UTN|FRGP
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL GENERAL PACHECO

Tecnicatura Universitaria en Programación

Base de Datos II

Equipo # 51

Integrantes:

Donozo, Roman Agustín – Legajo: 30319

Morandi, Valeria – Legajo: 27890

Vudi, Tomás Valentín – Legajo: 30651

Zarate, Matias Leandro - Legajo: 25632

Contenido

1. Explicación del Sistema

El sistema desarrollado permite gestionar de forma integral las operaciones de un supermercado de pequeña escala, brindando soporte a los procesos de compra, venta, control de stock y administración del personal.
Está diseñado sobre una base de datos relacional que centraliza toda la información de artículos, clientes, proveedores, empleados, movimientos de inventario y pagos, garantizando la integridad y consistencia de los datos.
El objetivo principal del sistema es optimizar la administración comercial y contable del supermercado, permitiendo registrar de manera precisa cada transacción y disponer de información actualizada para la toma de decisiones estratégicas y operativas.

Funcionalidades principales

- **Gestión de artículos y categorías:**
  Permite registrar, clasificar y mantener actualizada la información de los productos comercializados en el supermercado.
  Cada artículo se asocia a una categoría y posee datos de precio de compra, precio de venta, stock disponible y estado (activo/inactivo).

- **Gestión de clientes, proveedores y empleados:**
  Todos los actores del sistema se gestionan a través de la tabla común **Personas**, lo que evita duplicidades.
  Desde esta entidad se derivan los roles de cliente, proveedor o empleado, facilitando el mantenimiento centralizado de la información personal y de contacto.

- **Registro de compras:**
  El sistema permite registrar las compras efectuadas a proveedores, indicando los artículos adquiridos, cantidades, precios unitarios y forma de pago.
  Cada compra se detalla en la tabla **ComprasDetalle**, y al confirmarse, se actualiza automáticamente el stock de los artículos involucrados mediante los movimientos registrados en **MovimientosArticulos**.

- **Registro de ventas:**
  Se registran las operaciones de venta a clientes, vinculando los artículos vendidos, sus cantidades y precios de venta.
  Cada venta se compone de su respectivo detalle (**VentasDetalle**) y, al concretarse, el sistema descuenta el stock correspondiente, generando automáticamente los movimientos de salida en el inventario.

- **Gestión de stock y movimientos:**
  La tabla **MovimientosArticulos** centraliza todas las entradas y salidas de productos, ya sea por compras, ventas o ajustes de inventario.
  Esto permite conocer en todo momento el stock real de cada artículo y mantener la trazabilidad de sus variaciones.

