



Universidad Nacional Autónoma de México
Facultad de Ingeniería

Proyecto Final de Bases de Datos

Semestre 2026-1

Fecha de entrega: 06/12/2025

Equipo: Databased Team

Integrantes:

- Buendía López Sebastián
- Hernández Pérez Mariana Daniela
- Velarde Valencia Josue
- Velazquez Cortes Cristina del Carmen
- Vences Santillán Carlos Eduardo

Índice

Índice

| | |
|---|-----------|
| 1. Introducción | 2 |
| 1.1. Descripción del problema | 2 |
| 1.2. Objetivos | 2 |
| 1.3. Propuesta de solución | 3 |
| 2. Plan de trabajo | 3 |
| 2.1. Actividades | 3 |
| 2.2. Plan de actividades | 7 |
| 3. Diseño | 8 |
| 3.1. Modelo Entidad–Relación | 8 |
| 3.2. Modelo Relacional | 9 |
| 3.3. Normalización | 10 |
| 4. Implementación | 12 |
| 4.1. Creación de tablas en SQL | 12 |
| 4.2. Inserción de datos | 14 |
| 4.3. Índex | 17 |
| 4.4. Funciones | 18 |
| 4.5. Triggers | 21 |
| 4.6. Pruebas funciones | 24 |
| 4.7. Vistas | 26 |
| 4.8. Consultas requeridas | 30 |
| 5. Aplicación | 32 |
| 5.1. Dashboards | 32 |
| 6. Resultados | 35 |
| 7. Conclusiones | 39 |
| 7.1. Buendía López Sebastián | 39 |
| 7.2. Hernández Pérez Mariana Daniela | 39 |
| 7.3. Velarde Valencia Josue | 39 |
| 7.4. Velazquez Cortes Cristina del Carmen | 39 |
| 7.5. Vences Santillán Carlos Eduardo | 40 |
| 8. Referencias | 41 |

1. Introducción

1.1. Descripción del problema

La cadena de papelerías enfrenta dificultades para gestionar de manera organizada y consistente la información relacionada con sus operaciones diarias. Actualmente no cuenta con un sistema unificado que permita administrar proveedores, clientes, productos, inventarios y ventas, lo que genera procesos lentos, dependientes de registros manuales y con potenciales errores. Esta falta de estructura provoca problemas como pérdida de información, inconsistencias en el stock, dificultad para calcular utilidades y ausencia de reportes que apoyen la toma de decisiones.

Además, la operación del negocio requiere manejar datos específicos como precios de compra y venta, fechas, cantidades, marcas, descripciones y códigos de barras, así como registrar adecuadamente cada transacción realizada. El negocio también necesita automatizar tareas clave, como actualizar el inventario después de cada venta, generar alertas cuando un producto está por agotarse o validar que existan suficientes unidades antes de completar una transacción. Sin un sistema que implemente estas reglas de forma automática, el control operativo resulta ineficiente e inconsistente.

Finalmente, la empresa carece de herramientas que le permitan analizar su información histórica. No cuenta con reportes de ventas por periodo, métricas de ingreso, utilidad por producto ni visualizaciones que permitan evaluar el rendimiento del negocio. Todo esto dificulta la planeación y limita la capacidad de identificar tendencias o áreas de oportunidad.

1.2. Objetivos

Diseñar e implementar una base de datos integral que permita gestionar de forma eficiente la información de proveedores, clientes, productos, inventarios y ventas, automatizando procesos y facilitando la toma de decisiones mediante reportes y visualizaciones.

- Modelar correctamente las entidades y relaciones del negocio mediante un diseño conceptual y relacional consistente.
- Implementar en PostgreSQL las tablas, restricciones, funciones y triggers necesarios para asegurar la integridad de los datos y automatizar las reglas operativas.
- Crear vistas y consultas que permitan obtener información relevante como utilidades, ventas por periodo y productos con bajo inventario.
- Desarrollar dashboards que muestren de forma clara e intuitiva los indicadores clave del negocio.
- Garantizar que la solución final cumpla con los requerimientos funcionales establecidos y pueda ser utilizada para mejorar la administración del negocio.

1.3. Propuesta de solución

Para atender las necesidades del proyecto, proponemos una solución integral que cubre todo el proceso de creación de una base de datos: desde el modelado inicial hasta el análisis final de información. Para ello se emplearon tres herramientas principales: Drawio, PostgreSQL y Power BI, cada una elegida por su funcionalidad en una etapa específica del desarrollo.

En la fase de diseño conceptual, Drawio permite construir de manera clara y colaborativa el Modelo Entidad–Relación, facilitando la identificación de entidades, relaciones y reglas del negocio antes de implementar la solución. Lo seleccionamos gracias a su interfaz intuitiva y su capacidad para generar diagramas limpios, lo que reduce errores en fases posteriores.

Para la implementación, se seleccionó PostgreSQL debido a su robustez, seguridad y soporte avanzado para triggers, funciones y validaciones, los cuales se ocupan para automatizar procesos como el control de inventario, la generación de alertas y el cálculo de utilidades. El manejador PostgreSQL garantiza integridad de datos, rendimiento y flexibilidad.

Finalmente, proponemos utilizar Power BI para transformar la información almacenada en la base de datos en dashboards interactivos. Esta herramienta permite visualizar ingresos por periodo, comportamiento de ventas y métricas clave de manera accesible para usuarios no técnicos, lo que facilita la toma de decisiones basada en datos reales.

2. Plan de trabajo

2.1. Actividades

Para las actividades a realizar, hicimos una lista con todo lo necesario para llevar el proyecto de la mejor manera.

| Nombre | Descripción |
|---------------|--|
| Tarea 0 | Gestión y organización del equipo. |
| FASE 1 | Ánalisis de Requerimientos |
| Tarea 1 | Leer y analizar el documento del proyecto. |
| Tarea 2 | Identificar entidades necesarias (proveedores, clientes, inventario, ventas, artículos, etc.) |
| Tarea 3 | Identificar atributos obligatorios (RFC, emails, domicilio compuesto, códigos de barras, etc.) |
| Tarea 4 | Identificar reglas de negocio (stock, utilidad, alertas, formato VENT-000, etc.) |
| Tarea 5 | Elaborar el documento de requerimientos resumido. |
| FASE 2 | Diseño Conceptual (MER). |
| Tarea 6 | Diseñar el MER completo en Draw.io o herramienta elegida. |
| Tarea 7 | Definir entidades, relaciones, cardinalidades y atributos. |
| Tarea 8 | Ajustar MER según retroalimentación del equipo. |
| Tarea 9 | Exportar MER. |
| FASE 3 | Diseño Lógico (MR) |
| Tarea 10 | Convertir MER → MR (tablas, claves, relaciones, dominios). |
| Tarea 11 | Definir claves primarias y foráneas. |
| Tarea 12 | Definir atributos obligatorios y restricciones. |
| Tarea 13 | Validar normalización (1FN, 2FN, 3FN). |
| Tarea 14 | Ajustar MR según retroalimentación del equipo. |
| FASE 4 | Diseño Físico |
| Tarea 15 | Crear el script DDL de todas las tablas. |
| Tarea 16 | Crear tipos compuestos. |
| Tarea 17 | Elegir y crear al menos un índice, justificando: tipo y ubicación. |
| Tarea 18 | Definir estrategias de integridad referencial (ON DELETE/UPDATE). |
| Tarea 19 | Crear los primeros inserts de datos de prueba. |
| FASE 5 | Implementación |

| | |
|---------------|--|
| Tarea 20 | Trigger para decrementar stock después de una venta. |
| Tarea 21 | Validación de stock = 0 → abortar transacción. |
| Tarea 22 | Validación stock < 3 → alerta. |
| Tarea 23 | Cálculo automático de subtotal por artículo. |
| Tarea 24 | Cálculo automático de total de venta. |
| Tarea 25 | Generar número de venta formato “VENT-001”. |
| Tarea 26 | Función para obtener utilidad por código de barras. |
| Tarea 27 | Función para obtener productos con stock < 3. |
| Tarea 28 | Crear vista estilo factura. |
| Tarea 29 | Consulta: total vendido por fecha específica. |
| Tarea 30 | Consulta: total vendido por rango de fechas. |
| Tarea 31 | Consulta: ganancia total en periodo. |
| Tarea 32 | Consulta para dashboard: ingresos mensuales por artículo. |
| Tarea 33 | Segunda consulta para dashboard. |
| Tarea 34 | Crear dashboard en Power BI. |
| Tarea 35 | Conectar dashboard a la BD. |
| Tarea 36 | Generar gráficas requeridas. |
| FASE 6 | Pruebas |
| Tarea 37 | Probar triggers con ventas reales. |
| Tarea 38 | Probar consultas con datos finales. |
| Tarea 39 | Validar que la vista factura genera la información correcta. |
| Tarea 40 | Revisar integridad referencial con deletes/updates. |
| Tarea 41 | Validar que el dashboard refleja datos correctos. |
| Tarea 42 | Pruebas finales. |
| FASE 7 | Documentación y Presentación |
| Tarea 43 | Crear el repositorio del equipo. |
| Tarea 44 | Crear estructura del documento en LaTeX |
| Tarea 45 | Documentación del proyecto. |

| | |
|----------|--|
| Tarea 46 | Diseñar y crear presentación formal. |
| Tarea 47 | Recolectar capturas funcionales del software. |
| Tarea 48 | Subir todos los scripts con nombres correctos. |
| Tarea 49 | Subir el documento final. |
| Tarea 50 | Subir los PDFs del MER y MR. |
| Tarea 51 | Subir dashboard y presentación. |
| Tarea 52 | Revisión Final. |

| Persona | Nombre |
|----------------|--------------------------------------|
| Persona 1 | Buendía López Sebastián |
| Persona 2 | Velarde Valencia Josue |
| Persona 3 | Hernández Pérez Mariana Daniela |
| Persona 4 | Velazquez Cortes Cristina del Carmen |
| Persona 5 | Vences Santillán Carlos Eduardo |
| Todo el equipo | |

