UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

División de Ingeniería Eléctrica
Departamento de Ingeniería en Computación

Bases de Datos

Proyecto final de la asignatura de bases de datos

Ing. Fernando Arreola

Noviembre 2024

Equipo DataBenders:

Integrantes:

Luis Fernando Franco Arellano
Gustavo Isaac Soto Huerta
Eduardo Zavala Sánchez
José Eduardo Villeda Tlecuitl

Semestre 2025 - 1 **Fecha de entrega: 18 de noviembre de 2024**

Capítulo 1

Introducción

En el ámbito de la gestión empresarial, el manejo adecuado y eficiente de la información es fundamental para optimizar procesos y mejorar la toma de decisiones. Este proyecto tiene como propósito desarrollar una solución de base de datos que permita a una cadena de papelerías almacenar y organizar su información de forma estructurada y accesible, contribuyendo así a su modernización operativa. La implementación de esta base de datos busca satisfacer una serie de requerimientos específicos, tales como el almacenamiento de datos detallados de proveedores, clientes, inventarios y transacciones de ventas, asegurando a su vez la integridad de la información y la eficiencia en su manejo.

Este sistema no sólo se limita a almacenar información; también está diseñado para automatizar ciertos procesos críticos, como la generación de facturas, la actualización del inventario, y el cálculo de utilidades, entre otros. A lo largo del desarrollo, se deberán aplicar conceptos de diseño de bases de datos y lenguajes de consulta, tomando en cuenta las particularidades del negocio y las reglas de integridad que garanticen la consistencia de los datos. Además, se realizarán configuraciones específicas, tales como la creación de índices y estructuras adicionales para optimizar el acceso a la información y la generación de reportes.

De este modo, el proyecto permite a los estudiantes aplicar conocimientos de diseño y administración de bases de datos en un contexto práctico, explorando cómo una infraestructura de datos bien construida puede ofrecer un soporte fundamental a las operaciones y la estrategia de una empresa.

1.1. Objetivo

- **Optimizar la gestión de información clave para la operación de la papelería.** Desarrollar una base de datos eficiente para almacenar de forma organizada datos cruciales como clientes, proveedores, productos e inventario. Esto permitirá un acceso centralizado y rápido a la información, mejorando la precisión en la toma de decisiones comerciales y agilizando la gestión diaria.

- **Automatizar procesos de inventario y control de ventas para aumentar la eficiencia** Implementar un sistema automatizado que actualice el inventario tras cada venta y genere alertas ante niveles bajos de stock. Además, se rastrearán las transacciones en tiempo real para evitar desabastecimientos y mejorar la respuesta ante la demanda, optimizando así la eficiencia operativa de la papelería.

- **Facilitar la generación de reportes financieros para apoyar decisiones estratégicas** Crear herramientas que generen reportes detallados sobre ventas, ingresos y márgenes de utilidad, facilitando el análisis del rendimiento del negocio. Esto permitirá identificar tendencias de productos, optimizar las compras y diseñar estrategias de ventas basadas en datos precisos.

- **Asegurar la escalabilidad y adaptabilidad del sistema para el futuro** Diseñar la base de datos con tecnologías escalables que permitan su crecimiento a medida que la papelería expanda sus operaciones. Esto garantizará que el sistema soporte mayores volúmenes de datos y usuarios sin comprometer el rendimiento, permitiendo una adaptación flexible a cambios futuros.

1.2. Propuesta de solución.

El proyecto se centra en el diseño e implementación de un sistema de base de datos para una cadena de papelerías, con el objetivo de optimizar el almacenamiento, gestión y análisis de datos relacionados con proveedores, clientes, productos, ventas y empleados. Para ello se tiene una propuesta de solución dividida en varias fases basada en el los requerimientos dados por la papeleria.

1. Analisis de requerimientos.
2. Diseño del modelo entidad-relación para estructurar la base de datos.
3. Conversión del modelo entidad-relación al modelo relacional.
4. Implementación del diseño en un Sistema de Gestión de Bases de Datos. (DBMS)
5. Desarrollo de funciones adicionales. (Parte dos)

1.3. Plan de trabajo

El plan de trabajo para el desarrollo del sistema de gestión de base de datos para la cadena de papelerías se basa en los requerimientos identificados en el análisis previo y sigue un enfoque estructurado que garantiza una implementación exitosa.

1. **Analisis de requerimientos**

   El primer paso para la implementación de la base de datos es analizar en profundidad los requerimientos específicos proporcionados por la cadena de papelerías. Este análisis inicial es esencial para asegurar que la base de datos diseñada cubra todas las necesidades operativas y comerciales del negocio, y para evitar posibles problemas durante las fases posteriores del proyecto.

2. **Diseño del modelo entidad-relación para estructurar la base de datos**

   El punto de partida en la creación del sistema será desarrollar un diagrama entidad-relación (ERD) detallado que represente de manera clara y precisa la estructura lógica de los datos. Este proceso implica identificar todas las entidades que forman parte del entorno de la papelería, como clientes, productos, proveedores, entre otros. Además, se definirán los atributos específicos para cada entidad, como el nombre del cliente, la descripción del producto, los detalles de los pedidos, etc.

   Posteriormente, se establecerán las relaciones entre estas entidades para reflejar cómo interactúan en el contexto del negocio.

3. **Conversión del modelo entidad-relación al modelo relacional**

   Después de haber finalizado el diseño del diagrama entidad-relación (ERD), el siguiente paso en el desarrollo de la base de datos consiste en convertir este modelo conceptual en un modelo relacional detallado. Esta fase implica trasladar los elementos del ERD a un esquema físico que sea compatible con el sistema de gestión de bases de datos (DBMS) que se utilizará. La conversión del modelo ERD al relacional es fundamental, ya que proporciona la base estructural para el almacenamiento y la manipulación eficiente de los datos en la base de datos.

4. **Implementación del diseño en un Sistema de Gestión de Bases de Datos (DBMS)**

   En esta etapa esencial del proyecto, se implementará el diseño conceptual y relacional previamente definido en un Sistema de Gestión de Bases de Datos (DBMS) concreto, utilizando PostgreSQL. El objetivo es convertir el modelo lógico en una infraestructura física que posibilite la gestión eficiente de la información de la papelería en un entorno real y funcional.

5. **Desarrollo de funciones adicionales.**

   Se desarrollará un módulo adicional para la generación automática de facturas y reportes financieros, junto con la implementación de alertas que notifiquen niveles bajos de stock. Además, se creará un dashboard interactivo que permita visualizar métricas clave, como ventas, ingresos y los productos más vendidos, mejorando así la gestión y el análisis del negocio.

6. **Pruebas del sistema**

   Se llevarán a cabo pruebas unitarias y de integración para asegurar el correcto funcionamiento del sistema, junto con la validación de su desempeño bajo distintos escenarios de carga. A partir de los resultados obtenidos, se realizarán ajustes y correcciones para solucionar los errores detectados durante las pruebas.

7. **Documentación y presentación del proyecto**

   Se elabora la documentación formal en LaTeX, abarcando el diseño y la implementación del sistema. Además, se subirá el código fuente, los scripts y toda la documentación a un repositorio en GitHub, y se preparará una presentación para la exposición del proyecto final.

