

# Universidad Nacional Autónoma de México

## Facultad de Ingeniería

**Proyecto Final:** Sistema de Gestión de Base de Datos para una Papelería

**Equipo:** Chicuarotes

**Integrantes:**


- Aguilar Chávez Cristian Michael
- Gutiérrez Gutiérrez Arturo
- Ordoñez Figueroa Maria Fernanda
- Ortiz Valles Joaquín Rafael
- Rueda De Oliveira Chun Shik
- Torres Chávez Juan Carlos
- Villalba Reyes David


**Asignatura:** Bases de Datos

**Semestre:** 2026-1

**Profesor:** Ing. Fernando Arreola Franco

**Fecha de entrega:** 6 de Diciembre del 2025

# Índice

# 1 Introducción

## 1.1 Análisis del Problema

El proyecto consiste en el diseño y desarrollo de una solución integral para la administración de una cadena de papelerías, con el fin de modernizar su gestión de información. Actualmente, la empresa requiere transitar de registros manuales a un sistema robusto que permita administrar de manera eficiente sus procesos críticos de negocio.

Los requerimientos principales identificados son:

- **Gestión de Proveedores y Clientes:** El almacenamiento centralizado de información de proveedores y clientes, considerando la existencia de múltiples medios de contacto (teléfonos y correos electrónicos).
- **Control de Inventario:** La gestión detallada de productos, incluyendo código de barras, costos, precios de venta, control estricto de stock y categorización de artículos.
- **Registro de Ventas:** Un sistema transaccional que permita generar facturas , calculando totales automáticos y manteniendo un histórico de precios unitarios para asegurar la integridad financiera.

## 1.2 Objetivos

El objetivo general es diseñar e implementar una base de datos relacional en PostgreSQL que satisfaga los requerimientos de integridad, almacenamiento y consistencia de la papelería, asegurando la escalabilidad del negocio.

Como objetivos específicos, se busca optimizar los tiempos de consulta y garantizar la actualización automática de la información mediante lógica de negocio en la base de datos.

El presente documento integra la documentación técnica del desarrollo, abarcando desde la organización de los equipos de trabajo hasta la implementación del esquema de base de datos, incluyendo triggers y funciones almacenadas.

# 2 Plan de Trabajo

## 2.1 Metodología de Gestión

Para garantizar el cumplimiento de los objetivos, el equipo adoptó la metodología ágil Scrum. Se utilizó la plataforma **Atlassian Jira** para la planificación, asignación y seguimiento de tareas.

El trabajo se estructuró en **Sprints**, comenzando con el *Sprint 0: Diseño y Estructura* para el modelado, seguido del *Sprint 1: Lógica de Negocios* para la implementación de reglas complejas, y finalizando con el *Sprint 2: Optimizar y Documentar*, centrado en la elaboración del reporte técnico y el afinamiento de la base de datos.

## 2.2 Estructura Organizacional

Debido a la magnitud del proyecto, el equipo se dividió en tres células de trabajo especializadas:

- **Sub-equipo 1 (Gestión y Documentación):** Responsables del reporte técnico, control de versiones y generación de la presentación de negocio.
- **Sub-equipo 2 (Diseño y DDL):** Encargados del modelado de la BD, normalización y scripts SQL.
- **Sub-equipo 3 (Lógica de Negocio y Dashboard):** Responsables de Triggers y visualización de datos.

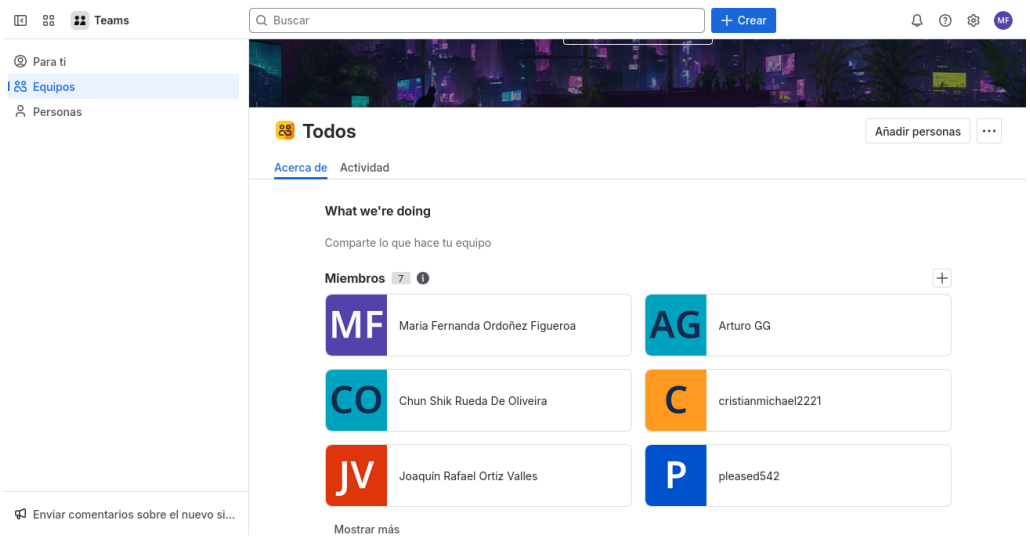


Figura 1: Vista general del equipo en Jira.

