Proyecto #2: Tienda Virtual “MegaShop”

Tercera Entrega

GRUPO 3:

Angelo Alexander Arango Graciano

Laura Mejia Delgado

Selvio Bedoya Heredia

Joimar Danilo Urrego David

Docente: Jeisson Ibarguen Maturana

Asignatura: Bases de datos II

05/06/2025

Medellin, Colombia

Tabla de Contenido

Introducción

En el siguiente documento se presenta la documentación del proyecto de bases de datos II llamado “MegaShop”, la cual es una tienda virtual, este es un proyecto ficticio que busca mejorar el sistema de ventas e inventario a través de una base de datos para una tienda debido a que no tienen stock y no se generan reportes claros de las ventas realizadas.

Este proyecto fue desarrollado utilizando tecnologías como SQL Server y para hacer su respectiva integración con Firebase se utilizó javascript junto a node.js y como motor de bases de datos se utilizó Firestore Database.

Justificación

Nuestro cliente es una megatienda virtual llamada “MegaShop”, la cual en los últimos meses ha presentado un gran aumento de clientes y popularidad, lo ha generado varios problemas operativos. El principal problema es que no cuentan con un sistema de gestión de inventario, lo que causa que vendan productos que ya no se encuentran en stock, afectando así la experiencia del cliente y la confiabilidad del servicio.

Además, no se generan reportes claros sobre las ventas realizadas, lo que dificulta llevar un control sobre los ingresos, los productos más vendidos, los métodos de pago más utilizados y la gestión del stock restante. Por esta razón, se hace necesario desarrollar una base de datos que permita automatizar estos procesos y mantener la información actualizada.

El sistema propuesto debe permitir registrar productos, gestionar inventario, realizar ventas, emitir factura y disminuir automáticamente el stock cada vez que se realice una venta. También estará la posibilidad de generar reportes.

El proyecto surge como respuesta a las necesidades actuales de “MegaShop” y tienen como objetivo mejorar la eficiencia en la administración de inventario y ventas, asegurando una mejor experiencia tanto para el cliente como los administradores.

Objetivos

Para una mejor administración y control de las operaciones de la tienda virtual MegaShop, se ha planteado el desarrollo de este proyecto, el cual deberá cumplir con las necesidades establecidas por el cliente:

- El sistema contará con una interfaz gráfica intuitiva, que permita a los usuarios gestionar el inventario, ventas y reportes de manera eficiente.

- Por seguridad, habrán usuarios catalogados para la correcta gestión de la base de datos.

- Se debe entregar la documentación detallada al cliente, en la cual se describe el funcionamiento completo del sistema, la estructura de la base de datos, así como los procedimientos almacenados, vistas y triggers implementados.

- En la documentación se explicará paso a paso el desarrollo del sistema, la implementación de la base de datos y la integración con Firebase, permitiendo la sincronización de inventario en tiempo real con la tienda en línea.

- El sistema deberá disminuir de manera automática el stock de los productos al confirmar una venta, mediante el uso de triggers programados.

- Se deberán implementar procedimientos almacenados para calcular comisiones de ventas según el método de pago y generar reportes de ingresos por períodos.

- Se desarrollarán vistas que permitan monitorear el inventario actualizado, resaltando productos con bajo stock, así como las ventas diarias y sus estados.

- Finalmente, el sistema deberá garantizar la integridad y el correcto funcionamiento de todas las funciones, cumpliendo con las reglas de negocio establecidas desde el inicio del proyecto.

Reglas de negocio

1. **Cada producto tiene un stock inicial. Al venderse, se disminuye el stock (trigger al confirmar la venta).**

   Al momento de generar la tabla de **productos**, esta debe contener un campo de **stock** el cual debe estar inicializado en 250, a su vez debe haber un procedimiento almacenado que ejecutará las compras, esta debe contener un condicional que verifica si hay stock suficiente de un producto, cuyo mínimo valor debe ser 10.