2.2. Plan de actividades

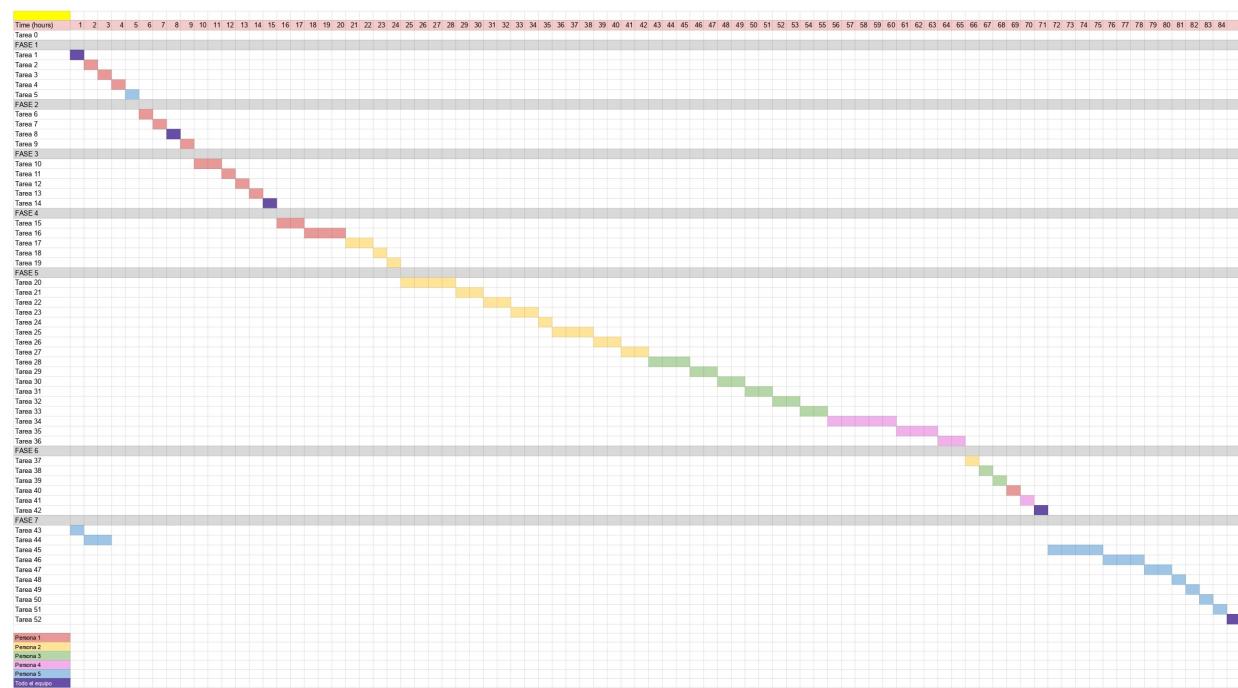


Figura 1: Cronograma utilizado para el proyecto

Para ver la versión detallada de click aquí.

3. Diseño

3.1. Modelo Entidad–Relación

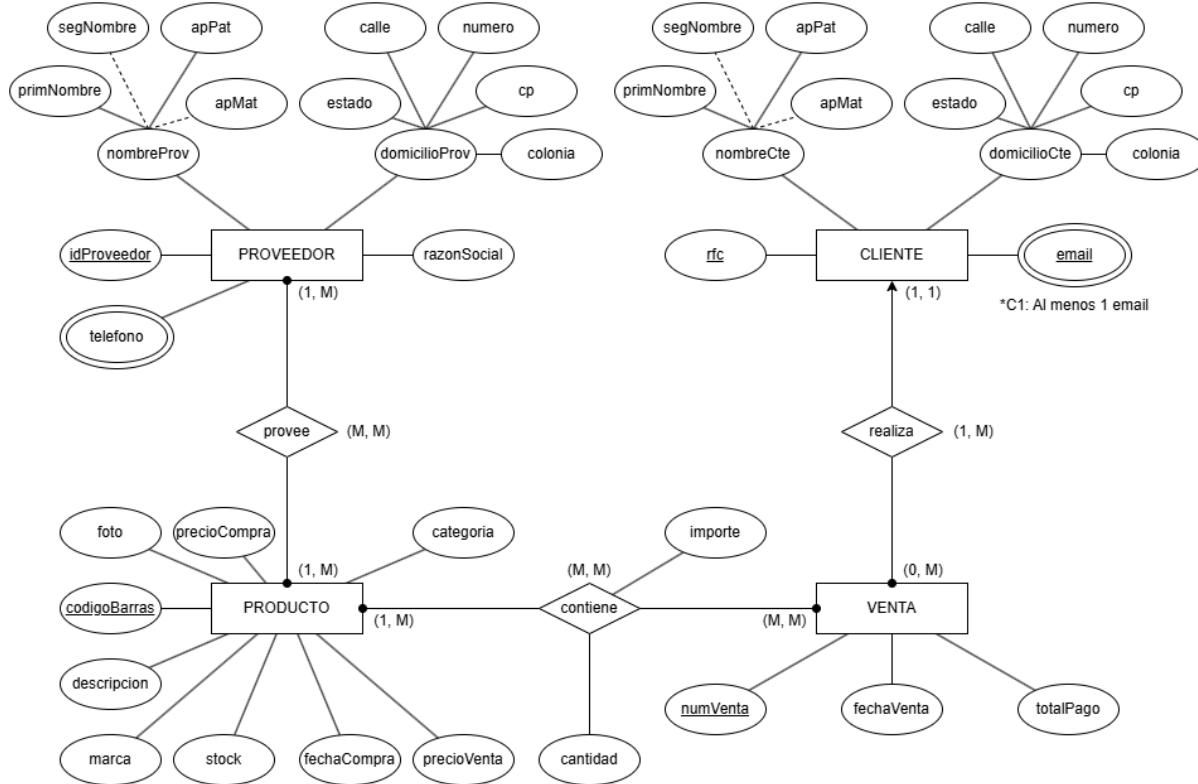


Figura 2: Modelo Entidad-Relación

El Modelo Entidad–Relación detalla los elementos clave y sus interconexiones en la operación de la papelería. La entidad PROVEEDOR, que almacena la información de cada suministrador, se relaciona con PRODUCTO a través de la relación provee. Esta relación refleja que un único proveedor puede abastecer una variedad de productos.

La entidad central del inventario es PRODUCTO, que almacena información esencial como el código de barras, precios, fechas, categoría, marca, descripción, foto y stock. La participación de un producto en varias ventas se gestiona mediante la relación contiene (M,M) entre PRODUCTO y VENTA, que además registra la cantidad vendida.

La entidad CLIENTE registra los datos personales y fiscales del cliente, incluyendo un email obligatorio. Cada cliente puede tener múltiples ventas asociadas, lo que se modela con la relación realiza (1,M) hacia la entidad VENTA. Esta registra detalles como el número de venta, la fecha de transacción y el monto total pagado.

3.2. Modelo Relacional

```
PROOVEDOR: {
    idProovedor int      PK,
    primNombre  varchar(50),
    segNombre   varchar(50),
    apPat       varchar(50),
    apMat       varchar(50) N,
    estado      varchar(50),
    colonia     varchar(50),
    calle        varchar(50),
    numero      int,
    cp          int,
    razonSocial varchar(250) }

TELEFONO_PROOVEDOR: {
    (idProovedor int      FK,
    telefono    int      FK) PK }

PRODUCTO: {
    codigoBarras  varchar(50) PK,
    categoria     varchar(50),
    descripcion  varchar(150),
    marca        varchar(50),
    foto         varchar(255),
    stock        int,
    fechaCompra  date,
    precioCompra float,
    precioVenta float}

PROVEE_PRODUCTO: {
    (idProovedor int      FK,
    codigoBarras varchar(50) FK) PK }

VENTA: {
    numVenta    int      PK,
    rfc         varchar(13) FK,
    fechaVenta  date,
    totalPago   float }
```

```

CONTIENE_PRODUCTO: {
    (codigoBarras  varchar(50) FK,
     numVenta    int      FK) PK,
    cantidad    int,
    importe float }

CLIENTE: {
    rfc      varchar(13) PK,
    primNombre  varchar(50),
    segNombre  varchar(50),
    apPat      varchar(50),
    apMat      varchar(50) N,
    estado     varchar(50),
    colonia    varchar(50),
    calle      varchar(50),
    numero    int,
    cp        int, }

EMAIL_CLIENTE: {
    (rfc      varchar(13) FK,
     email    varchar(50) FK) PK }

```

Figura 3: Modelo Relacional

3.3. Normalización

Primera Forma Normal (1FN)

El modelo cumple con 1FN porque todos los atributos son atómicos y no existen grupos repetitivos dentro de una misma tabla. Los atributos compuestos, como el nombre y el domicilio, fueron desglosados en sus componentes (`primNombre`, `segNombre`, `apPat`, `apMat`, `estado`, `colonia`, `calle`, `número`, `cp`).

De igual forma, los atributos multivaluados, como los teléfonos de proveedor o los correos electrónicos de cliente, fueron separados en tablas propias, evitando almacenar más de un valor en un solo campo.

De forma resumida, se cumple la primera forma normal ya que no existen atributos multivaluados ni campos de repetición.

Segunda Forma Normal (2FN)

La 2FN aplica únicamente a tablas con **llaves primarias compuestas**, ya que requiere que ningún atributo no clave dependa parcialmente de solo una parte de la llave.

En este modelo, las tablas con llaves compuestas son:

- **TELEFONO_PROVEEDOR**: {idProveedor, telefono}
- **PROVEE_PRODUCTO**: {idProveedor, codigoBarras}
- **CONTIENE_PRODUCTO**: {codigoBarras, numVenta}
- **EMAIL_CLIENTE**: {rfc, email}

En todas ellas, los atributos no clave dependen completamente de la llave compuesta, lo que significa que no existen dependencias parciales y que cada atributo está correctamente asociado a su llave correspondiente.

Tercera Forma Normal (3FN)

El modelo cumple con 3FN, ya que no existen dependencias transitivas dentro de las tablas. Ningún atributo no clave depende de otro atributo no clave.

- En **PRODUCTO**, atributos como `categoría`, `marca`, `descripción`, `precioCompra`, `precioVenta`, etc., dependen únicamente de la clave primaria `codigoBarras`.
- En **CLIENTE**, los atributos de domicilio y nombre dependen del `rfc` y no de otros atributos entre ellos.
- En **VENTA**, `fechaVenta` y `totalPago` dependen únicamente de `numVenta`.