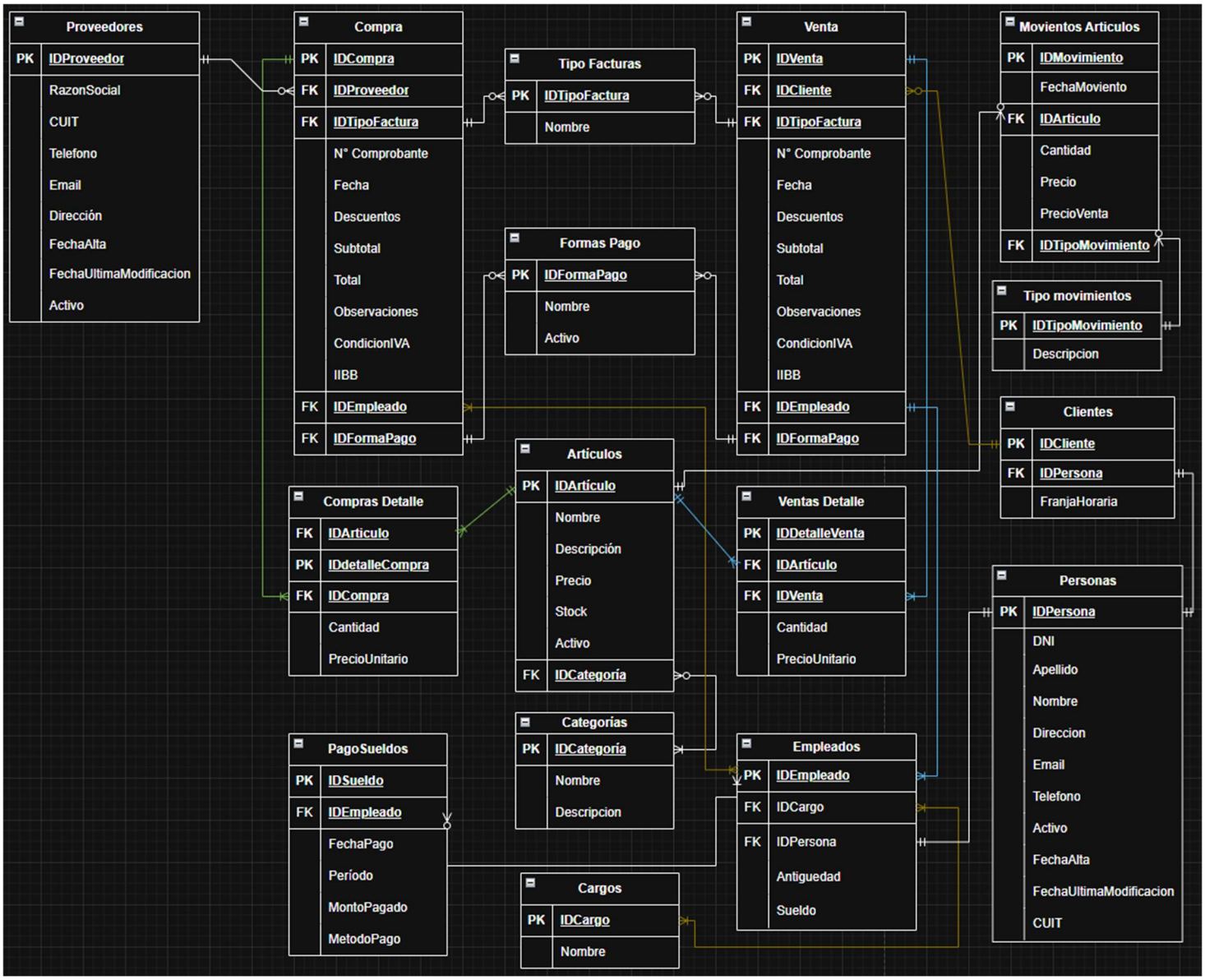
- **Gestión de empleados y pagos de sueldos:**
  El sistema almacena los datos del personal, sus cargos y sueldos, y permite registrar los pagos periódicos realizados a cada empleado a través de la tabla **PagoSueldos**, que se vincula a las **Formas de Pago** habilitadas.

- **Gestión de formas de pago:**
  Las operaciones comerciales (compras, ventas y pagos de sueldos) se asocian a una forma de pago registrada en la tabla **FormasPago**, donde se pueden activar, desactivar o agregar nuevos métodos según la necesidad del negocio.

- **Consultas y reportes:**
  El modelo de datos soporta consultas complejas a través de **vistas predefinidas**, que permiten generar reportes de ventas por cliente, compras por proveedor, stock valorizado y gastos en sueldos.
  Estas vistas facilitan el análisis económico-financiero del supermercado y la toma de decisiones gerenciales.

El sistema está construido con una base de datos relacional que almacena toda la información necesaria para gestionar estas funcionalidades.

2. Diagrama de Entidad Relación



3. Objetos de Base de Datos clave en el sistema

3.1 Vistas

3.1.1 VW_VentasPorCliente

Descripción:
Permite obtener un resumen de las ventas realizadas a cada cliente, mostrando la cantidad de operaciones, monto total facturado y fecha de la última compra.

```
CREATE VIEW VW_VentasPorCliente AS
SELECT
    c.IdCliente,
    p.Apellido + ', ' + p.Nombre AS Cliente,
    COUNT(DISTINCT v.IdVenta) AS CantidadVentas,
    SUM(d.Cantidad * d.PrecioUnitario) AS TotalFacturado,
    MAX(v.Fecha) AS UltimaCompra
FROM Clientes c
INNER JOIN Personas p ON c.IdPersona = p.IdPersona
INNER JOIN Ventas v ON v.IdCliente = c.IdCliente
INNER JOIN VentasDetalles d ON d.IdVenta = v.IdVenta
GROUP BY c.IdCliente, p.Apellido, p.Nombre;
```

Nos arroja un listado de clientes con su total de compras y monto acumulado de ventas.