2. **Existen varias formas de pago (tarjeta, PayPal, etc.) y se deben controlar los estados de pago.**

   La estructura de la tabla **factura**, debe contener un campo de **método de pago**, el cual será una llave foránea que llevará a otra tabla con los múltiples métodos de pago que puede ser un **enum**, en esta misma tabla de factura se maneja otro campo que será el estado de esta, la cual contendrá campos como de que si está pagada, pendiente o cancelada

3. **Procedimientos almacenados para:**

   - **Calcular comisiones de venta por método de pago.**

     Se dispondrá de una tabla de **comisiones** la cual guardará las comisiones que habrá según los **tipos de pagos**, cada **registro** tendrá su propio porcentaje que finalmente afectará a los **vendedores**.

- **Generar reporte de ingresos por período.**

    La base de datos dispondrá de un **procedimiento almacenado** que mostrará un reporte de los ingresos según **dos fechas** que se le pasen por parámetros.

4. **Vistas para monitorear:**

    - **Inventario actual (destacando productos con stock bajo).**

        La base de datos debe tener una vista donde se muestre el nombre, stock y tipo de todos los productos.

    - **Ventas diarias y su estado.**

        La base de datos debe tener una **vista** con el **reporte** de ventas diarias el cual contendrá la **cantidad de ventas** realizadas de cada **producto** por día y el **total** por **producto**.

5. **Deben haber descuentos para cierto tipo de productos x días.**

    Se planifica tener una tabla de **descuentos** donde se agregarán los descuentos que se realizarán a los productos según el tipo de estos.

6. **Se debe poder visualizar un historial con las compras de un cliente.**

    Creando una **vista** se pueden visualizar los datos del **historial de compras** de los **clientes**.

7. **Un usuario puede ser tanto cliente como vendedor.**

   En la estructura de las tablas de **vendedor** y **cliente,** los foreign key hacía la tabla usuario **no deben ser únicos** para permitir que se pueda registrar en ambas tablas, de forma que pueda ser tanto vendedor como cliente (Como si el vendedor estuviera en vacaciones y pudiera comprar en la tienda).

Requerimientos técnicos mínimos.

Software:

- Sistema de gestión de bases de datos: **SQL Server 2019** o superior.
- Entorno de desarrollo: **SQL Server Management Studio (SSMS)**.
- Sistema operativo compatible: **Windows 10 o superior**.

Hardware:

- Procesador: Intel Core i3 (8ª generación)
- Memoria RAM: mínimo 8 GB.
- Espacio en disco: mínimo 10 GB.
- Conexión a internet.

