UNIVERSIDAD NACIONAL DE INGENIERÍA

Cultura de paz y derechos humanos.

Elaborado por:

Br. Aldo Omar

Molina Hernández

Br Yosuer Josué

Martínez Montenegro

Docente:

Evelyn Sanchez

Introducción:

En un entorno empresarial cada vez más dinámico y competitivo, la gestión eficiente de datos se ha vuelto esencial para el éxito de cualquier negocio, independientemente de su tamaño o sector. En este contexto, las pulperías, como el pequeño pero vital engranaje de las comunidades locales, no son una excepción. La pulpería se encuentra en una posición única para comprender la importancia de una gestión de datos eficiente, ya que se enfrenta diariamente a la gestión de ventas, compras y devoluciones de productos que abastecen a sus clientes.

Objetivo General:

Diseñar una Base de datos para el control de las compras y ventas de una pulpería

Objetivos Específicos:

Realización Personalizado de la Estructura de Base de Datos que se desarrollará una estructura de Base de Datos

Implementación de un Sistema de Seguimiento de Inventario Especializado se creará un sistema de seguimiento de inventario

Usuario Altamente Intuitiva se diseñará una interfaz de usuario altamente intuitiva que simplificará la entrada de datos en la Base de Datos.

Implementación de Rigurosos Mecanismos de Seguridad se establecerán mecanismos de seguridad sólidos y personalizados para proteger la confidencialidad.

Generación de Informes y Análisis Específicos para funciones de generación de informes y análisis que se adaptarán específicamente a las necesidades de la pulpería.

Antecedentes:

La gestión de datos en las pulperías y pequeños negocios ha evolucionado considerablemente en las últimas décadas. En el pasado, estas operaciones se basaban en registros manuales y métodos en papel, lo que a menudo resultaba en errores de registro y dificultades para acceder a información crucial. Sin embargo, la tecnología ha avanzado y ha proporcionado soluciones informáticas que han revolucionado la forma en que se lleva a cabo la gestión de datos en estos establecimientos.

La implementación de sistemas de gestión de bases de datos en pulperías y negocios similares ha demostrado ser altamente beneficiosa. Ha brindado una plataforma centralizada para el almacenamiento y la gestión de datos, lo que ha llevado a mejoras significativas en la eficiencia operativa y la calidad de la toma de decisiones. Además, la automatización de procesos ha liberado recursos que antes se dedicaban a tareas manuales y ha permitido una mayor atención a los clientes y un crecimiento empresarial sostenible.

Justificación:

La implementación de una Base de Datos en la pulpería "Reparto Schick" que gestiona ventas, compras y devoluciones es completamente justificada y se basa en múltiples factores fundamentales. Estos factores garantizan que este proyecto sea una inversión valiosa y estratégica para mejorar la gestión de la pulpería y su competitividad en el mercado local. Aquí se presentan las razones clave para justificar este proyecto:

1. **Optimización Integral de la Gestión:** La Base de Datos permitirá una gestión holística de las operaciones comerciales en "Reparto Schick". Esto abarcará la gestión de inventario, la contabilidad de ventas y compras, así como el seguimiento de las devoluciones. La integración de todas estas operaciones proporcionará una visión completa de la salud financiera de la pulpería.

2. **Precisión y Confianza en los Datos:** La automatización de procesos minimizará drásticamente la probabilidad de errores humanos en la entrada de datos y cálculos. Esto garantizará la precisión y la confiabilidad de la información almacenada, lo que es esencial para la toma de decisiones basada en datos precisos.

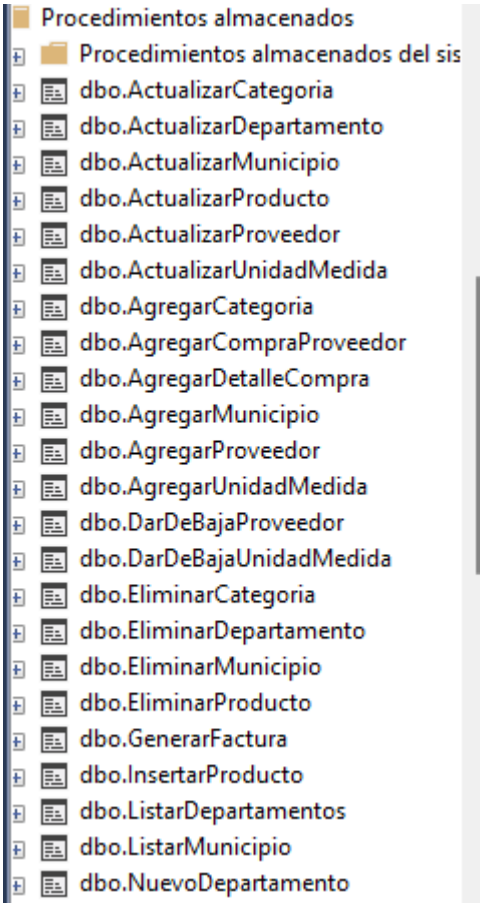
3. **Toma de Decisiones Informada:** La Base de Datos generará informes detallados y análisis personalizados que ofrecerán información valiosa sobre el rendimiento del negocio. Esto incluye la identificación de los productos más populares, la evaluación de la confiabilidad de los proveedores locales y el análisis de patrones de compra de los clientes leales. Estos datos informarán decisiones estratégicas para mejorar la rentabilidad y la satisfacción del cliente.