4. Implementación

4.1. Creación de tablas en SQL

```
● ● ●

-- 1. Tabla PROOVEDOR
CREATE TABLE PROOVEDOR (
    idProovedor SERIAL PRIMARY KEY,
    primNombre VARCHAR(50) NOT NULL,
    segNombre VARCHAR(50) NULL,
    apPat VARCHAR(50) NOT NULL,
    apMat VARCHAR(50) NULL,
    estado VARCHAR(50) NOT NULL,
    colonia VARCHAR(50) NOT NULL,
    calle VARCHAR(50) NOT NULL,
    numero INT NOT NULL,
    cp INT NOT NULL,
    razonSocial VARCHAR(250)
);

-- 2. Tabla TELEFONO_PROOVEDOR
CREATE TABLE TELEFONO_PROOVEDOR (
    idProovedor INT,
    telefono VARCHAR(20),
    PRIMARY KEY (idProovedor, telefono),
    CONSTRAINT fk_tel_prov FOREIGN KEY (idProovedor)
        REFERENCES PROOVEDOR(idProovedor) ON DELETE CASCADE
);

-- 3. Tabla PRODUCTO
CREATE TABLE PRODUCTO (
    codigoBarras VARCHAR(50) PRIMARY KEY,
    categoria VARCHAR(50) NOT NULL,
    descripcion VARCHAR(150),
    marca VARCHAR(50) NOT NULL,
    foto VARCHAR(255),
    stock INT DEFAULT 0,
    fechaCompra DATE NOT NULL,
    precioCompra DECIMAL(10, 2) NOT NULL,
    precioVenta DECIMAL(10, 2) NOT NULL
);

-- 4. Tabla PROVEE_PRODUCTO (Relación entre Proveedor y Producto)
CREATE TABLE PROVEE_PRODUCTO (
    idProovedor INT,
    codigoBarras VARCHAR(50),
    PRIMARY KEY (idProovedor, codigoBarras),
    CONSTRAINT fk_provee_prov FOREIGN KEY (idProovedor)
        REFERENCES PROOVEDOR(idProovedor) ON DELETE RESTRICT,
    CONSTRAINT fk_provee_prod FOREIGN KEY (codigoBarras)
        REFERENCES PRODUCTO(codigoBarras) ON DELETE RESTRICT
);
```

```

-- 5. Tabla CLIENTE
CREATE TABLE CLIENTE (
    rfc VARCHAR(13) PRIMARY KEY,
    primNombre VARCHAR(50) NOT NULL,
    segNombre VARCHAR(50) NULL,
    apPat VARCHAR(50) NOT NULL,
    apMat VARCHAR(50) NULL,
    estado VARCHAR(50) NOT NULL,
    colonia VARCHAR(50) NOT NULL,
    calle VARCHAR(50) NOT NULL,
    numero INT NOT NULL,
    cp INT NOT NULL
);

-- 6. Tabla EMAIL_CLIENTE
CREATE TABLE EMAIL_CLIENTE (
    rfc VARCHAR(13),
    email VARCHAR(50),
    PRIMARY KEY (rfc, email),
    CONSTRAINT fk_email_cte FOREIGN KEY (rfc)
        REFERENCES CLIENTE(rfc) ON DELETE CASCADE
);

-- 7. Tabla VENTA
CREATE TABLE VENTA (
    numVenta SERIAL PRIMARY KEY,
    rfc VARCHAR(13) NOT NULL,
    fechaVenta DATE DEFAULT CURRENT_DATE,
    totalPago DECIMAL(10, 2) NOT NULL,
    CONSTRAINT fk_venta_cte FOREIGN KEY (rfc)
        REFERENCES CLIENTE(rfc) ON DELETE RESTRICT
);

-- 8. Tabla CONTIENE_PRODUCTO (Relación entre Venta y Producto)
CREATE TABLE CONTIENE_PRODUCTO (
    codigoBarras VARCHAR(50),
    numVenta INT,
    cantidad INT NOT NULL,
    importe DECIMAL(10, 2) NOT NULL,
    PRIMARY KEY (codigoBarras, numVenta),
    CONSTRAINT fk_contiene_prod FOREIGN KEY (codigoBarras)
        REFERENCES PRODUCTO(codigoBarras) ON DELETE RESTRICT,
    CONSTRAINT fk_contiene_venta FOREIGN KEY (numVenta)
        REFERENCES VENTA(numVenta) ON DELETE CASCADE
);

```

Figura 4: Creando las tablas

Durante la creación de las tablas en PostgreSQL se definieron distintos tipos de constraints con el objetivo de garantizar la integridad, consistencia y confiabilidad de los datos en todo el sistema físico. Las primary keys permiten identificar de manera única cada registro y evitar duplicidades en entidades importantes como clientes, proveedores, productos y ventas. Las foreign keys aseguran la correcta relación entre tablas (por ejemplo, vinculando ventas con clientes o productos con proveedores), lo cual evita la existencia de datos huérfanos y manteniendo la coherencia entre las entidades del modelo.

Las restricciones NOT NULL se aplicaron a todos los atributos que representan información esencial, garantizando que no existan registros incompletos. Asimismo, se emplearon

valores DEFAULT en campos como fechas e inventarios iniciales para asegurar un comportamiento consistente incluso cuando el usuario no proporciona todos los datos manualmente.

Por otro lado, las decisiones sobre las políticas de eliminación (ON DELETE CASCADE o RESTRICT) se tomaron debido a la lógica del negocio: ciertas relaciones requieren eliminar datos dependientes automáticamente, mientras que otras deben proteger información histórica, como los detalles de ventas asociadas a productos ya existentes.

4.2. Inserción de datos



```
-- 1. PROOVEDORES
INSERT INTO PROOVEDOR (idProovedor, primNombre, apPat, estado, colonia, calle,
numero, cp, razonSocial)
VALUES
(1, 'Papelería', 'Luna', 'CDMX', 'Centro', 'Av. Reforma', 122, 01010, 'Papelería
Luna S.A.'),
(2, 'Distribuidora', 'Sol', 'CDMX', 'Roma', 'Durango', 77, 02220, 'Distribuidora
Sol S.A.');

-- Actualizamos la secuencia para que el siguiente insert sea el 3
SELECT setval('proovedor_idproovedor_seq', 2, true);
```

Figura 5: Insert

```

-- 2. TELÉFONO PROOVEDEDOR
INSERT INTO TELEFONO_PROOVEDEDOR (idProovedor, telefono)
VALUES
(1, '5511223344'),
(1, '5544556677'),
(2, '5588997766');

-- 3. PRODUCTOS
INSERT INTO PRODUCTO (codigoBarras, descripcion, marca, categoria, foto,
precioCompra, precioVenta, fechaCompra, stock)
VALUES
('750100000001', 'Pluma negra', 'Bic', 'Papeleria', NULL, 4.00, 8.00, '2025-01-
20', 50),
('750200000002', 'Cuaderno profesional', 'Scribe', 'Papeleria', NULL, 20.00,
35.00, '2025-01-15', 30),
('750300000003', 'Taza de regalo', 'Totto', 'Regalos', NULL, 40.00, 80.00, '2025-
01-10', 10);

```

Figura 6: Insert

```

-- 4. PROVEE_PRODUCTO
INSERT INTO PROVEE_PRODUCTO (idProovedor, codigoBarras)
VALUES
(1, '750100000001'), -- Papelería Luna Plumas
(1, '750200000002'), -- Papelería Luna Cuadernos
(2, '750300000003'); -- Dist. Sol provee Tazas

-- 5. CLIENTES
-- El RFC es la PK, así que lo usamos para identificar al cliente
INSERT INTO CLIENTE (rfc, primNombre, apPat, estado, colonia, calle, numero, cp)
VALUES
('PEJJ900101ABC', 'Juan', 'Pérez', 'CDMX', 'Polanco', 'Homero', 201, 01234),
('LOPM920202XYZ', 'María', 'López', 'CDMX', 'Condesa', 'Amsterdam', 12, 06700);

```

Figura 7: Insert

```

-- 6. EMAIL_CLIENTE
-- Relacionamos usando el RFC
INSERT INTO EMAIL_CLIENTE (rfc, email)
VALUES
('PEJJ900101ABC', 'juanp@gmail.com'),
('LOPM920202XYZ', 'marialopez@hotmail.com');

-- 7. VENTA
-- Asumo totalPago = 0.
INSERT INTO VENTA (numVenta, rfc, fechaVenta, totalPago)
VALUES
(1, 'PEJJ900101ABC', '2025-01-25', 0),
(2, 'LOPM920202XYZ', '2025-01-25', 0);

-- Actualizamos la secuencia del SERIAL de ventas
SELECT setval('venta_numventa_seq', 2, true);

```

Figura 8: Insert

```

-- 8. CONTIENE_PRODUCTO (Detalle de Venta)
-- Relacionamos la Venta #1 y #2 con los códigos de barras
INSERT INTO CONTIENE_PRODUCTO (numVenta, codigoBarras, cantidad, importe)
VALUES
(1, '750100000001', 2, 16.00), -- Venta 1: 2 Plumas
(1, '750200000002', 1, 35.00), -- Venta 1: 1 Cuaderno
(2, '750300000003', 1, 80.00); -- Venta 2: 1 Taza
CREATE INDEX idx_producto_stock ON producto(stock);

```

Figura 9: Insert

4.3. Índex

Como parte de los requerimientos del proyecto se solicitó la creación de al menos un índice dentro de la base de datos. Por ello, se decidió crear el siguiente índice:



```
CREATE INDEX idx_producto_stock ON producto(stock);
```

