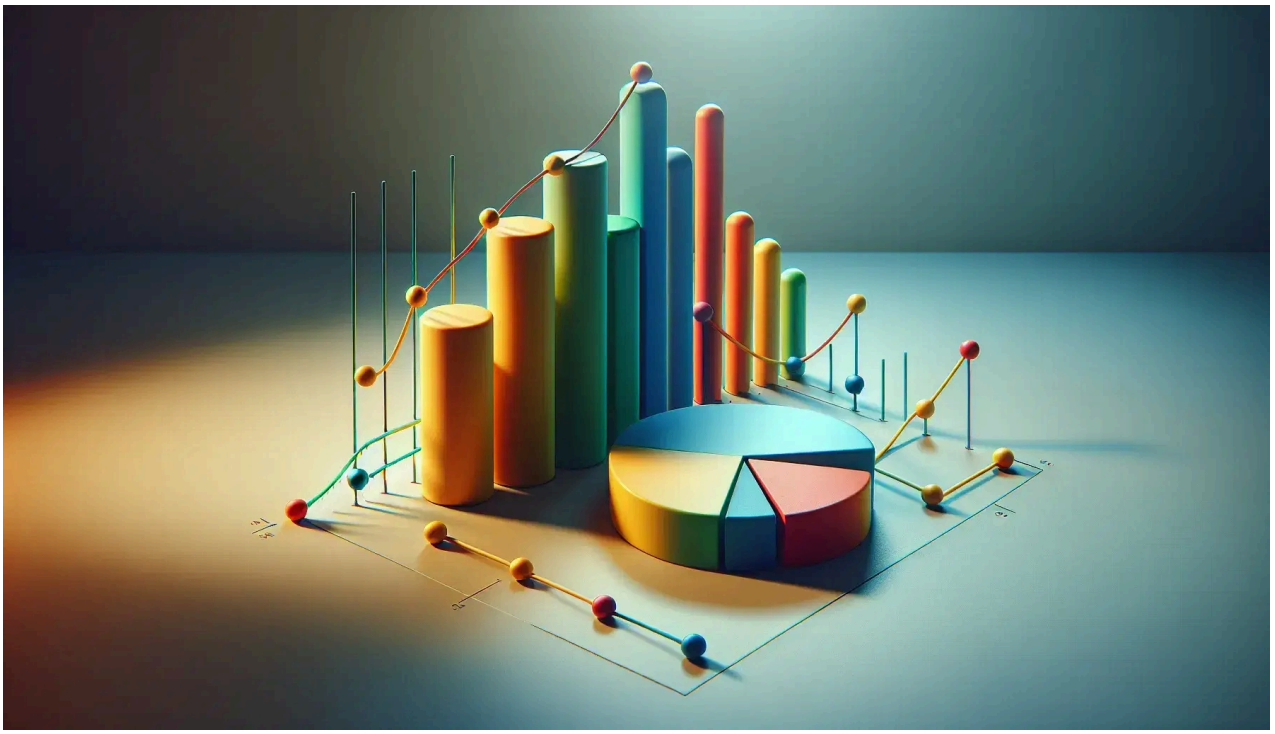
IDRA

TP-ECOMMERCE



Integrantes: Jose Ignacio Gutierrez, Franco Andres Chocou, Sergio Aaron Bravo, Ezequiel Martinez.

Profesor: Diego Mendoza.

Materia: Estadística y probabilidades segundo.

Carrera: Tecnicatura Superior en Desarrollo de Software.

Introducción:

El objetivo del proyecto fue implementar un sistema básico de e-commerce que permite registrar ventas y generar indicadores estadísticos a partir de los datos almacenados. El sistema incluye tanto información precargada como datos que pueden ser ingresados por el administrador.

Este informe describe el diseño, el desarrollo y el análisis estadístico realizados.