4. **Eficiencia Operativa y Reducción de Costos:** La automatización de procesos administrativos liberará tiempo y recursos que se podrán reinvertir en actividades que agreguen más valor al negocio. Esto incluye brindar un servicio al cliente más eficiente y centrarse en la expansión y el crecimiento sostenible de la pulpería.

5. **Mejora de la Satisfacción del Cliente:** La gestión eficiente del inventario y la capacidad de proporcionar información precisa sobre productos y transacciones permitirán a "Reparto Schick" brindar un servicio al cliente más rápido y confiable. Esto fomentará la satisfacción del cliente y la fidelidad a largo plazo.

Algebra relacional

Procedimientos almacenados.



Nuevo departamento

AgregarDepartamento(nombre: VARCHAR(30)) := ΔDepartamentos(Nombre) ← σNombre ≠ NULL ∧ Nombre ∉ πNombre(Departamentos);

```
Create procedure NuevoDepartamento
@Nombre nvarchar(30)
as
declare @VerificarNombre as nvarchar(30)
set @VerificarNombre = (select NombreDepartamento from Departamentos where NombreDepartamento=@Nombre)
If (@Nombre = '')
Begin
    print 'Nombre no puede estar vacio'
end
Else
Begin
    if(@Nombre=@VerificarNombre)
    Begin
        print 'No puede repetir'
    End
    Else
    Begin
    insert into Departamentos(NombreDepartamento) values (@Nombre)
    End
End
Go
select * from Departamentos
go
```

Eliminar Departamento

EliminarDepartamento(ID: INT) := ΔDepartamentos ← ρID/IdDepartamento(ID = IdDepartamento)(Departamentos);

```
--Eliminar Departamento
Create procedure EliminarDepartamento
@Identificador int
as
declare @IdDepartamento as int
set @IdDepartamento=(select IdDepartamento from Departamentos where IdDepartamento=@Identificador)
if(@Identificador=@IdDepartamento)
begin
  update Departamentos set Estado = 0 where IdDepartamento=@Identificador and Estado = 1
end
else
begin
   print 'Departamento no encontrado'
end
GO
```

Actualizar Departamento

ActualizarDepartamento(ID: INT, NuevoNombre: VARCHAR(30)) := ΔDepartamentos(Nombre) ← ρID/IdDepartamento(ID = IdDepartamento)(Departamentos) ⋈ πNombre(Departamentos) ⋈ Δ(ID, NuevoNombre);

```
--Actualizar Departamento
create procedure ActualizarDepartamento
@IdDepartamento int,
@NuevoNombreDepartamento nvarchar(30)
as
declare @IdentificadorID as int
set @IdentificadorID =(select IdDepartamento from Departamentos where IdDepartamento=@IdDepartamento)
declare @IdentificadorNombre as nvarchar(45)
set @IdentificadorNombre=(select NombreDepartamento from Departamentos where NombreDepartamento=@NuevoNombreDepartamento)

    if(@IdDepartamento=@IdentificadorID)
    begin
            if(@NuevoNombreDepartamento = '')
            begin
            print 'No puede estar en blanco'
            end
            else
            begin
                if(@NuevoNombreDepartamento = @IdentificadorNombre)
                begin
             print 'no se puede duplicar'
            end
            else
            begin
            update Departamentos set NombreDepartamento=@NuevoNombreDepartamento
            where IdDepartamento = @IdDepartamento and Estado = 1
            end
   end
end
else
begin
  print 'No se localizo'
end
Go
```