Figura 10: Creando el índice

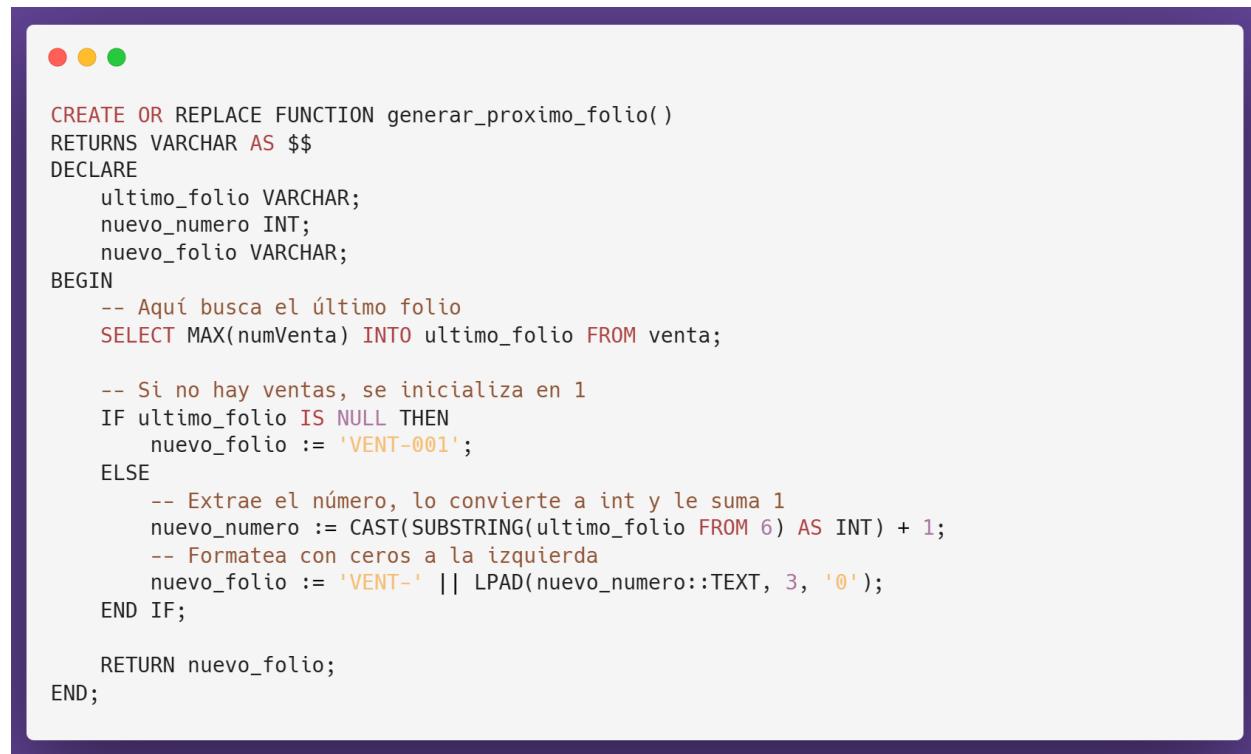
El índice se aplicó sobre el atributo *stock* de la tabla **producto** debido a que varias reglas de negocio requieren consultar con frecuencia los niveles de inventario. Entre las operaciones que dependen de este atributo se encuentran:

- Identificar productos con menos de 3 unidades disponibles.
- Verificar si un producto cuenta con suficiente stock antes de completar una venta.
- Consultar artículos agotados o que se agotarán pronto.

Dado que estas consultas utilizan condiciones como `stock <3` o `stock = 0`, un índice tipo *B-Tree* es lo más óptimo, ya que optimiza las búsquedas basadas en comparaciones y rangos. Además, el atributo *stock* es consultado y modificado de manera recurrente, por lo que el índice reduce de manera eficiente el tiempo de respuesta de las operaciones relacionadas con el inventario de estos productos.

4.4. Funciones

Funcion para generar proximo folio de venta en formato VENT-XXX:



```
CREATE OR REPLACE FUNCTION generar_proximo_folio()
RETURNS VARCHAR AS $$
DECLARE
    ultimo_folio VARCHAR;
    nuevo_numero INT;
    nuevo_folio VARCHAR;
BEGIN
    -- Aquí busca el último folio
    SELECT MAX(numVenta) INTO ultimo_folio FROM venta;

    -- Si no hay ventas, se inicializa en 1
    IF ultimo_folio IS NULL THEN
        nuevo_folio := 'VENT-001';
    ELSE
        -- Extrae el número, lo convierte a int y le suma 1
        nuevo_numero := CAST(SUBSTRING(ultimo_folio FROM 6) AS INT) + 1;
        -- Formatea con ceros a la izquierda
        nuevo_folio := 'VENT-' || LPAD(nuevo_numero::TEXT, 3, '0');
    END IF;

    RETURN nuevo_folio;
END;
```

Figura 11: Función “generar_proximo_folio()”

Esta función tiene como objetivo automatizar la creación del número de venta siguiendo el formato VENT-XXX, donde la parte numérica siempre incrementa de manera consecutiva y mantiene ceros a la izquierda.

Para hacer esto, la función primero consulta cuál es el último folio existente en la tabla venta. Si la base de datos todavía no tiene ventas registradas, significa que no existe un folio previo, por lo que la función devuelve directamente VENT-001 como si fuera el inicio.

En caso de que sí existan ventas previas, la función toma el folio más alto, extrae únicamente la parte numérica y la convierte a un entero para poder sumarle uno. Despues, formatea este nuevo número para que (siempre) tenga tres dígitos, agregando ceros a la izquierda cuando sea necesario. Para terminar, construye el nuevo folio uniendo el prefijo ”VENT-“ con el número resultante, por lo que obtenemos valores como VENT-006 o VENT-127 dependiendo de la secuencia.

Función para poder calcular las utilidades por el código de barras:

```
CREATE OR REPLACE FUNCTION calcular_utilidad(p_codigoBarras VARCHAR)
RETURNS NUMERIC AS $$

DECLARE
    v_costo NUMERIC;
    v_precio NUMERIC;
BEGIN
    SELECT precioCompra, precioVenta INTO v_costo, v_precio
    FROM producto
    WHERE codigoBarras = p_codigoBarras;

    IF v_costo IS NULL THEN
        RETURN 0;
    END IF;

    RETURN v_precio - v_costo;
END;
```

Figura 12: Función “calcular_utilidad()”

Esta función tiene como propósito obtener la ganancia unitaria de un producto a partir de su código de barras.

La función recibe como parámetro el código de barras del producto y busca en la tabla *producto* los valores correspondientes a su precio de compra y precio de venta. Estos valores se guardan en dos variables (*v_costo* y *v_precio*), las cuales servirán para hacer el cálculo.

Una vez obtenidos los datos, la función verifica si el producto existe o si alguno de los valores es nulo. En caso de no encontrar un registro válido, devuelve un valor de **0**.

Por último, si los valores son correctos, la función simplemente retorna la diferencia entre el precio de venta y el precio de compra, lo que representa la utilidad unitaria del producto.

Función para poder obtener los productos cuyo stock sea crítico o sea menor a tres:

```
CREATE OR REPLACE FUNCTION obtener_productos_bajo_stock()
RETURNS TABLE (
    codigo VARCHAR,
    nombre_prod VARCHAR,
    stock_actual INT
) AS $$

BEGIN
    RETURN QUERY
    SELECT codigoBarras, descripcion, stock
    FROM producto
    WHERE stock < 3;
END;
```

Figura 13: Función “obtener_productos_bajo_stock()”

Finalmente, diseñamos esta función para identificar de manera rápida los productos cuyo inventario se encuentra en un nivel crítico.

Este método nos retorna una tabla. La función no recibe parámetros, ya que su propósito es evaluar el inventario completo. Lo que hace es ejecutar una consulta sobre la tabla producto, seleccionando únicamente aquellos registros donde el valor del campo stock es menor a tres unidades. De cada producto encontrado, la función devuelve su código de barras, la descripción y el stock actual.

4.5. Triggers

Primer trigger:

Es el que gestiona los detalles de la venta antes de insertarlos, se ocupa de calcular el importe automatico (cantidad* precio), validar si hay suficiente stock (si no es el caso aborta), disminuye el stock y por ultimo alerta si nuestro stock es menor a tres.

```
CREATE OR REPLACE FUNCTION tf_gestionar_detalle_venta()
RETURNS TRIGGER AS $$

DECLARE
    v_stock_actual INT;
    v_precio_venta NUMERIC;
    v_descripcion VARCHAR;

BEGIN
    -- Obteniendo datos actuales del producto
    SELECT stock, precioVenta, descripcion INTO v_stock_actual, v_precio_venta, v_descripcion
    FROM producto
    WHERE codigoBarras = NEW.codigoBarras;

    -- Validando que el producto exista
    IF NOT FOUND THEN
        RAISE EXCEPTION 'El producto % no existe.', NEW.codigoBarras;
    END IF;

    -- Validar que haya Stock Suficiente (Aborta si llega a 0 o si es insuficiente)
    IF v_stock_actual < NEW.cantidad THEN
        RAISE EXCEPTION 'Stock insuficiente para el producto %. Disponible: %, Solicitado: %',
                         v_descripcion, v_stock_actual, NEW.cantidad;
    END IF;

    -- Cálculo automático del importe
    NEW.importe := NEW.cantidad * v_precio_venta;

    -- Disminuyendo el Stock
    UPDATE producto
    SET stock = stock - NEW.cantidad
    WHERE codigoBarras = NEW.codigoBarras;

    -- Verifica el nivel de stock para alerta (Post-venta)
    -- Se calcula el stock resultante
    IF (v_stock_actual - NEW.cantidad) < 3 THEN
        RAISE NOTICE 'ALERTA DE INVENTARIO: El producto "%" (%) tiene stock bajo (% unidades restantes).',
                      v_descripcion, NEW.codigoBarras, (v_stock_actual - NEW.cantidad);
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_gestion_detalle_venta
BEFORE INSERT ON contiene
FOR EACH ROW
EXECUTE FUNCTION tf_gestionar_detalle_venta();
```

Figura 14: Trigger “tf_gestionar_detalle_venta()”

El trigger `tg_gestion_detalle_venta`, junto con la función `tf_gestionar_detalle_venta()`, nos aseguran que cada línea de venta sea consistente con el inventario actual y que se calculen correctamente los importes, lo cual evita errores manuales y ventas que no puedan

completarse por falta de stock.

En primer lugar, el trigger se ejecuta antes de insertar un nuevo registro en la tabla de detalle de venta (**contiene**). Cuando esto ocurre, la función obtiene el stock actual, el precio de venta y la descripción del producto a partir de la tabla *producto*, usando el código de barras que viene en el nuevo registro. Si el producto no existe, se genera una excepción y la operación se detiene.