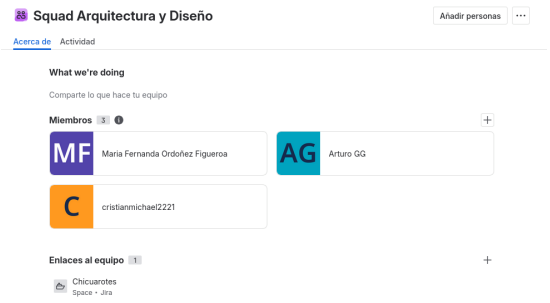


Figura 2: Squad Arquitectura y Diseño.

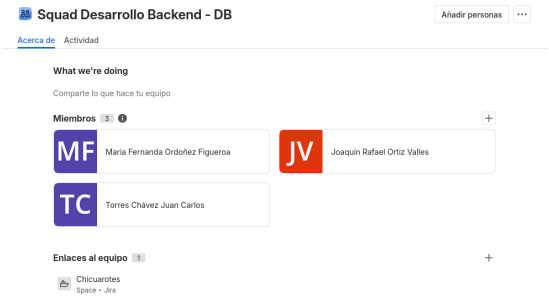


Figura 3: Squad Desarrollo Backend.

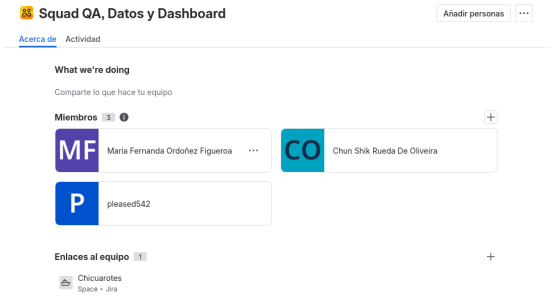


Figura 4: Squad QA y Datos.

## 2.3 Flujo de Trabajo y Comunicación

La estrategia de división en células favoreció un desarrollo continuo e iterativo, estableciendo puntos de control donde los entregables parciales requerían la validación del equipo general antes de integrarse al proyecto final.

Un caso práctico de este flujo fue el diseño del Modelo Entidad-Relación (MER):

1. El sub-equipo de **Diseño y DDL** desarrolló la propuesta inicial.
2. Se solicitó una revisión cruzada a todo el equipo mediante el tablero Kanban.
3. Gracias a la integración de herramientas, el movimiento de la tarjeta en Jira disparaba automáticamente una notificación en el canal de **Slack** del equipo, alertando que existía trabajo en estado de *Revisión"*.

Este mecanismo de notificaciones automatizadas redujo los tiempos de espera y aseguró que todos los integrantes estuvieran sincronizados con los cambios estructurales de la base de datos.

Asi mismo utilizamos una carpeta de Drive para tener los entregrables acomodados y disponibles para la entrega.

## 2.4 Cronograma y Seguimiento

La planificación temporal se gestionó mediante un diagrama de Gantt en Jira. Como se observa en la Figura 5, se definieron hitos claros para evitar cuellos de botella.