3.1.2 VW_ComprasPorProveedor

Descripción:
Muestra el total de compras realizadas a cada proveedor y la fecha de la última transacción.

```
CREATE VIEW VW_ComprasPorProveedor AS
SELECT
    pr.IdProveedor,
    pr.RazonSocial AS Proveedor,
    COUNT(DISTINCT c.IdCompra) AS CantidadCompras,
    SUM(cd.Cantidad * cd.PrecioUnitario) AS TotalComprado,
    MAX(c.Fecha) AS UltimaCompra
FROM Proveedores pr
INNER JOIN Compras c ON c.IDProveedor = pr.IDProveedor
INNER JOIN ComprasDetalles cd ON cd.IdCompra = c.IDCompra
GROUP BY pr.IdProveedor, pr.RazonSocial;
```

Esta vista nos muestra: resumen del volumen de compras por proveedor, mostrando los principales socios comerciales.

3.1.3 VW_StockValorizado

Descripción:
Calcula el valor total del inventario actual considerando cantidad y precio de compra.

```
CREATE VIEW VW_StockValorizado AS
SELECT
    a.IDArticulo,
    a.Nombre AS Articulo,
    a.Stock,
    a.Precio,
    (a.Stock * a.Precio) AS ValorInventario,
    c.Nombre AS Categoria
FROM Articulos a
INNER JOIN Categorias c ON a.IDCategoria = c.IDCategoria
WHERE a.Activo = 1;
```

Esta vista muestra el listado de artículos activos con su stock disponible, precio de compra y valor total en inventario, permitiendo evaluar el nivel y la valorización del stock actual.

3.1.4 VW_VentasPorClienteConNivel

Descripción:
Esta vista utiliza la función de usuario ***fn_NivelFacturacion*** para clasificar los clientes según su nivel de facturación total.

Amplía la vista original **VW_VentasPorCliente** agregando la columna **NivelFacturacion**, que categoriza los montos como BAJO, MEDIO o ALTO según el total vendido.

Devuelve el listado de clientes con su cantidad de compras, total facturado, último movimiento y el **nivel de facturación** (BAJO, MEDIO o ALTO) según los criterios definidos en la función fn_NivelFacturacion.

```
CREATE FUNCTION fn_NivelFacturacion (@Monto DECIMAL (10,2))
RETURNS VARCHAR (10)
AS
BEGIN
    DECLARE @Nivel VARCHAR (10);

    IF @Monto < 50000
        SET @Nivel = 'BAJO';
    ELSE IF @Monto BETWEEN 50000 AND 200000
        SET @Nivel = 'MEDIO';
    ELSE
        SET @Nivel = 'ALTO';

    RETURN @Nivel;
END;
```

```sql
CREATE VIEW VW_VentasPorClienteConNivel AS
SELECT
    c.IdCliente,
    p.Apellido + ', ' + p.Nombre AS Cliente,
    COUNT (DISTINCT v.IdVenta) AS CantidadVentas,
    SUM (d.Cantidad * d.PrecioUnitario) AS TotalFacturado,
    dbo.fn_NivelFacturacion(SUM (d.Cantidad * d.PrecioUnitario)) AS
NivelFacturacion,
    MAX(v.Fecha) AS UltimaCompra
FROM Ventas v
INNER JOIN VentasDetalles d ON v.IdVenta = d.IdVenta
INNER JOIN Clientes c ON v.IdCliente = c.IdCliente
INNER JOIN Personas p ON c.IdPersona = p.IdPersona
GROUP BY c.IdCliente, p.Apellido, p.Nombre;
```