2. Consultas SQL principales

Promedio de ventas diarias:

```sql
SELECT DATE(fecha) AS dia,
       AVG(total) AS promedio_ventas_diarias,
       COUNT(*) AS cantidad_ventas
FROM ventas
GROUP BY DATE(fecha);
```

Venta por categoría de productos:

```sql
SELECT categoria, ingreso_total, promedio_venta, unidades_vendidas
FROM vista_resumen_productos;
```

Correlación entre precio y cantidad vendida:

```sql
SELECT precio_unitario, cantidad, total
FROM ventas
ORDER BY precio_unitario DESC;
```

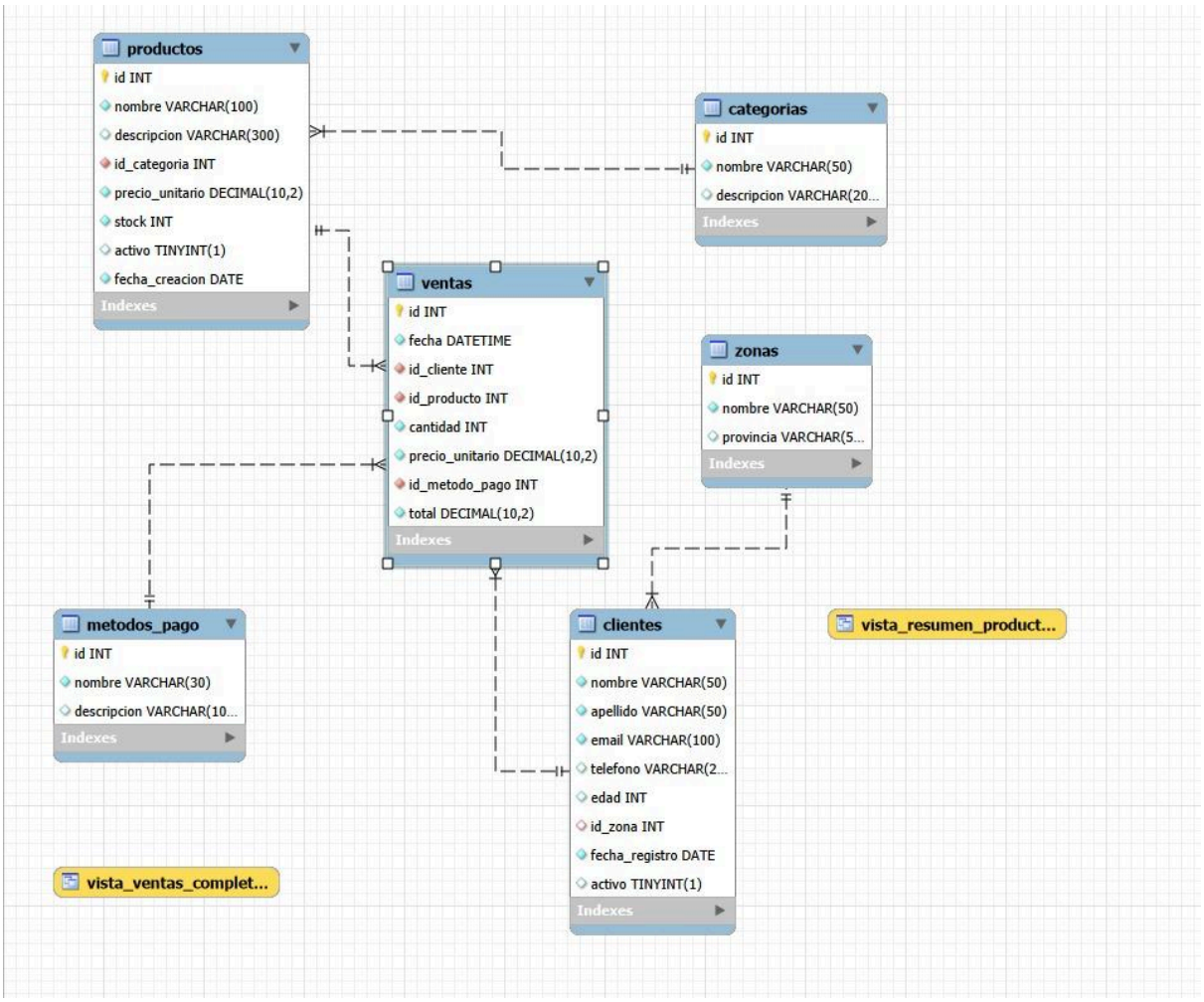
Análisis de clientes por zona:

```sql
SELECT zona, COUNT(*) as ventas_por_zona
FROM vista_ventas_completas
GROUP BY zona;
```