Modelo entidad relación

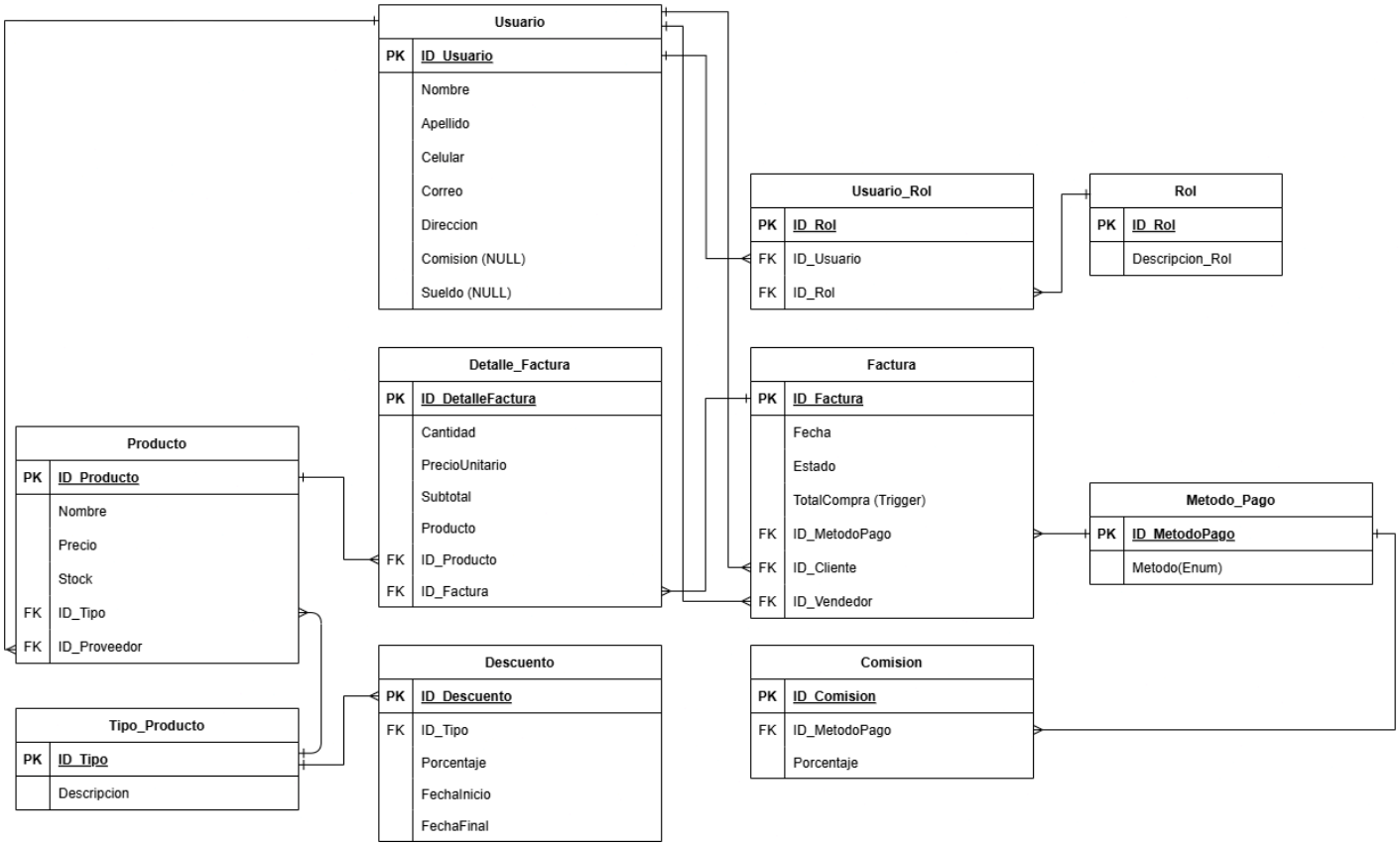

Tabla de datos

- **Tabla usuario**

  En esta tabla se registrará toda la información relacionada al usuario.

Campo	Tipo	Descripción
ID_Usuario	Llave primaria de tipo entero y auto incrementable	Identificador único del usuario
Nombre	Campo de tipo varchar (50)	Nombre(s) del usuario
Apellido	Campo de tipo varchar (50)	Apellido(s) del usuario
Celular	Campo de tipo varchar (20)	Número de teléfono móvil
Correo	Campo de tipo varchar (50)	Dirección de correo electrónico
Dirección	Campo de tipo varchar (50)	Domicilio del usuario
Comisión	Campo de tipo decimal (10,2)	Porcentaje de comisión (para empleados)
Sueldo	Campo de tipo decimal (10,2)	Salario base (para empleados)

- **Tabla rol**

  En esta tabla se registra los roles definidos dentro de la tienda.

Campo	Tipo	Descripción
ID_Rol	Llave primaria de tipo entero auto incrementable	Identificador único del rol
Descripcion_Rol	Campo de tipo varchar (100)	Nombre o descripción del rol

- **Tabla Usuario_Rol**

  En esta tabla se registran los roles que les corresponde a cada usuario dentro de la tienda.

Campo	Tipo	Descripción
ID_Rol	Llave primaria de tipo entero	Identificador del rol (relacionado con tabla rol)
ID_Usuario	Llave foránea de tipo entero	Identificador del usuario (relacionado con tabla usuario)

- **Tabla Método de Pago**

  En esta tabla se almacenará toda la información relacionada con los pagos de los productos.