3.2 Procedimientos almacenados

3.2.1 sp_RegistrarVenta

Descripción:
Registra una venta realizada a un cliente, actualiza el stock y deja trazabilidad del movimiento. Incluye control de existencia y manejo de errores para garantizar la integridad de datos.

```sql
CREATE PROCEDURE Sp_RegistrarVenta
(
    @IDCliente INT,
    @IDFormaPago INT,
    @IDTipoFactura INT,
    @NumComprobante INT,
    @IDEmpleado INT,
    @IDProducto INT,
    @Cantidad INT,
    @PrecioUnitario DECIMAL(10,2),
    @DescuentoAplicado MONEY = 0,
    @Observaciones NVARCHAR(250) = NULL,
    @CondicionIVA NVARCHAR(100) = NULL,
    @IIBB NVARCHAR(100) = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Subtotal MONEY;
    DECLARE @Total MONEY;
    DECLARE @IDVenta INT;

    SET @Cantidad = ISNULL(@Cantidad, 0);
    SET @PrecioUnitario = ISNULL(@PrecioUnitario, 0);
    SET @DescuentoAplicado = ISNULL(@DescuentoAplicado, 0);

    SET @Subtotal = @Cantidad * @PrecioUnitario;
```

```
    SET @Total = @Subtotal - @DescuentoAplicado;

    -- INSERT VENTA
    INSERT INTO Ventas (
        IDCliente,
        IDTipoFactura,
        NumComprobante,
        Fecha,
        Descuentos,
        Subtotal,
        Total,
        Observaciones,
        CondicionIVA,
        IIBB,
        IDEmpleado,
        IDFormaPago
    )
    VALUES (
        @IDCliente,
        @IDTipoFactura,
        @NumComprobante,
        GETDATE(),
        @DescuentoAplicado,
        @Subtotal,
        @Total,
        @Observaciones,
        @CondicionIVA,
        @IIBB,
        @IDEmpleado,
        @IDFormaPago
    );

    -- ID de la venta generada
    SET @IDVenta = SCOPE_IDENTITY();

    -- INSERT DETALLE
    INSERT INTO VentasDetalles (
        IDVenta,
        IDArticulo,
        Cantidad,
        PrecioUnitario
    )
    VALUES (
        @IDVenta,
        @IDProducto,
        @Cantidad,
        @PrecioUnitario
    );
END;
```

3.2.2 sp_RegistrarCompra

Descripción:
Registra una nueva compra a un proveedor, actualiza el stock de artículos y genera trazabilidad en movimientos.

```
CREATE PROCEDURE sp_RegistrarCompra
    @IdProveedor INT,
    @IdEmpleado INT,
    @IdFormaPago INT,
    @IDTipoFactura INT,
    @NumComprobante INT,
    @FechaCompra DATE,
    @IdArticulo INT,
    @Cantidad INT,
    @PrecioUnitario DECIMAL(10,2),
    @Observaciones NVARCHAR(250) = NULL,
    @CondicionIVA NVARCHAR(100) = NULL,
    @IIBB NVARCHAR(100) = NULL
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @IdCompra INT;
        DECLARE @Subtotal MONEY;
        DECLARE @Total MONEY;

        IF @Cantidad <= 0 OR @PrecioUnitario <= 0
            THROW 50010, 'La cantidad y el precio deben ser mayores
a cero.', 1;

        IF NOT EXISTS (SELECT 1 FROM Articulos WHERE IDArticulo =
@IdArticulo)
            THROW 50011, 'El artículo especificado no existe.', 1;

        SET @Subtotal = @Cantidad * @PrecioUnitario;
        SET @Total = @Subtotal;

        INSERT INTO Compras (
            IdProveedor, IDTipoFactura, NumComprobante, Fecha,
            Descuentos, Subtotal, Total,
            Observaciones, CondicionIVA, IIBB,
            IdEmpleado, IdFormaPago
        )
        VALUES (
            @IdProveedor, @IDTipoFactura, @NumComprobante, @Fecha-
Compra,
            0, @Subtotal, @Total,
            @Observaciones, @CondicionIVA, @IIBB,
            @IdEmpleado, @IdFormaPago
        );
```

```
        SET @IdCompra = SCOPE_IDENTITY();

        INSERT INTO ComprasDetalles (IdCompra, IdArticulo, Cantidad,
PrecioUnitario)
        VALUES (@IdCompra, @IdArticulo, @Cantidad, @PrecioUnitario);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

3.2.3 sp_RegistrarPagoSueldo

Descripción:
Registra un pago de sueldo a un empleado activo.
Si no se especifica el monto, toma el sueldo base del empleado.
Cuenta con manejo de errores y validación automática mediante trigger.