Mostrar Departamentos activos

DepartamentosActivos := σEstado = 1(Departamentos);

```
--
-- Listar Deptos Activos
create procedure ListarDepartamentos
as
select * from Departamentos where Estado = 1
Go
EXEC SP_HELPCONSTRAINT @OBJNAME = 'Proveedores'
```

Agregar municipio

AgregarMunicipio(IdMunicipio: INT, Nombre: VARCHAR(30), IdDepartamento: INT) :=

ΔMunicipios ← ρID/IdMunicipio(IdMunicipio = IdMunicipio)(Municipios)

⋈ ρN/Nombre(Nombre = Nombre)(Municipios)

⋈ ρD/IdDepartamento(IdDepartamento = IdDepartamento)(Departamentos);

```
/*********************************************/
CREATE PROCEDURE AgregarMunicipio
    @NombreMunicipio NVARCHAR(30),
    @IdDepartamento INT
AS
BEGIN
    -- Verificar si el departamento existe
    IF EXISTS (SELECT 1 FROM Departamentos WHERE IdDepartamento = @IdDepartamento)
    BEGIN
    If (@NombreMunicipio = '' or @IdDepartamento = '')
    Begin
        print 'Campo requerido vacio'
    End
        Else
        Begin
        -- Insertar el nuevo municipio
        INSERT INTO Municipios(NombreMunicipio, IdDepartamento)
        VALUES (@NombreMunicipio, @IdDepartamento);
        PRINT 'Municipio agregado exitosamente.';
        End
    END
    ELSE
    BEGIN
        PRINT 'El departamento no existe. No se pudo agregar el municipio.';
    END
END
GO
```

Actualizar Municipio

ActualizarMunicipio(IdMunicipio: INT, NuevoNombre: VARCHAR(30), NuevoIdDepartamento: INT) :=

ΔMunicipios(Nombre, IdDepartamento) ← ρID/IdMunicipio(IdMunicipio = IdMunicipio)(Municipios)

⋈ ρN/NuevoNombre(NuevoNombre = Nombre)(Municipios)

⋈ ρD/NuevoIdDepartamento(NuevoIdDepartamento = IdDepartamento)(Departamentos);

```
--Actualizar Municipio
create procedure ActualizarMunicipio
@IdMunicipio int,
@NuevoNombreMunicipio nvarchar(30)
as
declare @IdentificadorID as int
set @IdentificadorID =(select IdMunicipio from Municipios where IdMunicipio=@IdMunicipio)
declare @IdentificadorNombre as nvarchar(45)
set @IdentificadorNombre=(select NombreMunicipio from Municipios where NombreMunicipio=@NuevoNombreMunicipio)

    if(@IdMunicipio=@IdentificadorID)
    begin
            if(@NuevoNombreMunicipio = '')
            begin
            print 'No puede estar en blanco'
            end
            else
            begin
                if(@NuevoNombreMunicipio = @IdentificadorNombre)
                begin
             print 'no se puede duplicar'
            end
            else
            begin
            update Municipios set NombreMunicipio=@NuevoNombreMunicipio
            where IdMunicipio = @IdMunicipio
            end
   end
end
else
begin
  print 'No se localizo'
end
Go
```

Eliminar Municipio.

EliminarMunicipio(IdMunicipio: INT) :=

ΔMunicipios ← σIdMunicipio ≠ IdMunicipio(Municipios);

Agregar proveedor

AgregarProveedor(IdProveedor: INT, RazonSocial: VARCHAR(50), RUC: VARCHAR(11), NombreComercial: VARCHAR(50), Representante: VARCHAR(50), Direccion: VARCHAR(100), IdMunicipio: INT, Telefono: VARCHAR(15), Estado: BIT) :=

ΔProveedores ← ρID/IdProveedor(IdProveedor = IdProveedor)(Proveedores)

⋈ ρR/RazonSocial(RazonSocial = RazonSocial)(Proveedores)

⋈ ρRUC/RUC(RUC = RUC)(Proveedores)

⋈ ρNC/NombreComercial(NombreComercial = NombreComercial)(Proveedores)

⋈ ρRep/Representante(Representante = Representante)(Proveedores)

⋈ ρDir/Direccion(Direccion = Direccion)(Proveedores)

⋈ ρM/IdMunicipio(IdMunicipio = IdMunicipio)(Municipios)

⋈ ρTel/Telefono(Telefono = Telefono)(Proveedores)