Métodos de pago más utilizados:

```
    SELECT metodo_pago, COUNT(*) as cantidad_ventas, AVG(total) as ticket_promedio
FROM vista_ventas_completas
GROUP BY metodo_pago;
```

Tabla relación entidad:



3. Variables analizadas y su significado

Variable	Tipo	Escala	Significado en el Análisis
total	Cuantitativa	Continua	Monto total de la transacción en pesos
precio_unitario	Cuantitativa	Continua	Valor unitario del producto en pesos
cantidad	Cuantitativa	Discreta	Número de unidades vendidas por transacción
edad	Cuantitativa	Discreta	Edad del cliente en años (segmentación)
fecha	Temporal	-	Fecha y hora de registro de la venta
id_metodo_pago	Cualitativa	Nominal	Medio de pago utilizado (efectivo, tarjeta, etc.)
id_zona	Cualitativa	Nominal	Ubicación geográfica del cliente (por provincia)
stock	Cuantitativa	Discreta	Unidades disponibles del producto
fecha_registro	Temporal	-	Fecha de registro del cliente en el sistema
fecha_creacion	Temporal	-	Fecha de ingreso del producto al catálogo
id_categoria	Cualitativa	Nominal	Tipo de producto (electrónica, ropa, hogar, etc.)

- precio_unitario vs cantidad: Para identificar si productos más caros se venden menos unidades
- edad vs total: Para analizar el comportamiento de compra por grupos etarios
- id_categoria vs ventas: Para identificar las categorías de productos más rentables
- id_zona vs frecuencia_compra: Para análisis geográfico de clientes
- id_metodo_pago vs monto_promedio: Para optimizar medios de pago ofrecidos
- fecha (temporal) vs ventas: Para identificar tendencias estacionales y picos de demanda

-**Cualitativas**: (son números).

-**Continua**: Puede tomar cualquier valor (incluyendo decimales).

-**Discreta**: Solo toma valores enteros (no decimales),

-**Cualitativa**: (Son categorías/texto).

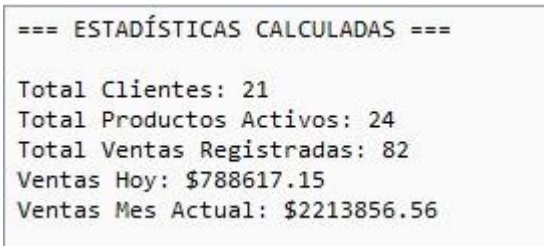
-**Nominal**: Son categorías sin orden específico (método de pago)

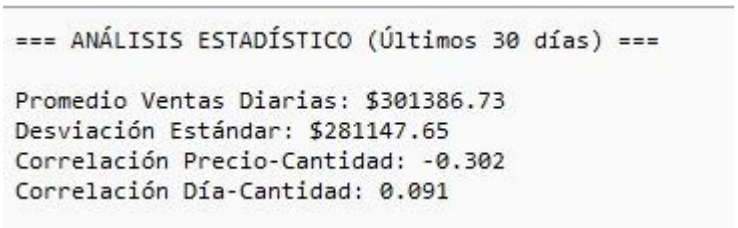
-**Temporales**: Son fechas que permiten análisis en el tiempo (fecha = 2025-11-18 para ver qué ventas por día/mes/año).