```
CREATE PROCEDURE sp_RegistrarPagoSueldo
    @IdEmpleado INT,
    @FechaPago DATE,
    @Monto DECIMAL(10,2) = NULL,
    @MetodoPago nvarchar(25)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (
            SELECT 1
            FROM Empleados E
            INNER JOIN Personas P ON E.IDPersona = P.IDPersona
            WHERE E.IDEmpleado = @IdEmpleado AND P.Activo = 1
        )
            THROW 53001, 'El empleado no existe o está inactivo.',
1;

        IF @Monto IS NULL
            SELECT @Monto = Sueldo FROM Empleados WHERE IDEmpleado =
@IdEmpleado;

        IF @Monto IS NULL OR @Monto <= 0
            THROW 53002, 'Monto de pago inválido.', 1;

        DECLARE @Periodo NVARCHAR(7);
        SET @Periodo = FORMAT(DATEADD(MONTH, -1, @FechaPago), 'yyyy-
MM');

        INSERT INTO PagoSueldos (IDEmpleado, FechaPago, Periodo,
MontoPagado, MetodoPago)
```

```
        VALUES (
            @IdEmpleado,
            @FechaPago,
            @Periodo,
            @Monto,
            @MetodoPago
        );
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

3.2.4 sp_EliminarProveedor

Descripción:
Permite eliminar o desactivar un proveedor dependiendo si tiene compras registradas.
Realiza eliminación lógica para mantener historial o física si no hay movimientos.

```
CREATE PROCEDURE Sp_EliminarProveedor
    @IdProveedor INT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Proveedores WHERE IDProveedor =
@IdProveedor)
          THROW 50001, 'El proveedor especificado no existe.', 1;

       IF EXISTS (SELECT 1 FROM Compras WHERE IDProveedor = @IdPro-
veedor)
        BEGIN

            UPDATE Proveedores
            SET Activo = 0,
                FechaUltimaModificacion = GETDATE()
            WHERE IDProveedor = @IdProveedor;
        END
        ELSE
        BEGIN

            DELETE FROM Proveedores
            WHERE IDProveedor = @IdProveedor;
        END

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
```

```
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

3.2.5 sp_ReporteGeneral (parametrizado y con función de usuario)

Descripción:
Este procedimiento genera un reporte consolidado mensual de las ventas, incluyendo el **nivel de facturación** calculado con la función de usuario `fn_NivelFacturacion`.
De esta manera, combina resultados numéricos con una clasificación cualitativa, reutilizando la lógica de la función previamente definida.

```
CREATE PROCEDURE sp_ReporteGeneral
    @FechaDesde DATE = NULL,
    @FechaHasta DATE = NULL,
    @IdCliente INT = NULL,
    @IdProveedor INT = NULL
AS
BEGIN
    PRINT '==== REPORTE GENERAL DEL SUPERMERCADO ====';
    PRINT ' ';


/*********************************************************************
*
      BLOQUE 1: VENTAS POR CLIENTE

*********************************************************************
/
    PRINT '>> VENTAS POR CLIENTE';
    SELECT
        c.[IdCliente],
        p.[Apellido] + ', ' + p.[Nombre] AS Cliente,
        COUNT(DISTINCT v.[IdVenta]) AS CantidadVentas,
        SUM(d.[Cantidad] * d.[PrecioUnitario]) AS TotalFacturado,
        dbo.fn_NivelFacturacion(SUM(d.[Cantidad] * d.[PrecioUnita-
rio])) AS NivelFacturacion,
        MAX(v.[Fecha]) AS UltimaCompra
    FROM [Ventas] v
    INNER JOIN [VentasDetalles] d ON v.[IdVenta] = d.[IdVenta]
    INNER JOIN [Clientes] c ON v.[IdCliente] = c.[IdCliente]
    INNER JOIN [Personas] p ON c.[IdPersona] = p.[IdPersona]
    WHERE
        (@IdCliente IS NULL OR c.[IdCliente] = @IdCliente)
        AND (@FechaDesde IS NULL OR v.[Fecha] >= @FechaDesde)
        AND (@FechaHasta IS NULL OR v.[Fecha] <= @FechaHasta)
    GROUP BY c.[IdCliente], p.[Apellido], p.[Nombre]
    ORDER BY TotalFacturado DESC;
```