Campo	Tipo	Descripción
ID_MetodoPago	Llave primaria de tipo entero auto incrementable	Identificador único del método de pago
Método	Campo de tipo varchar (50)	Nombre del método (ej: "Tarjeta de crédito")

- **Tabla comisión**

  En esta tabla están ubicadas las comisiones respectivas a cada trabajador de la tienda.

Campo	Tipo	Descripción
ID_Comision	Llave primaria de tipo entero auto incrementable	Identificador único de la comisión
ID_MetodoPago	Llave foránea de tipo entero	Método de pago asociado (relacionado con Método de Pago)
Porcentaje	Campo de tipo decimal (10,2)	Porcentaje de comisión aplicable

- **Tabla Tipo_Producto**

  En esta tabla se registran los tipos de todos los productos disponibles en la tienda.

Campo	Tipo	Descripción
ID_Tipo	Llave primaria de tipo entero auto incrementable	Identificador único del tipo de producto
Descripción	Campo de tipo varchar (100)	Nombre de la categoría (ej: "Electrodomésticos")

- **Tabla Producto**

  En esta tabla se almacena toda la información de los productos de la tienda.

Campo	Tipo	Descripción
ID_Producto	Llave primaria de tipo entero auto incrementable	Identificador único del producto
Nombre	Campo de tipo varchar (100)	Nombre del producto
Precio	Campo de tipo decimal (10,2)	Precio unitario
Stock	Campo de tipo entero	Cantidad disponible en inventario
ID_Tipo	Llave foránea de tipo entero	Tipo de producto (relacionado con Tipo_Producto)
ID_Proveedor	Llave foránea de tipo entero	Identificador del proveedor

- **Tabla Descuentos**

  En esta tabla se almacena toda la información relacionada a los descuentos de los productos.

Campo	Tipo	Descripción
ID_Descuento	Llave primaria de tipo entero auto incrementable	Identificador único del descuento
ID_Tipo	Llave foránea de tipo entero	Tipo de producto con descuento (relacionado con Tipo_Producto)

Porcentaje	Campo de tipo varchar (20)	Porcentaje o valor del descuento
Fecha_Inicio	Campo de tipo Date	Fecha de inicio de la promoción
Fecha_Final	Campo de tipo Date	Fecha de finalización de la promoción

- **Tabla Factura**

  En esta tabla se registra toda la información de la compra de productos.

Campo	Tipo	Descripción
ID_Factura	Llave primaria de tipo entero auto incrementable	Identificador único de la factura
Fecha	Campo de tipo Date	Fecha de emisión
Estado	Campo varchar (20)	Estado de la factura (ej: "Pagada", "Pendiente")
Total_Compra	Campo de tipo decimal (10,2)	Monto total de la compra
ID_MetodoPago	Llave foránea de tipo entero	Método de pago utilizado (relacionado con Método de Pago)
ID_Cliente	Llave foránea de tipo entero	Cliente asociado (relacionado con usuario)
ID_Vendedor	Llave foránea de tipo entero	Empleado que realizó la venta (relacionado con usuario)

- **Tabla Detalle Factura**

  En esta tabla se almacenarán todos los detalles de la factura realizada por una compra.

Campo	Tipo	Descripción
ID_DetalleFactura	Llave primaria de tipo entero auto incrementable	Identificador único del detalle
Cantidad	Campo de tipo entero	Cantidad de unidades compradas
PrecioUnitario	Campo de tipo decimal (10,2)	Precio por unidad al momento de la compra
Subtotal	Campo de tipo decimal (10,2)	Monto parcial (Cantidad × PrecioUnitario)
ID_Producto	Llave foránea de tipo entero	Producto comprado (relacionado con Producto)
ID_Factura	Llave foránea de tipo entero	Factura asociada (relacionado con Factura)

Documentación de código SQL y relaciones

- **Creación de la Base de Datos**

```sql
CREATE DATABASE DB_MegaShoop
USE DB_MegaShoop
```