Actividades	Duración estimada	Fecha de inicio	Fecha de fin
Análisis de requerimientos.	3 días.	07/10/24	10/10/24
Diseño del modelo entidad-relación. (ERD)	5 días.	11/10/24	16/10/24
Conversión al modelo relacional.	5 días.	17/10/24	22/10/24
Implementación en PostgreSQL	1 semana.	23/10/24	30/10/24
Desarrollo de funcionalidades adicionales.	1 semana.	31/10/24	07/11/24
Pruebas y validación.	3 días.	08/11/24	11/11/24
Documentación y entrega del proyecto.	4 días.	12/11/24	16/11/24

1.4. Diseño.

1. **Analisis de requerimientos.**

   Para llevar a cabo esta fase, se dedicó un tiempo considerable al análisis de los requerimientos que la cadena de papelerías había planteado. El proceso comenzó con un estudio detallado de la documentación proporcionada por la empresa, la cual contenía información clave sobre sus flujos de trabajo, gestión de inventario, control de proveedores, facturación y seguimiento de ventas. A partir de esta revisión, se identificaron los puntos críticos y áreas de mejora en sus procesos actuales.

2. **Diseño del modelo entidad-relación para estructurar la base de datos.**

   Para el desarrollo del modelo, se identificaron 10 entidades clave.

   Optamos por incluir catálogos individuales para estados y países, lo que facilita tanto la búsqueda como la inserción de datos, garantizando la consistencia al gestionar ubicaciones.

   En el análisis de relaciones, se determinó la existencia de una única entidad débil, denominada StockBajo, que depende exclusivamente de la existencia de un producto.

   A continuación, se detallan las principales entidades junto con sus respectivos atributos.

   - **Proveedores**
     - **ProveedorID** (Clave Primaria)
     - NombreComercial
     - Domicilio
       - Domicilio_Calle
       - Domicilio_Número
       - Domicilio_Colonia
       - Domicilio_CódigoPostal
       - Domicilio_Estado
     - Nombre

        - Nombre
        - ApellidoPaterno
        - ApellidoMaterno (Opcional)
    - Teléfonos (Multivalorado)
    - RFC (Clave Candidata)
- **Clientes**
    - **ClienteID** (Clave Primaria)
    - Nombre
        - Nombre
        - ApellidoPaterno
        - ApellidoMaterno (Opcional)
    - Domicilio
        - Domicilio_Calle
        - Domicilio_Número
        - Domicilio_Colonia
        - Domicilio_CódigoPostal
        - Domicilio_Estado
    - CorreosElectrónicos (Multivalorado)
    - RFC (Clave Candidata) (Opcional)
- **Productos**
    - **ProductoID** (Clave Primaria)
    - CódigoBarras (Clave Candidata)
    - Nombre
    - PrecioCompra
    - PrecioVenta
    - Foto
    - CantidadStock
- **Categorías**
    - **CategoríaID** (Clave Primaria)
    - Nombre
    - Descripción
- **Ventas**
    - **VentaID** (Clave Primaria)
    - Folio (Clave Candidata)
    - FechaVenta
    - MontoTotalPagar
- **Empleados**
    - **EmpleadoID** (Clave Primaria)
        - Nombre
        - ApellidoPaterno
        - ApellidoMaterno (Opcional)
    - FechaNacimiento
    - FechaIngreso
- **Producto_Venta** (Relación)
    - CantidadProducto
    - PrecioTotalProducto
- **StockBajo**
    - **ProductoID** (Clave Primaria)
    - EsStockBajo
- **Países**

- **PaísID** (Clave Primaria)
- Nombre

- **Estados**
  - **EstadoID** (Clave Primaria)
  - Nombre

1.4.1. Relaciones

- **Proveedor** a **Producto:** Un proveedor puede ofrecer múltiples productos (1:M).
- **Cliente** a **Venta:** Un cliente puede realizar múltiples ventas (1:M).
- **Venta** a **Producto:** Muchos productos pueden estar contenidos en muchas ventas (M:M). Esta relación puede tener atributos adicionales, como *Cantidad* y *PrecioTotal*.
- **Categoría** a **Producto:** Cada producto pertenece a una categoría (M:1).
- **Empleado** a **Venta:** Un empleado puede registrar múltiples ventas (1:M).
- **Estado** a **Cliente:** Un cliente reside en un estado (1:M).
- **Estado** a **Proveedor:** Un proveedor reside en un estado (1:M).
- **País** a **Estado:** Un estado pertenece a un país (1:M).
- **Cliente** a **CorreosElectrónicosCliente:** Un cliente puede tener múltiples direcciones de correo electrónico (1:M).

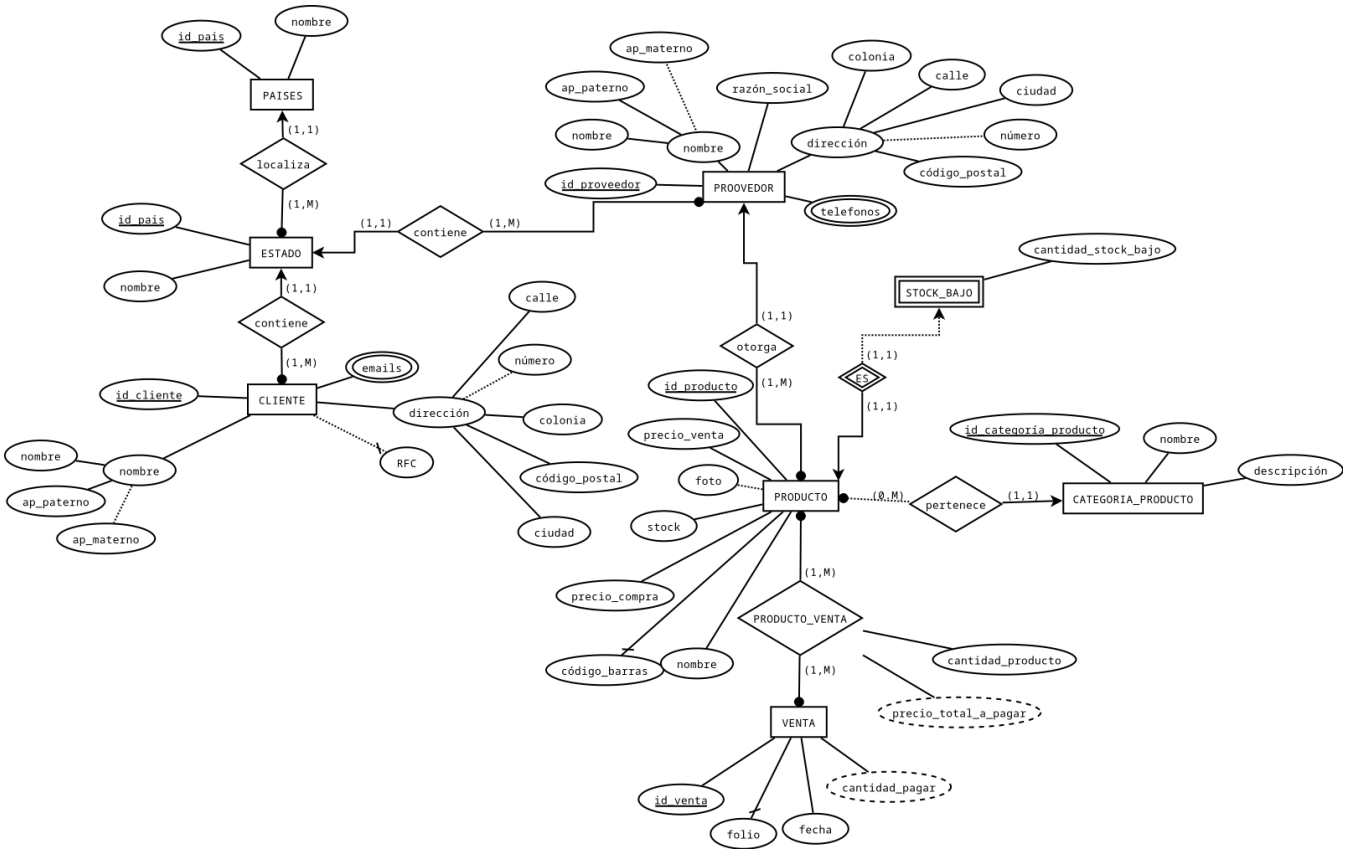

Figura 1.1: Modelo Entidad-Relación de nuestro caso de estudio

3. **Conversión del modelo entidad-relación al modelo relacional.**

   Basamos nuestra representación en las recomendaciones de la documentación de PGModeler , donde revisamos cómo representar nuestro caso de estudio en el modelo relacional.