```
    PRINT ' ';


/*********************************************************************
*
      BLOQUE 2: COMPRAS POR PROVEEDOR

*********************************************************************
/
    PRINT '>> COMPRAS POR PROVEEDOR';
    SELECT
        pr.[IdProveedor],
        pr.[RazonSocial] AS Proveedor,
        COUNT(DISTINCT c.[IdCompra]) AS CantidadCompras,
        SUM(cd.[Cantidad] * cd.[PrecioUnitario]) AS TotalComprado,
        dbo.fn_NivelFacturacion(SUM(cd.[Cantidad] * cd.[PrecioUnita-
rio])) AS NivelInversion,
        MAX(c.[Fecha]) AS UltimaCompra
    FROM [Compras] c
    INNER JOIN [ComprasDetalles] cd ON c.[IdCompra] = cd.[IdCompra]
    INNER JOIN [Proveedores] pr ON c.[IdProveedor] = pr.[IdProvee-
dor]
    WHERE
        (@IdProveedor IS NULL OR pr.[IdProveedor] = @IdProveedor)
        AND (@FechaDesde IS NULL OR c.[Fecha] >= @FechaDesde)
        AND (@FechaHasta IS NULL OR c.[Fecha] <= @FechaHasta)
    GROUP BY pr.[IdProveedor], pr.[RazonSocial]
    ORDER BY TotalComprado DESC;

    PRINT ' ';


/*********************************************************************
*
      BLOQUE 3: GASTOS EN SUELDOS
      ps.MontoPagado confirmado

*********************************************************************
/
    PRINT '>> GASTOS EN SUELDOS';
    SELECT
        e.[IdEmpleado],
        p.[Apellido] + ', ' + p.[Nombre] AS Empleado,
        SUM(ps.[MontoPagado]) AS TotalPagado,
        dbo.fn_NivelFacturacion(SUM(ps.[MontoPagado])) AS NivelGas-
toSueldo,
        MAX(ps.[FechaPago]) AS UltimoPago
    FROM [PagoSueldos] ps
    INNER JOIN [Empleados] e ON ps.[IdEmpleado] = e.[IdEmpleado]
    INNER JOIN [Personas] p ON e.[IdPersona] = p.[IdPersona]
    WHERE
        (@FechaDesde IS NULL OR ps.[FechaPago] >= @FechaDesde)
        AND (@FechaHasta IS NULL OR ps.[FechaPago] <= @FechaHasta)
    GROUP BY e.[IdEmpleado], p.[Apellido], p.[Nombre]
```

```
    ORDER BY TotalPagado DESC;

    PRINT ' ';


/*****************************************************************
*
      BLOQUE 4: STOCK VALORIZADO ACTUAL

*****************************************************************
/
    PRINT '>> STOCK VALORIZADO ACTUAL';
    SELECT
        a.[IdArticulo],
        a.[Nombre] AS Articulo,
        a.[Stock],
        a.[Precio],
        (a.[Stock] * a.[Precio]) AS ValorInventario,
        dbo.fn_NivelFacturacion((a.[Stock] * a.[Precio])) AS Nivel-
ValorInventario,
        c.[Nombre] AS Categoria
    FROM [Articulos] a
    INNER JOIN [Categorias] c ON a.[IdCategoria] = c.[IdCategoria]
    WHERE a.[Activo] = 1
    ORDER BY ValorInventario DESC;

    PRINT ' ';


/*****************************************************************
*
      BLOQUE 5: MOVIMIENTOS DE STOCK
      ma.IDMovimientoART y ma.PrecioVente confirmados