- **Estructura de Tablas Principales**

  - **Tabla Usuario**

    Esta tabla centraliza toda la información de usuarios del sistema (clientes, vendedores, proveedores y administradores).

```sql
CREATE TABLE Usuario (
    ID_Usuario INT IDENTITY(1,1) PRIMARY KEY,
    Nombre VARCHAR(50) NOT NULL,
    Apellido VARCHAR(50) NOT NULL,
    Celular VARCHAR(20),
    Correo VARCHAR(100) UNIQUE NOT NULL,
    Direccion VARCHAR(150),
    Comision DECIMAL(10, 2) NULL,
    Sueldo DECIMAL(10, 2) NULL
);
```

➢ **Características importantes:** Los campos Comision y Sueldo son opcionales (NULL) ya que solo aplican para empleados.

- **Sistema de Roles**

```sql
CREATE TABLE Rol (
    ID_Rol INT IDENTITY(1,1) PRIMARY KEY,
    Descripcion_Rol VARCHAR(50) NOT NULL
);

CREATE TABLE Usuario_Rol (
    ID_Rol INT NOT NULL,
    ID_Usuario INT NOT NULL,
    PRIMARY KEY (ID_Rol, ID_Usuario),
    FOREIGN KEY (ID_Rol) REFERENCES Rol(ID_Rol),
    FOREIGN KEY (ID_Usuario) REFERENCES Usuario(ID_Usuario)
);
```

➢ **Ventajas del diseño:** Un usuario puede tener múltiples roles (cliente y vendedor simultáneamente).

- **Gestión de Productos e Inventario**

```sql
CREATE TABLE Tipo_Producto (
    ID_Tipo INT IDENTITY(1,1) PRIMARY KEY,
    Descripcion VARCHAR(100) NOT NULL
);

CREATE TABLE Producto (
    ID_Producto INT IDENTITY(1,1) PRIMARY KEY,
    Nombre VARCHAR(100),
    Precio DECIMAL(10, 2),
    Stock INT,
    ID_Tipo INT,
    ID_Proveedor INT,
    FOREIGN KEY (ID_Tipo) REFERENCES Tipo_Producto(ID_Tipo),
    FOREIGN KEY (ID_Proveedor) REFERENCES Usuario(ID_Usuario)
);
```

- **Sistema de Facturación**

```sql
CREATE TABLE Factura (
    ID_Factura INT IDENTITY(1,1) PRIMARY KEY,
    Fecha DATETIME NOT NULL DEFAULT GETDATE(),
    Estado VARCHAR(20) NOT NULL DEFAULT 'PENDIENTE' CHECK (Estado IN ('Pendiente', 'Pagada', 'Cancelada'))
    TotalCompra DECIMAL(12, 2),
    ID_MetodoPago INT,
    ID_Cliente INT,
    ID_Vendedor INT,
    FOREIGN KEY (ID_MetodoPago) REFERENCES Metodo_Pago(ID_MetodoPago),
    FOREIGN KEY (ID_Cliente) REFERENCES Usuario(ID_Usuario),
    FOREIGN KEY (ID_Vendedor) REFERENCES Usuario(ID_Usuario)
);
```

- ## Relaciones Claves

1. **Usuario** → **Múltiples Roles**: Relación muchos a muchos
2. **Producto** → **Proveedor**: Cada producto tiene un proveedor (Usuario con rol proveedor)
3. **Factura** → **Cliente/Vendedor**: Referencias a la misma tabla Usuario pero roles diferentes
4. **Detalle_Factura**: Tabla que conecta Factura con Producto

Documentación de triggers.