Después, el trigger valida que haya stock suficiente para la cantidad solicitada. Si el inventario disponible es menor que la cantidad que se intenta vender, se lanza un error indicando el stock disponible y la cantidad solicitada, y la venta no se completa.

Si el stock es suficiente, la función calcula automáticamente el importe de la línea de detalle (cantidad por precio de venta) y se lo asigna al campo correspondiente.

Como último paso, se actualiza el stock del producto restando la cantidad vendida. Finalmente, el trigger verifica si el stock resultante queda por debajo de tres unidades. En ese caso, se lanza un mensaje de aviso (**RAISE NOTICE**), que funciona como una **alerta de inventario bajo**; esta nos informa qué producto está en nivel crítico y cuántas unidades quedan.

Segundo trigger:

Actualiza los totales, despues de insertar/borrar o actualizar. Calcula el total del pago en la tabla VENTA sumando todos los importes de la misma.



```
CREATE OR REPLACE FUNCTION tf_actualizar_total_venta( )
RETURNS TRIGGER AS $$

BEGIN
    -- Actualizar el total de la venta afectada
    -- Usamos un TG_OP para saber el tipo de operación que usamos, si usamos NEW
    -- (Insert/Update) o OLD (Delete)

    IF (TG_OP = 'DELETE') THEN
        UPDATE venta
        SET totalPago = (
            SELECT COALESCE(SUM(importe), 0)
            FROM contiene
            WHERE numVenta = OLD.numVenta
        )
        WHERE numVenta = OLD.numVenta;
        RETURN OLD;
    ELSE
        UPDATE venta
        SET totalPago = (
            SELECT COALESCE(SUM(importe), 0)
            FROM contiene
            WHERE numVenta = NEW.numVenta
        )
        WHERE numVenta = NEW.numVenta;
        RETURN NEW;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_actualizar_total_venta
AFTER INSERT OR UPDATE OR DELETE ON contiene
FOR EACH ROW
EXECUTE FUNCTION tf_actualizar_total_venta();
```

Figura 15: Trigger “tf_actualizar_total_venta()”

El trigger `tg_actualizar_total_venta`, junto con la función `tf_actualizar_total_venta()`, mantiene siempre actualizado el total a pagar de cada venta. Debido a que el total de una venta depende del conjunto de productos que contiene, tenemos la necesidad de recalcularlo automáticamente cada vez que se agregan, modifican o se eliminan renglones del detalle de venta.

Este trigger mantiene siempre actualizado el total a pagar de cada venta. La función identifica el tipo de operación realizada mediante la variable especial `TG_OP`, que indica si se trata de un *INSERT*, *UPDATE* o *DELETE*. Dependiendo del caso, la función toma el número de venta afectado ya sea desde `NEW` (para inserciones y actualizaciones) o desde `OLD` (para eliminaciones).

Con ese número de venta, la función ejecuta una consulta que suma todos los importes de la tabla `contiene` asociados a esa venta. Esta suma representa el total actualizado, y se asigna directamente al campo `totalPago` en la tabla `venta`.

4.6. Pruebas funciones

Prueba 1: En la prueba 1 se verifica la función de utilidad y la generación de folio, se verificamos cuanto ganamos por cada pluma negra usando el generador de utilidad junto con el código y llamando al trigger el generador de folio.

| | <code>proximo_folio_sugerido</code>  |
|---|---|
| | <code>character varying</code> |
| 1 | <code>VENT-007</code> |

Figura 16: Output prueba 1

Prueba 2: Se va a crear una nueva venta con el trigger de Actualización de total, insertamos una cabecera de venta y se manda un mensaje de confirmación junto con el ID de la nueva venta.

```
NOTICE: Venta creada con ID: 7
DO
```

Figura 17: Output prueba 2

Prueba 3: Insertar detalles de triggers de stock y cálculo, en donde se van a vender 10 plumas poniendo dando desde el script su código de barra y la verificación inmediata.

| | <code>descripcion</code>  <code>character varying (150)</code> | <code>stock</code>  <code>integer</code> |
|---|--|--|
| 1 | <code>Pluma negra</code> | 10 |

Figura 18: Output prueba 3

Prueba 4: Alerta de stock bajo, se mete el código de barras de la taza de regalo, y se vera la pestaña de stock insuficiente y viendo cuantos hay disponibles y cuanto se solicitan.

```
ERROR: Stock insuficiente para el producto Taza de regalo. Disponible: 2, Solicitud: 8
```

Figura 19: Output prueba 4

Prueba 5: Intento de venta sin stock. Para este caso trataremos de vender mas stock de lo que hay.

```
NOTICE: ¡ÉXITO! El sistema bloqueó la venta incorrecta: Stock insuficiente para el producto Taza de regalo. Disponible: 2, Solicitud: 50
```

Figura 20: Output prueba 5

Prueba 6: Verificar el total actual con el trigger de actualizar total.

| | numventa [PK] integer | totalpago numeric (10,2) |
|---|--------------------------|-----------------------------|
| 1 | 7 | 80.00 |

Figura 21: Output prueba 6

Prueba 7: Borrar las plumas de la venta actual y se pone el total que deberia de haber bajado.

| | numventa [PK] integer | nuevo_total_tras_borrado |
|---|--------------------------|--------------------------|
| 1 | 7 | 0.00 |

Figura 22: Output prueba 7

Prueba 8: Reporte de Stock bajo.

| | codigo character varying | nombre_prod character varying | stock_actual integer |
|---|-----------------------------|----------------------------------|-------------------------|
| 1 | 750300000003 | Taza de regalo | 2 |

```
NOTICE: --- REPORTE FINAL: Productos con bajo stock ---
```

Figura 23: Output prueba 8

4.7. Vistas

Vista factura:



```
CREATE OR REPLACE VIEW V_FACTURA_DETALLADA AS
SELECT
    v.numVenta AS "Folio Factura",
    v.fechaVenta AS "Fecha Emisión",
    -- Datos del Cliente
    c.rfc AS "RFC Cliente",
    CONCAT(c.primNombre, ' ', c.apPat, ' ', COALESCE(c.apMat, '')) AS "Nombre Cliente",
    CONCAT(c.calle, '#', c.numero, ' Col. ', c.colonia, ' ', c.estado) AS "Dirección Cliente",
    -- Detalle del Producto
    cp.codigoBarras AS "Código Producto",
    p.descripcion AS "Descripción",
    p.marca AS "Marca",
    cp.cantidad AS "Cantidad",
    p.precioVenta AS "Precio Unitario",
    cp.importe AS "Subtotal Línea",
    -- Total General de la Venta (repetido en cada línea)
    v.totalPago AS "Total a Pagar"
FROM VENTA v
JOIN CLIENTE c ON v.rfc = c.rfc
JOIN CONTIENE_PRODUCTO cp ON v.numVenta = cp.numVenta
JOIN PRODUCTO p ON cp.codigoBarras = p.codigoBarras
ORDER BY v.numVenta DESC;
```

Figura 24: Vista “V_FACTURA_DETALLADA()”

Esta vista integra datos provenientes de varias tablas: la información general de la venta, los datos del cliente y el detalle de cada producto incluido en la transacción. De esta manera, facilita la generación de comprobantes o reportes sin necesidad de realizar múltiples consultas o combinaciones manuales.

Para lograrlo, la vista realiza algunas instrucciones JOIN entre las tablas VENTA, CLIENTE, CONTIENE_PRODUCTO y PRODUCTO. De VENTA se obtiene el número de venta, la fecha de emisión y el total a pagar; de CLIENTE se recupera el RFC, el nombre completo construido mediante concatenación y la dirección del cliente; de CONTIENE_PRODUCTO se obtiene la cantidad y el subtotal de cada línea; mientras que de PRODUCTO se extraen atributos como la descripción, la marca y el precio unitario.

Como resultado, se genera una tabla en donde cada fila representa una línea de detalle de la factura, es decir, un producto vendido dentro de una venta. La consulta se ordena de

forma descendente por número de venta para mostrar primero las transacciones más recientes.

Reporte de Ventas por una fecha específica:

```
CREATE OR REPLACE FUNCTION reporte_ventas_por_fecha(p_fecha DATE)
RETURNS TABLE (
    folio INT,
    rfc_cliente VARCHAR,
    total_venta DECIMAL(10,2)
) AS $$

BEGIN
    RETURN QUERY
    SELECT numVenta, rfc, totalPago
    FROM VENTA
    WHERE fechaVenta = p_fecha;
END;
$$ LANGUAGE plpgsql;
```

Figura 25: Función “`reporte_ventas_por_fecha()`”

Hicimos esta función para generar un reporte de ventas correspondientes a una fecha específica. Como está parametrizada, la función recibe como entrada una fecha y devuelve únicamente la información relevante de ese día.

Toma el parámetro `p_fecha` y realiza una selección sobre la tabla `VENTA`, lo cual filtra las filas cuya fecha de venta coincide exactamente con el valor. La salida de la función es en forma de tabla, incluyendo el folio de la venta, el RFC del cliente y el total pagado.

Reporte de Ventas por rango de fechas:

```
CREATE OR REPLACE FUNCTION reporte_ventas_por_rango(p_fecha_inicio DATE,
p_fecha_fin DATE)
RETURNS TABLE (
    fecha DATE,
    folio INT,
    rfc_cliente VARCHAR,
    total_venta DECIMAL(10,2)
) AS $$

BEGIN
    RETURN QUERY
    SELECT fechaVenta, numVenta, rfc, totalPago
    FROM VENTA
    WHERE fechaVenta BETWEEN p_fecha_inicio AND p_fecha_fin
    ORDER BY fechaVenta ASC;
END;
$$ LANGUAGE plpgsql;
```

Figura 26: Función “reporte_ventas_por_rango()”

Esta función nos permite obtener un reporte de todas las ventas realizadas dentro de un intervalo de fechas.

El funcionamiento consiste en filtrar la tabla VENTA utilizando el operador BETWEEN, que selecciona todas las ventas cuya fecha se encuentra dentro del rango especificado por los parámetros. La función devuelve (para cada una de las ventas encontradas) la fecha, el folio, el RFC del cliente y el monto total de la transacción.