4. Resultados de los cálculos estadísticos

Imágenes tomadas del sistema:

Datos cargados de prueba:





```
=== INTERPRETACIÓN CORRELACIONES ===
Precio vs Cantidad Vendida: Correlación NEGATIVA MODERADA
Día de Semana vs Cantidad Vendida: Prácticamente NO HAY correlación
```

Las imágenes muestran la implementación real de los cálculos estadísticos en Java, incluyendo la fórmula de correlación de Pearson para analizar relaciones entre variables y los métodos para calcular métricas de negocio en tiempo real:

Función Calcular Correlacion Precio Cantidad

```java
@Override
public double calcularCorrelacionPrecioCantidad() throws ServiceException {
    try {
        List<Venta> ventas = ventaDAO.buscarTodos();
        if (ventas.isEmpty()) return 0.0;

        // Calcular correlación de Pearson: precio vs cantidad
        double sumXY = 0, sumX = 0, sumY = 0, sumX2 = 0, sumY2 = 0;
        int n = ventas.size();

        for (Venta venta : ventas) {
            double x = venta.getPrecioUnitario(); // Precio
            double y = venta.getCantidad();       // Cantidad

            sumXY += x * y;
            sumX += x;
            sumY += y;
            sumX2 += x * x;
            sumY2 += y * y;
        }

        // Fórmula correlación de Pearson
        double numerador = n * sumXY - sumX * sumY;
        double denominador = Math.sqrt((n * sumX2 - sumX * sumX) * (n * sumY2 - sumY * sumY));

        return denominador != 0 ? numerador / denominador : 0.0;

    } catch (DAOException e) {
        throw new ServiceException("Error al calcular correlación precio-cantidad: " + e.getMessage(), e);
    }
}
```

Funcion Calcular Correlacion Dia Cantidad

```java
@Override
public double calcularCorrelacionDiaCantidad() throws ServiceException {
    try {
        List<Venta> ventas = ventaDAO.buscarTodos();
        if (ventas.isEmpty()) return 0.0;

        // Calcular correlación de Pearson: día de semana (1-7) vs cantidad
        double sumXY = 0, sumX = 0, sumY = 0, sumX2 = 0, sumY2 = 0;
        int n = ventas.size();

        for (Venta venta : ventas) {
            double x = venta.getFecha().getDayOfWeek().getValue();
            double y = venta.getCantidad();

            sumXY += x * y;
            sumX += x;
            sumY += y;
            sumX2 += x * x;
            sumY2 += y * y;
        }

        // Fórmula correlación de Pearson
        double numerador = n * sumXY - sumX * sumY;
        double denominador = Math.sqrt((n * sumX2 - sumX * sumX) * (n * sumY2 - sumY * sumY));

        return denominador != 0 ? numerador / denominador : 0.0;

    } catch (DAOException e) {
        throw new ServiceException("Error al calcular correlación día-cantidad: " + e.getMessage(), e);
    }
}
```

IDRA

Función Calcular Promedio Ventas Diarias

```java
@Override
public double calcularPromedioVentasDiarias(LocalDate fechaInicio, LocalDate fechaFin) throws DAOException {
    String sql = "SELECT AVG(total_dia) as promedio FROM (" +
                 "SELECT DATE(fecha) as dia, SUM(total) as total_dia " +
                 "FROM ventas " +
                 "WHERE DATE(fecha) BETWEEN ? AND ? " +
                 "GROUP BY DATE(fecha)" +
                 ") as ventas_por_dia";

    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setDate(1, Date.valueOf(fechaInicio));
        stmt.setDate(2, Date.valueOf(fechaFin));
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            return rs.getDouble("promedio");
        }

    } catch (SQLException e) {
        throw new DAOException("Error al calcular promedio de ventas diarias", e);
    }

    return 0.0;
}
```

