle int(11) 4
idVenta int(11) 3
idProducto int(11) 1 - Monitor
cantidad int(11) 5
precioUnitario decimal(10,2) 80000.00
subtotal decimal(10,2) 400000.00
Continuar
Guardar y luego Volver
Continuar Previsualizar SQL Reiniciar
Consola

Figura 13. Generar «Reporte de Ventas»

Desde el menú principal del sistema, es posible generar un **reporte de ventas totales**, el cual calcula el monto acumulado de todas las ventas registradas.





Justificación del diseño

El proyecto aplica **encapsulamiento, herencia, polimorfismo y abstracción**, manteniendo una arquitectura limpia y escalable.
Además:

- Se utiliza el patrón **Modelo–Servicio–Controlador (MSC)**, que facilita la separación de responsabilidades.
- Se implementan **excepciones controladas** para el manejo de errores en la conexión y consultas SQL.

- Se usan **ArrayList** para la manipulación de colecciones de objetos en memoria.

El proyecto compila y ejecuta correctamente en Eclipse, estableciendo conexión mediante el driver JDBC a la base de datos remota FreeSQLDatabase.com.

Conclusión

El desarrollo del proyecto *PYME Ventas* permitió integrar los conocimientos de Java y bases de datos, logrando un sistema funcional que cumple con las consignas del trabajo práctico.
El sistema demuestra la capacidad de persistencia de datos, la aplicación de POO y el uso adecuado de excepciones y estructuras dinámicas.

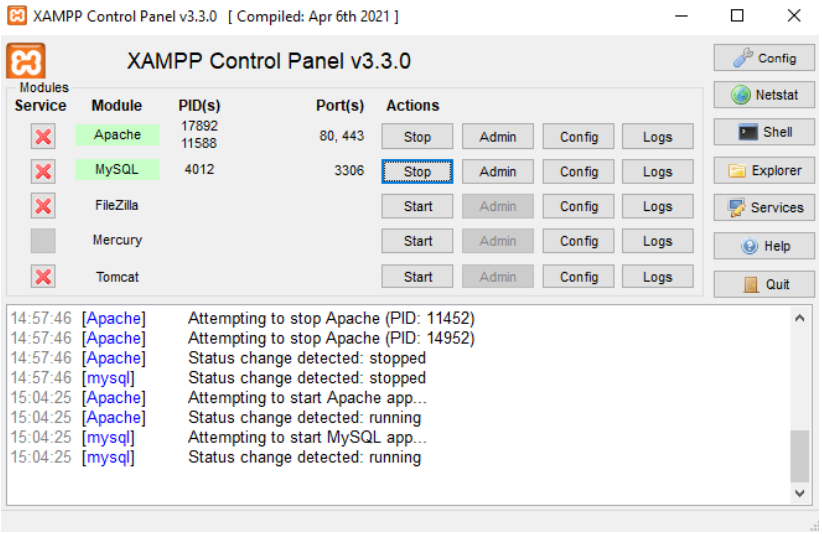
Nota Importante:

Configuración de la base de datos en entorno local (XAMPP)

Durante el desarrollo del proyecto, inicialmente se había utilizado un servicio gratuito de base de datos en línea (*freesqldatabase.com*) para la conexión del sistema con MySQL.
Sin embargo, este servicio tenía una **vigencia limitada**, y al expirar la cuenta, la base de datos dejó de estar disponible.

Por ese motivo, se decidió **migrar la base de datos a un entorno local utilizando XAMPP**, configurando el servidor MySQL en localhost, lo que permitió mantener la persistencia de datos de manera estable y sin restricciones de tiempo.

Figura 14. XAMPP Conexión



Configuración utilizada en XAMPP:

- **Servidor:** localhost

- **Puerto:** 3306
- **Base de datos:** pyme_ventas
- **Usuario:** root
- **Contraseña:** *(vacía)*
- **Gestor:** phpMyAdmin (incluido en XAMPP)

La conexión se gestiona a través de la clase ConexionBD, adaptada para funcionar con el servidor local:

Figura 15. Nueva ConexionBD

```java
package servicio;

import java.sql.Connection;

public class ConexionBD {
    private static final String URL = "jdbc:mysql://localhost:3306/pyme_ventas";
    private static final String USER = "root";
    private static final String PASSWORD = "";

    // devuelve conexión o lanza SQLException
    public static Connection obtenerConexion() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

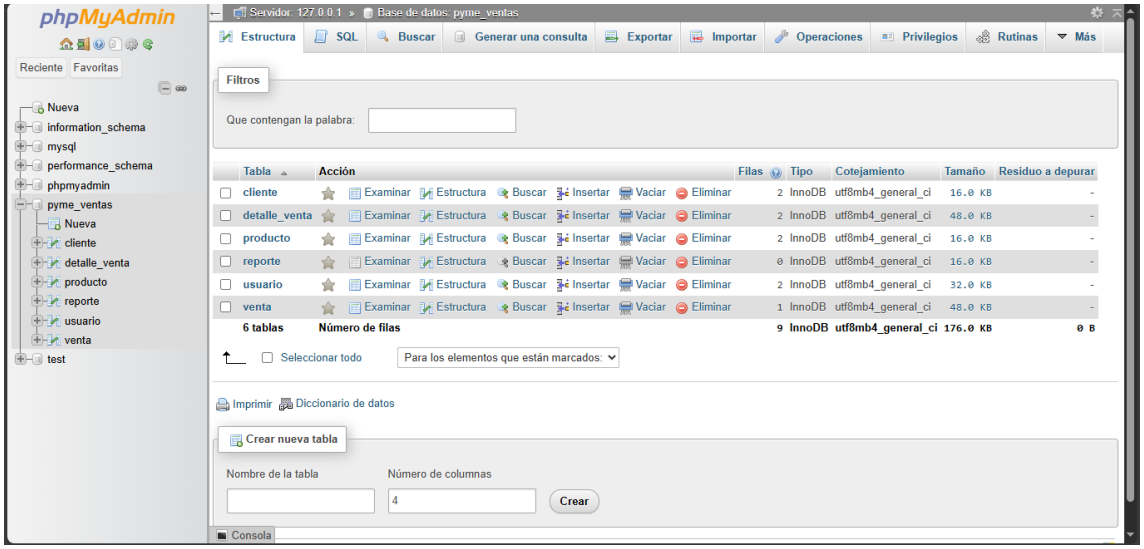
Comprobación del funcionamiento:

El sistema fue probado exitosamente realizando operaciones desde Eclipse que se reflejan en la base de datos local:

- **INSERT:** Registrar nuevos clientes, usuarios y productos.
- **UPDATE:** Modificar información existente (por ejemplo, el stock de un producto).
- **DELETE:** Eliminar registros.
- **SELECT:** Consultar listados de productos, clientes, usuarios y ventas.

Cada operación fue verificada desde phpMyAdmin, confirmando que los datos se modificaban en tiempo real.
Este proceso demuestra la correcta persistencia de datos y la interacción entre Java y MySQL solicitada en las consignas del TP4.

Figura 16. Base de Datos en Localhost



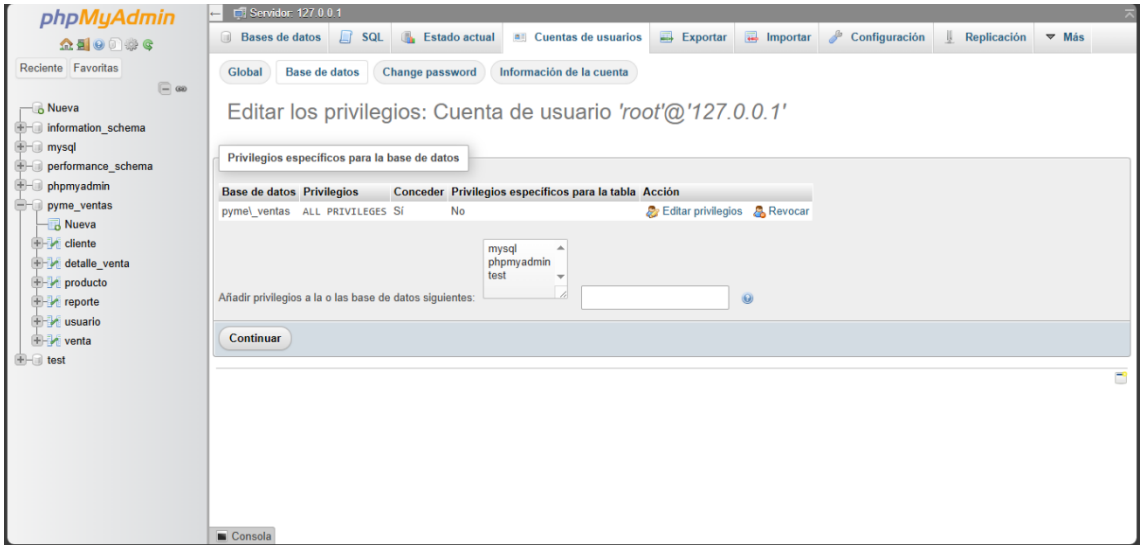