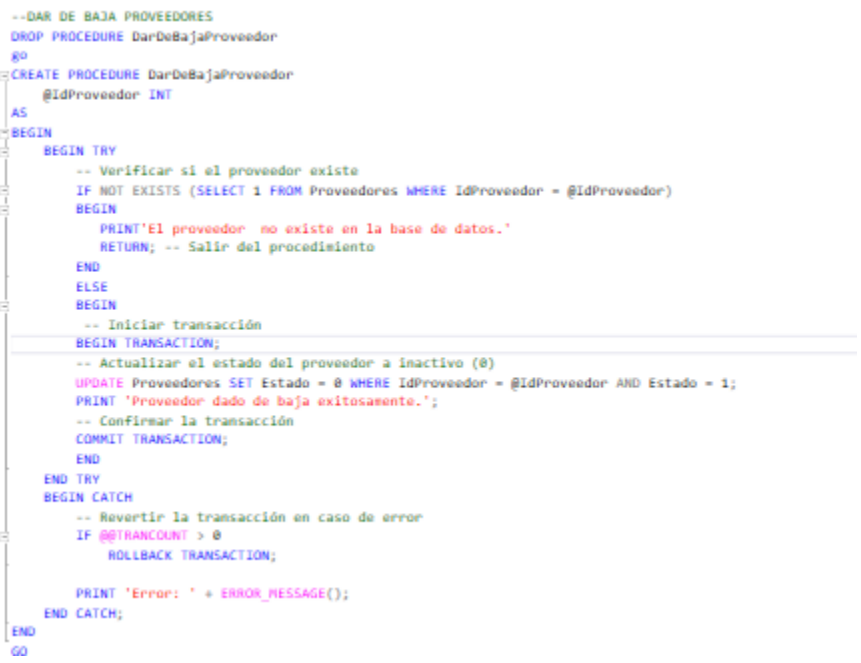
⋈ ρE/Estado(Estado = Estado)(Proveedores);

```sql
--Agregar proveedor
DROP PROCEDURE AgregarProveedor
GO
CREATE PROCEDURE AgregarProveedor
    @RazonSocial NVARCHAR(60),
    @RUC NVARCHAR(30),
    @NombreComercial NVARCHAR(50),
    @Representante NVARCHAR(50),
    @Direccion NVARCHAR(60),
    @IdMunicipio INT,
    @Telefono CHAR(8)
AS
BEGIN
    BEGIN TRY
        -- Verificar campos nulos
        IF (@RUC IS NULL OR @NombreComercial IS NULL OR @Telefono IS NULL OR @RazonSocial IS NULL OR @Direccion IS NULL OR @IdMunicipio IS NULL)
        BEGIN
            Print'Los campos RUC, Nombre Comercial, Teléfono y Razón Social no pueden ser nulos'
            RETURN; -- Salir del procedimiento
        END

        -- Verificar duplicados en RUC, Nombre Comercial, Teléfono y Razón Social
        IF EXISTS (SELECT 1 FROM Proveedores WHERE RUC = @RUC OR NombreComercial = @NombreComercial OR Telefono = @Telefono OR RazonSocial = @RazonSocial OR Direccion = @Direccion)
        BEGIN
            Print'El RUC, Nombre Comercial, Teléfono o Razón Social ya existen en la base de datos.'
            RETURN; -- Salir del procedimiento
        END

        -- Iniciar transacción
        BEGIN TRANSACTION;

        -- Insertar el nuevo proveedor
        INSERT INTO Proveedores ( RazonSocial,RUC, NombreComercial,Representante, Direccion, IdMunicipio,Telefono)
        VALUES ( @RazonSocial,@RUC, @NombreComercial,  @Representante, @Direccion, @IdMunicipio,@Telefono);

        -- Confirmar la transacción
        COMMIT TRANSACTION;

        PRINT 'Proveedor agregado exitosamente.';
    END TRY
    BEGIN CATCH
        -- Revertir la transacción en caso de error
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        PRINT 'Error: ' + ERROR_MESSAGE();
    END CATCH;
END
```

Eliminar proveedor

EliminarProveedor(IdProveedor: INT) :=

ΔProveedores ← σIdProveedor ≠ IdProveedor(Proveedores);



```
--DAR DE BAJA PROVEEDORES
DROP PROCEDURE DarDeBajaProveedor
go
CREATE PROCEDURE DarDeBajaProveedor
    @IdProveedor INT
AS
BEGIN
    BEGIN TRY
        -- Verificar si el proveedor existe
        IF NOT EXISTS (SELECT 1 FROM Proveedores WHERE IdProveedor = @IdProveedor)
        BEGIN
           PRINT'El proveedor  no existe en la base de datos.'
           RETURN; -- Salir del procedimiento
        END
        ELSE
        BEGIN
         -- Iniciar transacción
        BEGIN TRANSACTION;
        -- Actualizar el estado del proveedor a inactivo (0)
        UPDATE Proveedores SET Estado = 0 WHERE IdProveedor = @IdProveedor AND Estado = 1;
        PRINT 'Proveedor dado de baja exitosamente.';
        -- Confirmar la transacción
        COMMIT TRANSACTION;
        END
    END TRY
    BEGIN CATCH
        -- Revertir la transacción en caso de error
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        PRINT 'Error: ' + ERROR_MESSAGE();
    END CATCH;
END
GO
```

Actualizar proveedor.