Ganancia Total del Periodo:

```
CREATE OR REPLACE FUNCTION calcular_ganancia_periodo(p_fecha_inicio DATE,
p_fecha_fin DATE)
RETURNS DECIMAL(10,2) AS $$

DECLARE
    ganancia_total DECIMAL(10,2);
BEGIN
    SELECT COALESCE(SUM((p.precioVenta - p.precioCompra) * cp.cantidad), 0)
    INTO ganancia_total
    FROM VENTA v
    JOIN CONTIENE_PRODUCTO cp ON v.numVenta = cp.numVenta
    JOIN PRODUCTO p ON cp.codigoBarras = p.codigoBarras
    WHERE v.fechaVenta BETWEEN p_fecha_inicio AND p_fecha_fin;

    RETURN ganancia_total;
END;
$$ LANGUAGE plpgsql;
```

Figura 27: Función “calcular_ganancia_periodo()”

La ganancia total en un rango de fechas determinado se calcula tomando en cuenta la utilidad unitaria de cada producto vendido (la diferencia entre su precio de venta y su precio de compra) multiplicada por la cantidad vendida en cada transacción.

Funciona uniendo las tablas `VENTA`, `CONTIENE_PRODUCTO` y `PRODUCTO` para relacionar cada venta con sus productos y sus precios correspondientes. Una vez hechas las uniones, la función filtra las ventas utilizando el rango de fechas como parámetros.

Para cada línea de detalle, se calcula la utilidad individual del producto y se multiplica por la cantidad vendida. Al final, se suman todas esas utilidades para obtener la ganancia total del periodo.

Se utiliza `COALESCE` para asegurar que, si no se encontraron ventas en ese rango, la función retorne **0** en lugar de un valor nulo.

4.8. Consultas requeridas

Consultas para Dashboard

Ingresos mensuales por artículo

```
CREATE OR REPLACE VIEW DASHBOARD_INGRESOS_MENSUALES_ARTICULO AS
SELECT
    TO_CHAR(v.fechaVenta, 'YYYY-MM') AS "Mes",
    p.descripcion AS "Artículo",
    SUM(cp.importe) AS "Ingreso Total"
FROM VENTA v
JOIN CONTIENE_PRODUCTO cp ON v.numVenta = cp.numVenta
JOIN PRODUCTO p ON cp.codigoBarras = p.codigoBarras
GROUP BY TO_CHAR(v.fechaVenta, 'YYYY-MM'), p.descripcion
ORDER BY "Mes" DESC, "Ingreso Total" DESC;
```

Figura 28: Función “DAHBOARD_INGRESOS_MENSUALES_ARTICULO()”

Esta consulta sirve para proporcionar un resumen claro del ingreso generado por cada producto a lo largo del tiempo. Su propósito principal es servir como la base para que el dashboard pueda analizar tendencias de venta por mes y por artículo.

Para hacer esta vista, se combinan las tablas VENTA, CONTIENE_PRODUCTO y PRODUCTO mediante uniones que permiten relacionar cada venta con los productos involucrados y sus respectivos subtotales.

La función TO_CHAR se utiliza para convertir la fecha de venta en un formato “Año-Mes”, lo cual permite agrupar todas las ventas del mismo periodo mensual sin importar el día específico en que ocurrieron. Después, se agrupan los registros por mes y por producto, y se calcula la suma total de los importes vendidos para cada combinación.

El resultado es una tabla organizada cronológicamente que muestra, para cada mes y cada artículo, el ingreso total generado.

Margen Bruto por Categoría

```
CREATE OR REPLACE VIEW DASHBOARD_MARGEN_POR_CATEGORIA AS
SELECT
    p.categoría AS "Categoría",
    -- Calculamos cuánto se vendió en total (Ingresos)
    SUM(cp.importe) AS "Ventas Totales ($)",
    -- Calculamos el margen bruto: (Venta - Compra) * cantidad
    SUM((p.precioVenta - p.precioCompra) * cp.cantidad) AS "Margen Bruto
(Ganancia)"
FROM CONTIENE_PRODUCTO cp
JOIN PRODUCTO p ON cp.codigoBarras = p.codigoBarras
-- JOIN VENTA v ON cp.numVenta = v.numVenta -- Opcional: si quisieras filtrar por
fecha
GROUP BY p.categoría
ORDER BY "Margen Bruto (Ganancia)" DESC;
```

Figura 29: Función “DAHBOARD_MARGEN_POR_CATEGORIA()”

Para tener una mejor vista sobre la rentabilidad del negocio, mostramos qué categorías de productos generan mayor margen bruto. Para construirla, se utilizan las tablas `CONTIENE_PRODUCTO` y `PRODUCTO`, las cuales permiten relacionar cada artículo vendido con su categoría, sus precios y la cantidad vendida.

Primero se calcula el total vendido por categoría mediante la suma de los importes asociados a cada línea de venta. Después, calculamos el margen bruto aplicando la fórmula (precio de venta – precio de compra) × cantidad, lo que representa la ganancia real. Al sumar este valor para todos los productos de la misma categoría, la vista muestra de manera clara qué grupos de artículos son más rentables para el negocio.

5. Aplicación

5.1. Dashboards

Para nuestro Dashboard utilizamos la aplicación Power BI Desktop. Para visualizar nuestros gráfico primero tendremos que conectar Power BI a nuestra base de datos para lo que necesitaremos: servidor, nombre de la base de datos, usuario y contraseña.

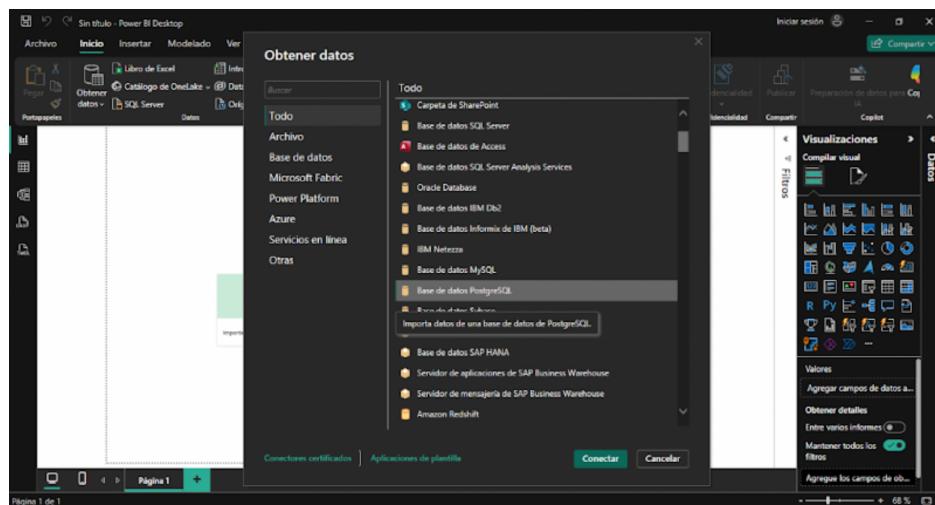
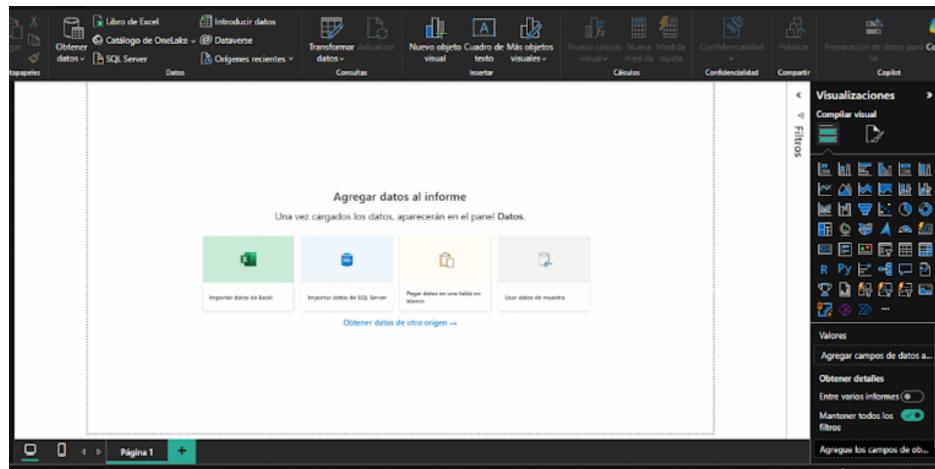


Figura 30: Conectando a la base de datos.

Una vez conectada con la base de datos, y con las consultas ya cargadas en la base, seleccionamos `public.dashboard_ingresos_mensuales_articulo`.

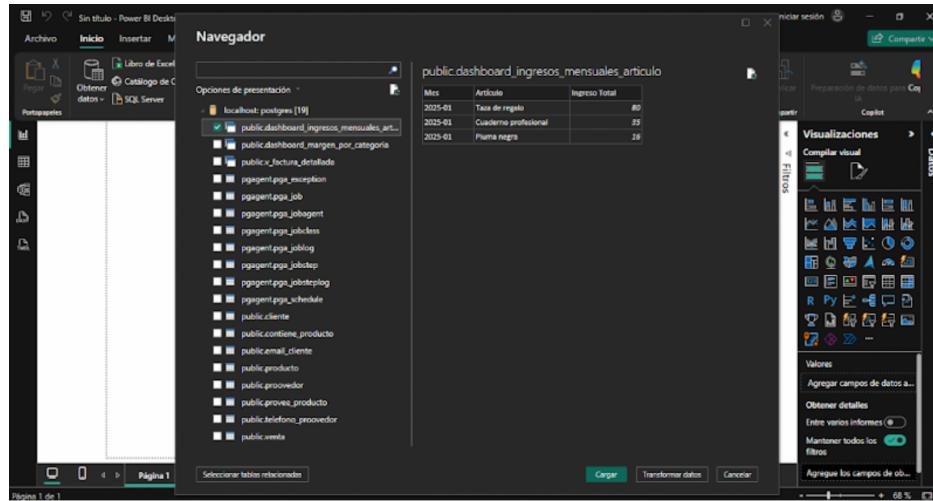


Figura 31: Usando la función

y “public.dashboard.margen_por_categoría” para crear nuestras dos tablas.