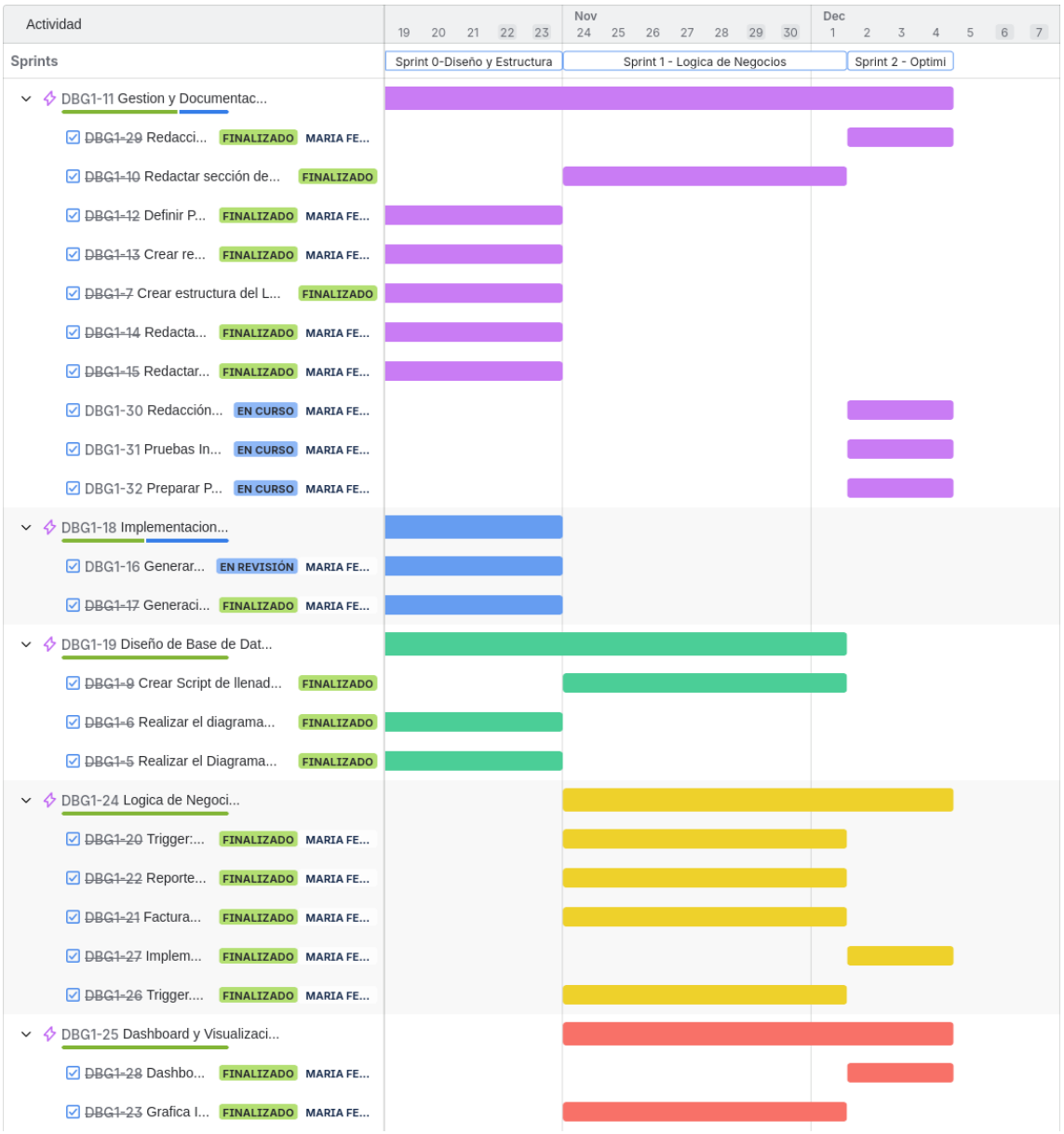


Figura 5: Línea de tiempo (Roadmap) mostrando Sprints y Epics.

Para el control diario, se utilizó un tablero Kanban que permitió visualizar el flujo de trabajo por sub-equipo (Figura 6).