- ## Trigger 1 - Validación de stock

  - **Nombre del trigger/evento:** trg_DetalleFactura
  - **Tabla asociada:** Detalle_Factura
  - **Condición de activación:** INSTEAD OF INSERT (Se ejecuta antes de insertar un registro)
  - **Acción realizada:** Valida que exista stock suficiente del producto antes de permitir la venta. Si hay stock disponible, calcula automáticamente el subtotal e inserta el detalle de factura. Si no hay stock suficiente, cancela la operación con un mensaje de error.

```sql
CREATE TRIGGER trg_DetalleFactura
ON Detalle_Factura
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;

    -- Verifica si algún producto tiene stock menor a 10
    IF EXISTS (
        SELECT 1
        FROM inserted I
        INNER JOIN Producto P ON I.ID_Producto = P.ID_Producto
        WHERE P.Stock < 10
    )
    BEGIN
        RAISERROR ('Error: Uno o más productos tienen stock menor al mínimo permitido (10). Venta no permitida.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END
```

```sql
    -- Si todo está bien, calcular subtotal e insertar
    INSERT INTO Detalle_Factura (Cantidad, PrecioUnitario, Subtotal, ID_Producto, ID_Factura)
    SELECT
        I.Cantidad,
        I.PrecioUnitario,
        I.Cantidad * I.PrecioUnitario AS Subtotal,
        I.ID_Producto,
        I.ID_Factura
    FROM inserted I;
END;
```

- ## Trigger 2 - Actualizar Stock

- **Nombre del trigger/evento:** trg_Actualizar_DetalleFactura
- **Tabla asociada:** Detalle_Factura
- **Condición de activación:** AFTER INSERT (Se ejecuta después de insertar exitosamente un registro)
- **Acción realizada:** Actualiza automáticamente el stock del producto restando la cantidad vendida y actualiza el total de la factura sumando el subtotal del detalle insertado.

```
CREATE TRIGGER trg_Actualizar_DetalleFactura
ON Detalle_Factura
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    -- Actualizar el stock del producto vendido
    UPDATE P
    SET P.Stock = P.Stock - I.Cantidad
    FROM Producto P
    INNER JOIN inserted I ON P.ID_Producto = I.ID_Producto;

    -- Actualizar el total de la factura sumando el subtotal
    UPDATE F
    SET F.TotalCompra = ISNULL(F.TotalCompra, 0) + I.Subtotal
    FROM Factura F
    INNER JOIN inserted I ON F.ID_Factura = I.ID_Factura;
END;
```

Flujo de trabajo:

1. Se intenta insertar un dato en Detalle_Factura
2. Se dispara trg_DetalleFactura para validar el stock disponible
3. Si hay stock suficiente, permite el insert
4. Se dispara trg_Actualizar_DetalleFactura actualiza stock y total automáticamente

Documentación de procedimiento almacenados.

- **Procedimiento 1: Cálculo de Comisiones**

- **Nombre:** CalcularComisionVenta
- **Objetivo del procedimiento:** Calcular y actualizar automáticamente las comisiones de los vendedores basándose en el método de pago utilizado y el total de la venta realizada.
- **Parámetros de entrada y salida:**

Parámetro	Tipo	Entrada/Salida	Descripción
p_factura_id	INT	Entrada	ID de la factura para la cual se calcula la comisión.
total_venta	DECIMAL(10,2)	Salida	Valor total de la factura.
porcentaje_comision	DECIMAL(5,2)	Salida	Porcentaje aplicado según el método de pago.
comision_ganada	DECIMAL(10,2)	Salida	Monto calculado como comisión para el vendedor.

- **Ejemplo de ejecución**:

```
-- Calcular comisión para la factura ID 5
EXEC CalcularComisionVenta @p_factura_id = 5;
```