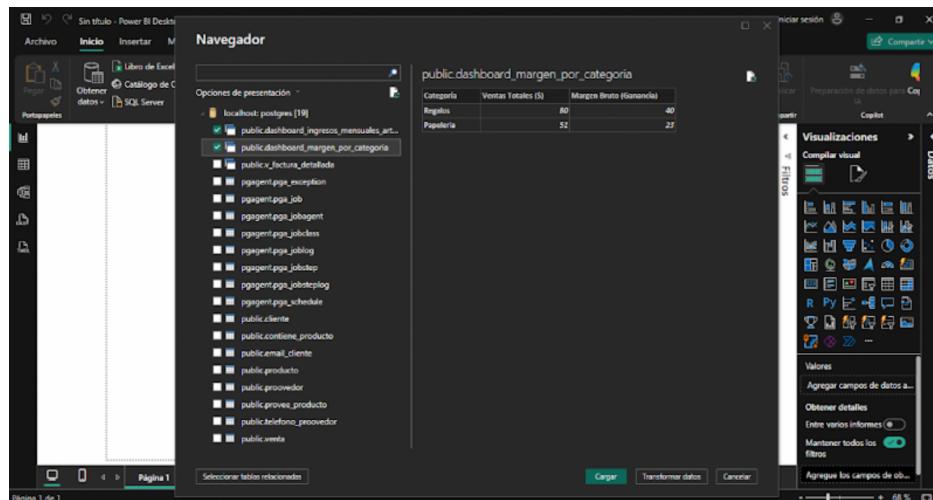


Figura 32: Usando la función

Una vez cargada nuestra base de datos y seleccionadas nuestras tablas ya podemos insertar el gráfico que más nos convenga según nuestro análisis

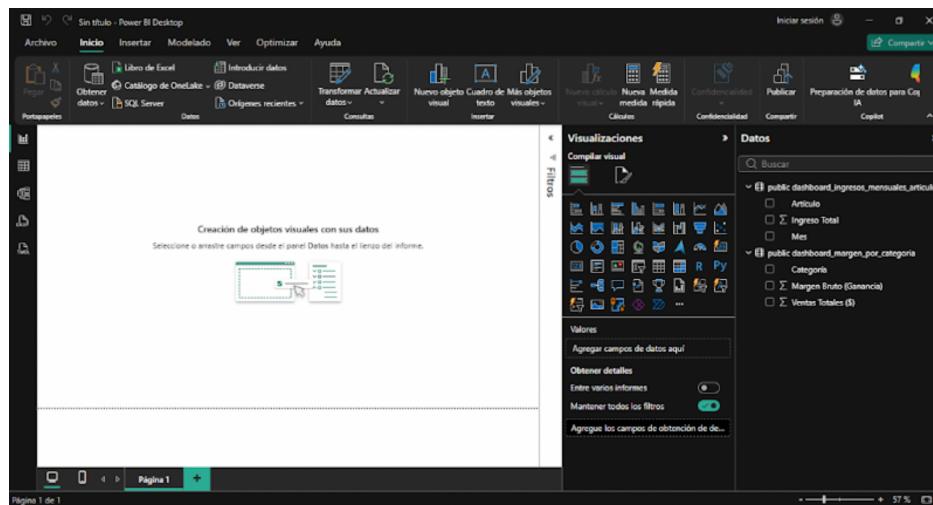


Figura 33: Insertar gráfico

1. Tabla Suma de ingreso total por mes y artículo: Nuestro gráfico toma la variable X como el mes (en este caso el primer periodo del 2025) mientras que nuestra Y representa la suma de los ingresos totales de cada artículo. Leyendo la gráfica podemos concluir que la mayor parte de ingresos en el primer mes del 2025 se obtuvieron gracias a las tazas de regalo.

2. Suma de margen bruto (ganancia) y suma de ventas totales por categoría : Nuestro gráfico toma los datos por categoría, suma de margen bruto y suma de las ventas totales. Establecimos nuestra X como categoría y nuestra Y como margen bruto y suma total. Gracias a nuestra gráfica concluimos que los regalos generaron más ganancia que los objetos de papelería, reflejando un margen de ventas mayor.

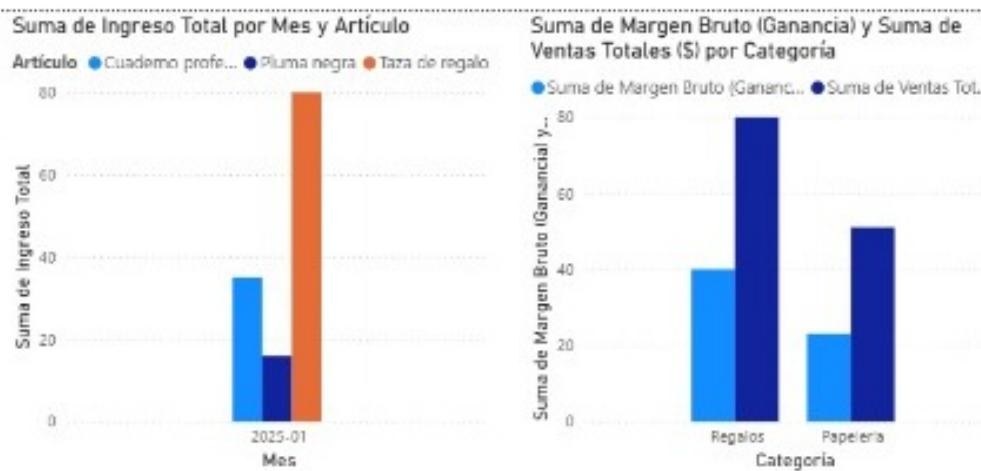


Figura 34: Generando las gráficas

6. Resultados

Como resultados de nuestras consultas y vistas, tenemos lo siguiente:

| | Folio Factura integer | Fecha Emisión date | RFC Cliente character varying (13) | Nombre Cliente text | Dirección Cliente text | Código Producto character varying (50) | Descripción character varying (|
|---|--------------------------|-----------------------|---------------------------------------|------------------------|---------------------------------|---|------------------------------------|
| 1 | 8 | 2025-12-06 | PEJJ900101ABC | Juan Pérez | Homero #201, Col. Polanco, CDMX | 750300000003 | Taza de regalo |
| 2 | 7 | 2025-12-06 | PEJJ900101ABC | Juan Pérez | Homero #201, Col. Polanco, CDMX | 750100000001 | Pluma negra |
| 3 | 6 | 2025-12-06 | PEJJ900101ABC | Juan Pérez | Homero #201, Col. Polanco, CDMX | 750100000001 | Pluma negra |
| 4 | 5 | 2025-12-06 | PEJJ900101ABC | Juan Pérez | Homero #201, Col. Polanco, CDMX | 750100000001 | Pluma negra |
| 5 | 4 | 2025-12-06 | PEJJ900101ABC | Juan Pérez | Homero #201, Col. Polanco, CDMX | 750100000001 | Pluma negra |
| 6 | 3 | 2025-12-06 | PEJJ900101ABC | Juan Pérez | Homero #201, Col. Polanco, CDMX | 750100000001 | Pluma negra |

| Marca character varying (50) | Cantidad integer | Precio Unitario numeric (10,2) | Subtotal Línea numeric (10,2) | Total a Pagar numeric (10,2) |
|---------------------------------|---------------------|-----------------------------------|----------------------------------|---------------------------------|
| Totto | 8 | 80.00 | 640.00 | 640.00 |
| Bic | 10 | 8.00 | 80.00 | 80.00 |
| Bic | 10 | 8.00 | 80.00 | 80.00 |
| Bic | 10 | 8.00 | 80.00 | 80.00 |
| Bic | 10 | 8.00 | 80.00 | 80.00 |
| Bic | 10 | 8.00 | 80.00 | 720.00 |

Figura 35: Factura

| | fecha date | folio integer | rfc_cliente character varying | total_venta numeric |
|---|---------------|------------------|----------------------------------|------------------------|
| 1 | 2025-12-06 | 3 | PEJJ900101ABC | 720.00 |
| 2 | 2025-12-06 | 4 | PEJJ900101ABC | 80.00 |
| 3 | 2025-12-06 | 5 | PEJJ900101ABC | 80.00 |
| 4 | 2025-12-06 | 6 | PEJJ900101ABC | 80.00 |
| 5 | 2025-12-06 | 7 | PEJJ900101ABC | 80.00 |
| 6 | 2025-12-06 | 8 | PEJJ900101ABC | 640.00 |

Figura 36: Fecha específica

| | |
|---|---|
| | ganancia_diciembre  |
| 1 | 840.00 |

Figura 37: Fecha del total del periodo

| | fecha date | folio integer  | rfc_cliente character varying  | total_venta numeric  |
|---|----------------------|---|---|---|
| 1 | 2025-12-06 | 3 | PEJJ900101ABC | 720.00 |
| 2 | 2025-12-06 | 4 | PEJJ900101ABC | 80.00 |
| 3 | 2025-12-06 | 5 | PEJJ900101ABC | 80.00 |
| 4 | 2025-12-06 | 6 | PEJJ900101ABC | 80.00 |
| 5 | 2025-12-06 | 7 | PEJJ900101ABC | 80.00 |
| 6 | 2025-12-06 | 8 | PEJJ900101ABC | 640.00 |

Figura 38: Fecha por periodo de tiempo

| | Mes text  | Artículo character varying (150)  | Ingreso Total numeric  |
|---|--|--|---|
| 1 | 2025-12 | Taza de regalo | 1280.00 |
| 2 | 2025-12 | Pluma negra | 400.00 |
| 3 | 2025-01 | Taza de regalo | 80.00 |
| 4 | 2025-01 | Cuaderno profesional | 35.00 |
| 5 | 2025-01 | Pluma negra | 16.00 |

Figura 39: Mensuales por articulo

| | Categoría character varying (50)  | Ventas Totales (\$) numeric  | Margen Bruto (Ganancia) numeric  |
|---|---|---|--|
| 1 | Regalos | 1360.00 | 680.00 |
| 2 | Papeleria | 451.00 | 223.00 |