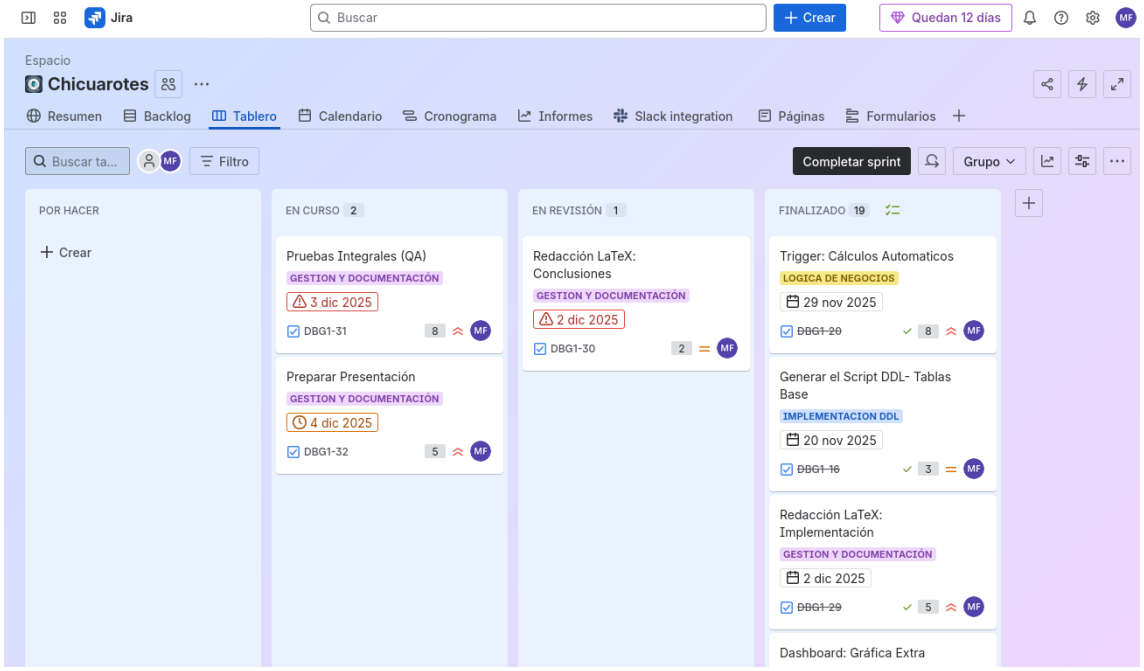


Figura 6: Tablero Kanban utilizado para el seguimiento de tareas.

# 3 Diseño de la Base de Datos

## 3.1 Modelo Entidad-Relación (MER)

Para la fase de diseño conceptual, se elaboró un Modelo Entidad-Relación utilizando la notación estándar de Peter Chen. El objetivo fue representar la semántica del negocio de papelería sin las restricciones físicas de la base de datos.

Se identificaron las siguientes características estructurales clave:

- **Entidades Principales:** Se definieron cuatro entidades fuertes: *Proveedor, Cliente, Producto y Venta*.
- **Manejo de Atributos Complejos:**
  - **Atributos Compuestos:** Para los domicilios (tanto de clientes como proveedores), se utilizó una estructura compuesta (calle, número, colonia, CP, estado) para facilitar futuras búsquedas geográficas y normalización.
  - **Atributos Multivaluados:** Se identificaron *teléfonos* (en Proveedor) y *emails* (en Cliente) como atributos multivaluados, reconociendo que en la realidad un sujeto puede tener múltiples medios de contacto.
- **Relaciones y Cardinalidades:**
  - **Cliente - Venta (1:M):** Un cliente puede generar múltiples ventas a lo largo del tiempo, pero una venta específica pertenece a un único cliente.
  - **Venta - Producto (M:M):** Se modeló la relación *"Contiene"* como de muchos a muchos, ya que una venta incluye varios productos y un producto puede estar en muchas ventas.
  - **Atributos de Relación:** Es importante destacar que los atributos *cantidad* y *precio unitario* se asociaron a la relación *"Contiene"* y no a las entidades, para garantizar el registro histórico exacto de cada transacción.

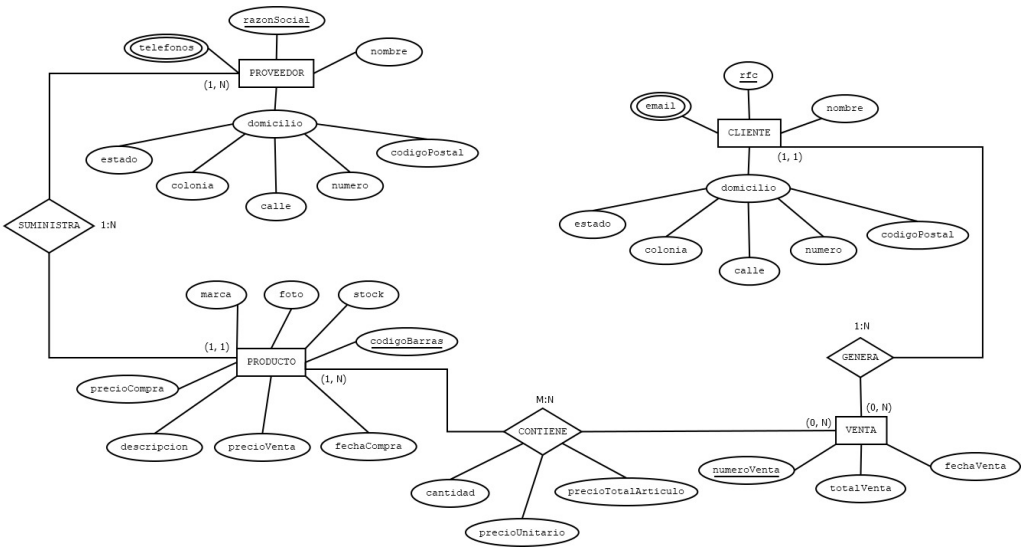


Figura 7: Diagrama Modelo-Entidad-Relación

## 3.2 Modelo Relacional (MR)

El esquema conceptual se transformó a un modelo relacional normalizado.