- **Comentarios adicionales**: El procedimiento solo procesa facturas en estado 'Pagada' y acumula las comisiones al total existente del vendedor. Los porcentajes de comisión varían según el método de pago (Efectivo: 10%, Tarjeta: 5%, PSE: 3%).

```sql
CREATE PROCEDURE CalcularComisionVenta
    @p_factura_id INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE
        @metodo_pago_id INT,
        @vendedor_id INT,
        @total_venta DECIMAL(10,2),
        @porcentaje_comision DECIMAL(5,2),
        @comision_calculada DECIMAL(10,2);

    -- Obtener datos de la factura
    SELECT
        @metodo_pago_id = ID_MetodoPago,
        @vendedor_id = ID_Vendedor,
        @total_venta = TotalCompra
    FROM Factura
    WHERE ID_Factura = @p_factura_id AND Estado = 'Pagada';

    -- Si la factura existe y esta pagada
    IF @total_venta IS NOT NULL
    BEGIN
        -- Obtener el porcentaje de comision para este metodo de pago
        SELECT @porcentaje_comision = Porcentaje
        FROM Comision
        WHERE ID_MetodoPago = @metodo_pago_id;

        -- Calcular la comision
        SET @comision_calculada = @total_venta * (@porcentaje_comision / 100);

        -- Actualizar la comision del vendedor (acumulandola)
        UPDATE Usuario
        SET Comision = ISNULL(Comision, 0) + @comision_calculada
        WHERE ID_Usuario = @vendedor_id;

        -- Devolver el resultado
        SELECT
            @total_venta AS total_venta,
            @porcentaje_comision AS porcentaje_comision,
            @comision_calculada AS comision_ganada;
    END
    ELSE
    BEGIN
      -- Si la factura no existe o no esta pagada
      RAISERROR ('Error: Factura no encontrada o no est♦ en estado Pagada', 16, 1);
   END
END;
```

- ## Procedimiento 2: Reporte de Ingresos

- **Nombre:** ReporteIngresosPorPeriodo
- **Objetivo del procedimiento:** Generar un reporte detallado de ingresos, facturas y estadísticas de ventas dentro de un rango de fechas específico, agrupado por fecha y método de pago.
- **Parámetros de entrada y salida:**

Parámetro	Tipo	Entrada/Salida	Descripción
@fecha_inicio	DATE	Entrada	Fecha de inicio del periodo a consultar.
@fecha_fin	DATE	Entrada	Fecha de finalización del periodo.
Fecha	DATE	Salida	Día específico dentro del rango consultado.
Total_Facturas	INT	Salida	Cantidad total de facturas generadas en ese día.
Ingresos_Totales	DECIMAL	Salida	Suma total de ingresos por ventas.
Promedio_Por_Factura	DECIMAL	Salida	Suma total de ingresos por ventas.
Metodo_Pago	VARCHAR	Salida	Nombre del método de pago utilizado.
Facturas_Pagadas	INT	Salida	Cantidad de facturas pagadas en ese día.
Facturas_Pendientes	INT	Salida	Cantidad de facturas pendientes en ese día.

- **Ejemplo de ejecución:**

```
-- Reporte de ingresos del último mes
EXEC ReporteIngresosPorPeriodo
    @fecha_inicio = '2024-01-01',
    @fecha_fin = '2024-01-31';
```

- **Comentarios adicionales:** El reporte incluye total de facturas, ingresos totales, promedio por factura, y cuenta de facturas por estado (pagadas/pendientes). Los resultados se ordenan por fecha descendente e ingresos totales.

```
CREATE PROCEDURE ReporteIngresosPorPeriodo
    @fecha_inicio DATE,
    @fecha_fin DATE
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        CAST(F.Fecha AS DATE) as Fecha,
        COUNT(F.ID_Factura) as Total_Facturas,
        SUM(F.TotalCompra) as Ingresos_Totales,
        AVG(F.TotalCompra) as Promedio_Por_Factura,
        MP.Metodo as Metodo_Pago,
        COUNT(CASE WHEN F.Estado = 'Pagada' THEN 1 END) as Facturas_Pagadas,
        COUNT(CASE WHEN F.Estado = 'Pendiente' THEN 1 END) as Facturas_Pendientes
    FROM Factura F
    INNER JOIN Metodo_Pago MP ON F.ID_MetodoPago = MP.ID_MetodoPago
    WHERE CAST(F.Fecha AS DATE) BETWEEN @fecha_inicio AND @fecha_fin
    GROUP BY CAST(F.Fecha AS DATE), MP.Metodo
    ORDER BY Fecha DESC, Ingresos_Totales DESC;
END;
```

Documentación de vistas.