ActualizarProveedor(IdProveedor: INT, NuevoNombreComercial: VARCHAR(50), NuevoRepresentante: VARCHAR(50), NuevaDireccion: VARCHAR(100), NuevoIdMunicipio: INT, NuevoTelefono: VARCHAR(15)) :=

ΔProveedores(NombreComercial, Representante, Direccion, IdMunicipio, Telefono) ← σIdProveedor = IdProveedor(Proveedores)

⋈ ρNC/NuevoNombreComercial(NuevoNombreComercial = NombreComercial)(Proveedores)

⋈ ρRep/NuevoRepresentante(NuevoRepresentante = Representante)(Proveedores)

⋈ ρDir/NuevaDireccion(NuevaDireccion = Direccion)(Proveedores)

⋈ ρM/NuevoIdMunicipio(NuevoIdMunicipio = IdMunicipio)(Municipios)

⋈ ρTel/NuevoTelefono(NuevoTelefono = Telefono)(Proveedores);

```sql
--Actualizar proveedor
CREATE PROCEDURE ActualizarProveedor
    @IdProveedor INT,
    @NuevoRUC NVARCHAR(30),
    @NuevoNombreComercial NVARCHAR(50),
    @NuevoTelefono CHAR(8),
    @NuevoRazonSocial NVARCHAR(60),
    @NuevoRepresentante NVARCHAR(50),
    @NuevaDireccion NVARCHAR(60),
    @NuevoIdMunicipio INT

AS
BEGIN
    BEGIN TRY
        -- Verificar si el proveedor existe
        IF NOT EXISTS (SELECT 1 FROM Proveedores WHERE IdProveedor = @IdProveedor)
        BEGIN
            PRINT 'El proveedor no existe en la base de datos.'
            RETURN; -- Salir del procedimiento
        END
        -- Verificar campos nulos en los nuevos valores
        IF (@NuevoRUC IS NULL OR @NuevoNombreComercial IS NULL OR @NuevoTelefono IS NULL OR @NuevoRazonSocial IS NULL)
        BEGIN
            PRINT'Los campos RUC, Nombre Comercial, Teléfono y Razón Social no pueden ser nulos.'
            RETURN; -- Salir del procedimiento
        END

        -- Verificar duplicados en RUC, Nombre Comercial, Teléfono y Razón Social
        IF EXISTS (SELECT 1 FROM Proveedores WHERE (RUC = @NuevoRUC OR NombreComercial = @NuevoNombreComercial OR Telefono = @NuevoTelefono OR RazonSocial = @NuevoRazonSocial) AND IdProveedor <> @IdProveedor)
        BEGIN
            PRINT 'El RUC, Nombre Comercial, Teléfono o Razón Social ya existen en la base de datos para otro proveedor.'
            RETURN; -- Salir del procedimiento
        END

        -- Iniciar transacción
        BEGIN TRANSACTION;

        -- Actualizar los datos del proveedor
        UPDATE Proveedores
        SET
            RazonSocial = @NuevoRazonSocial,
            RUC = @NuevoRUC,
            NombreComercial = @NuevoNombreComercial,
            Representante = @NuevoRepresentante,
            Direccion = @NuevaDireccion,
            IdMunicipio = @NuevoIdMunicipio,
            Telefono = @NuevoTelefono

        WHERE
            IdProveedor = @IdProveedor AND Estado = 1;
```

Agregar categoría.