- **Normalización:** Se cumple la 1FN extrayendo datos repetitivos a tablas auxiliares (`TELEFONO_PROVEEDOR`).
- **Estrategia de Proveedores:** Se optó por una relación 1:M entre Proveedor y Producto para simplificar la gestión de reabastecimiento.
- **Datos Históricos:** Se incluyó el campo `precio_unitario_aplicado` en el detalle de venta para preservar la integridad histórica de los precios.

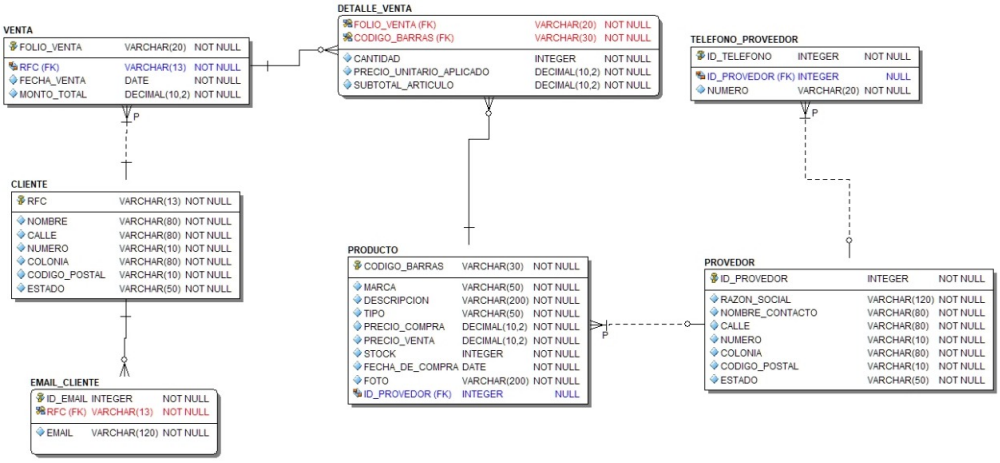


Figura 8: Modelo Relacional Final.

# 4 Implementación

La materialización de este proyecto fue resultado de un proceso de ingeniería incremental, ejecutado a lo largo de varios días de desarrollo. Siguiendo la planificación de nuestro tablero Kanban, la construcción del sistema no se limitó a escribir código, sino que siguió una metodología por capas para garantizar la estabilidad en cada paso.

El despliegue se realizó en cuatro fases críticas, asegurando que cada componente funcionara perfectamente antes de construir el siguiente nivel de complejidad:

1. **Fase 1 (Cimientos):** Definición estructural y restricciones de integridad (DDL).
2. **Fase 2 (Volumen):** Estrategias de poblado de datos y validación de relaciones (DML).
3. **Fase 3 (Inteligencia):** Programación de la lógica de negocio automatizada (Triggers y PL/pgSQL).
4. **Fase 4 (Seguridad y Control):** Implementación de auditoría avanzada y vistas de reporte.

A continuación, se documenta el trabajo técnico realizado en cada etapa.

## 4.1 Fase 1: Construcción de la Estructura (DDL)

El primer desafío fue traducir el Modelo Relacional a un esquema físico robusto en PostgreSQL. Esta etapa requirió un análisis cuidadoso del orden de creación para satisfacer la integridad referencial: se priorizaron los catálogos maestros (`proveedor`, `cliente`) antes de las tablas transaccionales.

Además, no solo se definieron los tipos de datos, sino que se integraron reglas de negocio defensivas directamente en la estructura (constraints), impidiendo desde el día uno que el sistema aceptara datos incoherentes como inventarios negativos.

```
-- Tabla PRODUCTO
CREATE TABLE producto (
    codigo_barras VARCHAR(30) PRIMARY KEY,
    marca VARCHAR(50),
    descripcion VARCHAR(200),
    tipo VARCHAR(50),
    precio_compra DECIMAL(10,2) NOT NULL,
    precio_venta DECIMAL(10,2) NOT NULL,
    -- Validacion critica como regla de negocio, el stock fisico
   nunca puede ser menor a 0
    stock INT NOT NULL CHECK (stock >= 0),
    fecha_compra DATE,
    id_proveedor INT REFERENCES proveedor(id_proveedor)
        ON UPDATE CASCADE ON DELETE SET NULL
);

-- Tabla DETALLE_VENTA
CREATE TABLE detalle_venta (
    folio_venta VARCHAR(20) REFERENCES venta(folio_venta) ON UPDATE
    CASCADE ON DELETE CASCADE,
```

```
    codigo_barras VARCHAR(30) REFERENCES producto(codigo_barras) ON
    UPDATE CASCADE ON DELETE CASCADE,
    cantidad INT NOT NULL CHECK (cantidad > 0),
    precio_unitario_aplicado DECIMAL(10,2),
    subtotal_articulo DECIMAL(10,2),
    PRIMARY KEY (folio_venta, codigo_barras)
);
```

Listing 1: Definición de tablas con validaciones nativas

### 4.2 Fase 2: Estrategia de Poblado de Datos (DML)

Con la estructura lista, se procedió a la inyección de datos. Esta fase fue crucial para validar que las relaciones 1:M como Proveedor-Producto funcionaran correctamente antes de programar la lógica compleja.