   - **Proveedores**

- **ProveedorID** (PK)
- RazonSocial (VARCHAR(100))
- Domicilio
  - Domicilio_Calle (VARCHAR(100))
  - Domicilio_Numero (VARCHAR(10))
  - Domicilio_Colonia (VARCHAR(100))
  - Domicilio_CodigoPostal (VARCHAR(10))
  - Domicilio_EstadoID (FK)
- Nombre (VARCHAR(100))
- RFC (VARCHAR(50), Unique)

- **ProveedorTelefonos**
  - **TelefonoID** (PK)
  - ProveedorID (FK)
  - NumeroTelefono (VARCHAR(20))
  - TipoTelefono (VARCHAR(50))
- **Clientes**
  - **ClienteID** (PK)
  - Nombre (VARCHAR(100))
  - ApellidoPaterno (VARCHAR(100))
  - ApellidoMaterno (VARCHAR(100))
  - Domicilio
    - Domicilio_Calle (VARCHAR(100))
    - Domicilio_Numero (VARCHAR(10))
    - Domicilio_Colonia (VARCHAR(100))
    - Domicilio_CodigoPostal (VARCHAR(10))
    - Domicilio_EstadoID (FK)
  - Emails (VARCHAR(100), Multivaluado)
  - RFC (VARCHAR(50), Unique) (NULL)
- **Productos**
  - **ProductoID** (PK)
  - CodigoBarras (VARCHAR(50), UNIQUE)
  - Nombre (VARCHAR(100))
  - PrecioCompra (DECIMAL(10, 2))
  - PrecioVenta (DECIMAL(10, 2))
  - Foto (BLOB)
  - CantidadStock (INT)
  - CategoriaID (FK)
  - ProveedorID (FK)
- **Categorias**
  - **CategoriaID** (PK)
  - Nombre (VARCHAR(100))
  - Descripcion (TEXT)
- **Ventas**
  - **VentaID** (PK)
  - Folio (VARCHAR(20), Unique)
  - FechaVenta (DATE)
  - CantidadTotalPagar (NUMERIC(10, 2))
  - EmpleadoID (FK)

- **Empleados**
  - **EmpleadoID** (PK)
  - Nombre (VARCHAR(100))
  - ApellidoPaterno (VARCHAR(100))
  - ApellidoMaterno (VARCHAR(100))
  - FechaNacimiento (DATE)
  - FechaIngreso (DATE)
- **DetalleVenta**
  - **DetalleVentaID** (PK)
  - VentaID (FK)
  - ProductoID (FK)
  - CantidadProducto (INT)
  - PrecioTotalArticulo (NUMERIC(10, 2))
- **StockBajo**
  - **ProductoID** (PK, FK)
  - EnBajoStock (BOOLEAN, DEFAULT FALSE)
- **Paises**
  - **PaisID** (PK)
  - Nombre (VARCHAR(100))
- **Estados**
  - **EstadoID** (PK)
  - Nombre (VARCHAR(100))
  - PaisID (FK)
- **ClienteEmails**
  - ClienteID (FK)
  - Email (VARCHAR(100))
  - **PRIMARY KEY (ClienteID, Email)**

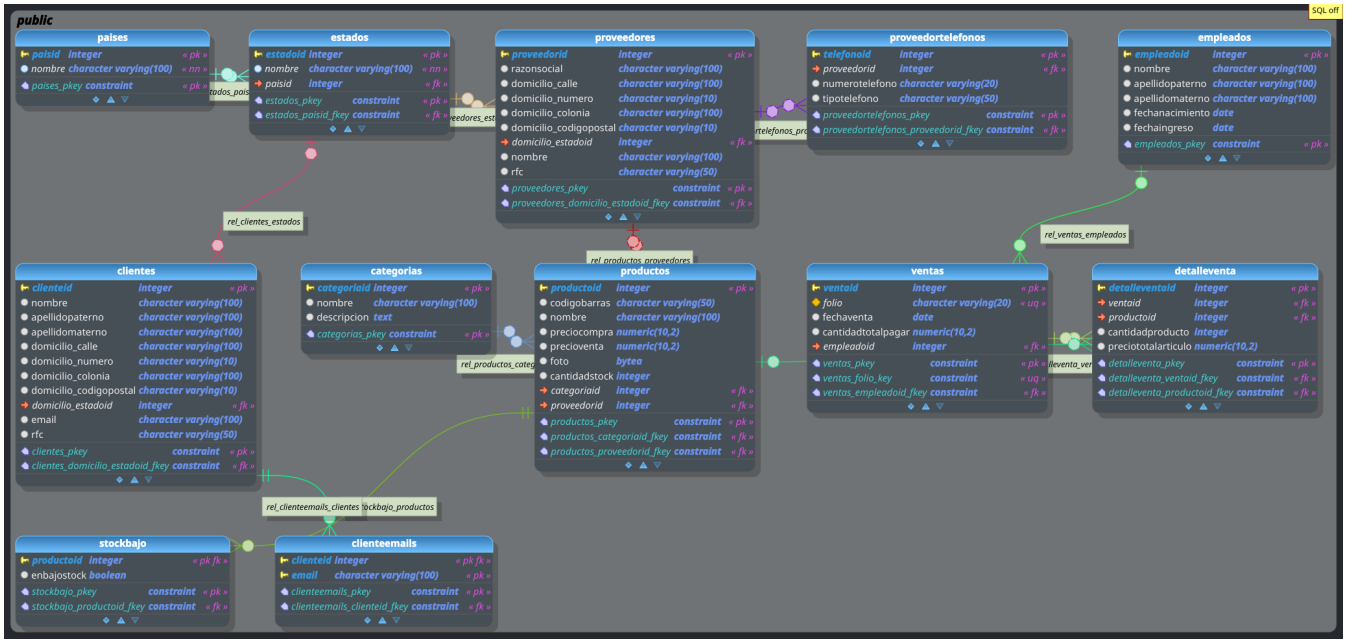

Figura 1.2: Modelo relacional de nuestro modelo

4. **Implementación del diseño en un Sistema de Gestión de Bases de Datos. (DBMS)**

   Basándonos en nuestro modelo relacional y utilizando la herramienta PGModeler junto con la documentación de sentencias SQL para PostgreSQL, generamos el código necesario para la creación de un sistema de gestión integral que incluye 12 tablas: Países, Estados, Proveedores, ProveedorTelefonos, Empleados, Clientes, Categorías, Productos, Ventas, DetalleVentas y ClienteEmails. Cada tabla fue diseñada con sus respectivos atributos y relaciones, optimizando la integridad referencial y la eficiencia del sistema. Las tablas gestionan información clave como ubicaciones (países y estados), proveedores y sus teléfonos, empleados, clientes y sus correos electrónicos, así como categorías y detalles de productos. Además, el sistema organiza las ventas y sus respectivos detalles, vinculando transacciones con los clientes y empleados responsables, asegurando una estructura escalable y eficiente para el manejo de datos comerciales y operativos.