*****************************************************************
/
    PRINT '>> MOVIMIENTOS DE STOCK (ENTRADAS Y SALIDAS)';
    SELECT
        ma.[IDMovimientoART],
        ma.[FechaMovimiento],
        tm.[Descripcion] AS TipoMovimiento,
        a.[Nombre] AS Articulo,
        ma.[Cantidad],
        ma.[Precio],
        ma.[PrecioVenta] AS PrecioVenta
    FROM [MovimientosArticulos] ma
    INNER JOIN [TiposMovimientos] tm ON ma.[IDTipoMovimiento] =
tm.[IDTipoMovimiento]
    INNER JOIN [Articulos] a ON ma.[IDArticulo] = a.[IDArticulo]
    WHERE
        (@FechaDesde IS NULL OR ma.[FechaMovimiento] >= @FechaDesde)
        AND (@FechaHasta IS NULL OR ma.[FechaMovimiento] <= @Fe-
chaHasta)
    ORDER BY ma.[FechaMovimiento] DESC;
```

```
    PRINT ' ';
    PRINT '==== FIN DEL REPORTE GENERAL ====';
END;
```

3.3 Triggers

3.3.1 TR_ActualizarStockCompra

Descripción:
Este trigger se ejecuta automáticamente después de insertar un registro en la tabla **ComprasDetalles**.
Su función es actualizar el **stock** del artículo comprado y registrar el **movimiento de entrada** en la tabla **MovimientosArticulos**, asegurando la trazabilidad del inventario y la consistencia de los datos.

```
CREATE TRIGGER TR_ActualizarStockCompra
ON ComprasDetalles
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE A
    SET A.Stock = A.Stock + I.Cantidad
    FROM Articulos A
    INNER JOIN INSERTED I ON A.IDArticulo = I.IDArticulo;

    DECLARE @IdTipoEntrada INT;
    SELECT TOP 1 @IdTipoEntrada = IDTipoMovimiento FROM TiposMovi-
mientos WHERE Descripcion = 'ENTRADA';
    IF @IdTipoEntrada IS NULL
        THROW 52001, 'Falta el tipo de movimiento ENTRADA en Tipos-
Movimientos.', 1;

    INSERT INTO MovimientosArticulos (IDArticulo, FechaMovimiento,
IDTipoMovimiento, Cantidad, Precio, PrecioVenta)
    SELECT I.IDArticulo, GETDATE(), @IdTipoEntrada, I.Cantidad,
COALESCE(I.PrecioUnitario,0), 0
    FROM INSERTED I;
END;
```

3.3.2 TR_ActualizarStockVenta

Descripción:
Este trigger se ejecuta automáticamente después de registrar el detalle de una venta en la tabla **VentasDetalles**.

Su objetivo es mantener la coherencia del inventario actualizando el **stock** de los artículos vendidos y registrando el **movimiento de salida** correspondiente en la tabla **MovimientosArticulos**. De esta manera, cada vez que se concreta una venta, el sistema descuenta las unidades del producto involucrado y deja constancia del egreso, garantizando la trazabilidad de los movimientos y la integridad de los datos del stock sin requerir intervención manual.

```
CREATE TRIGGER TR_ActualizarStockVenta
ON VentasDetalles
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE A
    SET A.Stock = A.Stock - I.Cantidad
    FROM Articulos A
    INNER JOIN INSERTED I ON A.IDArticulo = I.IDArticulo;

    DECLARE @IdTipoSalida INT;
    SELECT TOP 1 @IdTipoSalida = IDTipoMovimiento FROM TiposMovi-
mientos WHERE Descripcion = 'SALIDA';
    IF @IdTipoSalida IS NULL
        THROW 52002, 'Falta el tipo de movimiento SALIDA en TiposMo-
vimientos.', 1;

    INSERT INTO MovimientosArticulos (IDArticulo, FechaMovimiento,
IDTipoMovimiento, Cantidad, Precio, PrecioVenta)
    SELECT I.IDArticulo, GETDATE(), @IdTipoSalida, I.Cantidad, 0,
COALESCE(I.PrecioUnitario,0)
    FROM INSERTED I;
END;
```

3.3.3 TR_ControlPagoSueldos

Descripción:
Evita pagos duplicados a un mismo empleado dentro del mismo mes y año.
Verifica los registros nuevos contra el historial de pagos y, si detecta más de un pago en el mismo período, genera un error e impide la operación.