AgregarCategoria(IdCategoria: INT, Nombre: VARCHAR(50), Descripcion: VARCHAR(100), Estado: BIT) :=

ΔCategorias ← ρID/IdCategoria(IdCategoria = IdCategoria)(Categorias)

⋈ ρN/Nombre(Nombre = Nombre)(Categorias)

⋈ ρD/Descripcion(Descripcion = Descripcion)(Categorias)

⋈ ρE/Estado(Estado = Estado)(Categorias);

```
--Agregar Categoria
CREATE PROCEDURE AgregarCategoria
    @Nombre varchar(30),
    @Descripcion varchar(100)
AS
BEGIN
    BEGIN TRY
        -- Verificar campos nulos
        IF (@Nombre IS NULL OR @Descripcion IS NULL)
        BEGIN
            Print'Los campos Nombre y Descripcion no pueden ser nulos'
            RETURN; -- Salir del procedimiento
        END

        -- Verificar duplicados en RUC, Nombre Comercial, Teléfono y Razón Social
        IF EXISTS (SELECT 1 FROM Categorias WHERE Nombre = @Nombre or Descripcion = @Descripcion)
        BEGIN
            Print'Los campos Nombre y Descripcion ya existen.'
            RETURN; -- Salir del procedimiento
        END

        -- Iniciar transacción
        BEGIN TRANSACTION;

        -- Insertar el nuevo proveedor
        INSERT INTO Categorias(Nombre,Descripcion)
        VALUES ( @Nombre,@Descripcion);

        -- Confirmar la transacción
        COMMIT TRANSACTION;

        PRINT 'Categoria agregada exitosamente.';
    END TRY
    BEGIN CATCH
        -- Revertir la transacción en caso de error
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        PRINT 'Error: ' + ERROR_MESSAGE();
    END CATCH;
END
GO
```

Eliminar categoría.

EliminarCategoria(IdCategoria: INT) := ΔCategorias ← σIdCategoria ≠ IdCategoria(Categorias);

```
--Eliminar Categorias
Create procedure EliminarCategoria
@Identificador int
as
declare @IdCategoria int
set @IdCategoria=(select IdCategoria from Categorias where IdCategoria = @Identificador)
if(@Identificador=@IdCategoria)
begin
  update Categorias set Estado = 0 where IdCategoria=@Identificador and Estado = 1
end
else
begin
   print 'Caegoria no encontrado'
end
```

Actualizar categoría.

ActualizarCategoria(IdCategoria: INT, NuevoNombre: VARCHAR(50), NuevaDescripcion: VARCHAR(100), NuevoEstado: BIT) :=

ΔCategorias(Nombre, Descripcion, Estado) ← σIdCategoria = IdCategoria(Categorias)

⋈ ρNN/NuevoNombre(NuevoNombre = Nombre)(Categorias)

⋈ ρND/NuevaDescripcion(NuevaDescripcion = Descripcion)(Categorias)

⋈ ρNE/NuevoEstado(NuevoEstado = Estado)(Categorias);

```sql
create procedure ActualizarCategoria
    @IdCategoria INT,
    @NuevoNombre NVARCHAR(50),
    @NuevaDescripcion NVARCHAR(255)
AS
BEGIN
    BEGIN TRY
        -- Verificar si la categoría existe y su estado es igual a 1
        IF NOT EXISTS (SELECT 1 FROM Categorias WHERE IdCategoria = @IdCategoria AND Estado = 1)
        BEGIN
            print 'La categoría con ID  no existe o su estado no es activo'
            RETURN; -- Salir del procedimiento
        END

        -- Iniciar transacción
        BEGIN TRANSACTION;

        -- Actualizar la categoría
        UPDATE Categorias
        SET
            Nombre = @NuevoNombre,
            Descripcion = @NuevaDescripcion
        WHERE
            IdCategoria = @IdCategoria;

        -- Confirmar la transacción
        COMMIT;

        PRINT 'Categoría actualizada exitosamente.';
    END TRY
    BEGIN CATCH
        -- Revertir la transacción en caso de error
        IF @@TRANCOUNT > 0
            ROLLBACK;

        PRINT 'Error: ' + ERROR_MESSAGE();
    END CATCH;
END

Go
```

Unidades de medidas.

AgregarUnidadDeMedida(IdUnidad: INT, Nombre: VARCHAR(50), Abreviatura: VARCHAR(10)) :=

ΔUnidadesDeMedida ← ρID/IdUnidad(IdUnidad = IdUnidad)(UnidadesDeMedida)

⋈ ρN/Nombre(Nombre = Nombre)(UnidadesDeMedida)