5. **Desarrollo de funciones adicionales. (Parte dos)**

   Una vez diseñada e implementada la base de datos, se debe integrar un sistema de visualización de datos que facilite el análisis y la toma de decisiones estratégicas. Para ello, se propone la creación de un dashboard interactivo que muestre indicadores clave, como los ingresos mensuales desglosados en inversiones, ganancias y total de ingresos, lo que permitirá un seguimiento claro del rendimiento financiero. Además, incluirá una sección con el Top 3 de artículos más vendidos, útil para identificar tendencias y ajustar estrategias de inventario y marketing. Por último, se incorporará un reporte de los empleados que registran más órdenes, proporcionando métricas de desempeño que pueden ser utilizadas para evaluar la productividad del personal y tomar decisiones de gestión. Este dashboard será una herramienta integral para la supervisión y optimización de las operaciones del negocio.

1.5. Implementación.

A continuación se presenta el DDL para la creación de la base de datos:

```
    CREATE TABLE Paises (
    PaisID INT GENERATED BY DEFAULT AS IDENTITY,
    Nombre VARCHAR(100) NOT NULL,
    CONSTRAINT Paises_PK PRIMARY KEY (PaisID)
);

CREATE TABLE Estados (
    EstadoID INT GENERATED BY DEFAULT AS IDENTITY,
    Nombre VARCHAR(100) NOT NULL,
    PaisID INT,
    CONSTRAINT ESTADOS_PK PRIMARY KEY (EstadoID),
    CONSTRAINT ESTADOS_PAISID_FK FOREIGN KEY (PaisID) REFERENCES Paises(PaisID)
);

CREATE TABLE Proveedores (
    ProveedorID INT GENERATED BY DEFAULT AS IDENTITY,
    RazonSocial VARCHAR(100) NOT NULL,
    Domicilio_Calle VARCHAR(100),
    Domicilio_Numero VARCHAR(10),
    Domicilio_Colonia VARCHAR(100),
    Domicilio_CodigoPostal VARCHAR(10),
    Domicilio_EstadoID INT,
    Nombre VARCHAR(100) NOT NULL,
    RFC VARCHAR(50) NOT NULL,
    CONSTRAINT Proveedores_pk PRIMARY KEY (ProveedorID),
    CONSTRAINT proveedores_domicilio_estado_fk FOREIGN KEY (Domicilio_EstadoID) REFERENCES
);

CREATE TABLE ProveedorTelefonos (
    TelefonoID INT GENERATED BY DEFAULT AS IDENTITY,
    ProveedorID INT,
    NumeroTelefono VARCHAR(20),
```

```
    TipoTelefono VARCHAR(50), -- Ejemplo: 'Movil', 'Oficina', 'Fax', etc.
    CONSTRAINT Proveedor_telefonos_pk PRIMARY KEY (TelefonoID),
    CONSTRAINT Proveedortelefono_fk FOREIGN KEY (ProveedorID) REFERENCES Proveedores(ProveedorID
);

CREATE TABLE Empleados (
    EmpleadoID INT GENERATED BY DEFAULT AS IDENTITY,
    Nombre VARCHAR(100) NOT NULL,
    ApellidoPaterno VARCHAR(100) NOT NULL,
    ApellidoMaterno VARCHAR(100),
    FechaNacimiento DATE NOT NULL,
    FechaIngreso DATE NOT NULL,
    CONSTRAINT Empledados_pk PRIMARY KEY (EmpleadoID)
);

CREATE TABLE Clientes (
    ClienteID INT GENERATED BY DEFAULT AS IDENTITY,
    Nombre VARCHAR(100) NOT NULL,
    ApellidoPaterno VARCHAR(100) NOT NULL,
    ApellidoMaterno VARCHAR(100),
    Domicilio_Calle VARCHAR(100),
    Domicilio_Numero VARCHAR(10),
    Domicilio_Colonia VARCHAR(100),
    Domicilio_CodigoPostal VARCHAR(10),
    Domicilio_EstadoID INT,
    Email VARCHAR(100),
    RFC VARCHAR(50) NOT NULL,
    CONSTRAINT Clientes_pk PRIMARY KEY (ClienteID),
    CONSTRAINT Clientes_Domicilio FOREIGN KEY (Domicilio_EstadoID) REFERENCES Estados(EstadoID)
);

CREATE TABLE Categorias (
    CategoriaID INT GENERATED BY DEFAULT AS IDENTITY,
    Nombre VARCHAR(100) NOT NULL,
    Descripcion TEXT,
    CONSTRAINT Categorias_pk PRIMARY KEY (CategoriaID)
);

CREATE TABLE Productos (
    ProductoID INT GENERATED BY DEFAULT AS IDENTITY,
    CodigoBarras VARCHAR(50),
    Nombre VARCHAR(100) NOT NULL,
    PrecioCompra DECIMAL(10, 2) NOT NULL,
    PrecioVenta DECIMAL(10, 2) NOT NULL,
    Foto BYTEA,
    Stock NUMERIC NOT NULL,
    CantidadStock INT NOT NULL,
    CategoriaID INT,
    ProveedorID INT,
    CONSTRAINT Productos_pk PRIMARY KEY (ProductoID),
    CONSTRAINT CATEGORIA_FK FOREIGN KEY (CategoriaID) REFERENCES Categorias(CategoriaID),
    CONSTRAINT PROVEEDOR_FK FOREIGN KEY (ProveedorID) REFERENCES Proveedores(ProveedorID)
);

CREATE TABLE Ventas (
    VentaID INT GENERATED BY DEFAULT AS IDENTITY,
    Folio VARCHAR(20) UNIQUE,
    FechaVenta DATE,
    CantidadTotalPagar DECIMAL(10, 2),
```

```
    EmpleadoID INT,
    CONSTRAINT VENTAS_PK PRIMARY KEY (VentaID),
    CONSTRAINT VENTAS_EMPLEADO_FK FOREIGN KEY (EmpleadoID) REFERENCES Empleados(EmpleadoID)
);

CREATE TABLE DetalleVenta (
    DetalleVentaID INT GENERATED BY DEFAULT AS IDENTITY,
    VentaID INT,
    ProductoID INT,
    CantidadProducto INT,
    PrecioTotalArticulo DECIMAL(10, 2),
    ClienteID INT,
    CONSTRAINT DETALL_VENTA_PK PRIMARY KEY (DetalleVentaID),
    CONSTRAINT DETALLE_VENTA_VENTAID FOREIGN KEY (VentaID) REFERENCES Ventas(VentaID),
    CONSTRAINT DETALLE_VENTA_PRODUCTO FOREIGN KEY (ProductoID) REFERENCES Productos(Product
    CONSTRAINT DETALLE_VENTA_CLIENTE FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID)
);


CREATE TABLE ClienteEmails (
    ClienteID INT,
    Email VARCHAR(100) NOT NULL,
    CONSTRAINT CLIENTE_MAILS_PK PRIMARY KEY (ClienteID, Email),
    CONSTRAINT EMAILS_CLIENTES_FK FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID)
);
```

Implementación de triggers:

- Trigger para menejar las ventas y el stock:

```
-- Funcion que se ejecutara antes de insertar en detalleventa
CREATE OR REPLACE FUNCTION fn_procesar_venta()
RETURNS TRIGGER AS $$
DECLARE
    stock_actual INT;
    nuevo_stock INT;
BEGIN
    -- Obtener el stock actual del producto
    SELECT cantidadstock INTO stock_actual
    FROM productos
    WHERE productoid = NEW.productoid;

    -- Verificar si se obtuvo el stock actual
    IF stock_actual IS NULL THEN
        RAISE EXCEPTION 'No se encontr  el producto con ID %.', NEW.productoid;
    END IF;

    -- Calcular el nuevo stock
    nuevo_stock := stock_actual - NEW.cantidadproducto;

    -- Verificar si hay suficiente stock
    IF nuevo_stock < 0 THEN
        RAISE EXCEPTION 'Stock insuficiente para el producto ID %.', NEW.productoid;
        -- Abortar  la transacci n
    END IF;

    -- Actualizar el stock del producto
    UPDATE productos
    SET cantidadstock = nuevo_stock
    WHERE productoid = NEW.productoid;
```

```
    -- Emitir alerta si el nuevo stock es menor que 3
    IF nuevo_stock < 3 THEN
        RAISE NOTICE 'Alerta: El stock del producto ID % es menor que 3 (Stock actual: %).',
        -- Aqu  podr as implementar l gica adicional para enviar una notificaci n
    END IF;

    -- Calcular el precio total por art culo
    SELECT precioventa INTO NEW.preciototalarticulo
    FROM productos
    WHERE productoid = NEW.productoid;

    NEW.preciototalarticulo := NEW.cantidadproducto * NEW.preciototalarticulo;

    -- Actualizar el total a pagar por la venta
    UPDATE ventas
    SET cantidadtotalpagar = COALESCE(cantidadtotalpagar, 0) + NEW.preciototalarticulo
    WHERE ventaid = NEW.ventaid;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Crear el trigger que llama a la funci n antes de insertar en detalleventa
CREATE TRIGGER trg_procesar_venta
BEFORE INSERT ON detalleventa
FOR EACH ROW
EXECUTE FUNCTION fn_procesar_venta();
```

Como se puede ver, se requería que se decrementara el stock a medida que de la cantidad que venderia por producto, la solución esta dada por el trigger "trg$_procesar_venta$" *elcualmedianteunafuncionque* :

- Obtiene el stock actual del producto (en caso de no encontrarlo indica error)
- Calcula el nuevo stock
- Verifica si hay suficiente stock
- Actualiza el stock del producto

	productoid [PK] integer	nombre character varying (100)	cantidadstock integer
1	1	Cuaderno Profesional	91

```
BEGIN;
INSERT INTO detalleventa (ventaid,productoid, cantidadproducto, clienteid)
VALUES (1, 1, 92, 1);
COMMIT;
```

```
ERROR:  Stock insuficiente para el producto ID 1.
CONTEXT:  función PL/pgSQL fn_procesar_venta() en la línea 21 en RAISE

SQL state: P0001
```

Figura 1.3: Implemetación del stock.

	productoid [PK] integer	nombre character varying (100)	cantidadstock integer
1	1	Cuaderno Profesional	91

```
BEGIN;


INSERT INTO detalleventa (ventaid, productoid, cantidadproducto, clienteid)
VALUES (1, 1, 89, 1);

COMMIT;
NOTICE:  Alerta: El stock del producto ID 1 es menor que 3 (Stock actual: 2).
COMMIT

Query returned successfully in 79 msec.
```

Figura 1.4: Implemetación del stock.

Factura de Compra:

Para la solución de este problema se pensó en una vista en la cual se puede obtener todos los registros que se verian dentro de una factura.La factura contiene datos sobre: Datos de la venta, Información del empleado que realizo la venta, Información del cliente (id, nombre y dirección), Información del producto y finalmente Detalles especifos de la venta

```
CREATE VIEW VistaFactura AS
SELECT
    -- Datos de la venta
    v.VentaID,                      -- ID nico de la venta
    v.Folio,                        -- Folio o n mero de referencia de la venta
    v.FechaVenta,                   -- Fecha en que se realiz  la venta
    v.CantidadTotalPagar,           -- Cantidad total a pagar por la venta

    -- Informaci n del empleado que realiz  la venta
    e.Nombre AS EmpleadoNombre,             -- Nombre del empleado
    e.ApellidoPaterno AS EmpleadoApellido,  -- Apellido paterno del empleado

    -- Informaci n del cliente
    c.ClienteID,                    -- ID nico del cliente
    c.Nombre AS ClienteNombre,              -- Nombre del cliente
    c.ApellidoPaterno AS ClienteApellido,   -- Apellido paterno del cliente
    c.RFC AS ClienteRFC,                    -- Registro Federal de Contribuyentes del clien

    -- Domicilio del cliente
    c.Domicilio_Calle,              -- Calle del domicilio del cliente
    c.Domicilio_Numero,             -- N mero exterior del domicilio
    c.Domicilio_Colonia,            -- Colonia del domicilio
    c.Domicilio_CodigoPostal,       -- C digo postal del domicilio

    -- Informaci n del producto vendido
    p.ProductoID,                   -- ID nico del producto
    p.Nombre AS ProductoNombre,             -- Nombre del producto
    p.CodigoBarras,                         -- C digo de barras del producto

    -- Detalles de la venta
    dv.CantidadProducto,            -- Cantidad de unidades vendidas del producto
    dv.PrecioTotalArticulo          -- Precio total por el art culo (cantidad * precio unitari
FROM
    Ventas v
    INNER JOIN Empleados e ON v.EmpleadoID = e.EmpleadoID
    INNER JOIN DetalleVenta dv ON v.VentaID = dv.VentaID
    INNER JOIN Productos p ON dv.ProductoID = p.ProductoID
    INNER JOIN Clientes c ON dv.ClienteID = c.ClienteID;
```

```
Proyecto=# SELECT * FROM VistaFactura WHERE VentaID = 1;
 ventaid |  folio   | fechaventa | cantidadtotalpagar | empleadonombre | empleadoapellido | clienteid | clientenombre | clienteapellido |  clienterfc
   |   domicilio_calle   | domicilio_numero | domicilio_colonia | domicilio_codigopostal | productoid |    productonombre    | codigobarras  | cantida
dproducto | preciototalarticulo
---------+----------+------------+--------------------+----------------+------------------+-----------+---------------+-----------------+------------
--+---------------------+------------------+-------------------+------------------------+------------+-----------------------+---------------+--------
----------+---------------------
       1 | VENT-001 | 2023-10-01 |             395.00 | Juan           | P0rez            |         1 | Ana           | Ramprez         | RAM12345678
9 | Calle Independencia | 456              | Col. Revoluci$n   | 06400                  |          1 | Cuaderno Profesional | 7501001234567 |
        2 |                70.00
       1 | VENT-001 | 2023-10-01 |             395.00 | Juan           | P0rez            |         1 | Ana           | Ramprez         | RAM12345678
9 | Calle Independencia | 456              | Col. Revoluci$n   | 06400                  |          2 | Pluma Bol¢grafo Azul | 7502002345678 |
        5 |                50.00
       1 | VENT-001 | 2023-10-01 |             395.00 | Juan           | P0rez            |         1 | Ana           | Ramprez         | RAM12345678
9 | Calle Independencia | 456              | Col. Revoluci$n   | 06400                  |          3 | Taza de Cer§mica      | 7503003456789 |
        1 |                30.00
       1 | VENT-001 | 2023-10-01 |             395.00 | Juan           | P0rez            |         1 | Ana           | Ramprez         | RAM12345678
9 | Calle Independencia | 456              | Col. Revoluci$n   | 06400                  |          1 | Cuaderno Profesional | 7501001234567 |
        2 |                70.00
       1 | VENT-001 | 2023-10-01 |             395.00 | Juan           | P0rez            |         1 | Ana           | Ramprez         | RAM12345678
9 | Calle Independencia | 456              | Col. Revoluci$n   | 06400                  |          1 | Cuaderno Profesional | 7501001234567 |
        2 |                70.00
       1 | VENT-001 | 2023-10-01 |             395.00 | Juan           | P0rez            |         1 | Ana           | Ramprez         | RAM12345678
9 | Calle Independencia | 456              | Col. Revoluci$n   | 06400                  |          1 | Cuaderno Profesional | 7501001234567 |
        3 |               105.00
(6 rows)
```

Figura 1.5: Vista de empleado.