```
CREATE TRIGGER TR_ControlPagoSueldos
ON PagoSueldos
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    ;WITH Nuevos AS (
        SELECT IdEmpleado, MONTH(FechaPago) AS MesPago, YEAR(Fecha-
Pago) AS AnioPago, COUNT(*) AS CantNuevos
        FROM INSERTED
        GROUP BY IdEmpleado, MONTH(FechaPago), YEAR(FechaPago)
    ),
```

```
    Existentes AS (
        SELECT P.IdEmpleado, MONTH(P.FechaPago) AS MesPago,
YEAR(P.FechaPago) AS AnioPago, COUNT(*) AS CantExist
        FROM PagoSueldos P
        INNER JOIN Nuevos N ON P.IdEmpleado = N.IdEmpleado
            AND MONTH(P.FechaPago) = N.MesPago
            AND YEAR(P.FechaPago) = N.AnioPago
        WHERE P.IDSueldo NOT IN (SELECT IDSueldo FROM INSERTED)
        GROUP BY P.IdEmpleado, MONTH(P.FechaPago), YEAR(P.FechaPago)
    ),
    Conflictos AS (
        SELECT N.IdEmpleado, N.MesPago, N.AnioPago, N.CantNuevos,
COALESCE(E.CantExist,0) AS CantExist,
               (N.CantNuevos + COALESCE(E.CantExist,0)) AS Total
        FROM Nuevos N
        LEFT JOIN Existentes E ON N.IdEmpleado = E.IdEmpleado AND
N.MesPago = E.MesPago AND N.AnioPago = E.AnioPago
    )
    SELECT * INTO #ConflictosTemp FROM Conflictos WHERE Total > 1;

    IF EXISTS (SELECT 1 FROM #ConflictosTemp)
    BEGIN
        DECLARE @msg NVARCHAR(400) = 'Conflicto: pago duplicado de-
tectado para empleado(s): ';
        SELECT @msg = @msg + CAST(IdEmpleado AS NVARCHAR(10)) + '
(mes=' + CAST(MesPago AS NVARCHAR(2)) + '/anio=' + CAST(AnioPago AS
NVARCHAR(4)) + '); '
        FROM #ConflictosTemp;

        DROP TABLE #ConflictosTemp;
        THROW 52100, @msg, 1;
    END

    DROP TABLE IF EXISTS #ConflictosTemp;
END;
```

3.3.4 Trg_ValidarComprasProveedor

Descripción:
Este Trigger se ejecuta automáticamente cuando se intenta realizar una eliminación física de un proveedor.
Recibe como parámetro al **IDProveedor** verificando su existencia. Si el proveedor no tiene compras registradas, se elimina físicamente de la base de datos. Caso contrario, se realiza una eliminación lógica cambiando el estado del campo “Activo” a false con el objetivo de mantener su historial.

```
CREATE TRIGGER Trg_ValidarComprasProveedor
ON Proveedores
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;
```

```
    DECLARE @IdProveedor INT;

    DECLARE proveedor_cursor CURSOR FOR
        SELECT IDProveedor FROM DELETED;

    OPEN proveedor_cursor;
    FETCH NEXT FROM proveedor_cursor INTO @IdProveedor;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF EXISTS (SELECT 1 FROM Compras WHERE IDProveedor = @IdPro-
veedor)
        BEGIN

            UPDATE Proveedores
            SET Activo = 0,
                FechaUltimaModificacion = GETDATE()
            WHERE IDProveedor = @IdProveedor;
        END
        ELSE
        BEGIN

            DELETE FROM Proveedores
            WHERE IDProveedor = @IdProveedor;
        END

        FETCH NEXT FROM proveedor_cursor INTO @IdProveedor;
    END

    CLOSE proveedor_cursor;
    DEALLOCATE proveedor_cursor;
END;
```

Links a los recursos

Script de creación de base de datos con datos

https://github.com/MatiasLeandroZarate/DB_Supermercado.git

Video demo del sistema (hasta 25 minutos)

https://drive.google.com/file/d/1eP083YtzKybwZXNTyd1XIPGqyd-7Lexy/view?usp=sharing