⋈ ρA/Abreviatura(Abreviatura = Abreviatura)(UnidadesDeMedida);

```sql
CREATE PROCEDURE AgregarUnidadMedida
    @Nombre NVARCHAR(30),
    @Abreviatura NVARCHAR(5)
AS
BEGIN
    BEGIN TRY
        -- Verificar campos no nulos
        IF (@Nombre IS NULL OR @Abreviatura IS NULL)
        BEGIN
            print'Los campos Nombre y Abreviatura no pueden ser nulos.'
            RETURN; -- Salir del procedimiento
        END

        -- Iniciar transacción
        BEGIN TRANSACTION;

        -- Insertar la nueva unidad de medida
        INSERT INTO UnidadesMedida(Nombre, Abreviatura)
        VALUES (@Nombre, @Abreviatura);

        -- Confirmar la transacción
        COMMIT;

        PRINT 'Unidad de medida agregada exitosamente.';
    END TRY
    BEGIN CATCH
        -- Revertir la transacción en caso de error
        IF @@TRANCOUNT > 0
            ROLLBACK;

        PRINT 'Error: ' + ERROR_MESSAGE();
    END CATCH;
END
go
```

Dar de baja.

DarDeBajaUnidadDeMedida(IdUnidad: INT) :=

ΔUnidadesDeMedida ← σIdUnidad ≠ IdUnidad(UnidadesDeMedida);

```sql
CREATE PROCEDURE DarDeBajaUnidadMedida
    @IdUnidadMedida INT
AS
BEGIN
    BEGIN TRY
        -- Verificar si la unidad de medida existe
        IF NOT EXISTS (SELECT 1 FROM UnidadesMedida WHERE IdMedida = @IdUnidadMedida)
        BEGIN
            print 'La unidad de medida  no existe en la base de datos.'
            RETURN; -- Salir del procedimiento
        END

        -- Iniciar transacción
        BEGIN TRANSACTION;

        -- Actualizar el estado de la unidad de medida a 0 (inactivo)
        UPDATE UnidadesMedida
        SET Estado = 0
        WHERE IdMedida = @IdUnidadMedida;

        -- Confirmar la transacción
        COMMIT;

        PRINT 'Unidad de medida dada de baja exitosamente.';
    END TRY
    BEGIN CATCH
        -- Revertir la transacción en caso de error
        IF @@TRANCOUNT > 0
            ROLLBACK;

        PRINT 'Error: ' + ERROR_MESSAGE();
    END CATCH;
END
GO
```

Actualizar.

ActualizarUnidadDeMedida(IdUnidad: INT, NuevoNombre: VARCHAR(50), NuevaAbreviatura: VARCHAR(10)) := ΔUnidadesDeMedida(Nombre, Abreviatura) ← σIdUnidad = IdUnidad(UnidadesDeMedida)

⋈ ρNN/NuevoNombre(NuevoNombre = Nombre)(UnidadesDeMedida)

⋈ ρNA/NuevaAbreviatura(NuevaAbreviatura = Abreviatura)(UnidadesDeMedida);

```sql
CREATE PROCEDURE ActualizarUnidadMedida
    @IdUnidadMedida INT,
    @NuevoNombre NVARCHAR(50),
    @NuevaAbreviatura NVARCHAR(10)
AS
BEGIN
    BEGIN TRY
        -- Verificar si la unidad de medida existe y su estado es igual a 1
        IF NOT EXISTS (SELECT 1 FROM UnidadesMedida WHERE IdMedida = @IdUnidadMedida AND Estado = 1)
        BEGIN
            print 'La unidad de medida no existe'
            RETURN; -- Salir del procedimiento
        END

        -- Verificar campos no nulos
        IF (@NuevoNombre IS NULL OR @NuevaAbreviatura IS NULL)
        BEGIN
            print 'Los campos Nombre y Abreviatura no pueden ser nulos.'
            RETURN; -- Salir del procedimiento
        END

        -- Iniciar transacción
        BEGIN TRANSACTION;

        -- Actualizar la unidad de medida
        UPDATE UnidadesMedida
        SET
            Nombre = @NuevoNombre,
            Abreviatura = @NuevaAbreviatura
        WHERE
            IdMedida = @IdUnidadMedida;

        -- Confirmar la transacción
        COMMIT;

        PRINT 'Unidad de medida actualizada exitosamente.';
    END TRY
    BEGIN CATCH
        -- Revertir la transacción en caso de error
        IF @@TRANCOUNT > 0
            ROLLBACK;

        PRINT 'Error: ' + ERROR_MESSAGE();
    END CATCH;
END
```

Agregar productos.