Adicionalmente se creó otra vista la cual contiene a las ganancias de la venta, esta se implemento debido a que es un elemento importante dentro del sistema.

```
CREATE VIEW VistaVentasGanancias AS
SELECT
    -- Fecha de la venta
    v.FechaVenta,

    -- Cantidad total vendida en la fecha
    SUM(dv.PrecioTotalArticulo) AS CantidadTotalVendida,

    -- Ganancia total en la fecha (Ventas - Costo de los productos)
    SUM(dv.PrecioTotalArticulo - (dv.CantidadProducto * p.PrecioCompra)) AS GananciaTotal
FROM
    Ventas v
    INNER JOIN DetalleVenta dv ON v.VentaID = dv.VentaID    -- Relaciona las ventas con sus deta
    INNER JOIN Productos p ON dv.ProductoID = p.ProductoID  -- Relaciona los detalles con los pr
GROUP BY
    v.FechaVenta;
```

Figura 1.6: Vista.

```sql
CREATE VIEW VistaFactura AS
SELECT
    -- Datos de la venta
    v.VentaID,                -- ID único de la venta
    v.Folio,                  -- Folio o número de referencia de la venta
    v.FechaVenta,             -- Fecha en que se realizó la venta
    v.CantidadTotalPagar,     -- Cantidad total a pagar por la venta

    -- Información del empleado que realizó la venta
    e.Nombre AS EmpleadoNombre,            -- Nombre del empleado
    e.ApellidoPaterno AS EmpleadoApellido, -- Apellido paterno del empleado

    -- Información del cliente
    c.ClienteID,              -- ID único del cliente
    c.Nombre AS ClienteNombre,             -- Nombre del cliente
    c.ApellidoPaterno AS ClienteApellido,  -- Apellido paterno del cliente
    c.RFC AS ClienteRFC,                   -- Registro Federal de Contribuyentes del cliente

    -- Domicilio del cliente
    c.Domicilio_Calle,        -- Calle del domicilio del cliente
    c.Domicilio_Numero,       -- Número exterior del domicilio
    c.Domicilio_Colonia,      -- Colonia del domicilio
    c.Domicilio_CodigoPostal, -- Código postal del domicilio

    -- Información del producto vendido
    p.ProductoID,             -- ID único del producto
    p.Nombre AS ProductoNombre,            -- Nombre del producto
    p.CodigoBarras,                        -- Código de barras del producto

    -- Detalles de la venta
    dv.CantidadProducto,      -- Cantidad de unidades vendidas del producto
    dv.PrecioTotalArticulo    -- Precio total por el artículo (cantidad * precio unitario)
FROM
    Ventas v
    INNER JOIN Empleados e ON v.EmpleadoID = e.EmpleadoID     -- Relaciona la venta con el empleado
    INNER JOIN DetalleVenta dv ON v.VentaID = dv.VentaID      -- Relaciona la venta con sus detalles
    INNER JOIN Productos p ON dv.ProductoID = p.ProductoID    -- Relaciona los detalles con los productos
    INNER JOIN Clientes c ON dv.ClienteID = c.ClienteID;      -- Relaciona los detalles con los clientes
```

Figura 1.7: Vista.

Obtener el nombre de los productos con un stock bajo

Para implementar este requerimiento creó un trigger que se activa antes de insertar en un producto, y una función en la cual se crea una tabla para almacenar los productos con bajo stock, en ella se insertaran los productos con stock menor o igual a 3.

```sql
    CREATE TABLE IF NOT EXISTS productos_stock_bajo (
    productoid INT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    cantidadstock INT NOT NULL
);

-- 2. Insertar inicialmente los productos con stock menor a 3
INSERT INTO productos_stock_bajo (productoid, nombre, cantidadstock)
SELECT
    productoid,
    nombre,
    cantidadstock
FROM
    productos
WHERE
    cantidadstock < 3
ON CONFLICT (productoid) DO NOTHING; -- Evita errores si ya existen registros

-- 3. Crear una funci n que actualiza la tabla productos_stock_bajo
    despu s de INSERT o UPDATE en productos

CREATE OR REPLACE FUNCTION fn_actualizar_productos_stock_bajo()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.cantidadstock < 3 THEN
        -- Si el producto ya existe en productos_stock_bajo, actualizamos la cantidad
        IF EXISTS
```

```
            (SELECT 1 FROM productos_stock_bajo WHERE productoid = NEW.productoid) THEN
            UPDATE productos_stock_bajo
            SET cantidadstock = NEW.cantidadstock
            WHERE productoid = NEW.productoid;
        ELSE
            -- Si no existe, lo insertamos
            INSERT INTO productos_stock_bajo (productoid, nombre, cantidadstock)
            VALUES (NEW.productoid, NEW.nombre, NEW.cantidadstock);
        END IF;
    ELSE
        -- Si el stock es 3 o m s y el producto existe en productos_stock_bajo,
        lo eliminamos
        DELETE FROM productos_stock_bajo
        WHERE productoid = NEW.productoid;
    END IF;
    RETURN NULL; -- En triggers AFTER, el retorno no se utiliza
END;
$$ LANGUAGE plpgsql;

-- 4. Crear triggers para INSERT y UPDATE en la tabla productos
-- Trigger despu s de insertar en productos
DROP TRIGGER IF EXISTS trg_insert_productos_stock_bajo ON productos;
CREATE TRIGGER trg_insert_productos_stock_bajo
AFTER INSERT ON productos
FOR EACH ROW
EXECUTE FUNCTION fn_actualizar_productos_stock_bajo();

-- Trigger despu s de actualizar en productos
DROP TRIGGER IF EXISTS trg_update_productos_stock_bajo ON productos;
CREATE TRIGGER trg_update_productos_stock_bajo
AFTER UPDATE OF cantidadstock ON productos
FOR EACH ROW
EXECUTE FUNCTION fn_actualizar_productos_stock_bajo();
```

Como se puede ver se crea la tabla a la que se pondran los productos con bajo stock, para que posteriormente mediante una función se valide la cantidad, si es menor a 3 se actualiza la cantidad en caso de ya estar en stock bajo y en caso de no estar se agrega. El trigger esta implementado para aparecer antes de una actualización o inserción en productos.

```
DROP TRIGGER IF EXISTS trg_insert_productos_stock_bajo ON productos;
CREATE TRIGGER trg_insert_productos_stock_bajo
AFTER INSERT ON productos
FOR EACH ROW
EXECUTE FUNCTION fn_actualizar_productos_stock_bajo();

-- Trigger después de actualizar en productos
DROP TRIGGER IF EXISTS trg_update_productos_stock_bajo ON productos;
CREATE TRIGGER trg_update_productos_stock_bajo
AFTER UPDATE OF cantidadstock ON productos
FOR EACH ROW
EXECUTE FUNCTION fn_actualizar_productos_stock_bajo();
```

Figura 1.8: Nombre de producto con stock bajo.

```
Proyecto=# select *from productos_stock_bajo;
 productoid |        nombre        | cantidadstock
------------+----------------------+---------------
          1 | Cuaderno Profesional |             2
(1 row)


Proyecto=# update PRODUCTOS SET cantidadstock = 2 where productoid = 3;
UPDATE 1
Proyecto=# select *from productos_stock_bajo;
 productoid |        nombre        | cantidadstock
------------+----------------------+---------------
          1 | Cuaderno Profesional |             2
          3 | Taza de Cerßmica     |             2
(2 rows)
```

Figura 1.9: Implementación.