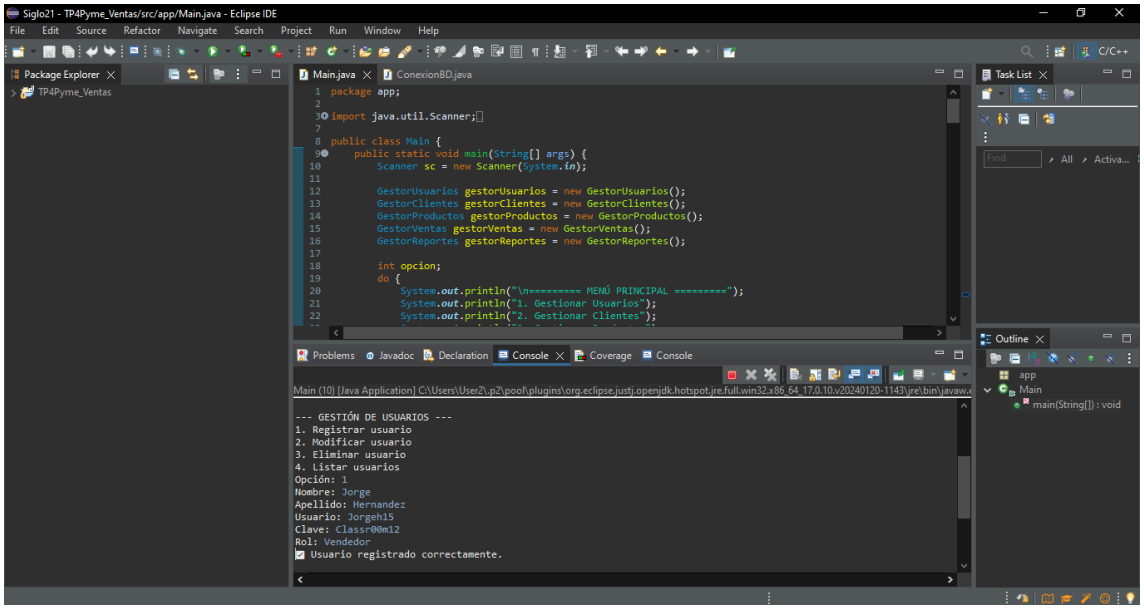
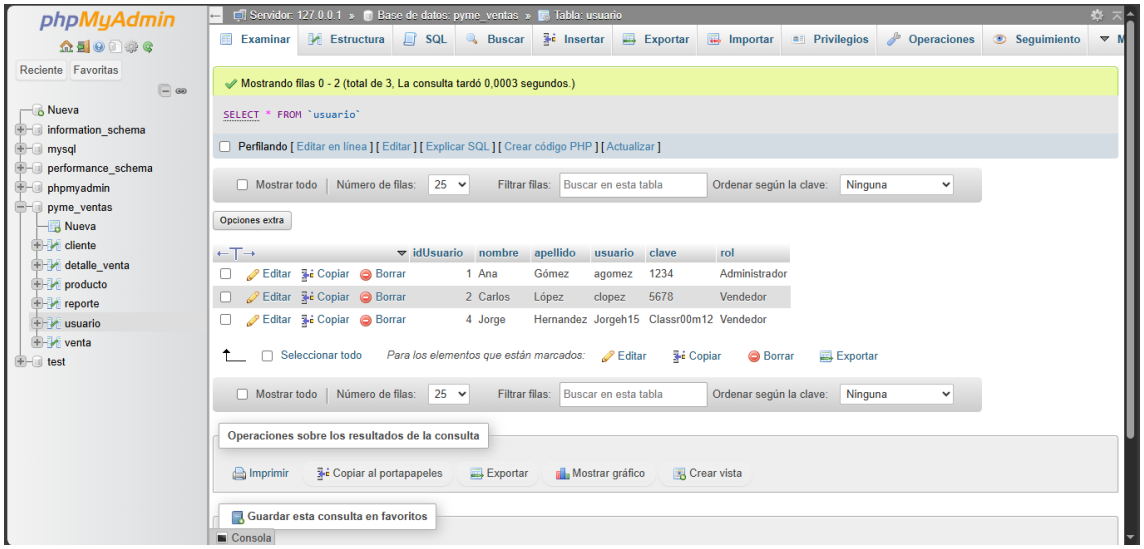


Figura 17. Ejemplo – Registrar Usuario





Presentación del Desarrollo en Java del TP N°4

A continuación, se adjuntan las clases Java y conexión a base de datos para hacer consultas al sistema y comprobar la persistencia. Más la presentación del proyecto en un vídeo de 3 minutos.

GitHub: https://github.com/agustingreco2020/TP4_PymesVentas/tree/main

Presentación del Proyecto en vídeo (Base de Datos MySQL en FreesqlDatabase): https://www.youtube.com/watch?v=4RahFvioQIs

Referencias

- Pressman, R. (2010). *Ingeniería del Software: Un enfoque práctico*. McGraw-Hill.
- Sommerville, I. (2011). *Ingeniería de software*. Pearson Educación.
- Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *El Lenguaje Unificado de Modelado (UML)*. Addison Wesley.
- Oracle. (2025). *Java Database Connectivity (JDBC) Overview*. Disponible en: https://docs.oracle.com
- MySQL Documentation. (2025). *MySQL Reference Manual*. Disponible en: https://dev.mysql.com/doc/
- Compilador Utilizado: https://onecompiler.com/
- Herramienta Utilizada para hacer los diagramas: https://plantuml.com/es/