Se diseñó un script de inserción jerárquico que simuló un entorno real de operación, registrando múltiples proveedores y un catálogo diverso de productos para someter a prueba la capacidad de almacenamiento del sistema.

```
-- Registro de Proveedores
INSERT INTO proveedor (razon_social, ...) VALUES
('Distribuidora Papelera Nacional S.A. de C.V.', ...),
('BIC M xico S.A. de C.V.', ...);

-- Alta de Inventario Inicial
INSERT INTO producto (codigo_barras, marca, tipo, stock,
   id_proveedor)
VALUES ('750100000001', 'Scribe', 'Papeler a', 100, 1);

-- Simulacion de Ventas Hist ricas
INSERT INTO venta (folio_venta, rfc_cliente, fecha_venta,
   monto_total)
VALUES ('VENT-001', 'HERN850505H22', '2025-11-01 10:00:00', 91.00);
```

Listing 2: Script de carga inicial de datos

### 4.3 Fase 3: Desarrollo de la Lógica de Negocio (Triggers)

La etapa más intensiva del desarrollo fue generar la logica en la base de datos. Se programaron disparadores en PL/pgSQL para automatizar el flujo financiero y de inventario, eliminando la dependencia de cálculos manuales y reduciendo el error humano.

Se cubrieron dos escenarios los insert y updates. Acontinuacion estan detallados

#### 4.3.1. Automatización de Ventas Nuevas (INSERT)

Se desarrolló un trigger que intercepta cada nueva venta para validar disponibilidad en tiempo real, descontar el producto del inventario y calcular los costos monetarios al instante.

```
CREATE OR REPLACE FUNCTION actualizar_venta_insert()
RETURNS TRIGGER AS $$
DECLARE
    stock_actual INT;
    subtotal_nuevo DECIMAL(10,2);
```

```
BEGIN
    -- Se verifica la disponibilidad
    SELECT stock INTO stock_actual FROM producto WHERE codigo_barras = NEW.codigo_barras;
    IF stock_actual < NEW.cantidad THEN
        RAISE EXCEPTION 'Stock insuficiente. Disponibles: %', stock_actual;
    END IF;

    -- Se ve reflejado en el inventario
    UPDATE producto SET stock = stock - NEW.cantidad WHERE codigo_barras = NEW.codigo_barras;

    -- Se hace el calculo financiero de forma automatizada
    subtotal_nuevo := NEW.cantidad * NEW.precio_unitario_aplicado;
    NEW.subtotal_articulo := subtotal_nuevo;

    UPDATE venta SET monto_total = COALESCE(monto_total, 0) + subtotal_nuevo
    WHERE folio_venta = NEW.folio_venta;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Listing 3: Lógica automática para nuevas ventas

### 4.3.2. Gestión de Correcciones y Cambios (UPDATE)

Reconociendo que los errores operativos ocurren, se implementó una lógica de reversión y reaplicación". Si una venta se modifica, el sistema es capaz de devolver el stock original y volver a calcular todo el escenario con los nuevos datos, manteniendo la contabilidad intacta.

```
CREATE OR REPLACE FUNCTION actualizar_venta_update() RETURNS TRIGGER AS $$
DECLARE
    stock_despues INT;
    subtotal_anterior DECIMAL(10,2);
    subtotal_nuevo DECIMAL(10,2);
BEGIN
    -- Se hace la revision
    subtotal_anterior := OLD.subtotal_articulo;
    UPDATE producto SET stock = stock + OLD.cantidad WHERE codigo_barras = OLD.codigo_barras;

    -- Se crea la nueva validaci n y descuento
    SELECT stock INTO stock_despues FROM producto WHERE codigo_barras = NEW.codigo_barras;
    IF stock_despues < NEW.cantidad THEN
        RAISE EXCEPTION 'Stock insuficiente para la modificaci n.';
    END IF;
    UPDATE producto SET stock = stock - NEW.cantidad WHERE codigo_barras = NEW.codigo_barras;

    -- Si el stock esta bajo, se manda una alerta
    IF (stock_despues - NEW.cantidad) < 3 THEN
```