Trigger para el uso del formato en folio

```
        CREATE OR REPLACE FUNCTION generar_folio_venta()
RETURNS TRIGGER AS $$
BEGIN
    NEW.Folio := 'VENT-' || LPAD(nextval('secuencia_ventas')::text, 3, '0');
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

--Asignando el triger a la tabla ventas
--Se activa antes de una insercion de ventas
CREATE TRIGGER trigger_folio_ventas
BEFORE INSERT ON Ventas
FOR EACH ROW
EXECUTE FUNCTION generar_folio_venta();
```

En este caso el trigger se imlementa desoues de hacer la insercion en ventas, por lo que nos aseguramos de mantener el formato para el folio.

Indice empleado:

En este caso se decidio usar el indice B-tree en codigo de barras pues mejora la eficiencia de las consultas que buscan productos especificos por su codigo de barras, esspecialmente en funiones como obtener$_u tilidad_p roducto$.

Dado que 'codigobarras' sera frecuentemente utilizada el indice optimizara el tiempo de respuesta y reduce el costo de ejecucion de las consultas.

```
CREATE INDEX idx_productos_codigobarras
ON productos (codigobarras);
```

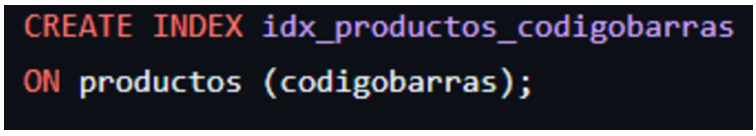

Figura 1.10: Creación de indice.

Parte 2 (DASHBOARD)

El código implementa un simulador básico de un sistema de gestión de bases de datos similar a PostgreSQL, en Java. Define una estructura de datos para manejar bases de datos, tablas, columnas, y registros. La clase principal, SimuladorPostgres, permite a los usuarios ejecutar comandos de base de datos como crear y seleccionar bases de datos, definir tablas, insertar registros, consultar datos, y guardar o cargar datos desde archivos en formato JSON.

El simulador utiliza la biblioteca Gson para la serialización y deserialización de datos en JSON. Los comandos que el simulador puede procesar incluyen CREATE DATABASE, USE, CREATE TABLE, INSERT INTO, SELECT * FROM, SAVE, y LOAD. Los usuarios interactúan con el simulador a través de un bucle de comandos que imita una consola de PostgreSQL, donde se pueden gestionar las estructuras y registros de datos dinámicamente.

```
        import com.google.gson.*;
import java.io.*;
import java.util.*;

// Clase para definir una columna con nombre y tipo de dato
class Columna {
    String nombre;
    String tipo;

    public Columna(String nombre, String tipo) {
        this.nombre = nombre;
        this.tipo = tipo.toUpperCase();
    }
}

// Clase para un registro con datos din micos (mapeo columna-valor)
class Registro {
    Map<String, Object> datos = new LinkedHashMap<>();

    public Registro(List<Columna> columnas, List<Object> valores) {
        for (int i = 0; i < columnas.size(); i++) {
            datos.put(columnas.get(i).nombre, valores.get(i));
        }
    }

    @Override
    public String toString() {
        return datos.toString();
    }
}

// Clase para representar una tabla con columnas y registros
class Tabla {
    String nombre;
    List<Columna> columnas = new ArrayList<>();
    List<Registro> registros = new ArrayList<>();

    public Tabla(String nombre) {
        this.nombre = nombre;
    }

    public void definirColumna(String nombre, String tipo) {
        columnas.add(new Columna(nombre, tipo));
    }

    public void insertarRegistro(List<Object> valores) {
        if (valores.size() != columnas.size()) {
            throw new IllegalArgumentException("N mero de valores no coincide con el n mero de
        }
```

```
        List<Object> valoresProcesados = new ArrayList<>();
        for (int i = 0; i < columnas.size(); i++) {
            valoresProcesados.add(validarTipo(columnas.get(i).tipo, valores.get(i)));
        }

        registros.add(new Registro(columnas, valoresProcesados));
        System.out.println("Registro insertado en la tabla " + nombre + ".");
    }

    public void consultarRegistros() {
        if (registros.isEmpty()) {
            System.out.println("No hay registros en la tabla " + nombre + ".");
            return;
        }
        for (Registro registro : registros) {
            System.out.println(registro);
        }
    }

    private Object validarTipo(String tipo, Object valor) {
        switch (tipo) {
            case "INT":
                if (valor instanceof Integer) return valor;
                if (valor instanceof String) return Integer.parseInt((String) valor);
                break;
            case "DOUBLE":
                if (valor instanceof Double) return valor;
                if (valor instanceof String) return Double.parseDouble((String) valor);
                break;
            case "VARCHAR":
                if (valor instanceof String) return valor;
                break;
            case "DATE":
                if (valor instanceof String && ((String) valor).matches("\\d{4}-\\d{2}-\\d
                break;
            default:
                throw new IllegalArgumentException("Tipo de dato no soportado: " + tipo);
        }
        throw new IllegalArgumentException("El valor " + valor + " no es v lido para el ti
    }
}

// Clase principal para el simulador
public class SimuladorPostgres {
    private Map<String, Map<String, Tabla>> basesDeDatos = new HashMap<>();
    private String baseDeDatosActual;

    public void crearBaseDeDatos(String nombre) {
        basesDeDatos.put(nombre, new HashMap<>());
        baseDeDatosActual = nombre;
        System.out.println("Base de datos creada: " + nombre);
    }

    public void usarBaseDeDatos(String nombre) {
        if (basesDeDatos.containsKey(nombre)) {
            baseDeDatosActual = nombre;
            System.out.println("Usando base de datos: " + nombre);
        } else {
            System.out.println("Base de datos " + nombre + " no existe.");
```

```
    }
}

public void crearTabla(String nombreTabla, List<Columna> columnas) {
    if (baseDeDatosActual == null) {
        System.out.println("Primero selecciona una base de datos con el comando 'USE <base>
        return;
    }
    Tabla tabla = new Tabla(nombreTabla);
    for (Columna columna : columnas) {
        tabla.definirColumna(columna.nombre, columna.tipo);
    }
    basesDeDatos.get(baseDeDatosActual).put(nombreTabla, tabla);
    System.out.println("Tabla creada: " + nombreTabla);
}

public void insertarEnTabla(String nombreTabla, List<Object> valores) {
    if (baseDeDatosActual == null) {
        System.out.println("Primero selecciona una base de datos con el comando 'USE <base>
        return;
    }
    Tabla tabla = basesDeDatos.get(baseDeDatosActual).get(nombreTabla);
    if (tabla == null) {
        System.out.println("La tabla " + nombreTabla + " no existe.");
        return;
    }
    tabla.insertarRegistro(valores);
}

public void consultarTabla(String nombreTabla) {
    if (baseDeDatosActual == null) {
        System.out.println("Primero selecciona una base de datos con el comando 'USE <base>
        return;
    }
    Tabla tabla = basesDeDatos.get(baseDeDatosActual).get(nombreTabla);
    if (tabla == null) {
        System.out.println("La tabla " + nombreTabla + " no existe.");
        return;
    }
    tabla.consultarRegistros();
}

public void guardarBaseDeDatos(String nombreArchivo) {
    try (Writer writer = new FileWriter(nombreArchivo)) {
        Gson gson = new GsonBuilder().setPrettyPrinting().create();
        gson.toJson(basesDeDatos, writer);
        System.out.println("Base de datos guardada en " + nombreArchivo);
    } catch (IOException e) {
        System.out.println("Error al guardar la base de datos: " + e.getMessage());
    }
}

public void cargarBaseDeDatos(String nombreArchivo) {
    try (Reader reader = new FileReader(nombreArchivo)) {
        Gson gson = new Gson();
        basesDeDatos = gson.fromJson(reader, new HashMap<String, Map<String, Tabla>>() {}.ge
        System.out.println("Base de datos cargada desde " + nombreArchivo);
    } catch (IOException e) {
        System.out.println("Error al cargar la base de datos: " + e.getMessage());
```

```
    }
}

public void ejecutarPrompt() {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Simulador de PostgreSQL. Escribe tus comandos (escribe 'EXIT' p

    while (true) {
        String prompt = baseDeDatosActual != null ? baseDeDatosActual + "#" : "postgres
        System.out.print(prompt + " ");
        String input = scanner.nextLine().trim();

        if (input.equalsIgnoreCase("EXIT")) {
            System.out.println("Saliendo...");
            break;
        }

        try {
            procesarComando(input);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    scanner.close();
}

private void procesarComando(String comando) {
    if (comando.startsWith("CREATE DATABASE")) {
        String nombre = comando.split(" ")[2].replace(";", "");
        crearBaseDeDatos(nombre);
    } else if (comando.startsWith("USE")) {
        String nombre = comando.split(" ")[1].replace(";", "");
        usarBaseDeDatos(nombre);
    } else if (comando.startsWith("CREATE TABLE")) {
        String[] partes = comando.replace("CREATE TABLE ", "").split("\\(");
        String nombreTabla = partes[0].trim();
        String[] columnasDef = partes[1].replace(");", "").split(",");
        List<Columna> columnas = new ArrayList<>();
        for (String columna : columnasDef) {
            String[] def = columna.trim().split(" ");
            columnas.add(new Columna(def[0], def[1]));
        }
        crearTabla(nombreTabla, columnas);
    } else if (comando.startsWith("INSERT INTO")) {
        String[] partes = comando.replace("INSERT INTO ", "").split("VALUES");
        String nombreTabla = partes[0].split("\\(")[0].trim();
        String[] valores = partes[1].replace("(", "").replace(");", "").split(",");
        List<Object> listaValores = new ArrayList<>();
        for (String valor : valores) {
            valor = valor.trim();
            if (valor.startsWith("'") && valor.endsWith("'")) {
                listaValores.add(valor.replace("'", ""));
            } else if (valor.matches("\\d+")) {
                listaValores.add(Integer.parseInt(valor));
            } else if (valor.matches("\\d+\\.\\d+")) {
                listaValores.add(Double.parseDouble(valor));
            } else {
                listaValores.add(valor);
```

```
                }
            }
            insertarEnTabla(nombreTabla, listaValores);
        } else if (comando.startsWith("SELECT * FROM")) {
            String nombreTabla = comando.replace("SELECT * FROM ", "").replace(";", "").trim();
            consultarTabla(nombreTabla);
        } else if (comando.startsWith("SAVE")) {
            String nombreArchivo = comando.replace("SAVE ", "").replace(";", "").trim();
            guardarBaseDeDatos(nombreArchivo);
        } else if (comando.startsWith("LOAD")) {
            String nombreArchivo = comando.replace("LOAD ", "").replace(";", "").trim();
            cargarBaseDeDatos(nombreArchivo);
        } else {
            System.out.println("Comando no reconocido.");
        }
    }

    public static void main(String[] args) {
        SimuladorPostgres simulador = new SimuladorPostgres();
        simulador.ejecutarPrompt();
    }
}
```