Figura 40: Por categoría

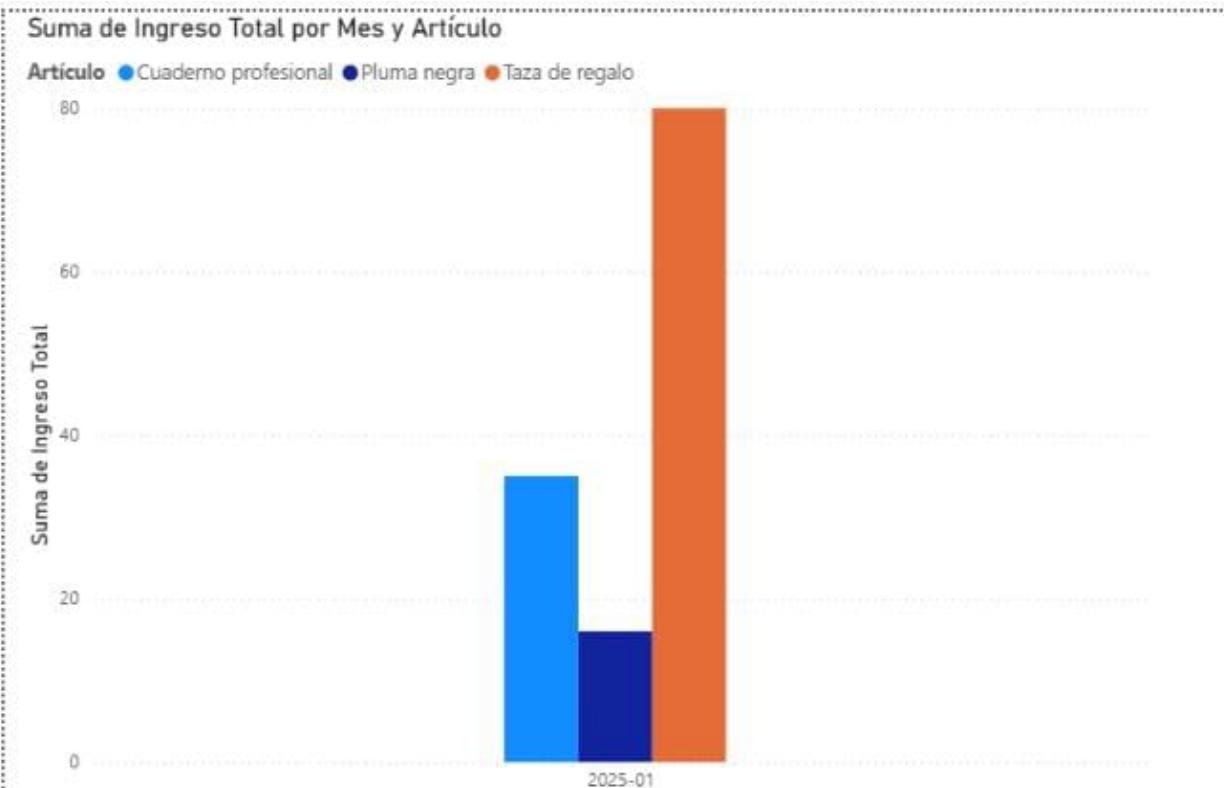


Figura 41: Producto más vendido por mes

Suma de Margen Bruto (Ganancia) y Suma de Ventas Totales (\$) por Categoría

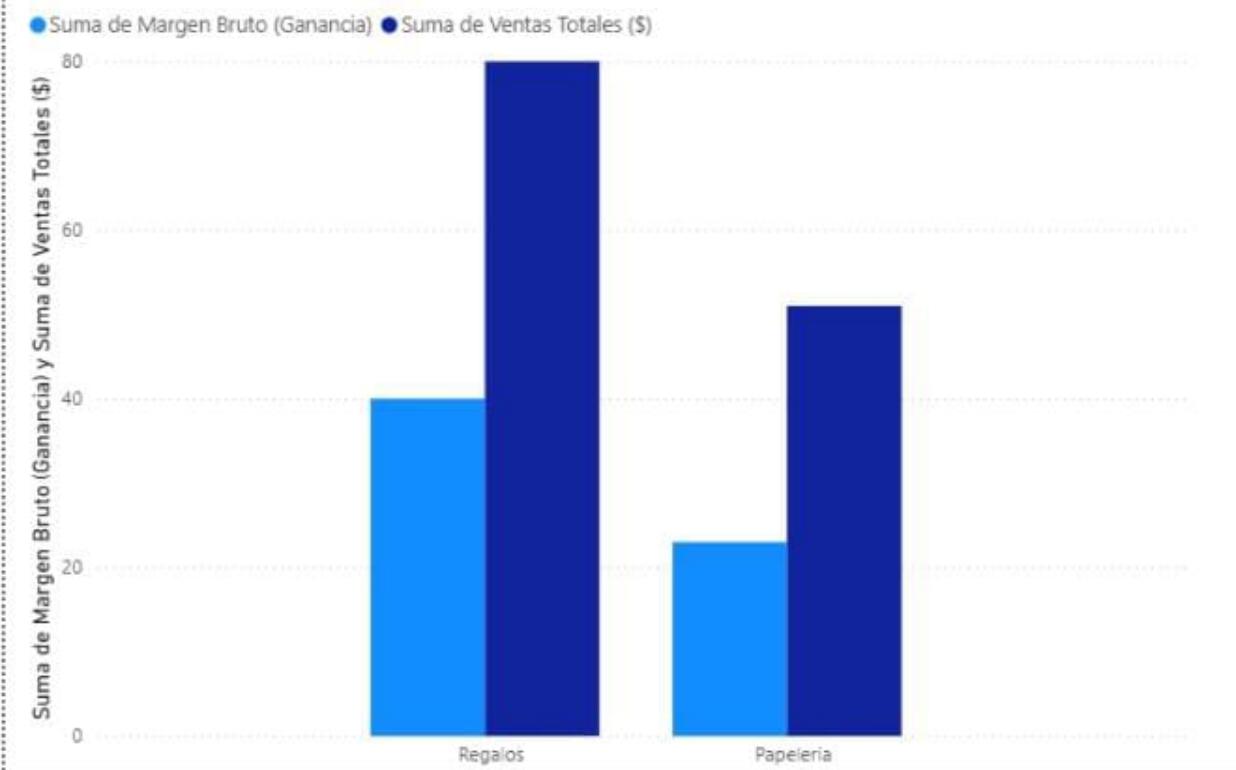


Figura 42: Márgenes de ventas

7. Conclusiones

7.1. Buendía López Sebastián

Durante el desarrollo del proyecto, mi responsabilidad se centró en las fases de diseño y estructuración de la base de datos, abarcando desde la elaboración del Modelo Entidad-Relación y el Modelo Relacional, hasta la creación física de las tablas y la inserción de los datos de prueba en SQL. El mayor reto que enfrenté fue realizar un análisis preciso para identificar correctamente las entidades y sus atributos; esta tarea era fundamental, pues un error en la definición inicial habría comprometido la integridad del modelo y dificultado el desarrollo de las actividades posteriores, como la programación de triggers y vistas. Afortunadamente, lograr un diseño normalizado y sólido permitió que la implementación fluyera correctamente. Afortunadamente, lograr un diseño normalizado y sólido permitió que la implementación fluyera correctamente.

7.2. Hernández Pérez Mariana Daniela

Desarrollar este sistema para la papelería fue una experiencia muy gratificante, ya que logramos traducir diagramas y código en una herramienta útil para el día a día. Aunque la automatización con triggers fue clave, considero que el punto de inflexión fue la implementación de Vistas para la generación de facturas; poder simplificar toda la información compleja de clientes y productos en una sola estructura virtual nos permitió simular la emisión de tickets de venta de forma ágil y ordenada. Ver cómo una vista bien diseñada facilita tanto la operación inmediata como el posterior análisis de datos en Power BI fue, sin duda, uno de los mayores aprendizajes técnicos del proyecto.

7.3. Velarde Valencia Josue

Para la realización de mi parte del proyecto no tuve muchas complicaciones con las tablas, su definición o codificación en SQL, el trabajo en conjunto fue muy eficiente y la comunicación me ayudó a saber exactamente qué necesitaba el proyecto para ser tal y como lo planteamos, por lo que encontré considero que el resultado fue el esperado.

7.4. Velazquez Cortes Cristina del Carmen

De las dificultades que tuve para la realización de mi parte del proyecto, ocurrieron al momento de realizar el dashboard, debido a que no podíamos conectar la base de datos pues nos encontrábamos trabajando en conjunto de manera remota y, investigando sobre nuestro problema, para conectarnos necesitábamos el driver NPGsql para trabajarla de forma remota.

7.5. Vences Santillán Carlos Eduardo

El desarrollo de este proyecto nos permitió implementar una solución integral para la gestión de una cadena de papelerías, abarcando desde el diseño conceptual hasta el análisis final de la información mediante dashboards.

Con ayuda de distintas herramientas como Drawio, PostgreSQL y Power BI, nos fue posible construir una base de datos, automatizar reglas de negocio mediante funciones y triggers, y generar visualizaciones que facilitan la toma de decisiones para el usuario final. El sistema resultante está alineado con los requerimientos del proyecto. Lo que más me costó trabajo fue realizar la documentación necesaria en formato Latex.

8. Referencias

- [1] The PostgreSQL Global Development Group, “PostgreSQL: The world’s most advanced open source database - Documentation,” <https://www.postgresql.org/docs/>, accedido: 15 de noviembre de 2025.
- [2] Neon, “PostgreSQL Tutorial,” <https://neon.com/postgresql/tutorial>, accedido: 18 de noviembre de 2025.
- [3] TodoPostgreSQL, “Manejando funciones en PostgreSQL,” <https://www.todopostgresql.com/manejando-funciones-en-postgresql/>, accedido: 22 de noviembre de 2025.
- [4] The PostgreSQL Global Development Group, “CREATE PROCEDURE - PostgreSQL Documentation,” <https://www.postgresql.org/docs/current/sql-createprocedure.html>, accedido: 28 de noviembre de 2025.
- [5] Neon, “PostgreSQL CREATE PROCEDURE By Examples,” <https://neon.com/postgresql/postgresql-plpgsql/postgresql-create-procedure>, accedido: 30 de noviembre de 2025.
- [6] The PostgreSQL Global Development Group, “CREATE TRIGGER - PostgreSQL Documentation,” <https://www.postgresql.org/docs/current/sql-createtrigger.html>, accedido: 5 de diciembre de 2025.
- [7] Imasgal, “Uso de disparadores en PostgreSQL - Blog Imasgal,” <https://imasgal.com/uso-de-disparadores-en-postgresql/>, accedido: 6 de diciembre de 2025.