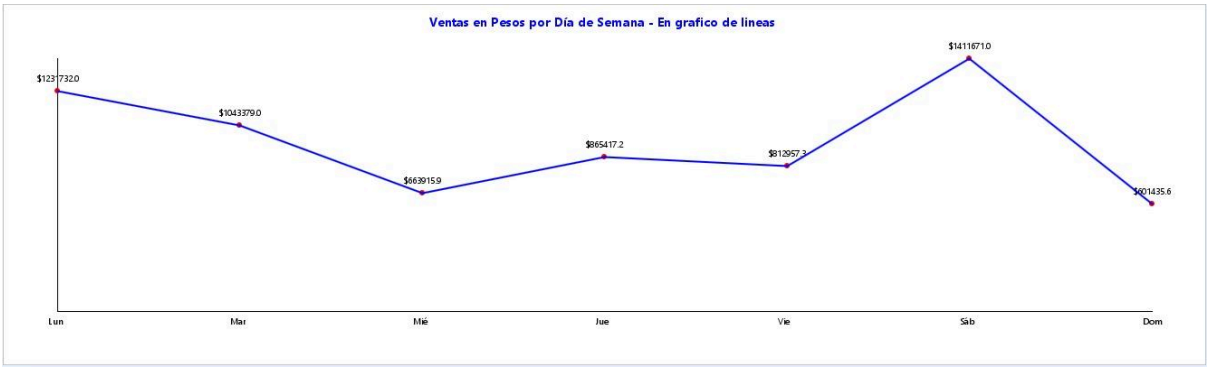
Función Calcular Desviación Estándar Ventas

```java
@Override
public double calcularDesviacionEstandarVentas(LocalDate fechaInicio, LocalDate fechaFin) throws DAOException {
    String sql = "SELECT STDDEV(total_dia) as desviacion FROM (" +
                 "SELECT DATE(fecha) as dia, SUM(total) as total_dia " +
                 "FROM ventas " +
                 "WHERE DATE(fecha) BETWEEN ? AND ? " +
                 "GROUP BY DATE(fecha)" +
                 ") as ventas_por_dia";

    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setDate(1, Date.valueOf(fechaInicio));
        stmt.setDate(2, Date.valueOf(fechaFin));
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            return rs.getDouble("desviacion");
        }

    } catch (SQLException e) {
        throw new DAOException("Error al calcular desviación estándar de ventas", e);
    }

    return 0.0;
}
```

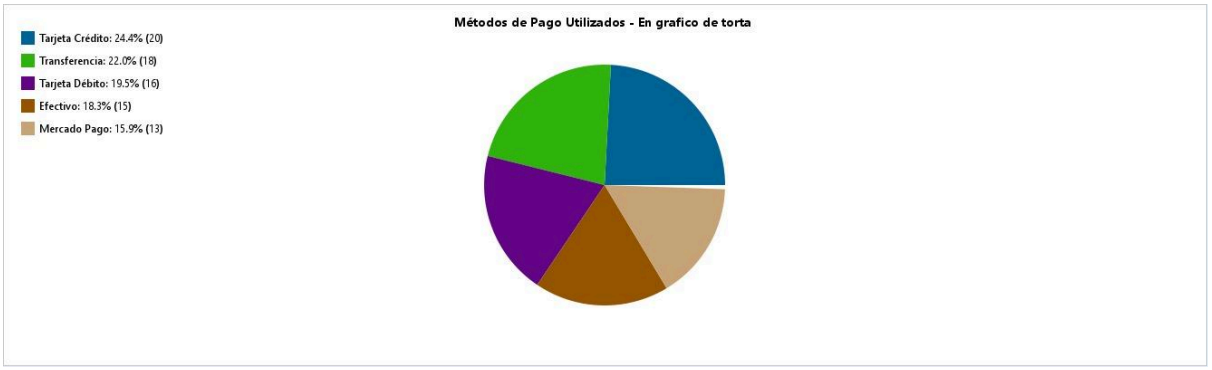
IDRA

Gráficos dentro del programa:

Gráfico de líneas - Ventas en pesos por día de semana.