AgregarProducto(IdProducto: INT, Nombre: VARCHAR(50), Descripcion: VARCHAR(100), PrecioVenta: DECIMAL(10, 2), IDCategoria: INT, IdUnidadMedida: INT, Existencia: INT) :=

ΔProductos ← ρID/IdProducto(IdProducto = IdProducto)(Productos)

⋈ ρN/Nombre(Nombre = Nombre)(Productos)

⋈ ρD/Descripcion(Descripcion = Descripcion)(Productos)

⋈ ρPV/PrecioVenta(PrecioVenta = PrecioVenta)(Productos)

⋈ ρC/IDCategoria(IDCategoria = IDCategoria)(Productos)

⋈ ρUM/IdUnidadMedida(IdUnidadMedida = IdUnidadMedida)(Productos)

⋈ ρE/Existencia(Existencia = Existencia)(Productos);

```
CREATE PROCEDURE InsertarProducto
    @Nombre NVARCHAR(50),
    @Descripcion NVARCHAR(50),
    @PrecioVenta FLOAT,
    @IDCategoria INT,
    @IDUnidadesMedida INT,
    @Existencia INT
AS
BEGIN
    -- Validar campos no nulos
    IF (@Nombre IS NULL OR @PrecioVenta IS NULL OR @IDCategoria IS NULL OR @IDUnidadesMedida IS NULL)
    BEGIN
        PRINT'Los campos Nombre, PrecioVenta, IDCategoria e IDUnidadesMedida no pueden ser nulos.'
        RETURN; -- Salir del procedimiento
    END

    -- Validar si el nombre ya existe
    IF EXISTS (SELECT 1 FROM Producto WHERE Nombre = @Nombre)
    BEGIN
        PRINT'Ya existe un producto con el mismo nombre.'
        RETURN; -- Salir del procedimiento
    END

    -- Insertar el nuevo producto
    INSERT INTO Producto (Nombre, Descripcion, PrecioVenta, IdCategoria, IdMedida, Existencia)
    VALUES (@Nombre, @Descripcion, @PrecioVenta, @IDCategoria, @IDUnidadesMedida, @Existencia)
END
```

Actualiar productos.

ActualizarProducto(IdProducto: INT, NuevoNombre: VARCHAR(50), NuevaDescripcion: VARCHAR(100), NuevoPrecioVenta: DECIMAL(10, 2), NuevaCategoria: INT, NuevaUnidadMedida: INT, NuevaExistencia: INT) :=

ΔProductos(Nombre, Descripcion, PrecioVenta, IDCategoria, IdUnidadMedida, Existencia) ← σIdProducto = IdProducto(Productos)

⋈ ρNN/NuevoNombre(NuevoNombre = Nombre)(Productos)

⋈ ρND/NuevaDescripcion(NuevaDescripcion = Descripcion)(Productos)

⋈ ρNPV/NuevoPrecioVenta(NuevoPrecioVenta = PrecioVenta)(Productos)

⋈ ρNC/NuevaCategoria(NuevaCategoria = IDCategoria)(Productos)

⋈ ρNUM/NuevaUnidadMedida(NuevaUnidadMedida = IdUnidadMedida)(Productos)

⋈ ρNE/NuevaExistencia(NuevaExistencia = Existencia)(Productos);

```sql
CREATE PROCEDURE ActualizarProducto
    @IDProducto INT,
    @NuevoNombre NVARCHAR(50),
    @NuevaDescripcion NVARCHAR(50),
    @NuevoPrecioVenta float,
    @NuevaIDCategoria INT,
    @NuevaIDUnidadesMedida INT,
    @NuevaExistencia INT
AS
BEGIN
    -- Validar campos no nulos
    IF (@NuevoNombre IS NULL OR @NuevoPrecioVenta IS NULL OR @NuevaIDCategoria IS NULL OR @NuevaIDUnidadesMedida IS NULL)
    BEGIN
        PRINT 'Los campos Nombre, PrecioVenta, IDCategoria e IDUnidadesMedida no pueden ser nulos.'
        RETURN; -- Salir del procedimiento
    END

    -- Validar si el nuevo nombre ya existe (excluyendo el producto actual)
    IF EXISTS (SELECT 1 FROM Producto WHERE Nombre = @NuevoNombre AND IDProducto <> @IDProducto)
    BEGIN
        PRINT 'Ya existe un producto con el mismo nombre.'
        RETURN; -- Salir del procedimiento
    END

    -- Actualizar el producto
    UPDATE Producto
    SET
        Nombre = @NuevoNombre,
        Descripcion = @NuevaDescripcion,
        PrecioVenta = @NuevoPrecioVenta,
        IdCategoria = @NuevaIDCategoria,
        IdMedida= @NuevaIDUnidadesMedida,
        Existencia = @NuevaExistencia
    WHERE
        IDProducto = @IDProducto AND Estado = 1;
END
GO
```