- **Vista 1: Inventario Actual con Stock Bajo**

```sql
CREATE VIEW VistaInventarioActual AS
SELECT
    P.ID_Producto,
    P.Nombre as Nombre_Producto,
    P.Precio,
    P.Stock,
    TP.Descripcion as Tipo_Producto,
    U.Nombre + ' ' + U.Apellido as Proveedor,
    CASE
        WHEN P.Stock <= 10 THEN 'STOCK BAJO'
        WHEN P.Stock <= 50 THEN 'STOCK MEDIO'
        ELSE 'STOCK ALTO'
    END as Estado_Stock,
    P.Stock * P.Precio as Valor_Inventario
FROM Producto P
INNER JOIN Tipo_Producto TP ON P.ID_Tipo = TP.ID_Tipo
INNER JOIN Usuario U ON P.ID_Proveedor = U.ID_Usuario;
```

- **Vista 2: Ventas Diarias y Estados**

```sql
CREATE VIEW VistaVentasDiarias AS
SELECT
    CAST(F.Fecha AS DATE) as Fecha_Venta,
    F.Estado,
    COUNT(F.ID_Factura) as Cantidad_Ventas,
    SUM(F.TotalCompra) as Total_Ingresos,
    AVG(F.TotalCompra) as Promedio_Venta,
    MP.Metodo as Metodo_Pago,
    COUNT(DISTINCT F.ID_Cliente) as Clientes_Diferentes,
    COUNT(DISTINCT F.ID_Vendedor) as Vendedores_Activos
FROM Factura F
INNER JOIN Metodo_Pago MP ON F.ID_MetodoPago = MP.ID_MetodoPago
GROUP BY CAST(F.Fecha AS DATE), F.Estado, MP.Metodo
ORDER BY Fecha_Venta DESC;
```

- **Vista 3: Historial de Compras por Cliente**

```sql
CREATE VIEW VistaHistorialCliente AS
SELECT
    C.ID_Usuario as ID_Cliente,
    C.Nombre + ' ' + C.Apellido as Cliente,
    C.Correo,
    F.ID_Factura,
    F.Fecha,
    F.Estado,
    F.TotalCompra,
    MP.Metodo as Metodo_Pago,
    V.Nombre + ' ' + V.Apellido as Vendedor,
    COUNT(DF.ID_DetalleFactura) as Productos_Comprados
FROM Usuario C
INNER JOIN Factura F ON C.ID_Usuario = F.ID_Cliente
INNER JOIN Metodo_Pago MP ON F.ID_MetodoPago = MP.ID_MetodoPago
INNER JOIN Usuario V ON F.ID_Vendedor = V.ID_Usuario
LEFT JOIN Detalle_Factura DF ON F.ID_Factura = DF.ID_Factura
GROUP BY C.ID_Usuario, C.Nombre, C.Apellido, C.Correo, F.ID_Factura,
         F.Fecha, F.Estado, F.TotalCompra, MP.Metodo, V.Nombre, V.Apellido;
```

- **Vista 4: Productos con Descuentos Activos**

```sql
CREATE VIEW VistaProductosConDescuento AS
SELECT
    P.ID_Producto,
    P.Nombre as Producto,
    P.Precio as Precio_Original,
    D.Porcentaje as Descuento_Porcentaje,
    P.Precio * (1 - D.Porcentaje/100) as Precio_Con_Descuento,
    P.Precio * (D.Porcentaje/100) as Ahorro,
    D.FechaInicio,
    D.FechaFinal,
    TP.Descripcion as Tipo_Producto,
    DATEDIFF(DAY, GETDATE(), D.FechaFinal) as Dias_Restantes
FROM Producto P
INNER JOIN Tipo_Producto TP ON P.ID_Tipo = TP.ID_Tipo
INNER JOIN Descuento D ON TP.ID_Tipo = D.ID_Tipo
WHERE GETDATE() BETWEEN D.FechaInicio AND D.FechaFinal;
```

Documentación de despliegue funcional.

Conclusiones.

Bibliografía.