1.6. Conclusiones.

- Luis Fernando Franco Arellano

  El desarrollo de este proyecto genero el complemento que necesitabamos para acabar de comprender como funcionan nuestras bases de datos en todos los niveles. Desde el modelo relacional a la implementación de vistas y disparadores. Además pudimos notar como estás bases de datos fueron complementadas con aplicaciones completas. Que son externas a las bases de datos que estuvimos creando

- Gustavo Isaac Soto Huerta

  Este proyecto me permitió aplicar mis conocimientos en bases de datos utilizando PostgreSQL. Al corregir errores en funciones y triggers, comprendí la importancia de la sintaxis y la consistencia en los nombres de tablas y columnas. Implementar un índice B-tree en `codigobarras` mejoró el rendimiento de las consultas, enseñándome cómo los índices optimizan una base de datos. Automatizar el folio de ventas con secuencias y triggers simplificó procesos y aseguró la integridad de los datos. Esta experiencia fue valiosa para mi desarrollo como ingeniero en computación, ya que apliqué teoría en situaciones prácticas y reforcé mis habilidades en SQL.

- José Eduardo Villeda Tlecuitl:

  Durante el desarrollo de este proyecto, se consiguió crear e implementar un sistema de base de datos sólido enfocado en optimizar la gestión de una cadena de papelerías. Iniciando con un análisis exhaustivo de los requerimientos proporcionados por la empresa, se diseñó un modelo entidad-relación que fue convertido en un modelo relacional eficiente. La implementación en PostgreSQL integró el uso de triggers, vistas e índices, lo que permitió automatizar procesos esenciales como la gestión de inventarios, la emisión de facturas y la detección de niveles bajos de stock. Además, se mejoró el rendimiento mediante la optimización de consultas críticas con índices específicos. Este sistema no solo agiliza las operaciones, sino que también ofrece herramientas avanzadas para el análisis de información, facilitando así una mejor toma de decisiones. En definitiva, este proyecto pone de manifiesto cómo un diseño de bases de datos bien estructurado puede impulsar la modernización y eficiencia en las operaciones empresariales.

- Eduardo Zavala Sánchez:

  El modelo esta diseñado para proporcionar una estructura en donde guardar la informacion de manera estructurada y eficiente para gestionar los requerimientos de la cadena de papelerias. Por lo que cumple con su proposito de diseño.

  La creación del proyecto fue un gran reto que me permitio poner en practica todos los conocimientos que se tenien de teoria, desde las primeras etapas en el modelado hasta la creacion del lenguaje y funciones, triggers. Me permitió reafirmar mis conocimienos y ademas de consolidarlos.

  La parte mas dificil del proyecto diria que fue el realizar el modelado de la base de datos mediante el modelo entidad relacion y relacional, puesto que una vez que se tiene bien definido el modelo lo demas a mi parecer no es tan dificl, otro reto fue estructurar bien los trigger y saber en que momento implementarlos.

  Un acierto me parece que fue la repartición de actividades y de trabajo, si bien todos colaboramos al momento de la creación del diseño creo que fue un ´punto clave repartitrnos las tareas, de otra manaera creo que no nos hubiera dado tiempo de terminar.

Bibliografía

[1] PostgreSQL Global Development Group, "PostgreSQL Documentation," [Online]. Available: https://www.postgresql.org/docs/. [Accessed: October 2023].

[2] E. F. Ribeiro, "pgModeler: PostgreSQL Database Modeler," [Software]. Available: https://pgmodeler.io/. [Accessed: October 2023].

[3] Dia Developers, "Dia Diagram Editor," [Software]. Available: https://wiki.gnome.org/Apps/Dia. [Accessed: October 2023].

[4] JFrame (Java Platform SE 7). (2020, 24 junio). https://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html. [Accessed: October 2023].

[5] . (2022, 25 enero). Oracle Help Center. [Accessed: October 2023]. https://docs.oracle.com/en/java/javase/17/