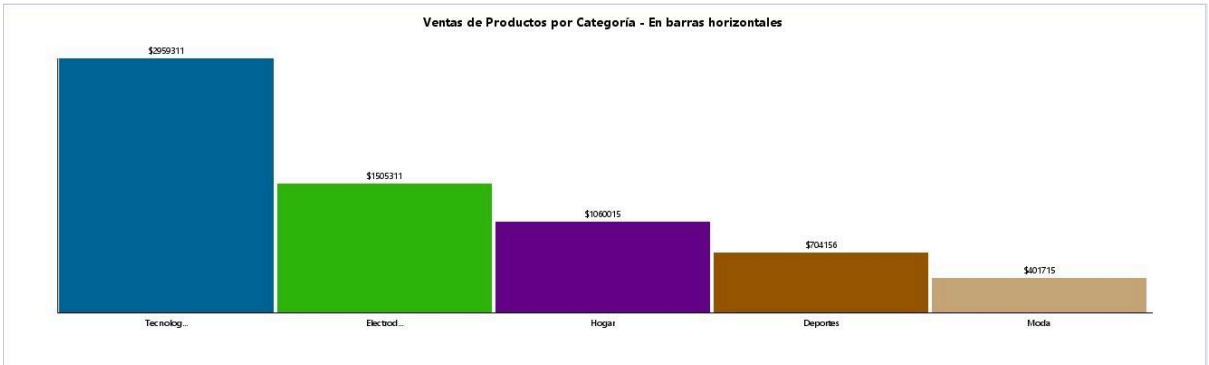


En gráficos de torta - Métodos de pago utilizados.



En barras horizontales - Productos más vendidos.

En barras horizontales - Ventas de productos por categoría.



ARQUITECTURA DE LA API - CÓMO SE CREÓ

-Tecnologías Usadas:
Spring Boot - Framework principal

Spring MVC - Para la arquitectura Controller-Service-DAO

JDBC - Conexión directa a MySQL

Arquitectura en capas - Controller → Service → DAO

Estructura de la API:

```
src/main/java/com/ecommerce/stats/
├── controller/     # Capa API - Endpoints REST
│   ├── VentaController.java
│   ├── ProductoController.java
│   └── ClienteController.java
├── service/        # Lógica de negocio y cálculos estadísticos
│   └── impl/VentaServiceImpl.java
├── dao/            # Acceso a datos y consultas SQL
│   └── impl/VentaDAOImpl.java
└── model/          # Entidades y DTOs
    ├── Venta.java
    ├── Producto.java
    └── Cliente.java
```

Endpoints Principales Implementados:

```
GET    /api/ventas                      # Listar todas las ventas
GET    /api/ventas/{id}                 # Obtener venta por ID
POST   /api/ventas                      # Registrar nueva venta
GET    /api/ventas/estadisticas/promedio    # Promedio de ventas diarias
GET    /api/ventas/estadisticas/desviacion  # Desviación estándar de ventas
GET    /api/ventas/estadisticas/correlacion # Correlación entre variables
GET    /api/ventas/categoria            # Ventas por categoría
```

Gestión de Productos:

```
GET    /api/productos                   # Listar todos los productos
POST   /api/productos                   # Crear nuevo producto
GET    /api/productos/mas-vendidos      # Productos más vendidos
```

Gestión de Clientes:

```
GET    /api/clientes                    # Listar todos los clientes
GET    /api/clientes/{id}/ventas        # Ventas por cliente específico
```

Frontend → Solicita datos al Controller
Controller → Delega lógica al Service
Service → Ejecuta cálculos y valida reglas
DAO → Accede a la base de datos via JDBC
MySQL → Retorna resultados de consultas SQL

5. División de tareas

Aaron Bravo: Diseño de base de datos, consultas SQL, configuración MySQL en WORKBENCH. Primera semana.

Ignacio Gutierrez: Desarrollo backend API Node.js, endpoints estadísticos, lógica de negocio. Segunda semana.

Franco Chocou: Desarrollo frontend, dashboard interactivo, diseño de botones, menús, análisis estadísticos e implementación de datos. Tercera semana.

Ezequiel Martinez: Documentación e informe teórico. Tercera semana.