```
        RAISE NOTICE 'ALERTA: Stock cr tico para producto %', NEW.codigo_barras;
    END IF;

    -- Se genera el ajuste en caja
    subtotal_nuevo := NEW.cantidad * NEW.precio_unitario_aplicado;
    NEW.subtotal_articulo := subtotal_nuevo;
    UPDATE venta
    SET monto_total = COALESCE(monto_total, 0) - subtotal_anterior + subtotal_nuevo
    WHERE folio_venta = NEW.folio_venta;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Listing 4: Lógica de corrección de ventas

### 4.4 Fase 4: Seguridad y Reportes (Auditoría y Vistas)

Finalmente, para elevar el nivel profesional del sistema, se implementó un módulo de auditoría utilizando tecnología `JSONB`, permitiendo rastrear cada cambio en la base de datos (quién, cuándo y qué cambió).

Además, se desarrollaron objetos de visualización para facilitar la operación diaria y la inteligencia de negocios:

- **Índice:** Se implementó un índice no agrupado (*Non-Clustered*) en la columna `fecha_venta` de la tabla `venta`.
  - *Justificación:* El sistema requiere generar reportes de ganancias por rangos de fechas (diario, mensual, anual). Este índice optimiza drásticamente el tiempo de respuesta al evitar lecturas secuenciales completas de la tabla histórica.
- **Vista :** Una vista que realiza un *JOIN* entre las cuatro tablas principales para presentar un resumen legible de la venta. Se añadió lógica de concatenación para mostrar la dirección fiscal completa en una sola columna.

```
-- Indice para los reportes de tiempo
CREATE INDEX idx_venta_fecha ON venta(fecha_venta);

-- Vista de la factura
CREATE OR REPLACE VIEW vista_factura AS
SELECT
    v.folio_venta AS "Folio",
    v.fecha_venta AS "Fecha Venta",
    c.rfc AS "RFC Cliente",
    c.nombre AS "Nombre Cliente",
    -- Concatenacion de direccion fiscal
    CONCAT(c.calle, ' #', c.numero, ', ', c.colonia, ', ',
           c.codigo_postal, ', ', c.estado) AS "Direccion Fiscal",
    p.descripcion AS "Producto",
    d.cantidad AS "Cantidad",
    d.precio_unitario_aplicado AS "Precio Unit.",
    d.subtotal_articulo AS "Importe",
```

```
    v.monto_total AS "Total Venta"
FROM venta v
JOIN cliente c ON v.rfc_cliente = c.rfc
JOIN detalle_venta d ON v.folio_venta = d.folio_venta
JOIN producto p ON d.codigo_barras = p.codigo_barras
ORDER BY v.folio_venta ASC, p.descripcion ASC;
```

Listing 5: Objetos para optimización y reporte

# 5 Aplicación

Como parte de la solución integral, se diseñó un módulo de visualización de datos orientado a la toma de decisiones estratégicas.

## 5.1 Análisis de Ingresos por Producto (Noviembre 2025)

Esta visualización da respuesta al requerimiento de negocio para identificar los artículos más rentables.

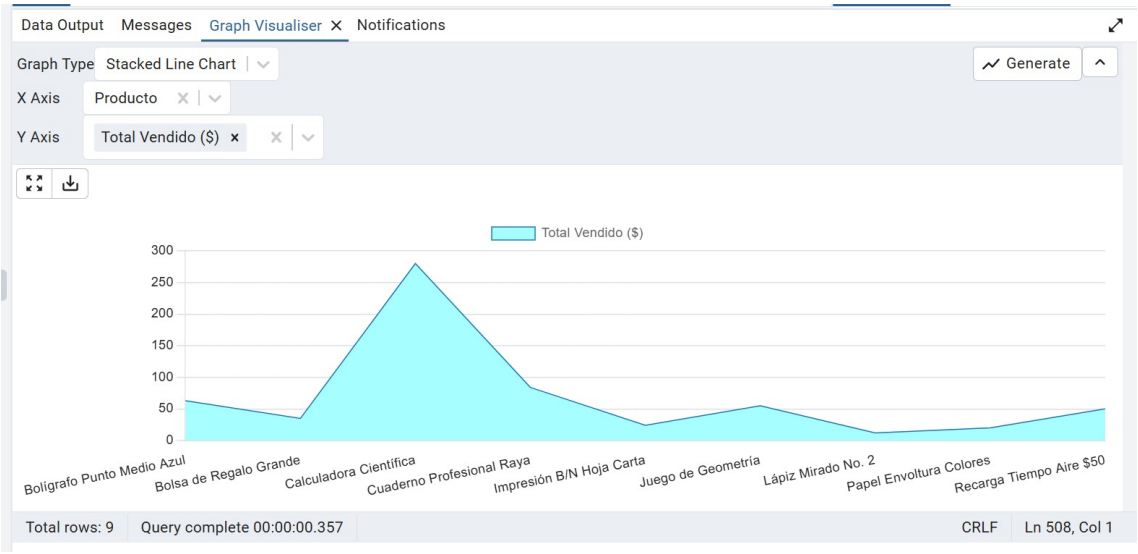


Figura 9: Distribución de ingresos totales por producto.

**Interpretación:** El producto *Çalculadora Científica"* representa el pico máximo de recaudación, validando la regla de Pareto (pocos productos generan la mayor parte del ingreso).

## 5.2 Monitoreo de Niveles de Stock

Para prevenir la pérdida de ventas, se generó un reporte visual de existencias.

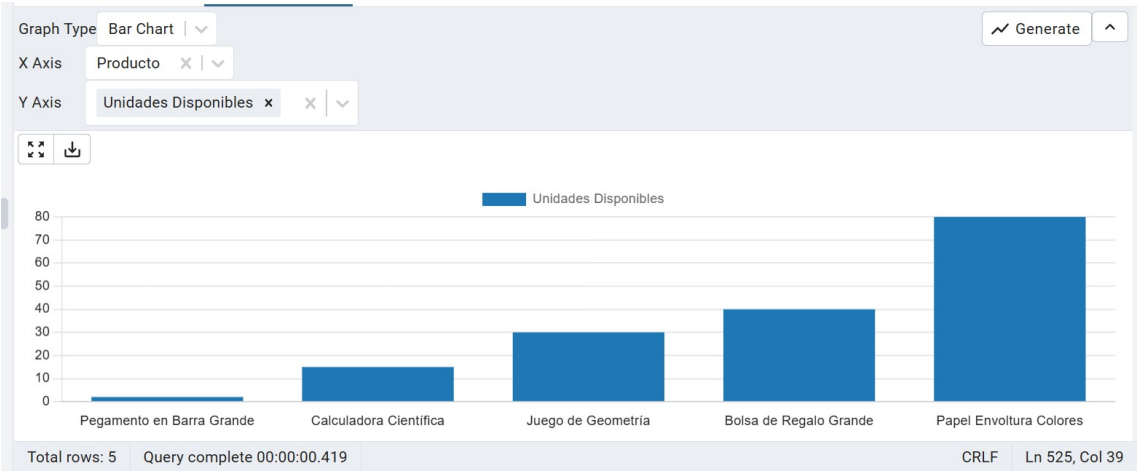


Figura 10: Estado actual del inventario por unidad.

**Interpretación:** Se observa claramente que el artículo *"Pegamento en Barra"* está en niveles críticos (cercanos a 0), activando las alertas del sistema, mientras que el *"Papel Envoltura"* tiene sobrestock.

# 6 Conclusiones

La realización de este proyecto ha representado para nuestro equipo, Chicuarotes, la consolidación práctica de los fundamentos que hemos adquirido durante el curso de Bases de Datos. A través de las metodologías y la división de trabajo, logramos pasar por las etapas del ciclo de vida de desarrollo de una base de datos, comenzando con la aplicación de las técnicas de modelado para transformar los problemas del negocio mediante entidades y relaciones, lo cual nos lleva al MER. Este primer paso fue muy importante para comprender la información antes de proceder a su transformación al Modelo Relacional , donde aplicamos las reglas de integridad referencial para garantizar la coherencia entre proveedores, productos y ventas.

Uno de los puntos más importantes de nuestro desarrollo fue el proceso de Normalización , el cual nos permitió refinar el diseño eliminando redundancias y evitando anomalías de actualización. Al aplicar todas las formas normales hasta la 3FN, logrando estructurar una base de datos que es eficiente, y separa los catálogos de las tablas. Después, llevamos este diseño al plano del software usando PostgreSQL, donde empleamos el Lenguaje de Definición de Datos (DDL) para establecer restricciones, y a su vez usamos el Lenguaje de Manipulación de Datos (DML) para gestionar el flujo operativo del sistema.

Finalmente, el proyecto nos permitió profundizar en la lógica de bases de datos mediante la implementación de los Triggers y Procedimientos Almacenados, aplicando los conceptos de Transacciones y sus propiedades ACID. Gracias a esto, logramos que el sistema garantice la atomicidad y también la consistencia en cada venta, así aseguramos que el inventario y los totales financieros se actualicen de manera sincronizada y automática. El uso de Vistas e Índices para la generación de reportes y optimización de consultas demuestra que nuestra base no solo almacena datos, sino que ayuda a los usuarios para la toma de decisiones en su negocio.